DB2 Information Management Software

IBM

DB2 Cube Views

A Primer

Introduce DB2 Cube Views as a key player in the OLAP world

Understand cube models, cubes and optimization

Improve your metadata flow and speed up queries

Corinne Baragoin
Geetha Balasubramaniam
Bhuvana Chandrasekharan
Landon DelSordo
Jan B Lillelund
Julie Maw
Annie Neroda
Paulo Pereira
Jo A Ramos

Redbooks

ibm.com/redbooks

International Technical Support Organization

DB2 Cube Views: A Primer

September 2003

SG24-7002-00

Note: Before using this information and the product it supports, read the information in "Notices" on page xxix.

First Edition (September 2003)

This edition applies to IBM DB2 Universal Database V8.1 Fixpack 2+, IBM DB2 Cube Views V8.1, IBM DB2 Office Connect Analytics Edition V4.0, IBM QMF For Windows V7.2f, Ascential MetaStage V7.0, Meta Integration Model Bridge V3.1, IBM DB2 OLAP Server V8.1, Cognos Series 7, BusinessObjects Enterprise 6, and MicroStrategy V7.2.3.

Note: We recommend that you consult the product documentation or follow-on versions of this redbook for more current information.

Contents

Figures xi

Tables xxv

Examples xxvii

Notices xxix
Trademarks xxx

Preface xxxi
The team that wrote this redbook xxxii
Become a published author xxxv
Comments welcome xxxv

Part 1. Understand DB2 Cube Views 1

Chapter 1. An OLAP-aware DB2 3
1.1 Business Intelligence and OLAP introduction 4
1.1.1 Online Analytical Processing 5
1.1.2 Metadata 8
1.2 DB2 UDB V8.1 becomes OLAP-aware 9
1.3 Challenges faced by DBA's in an OLAP environment 11
1.3.1 Manage the flow of metadata 11
1.3.2 Optimize and manage custom summary tables 11
1.3.3 Optimize MOLAP database loading 12
1.3.4 Enhance OLAP queries performance in the relational database . . . 13
1.4 How DB2 can help 13
1.4.1 Efficient multidimensional model: cube model 14
1.4.2 Summary tables optimization: Optimization Advisor 15
1.4.3 Interfaces 17
1.5 Metadata bridges to back-end and front-end tools 19

Chapter 2. DB2 Cube Views: scenarios and benefits 21
2.1 What can DB2 Cube Views do for you? 22
2.2 Feeding metadata into DB2 Cube Views 29
2.2.1 Feeding DB2 Cube Views from back-end tools 31
2.2.2 Feeding DB2 Cube Views from front-end tools 34
2.2.3 Feeding DB2 Cube Views from scratch 36
2.3 Feeding front-end tools from DB2 Cube Views 39

2.3.1 Supporting MOLAP tools with DB2 Cube Views 40
2.3.2 Supporting ROLAP tools with DB2 Cube Views 46
2.3.3 Supporting HOLAP tools with DB2 Cube Views 50
2.3.4 Supporting bridgeless ROLAP tools with DB2 Cube Views 54
2.4 Feeding Web services from DB2 Cube Views 56
2.4.1 A scenario 57
2.4.2 Flow and components 57
2.4.3 Benefits 58

Part 2. Build and optimize the DB2 Cube Model 61

Chapter 3. Building a cube model in DB2 63
3.1 What are the data schemas that can be modeled? 64
3.1.1 Star schemas 64
3.1.2 Snowflakes 66
3.1.3 Star and snowflakes characteristics 67
3.2 Cube model notion and terminology 67
3.2.1 Measures and facts 68
3.2.2 Attributes 70
3.2.3 Dimensions 71
3.2.4 Hierarchies 73
3.2.5 Attribute relationships 76
3.2.6 Joins 77
3.2.7 In a nutshell: cube model and cubes. 79
3.3 Building cube models using the OLAP Center 81
3.3.1 Planning for building a cube model 85
3.3.2 Preparing the DB2 relational database for DB2 Cube Views 86
3.3.3 Building the cube model by import 87
3.3.4 Building a cube model with Quick Start wizard 91
3.3.5 Creating a basic complete cube model from scratch 92
3.4 Enhancing a cube model. 118
3.4.1 Based on end-user analytics requirements. 119
3.4.2 Based on Optimization Advisor and MQT usage 122
3.5 Backup and recovery. 122
3.6 Summary 124

Chapter 4. Using the cube model for summary tables optimization . . . 125
4.1 Summary tables and optimization requirements 126
4.2 How cube model influences summary tables and query performance . . 127
4.3 MQTs: a quick overview 131
4.3.1 MQTs in general 132
4.3.2 MQTs in DB2 Cube Views 134
4.4 What you need to know before optimizing 136
4.4.1 Get at least a cube model and one cube defined 136

4.4.2 Define referential integrity or informational constraints 136
4.4.3 Do you know or have an idea of the query type? 143
4.4.4 Understand how Optimization Advisor uses cube model/cube 149
4.5 Using the Optimization Advisor 152
4.5.1 How does the wizard work 152
4.5.2 Check your cube model 154
4.5.3 Run the Optimization Advisor 157
4.5.4 Parameters for the Optimization Advisor 160
4.6 Deploying Optimization Advisor MQTs 169
4.6.1 What SQL statements are being run? 172
4.6.2 Are the statements using the MQTs? 173
4.6.3 How deep in the hierarchies do the MQTs go? 178
4.6.4 Check the DB2 parameters 181
4.6.5 Is the query optimization level correct? 185
4.7 Optimization Advisor and cube model interactions 185
4.7.1 Optimization Advisor recommendations 187
4.7.2 Query to the top of the cube 189
4.7.3 Querying a bit further down the cube 191
4.7.4 Moving towards the middle of the cube 194
4.7.5 Visiting the bottom of the cube 197
4.8 Performance considerations 197
4.9 Further steps in MQT maintenance 198
4.9.1 Refresh DEFERRED option 199
4.9.2 Refresh IMMEDIATE option 200
4.9.3 Refresh DEFERRED versus refresh IMMEDIATE 201
4.9.4 INCREMENTAL refresh versus FULL refresh 202
4.9.5 Implementation guidelines 203
4.9.6 Limitations for INCREMENTAL refresh 214
4.10 MQT tuning 215
4.11 Configuration considerations 216
4.11.1 Estimating memory required for MQTs 216
4.11.2 Estimating space required for MQTs 216
4.12 Conclusion 218

Part 3. Access dimensional data in DB2 219

Chapter 5. Metadata bridges overview 221
5.1 A quick summary 222

Chapter 6. Accessing DB2 dimensional data using Office Connect . . . 225
6.1 Product overview 226
6.2 Architecture and components 226
6.3 Accessing OLAP metadata and data in DB2 228
6.3.1 Prepare metadata 228

6.3.2 Launch Excel and load Office Connect Add-in 229
6.3.3 Connect to OLAP-aware database (data source) in DB2 230
6.3.4 Import cube metadata 230
6.3.5 Bind data to Excel worksheet 233
6.4 OLAP style operations in Office Connect 235
6.5 Saving and deleting reports. 240
6.6 Refreshing data 240
6.7 Optimizing for better performance. 240
6.7.1 Enable SQLDebug trace in Office Connect. 241
6.7.2 Use DB2 Explain to check if SQL is routed to the MQT 243
6.7.3 Scenario demonstrating benefit of optimization 244

Chapter 7. Accessing dimensional data in DB2 using QMF for Windows . . 247
7.1 QMF product overview 248
7.2 Evolution of QMF to DB2 Cube Views support 248
7.3 Components involved 249
7.4 Using DB2 Cube Views in QMF for Windows 250
7.4.1 QMF for Windows OLAP Query wizard. 251
7.4.2 Multidimensional data modeling 254
7.4.3 Object Explorer 254
7.4.4 Layout Designer 255
7.4.5 Query Results View. 257
7.5 OLAP report examples and benefits 263
7.5.1 Who can use OLAP functionality?. 264
7.5.2 Before starting. 264
7.5.3 Sales analysis scenario. 265
7.6 Maintenance 268
7.6.1 Invalidation of OLAP queries. 268
7.6.2 Performance issues 268
7.7 Conclusion. 269

Chapter 8. Using Ascential MetaStage and the DB2 Cube Views MetaBroker 271
8.1 Ascential MetaStage product overview 272
8.1.1 Managing metadata with MetaStage. 276
8.2 Metadata flow scenarios with MetaStage 281
8.2.1 Importing ERwin dimensional metadata into DB2 Cube Views. . . . 281
8.2.2 Leveraging existing enterprise metadata with MetaStage 289
8.2.3 Performing cross-tool impact analysis 295
8.2.4 Performing data lineage and process analysis in MetaStage 308
8.3 Conclusion: benefits 327

Chapter 9. Meta Integration of DB2 Cube Views within the enterprise toolset

329
9.1 Meta Integration Technology products overview 330
9.1.1 Meta Integration Works (MIW) 330
9.1.2 Meta Integration Repository (MIR) 332
9.1.3 Meta Integration Model Bridge (MIMB) 332
9.2 Architecture and components involved 333
9.3 Metadata flow scenarios 334
9.4 Metadata mapping and limitations considerations 337
9.4.1 Forward engineering from a relational model to a cube model 337
9.4.2 Reverse engineering of a cube model into a relational model 338
9.5 Implementation steps scenario by scenario 338
9.5.1 Metadata integration of DB2 Cube Views with ERwin v4.x 339
9.5.2 Metadata integration of DB2 Cube Views with ERwin v3.x 352
9.5.3 Metadata integration of DB2 Cube Views with PowerDesigner . . . 365
9.5.4 Metadata integration of DB2 Cube Views with IBM Rational Rose . 378
9.5.5 Metadata integration of DB2 Cube Views with CWM and XMI 393
9.5.6 Metadata integration of DB2 Cube Views with DB2 Warehouse Manager 399
9.5.7 Metadata integration of DB2 Cube Views with Informatica 408
9.6 Refresh considerations 412
9.7 Conclusion: benefits 414

Chapter 10. Accessing DB2 dimensional data using Integration Server Bridge 417
10.1 DB2 OLAP Server and Integration Server bridge 418
10.1.1 Integration Server 419
10.1.2 Hybrid Analysis 420
10.1.3 Integration Server Bridge 421
10.2 Metadata flow scenarios 422
10.2.1 DB2 OLAP Server and DB2 Cube Views not installed 423
10.2.2 DB2 OLAP Server and IS installed, but not DB2 Cube Views . . . 425
10.2.3 DB2 OLAP Server installed, but not IS and DB2 Cube Views . . . 426
10.2.4 DB2 Cube Views installed, but not DB2 OLAP Server 427
10.3 Implementation steps 427
10.3.1 Metadata flow from DB2 Cube Views to Integration Server 428
10.3.2 Metadata flow from Integration Server to DB2 Cube Views 439
10.4 Maintenance 448
10.5 DB2 OLAP Server examples and benefits 449
10.5.1 Data load 450
10.5.2 Hybrid Analysis 458
10.5.3 Drill through reports 477
10.6 Conclusions 482

Chapter 11. Accessing DB2 dimensional data using Cognos 483
11.1 The Cognos solution 484
11.1.1 Cognos Business Intelligence 484
11.2 Architecture and components involved 487
11.3 Implementation steps 492
11.4 Implementation considerations 498
11.4.1 Optimizing drill through 502
11.4.2 Optimizing Impromptu reports 510
11.4.3 Implementation considerations: mappings 513
11.4.4 Enhancing the DB2 cube model 520
11.5 Cube model refresh considerations. 523
11.6 Scenarios 524
11.6.1 Sales analysis scenario. 524
11.6.2 Financial analysis scenario 534
11.6.3 Performance results with MQT 539
11.7 Conclusion: benefits 541

Chapter 12. Accessing DB2 dimensional data using BusinessObjects 543
12.1 Business Objects product overview 544
12.1.1 BusinessObjects Enterprise 6 544
12.2 BusinessObjects Universal Metadata Bridge overview 546
12.2.1 Metadata mapping 548
12.2.2 Complex measure mapping 553
12.2.3 Data type conversion 558
12.3 Implementation steps 559
12.3.1 Export metadata from DB2 OLAP Center 560
12.3.2 Import the metadata in the universe using Application Mode 562
12.3.3 Import the metadata in the universe using API mode 567
12.3.4 Import the metadata in the universe using the batch mode 568
12.3.5 Warning messages 569
12.4 Reports and queries examples 571
12.4.1 Query 1 573
12.4.2 Query 2 576
12.4.3 Query 3 578
12.5 Deployment 581
12.5.1 Optimization tips 582
12.6 Conclusion: benefits 583
12.6.1 Universe creation 583
12.6.2 Improving response time with MQTs. 583

Chapter 13. Accessing DB2 dimensional data using MicroStrategy . . . 585
13.1 MicroStrategy product introduction 586
13.2 Architecture and components involved 587

13.3 Implementation steps 588
13.3.1 Installation 589
13.3.2 Prerequisites 589
13.3.3 Import 589
13.4 Mapping considerations and metadata refresh 592
13.4.1 Mapping fundamentals 592
13.4.2 Assumptions and best practices 594
13.4.3 Metadata refresh 597
13.5 Reports and query examples 597
13.5.1 The business case and the business questions 597
13.5.2 Question 1: department contributions to sales 598
13.5.3 Question 2: campaign contributions to sales 604
13.5.4 Question 3: ranking the campaigns by region 608
13.5.5 Question 4: obtaining the Top 5 campaigns 609
13.5.6 Question 5: campaign impact by age range 610
13.6 Conclusion: benefits 612

Chapter 14. Web services for DB2 Cube Views 613
14.1 Web services for DB2 Cube Views: advantages 614
14.2 Overview of the technologies used 615
14.2.1 Web services technology 615
14.2.2 XML 616
14.2.3 SOAP 617
14.2.4 WSDL 617
14.2.5 UDDI 618
14.2.6 XPath 618
14.3 Architecture of Web services for DB2 Cube Views 619
14.4 Web services for DB2 Cube Views 620
14.4.1 Describe 622
14.4.2 Members 627
14.4.3 Execute 631
14.5 Conclusion. 635

Part 4. Appendixes 637

Appendix A. DataStage: operational process metadata configuration and DataStage job example 639
Configure the operational metadata components 640
Configure the server machine 640
Configure the client 644
Creating DataStage Server jobs 647

Appendix B. Hybrid Analysis query performance results 661

Appendix C. FAQs, diagnostics, and tracing 669
Setup questions 669
Metadata questions 670
OLAP Center 670
Tracing 671

Appendix D. DB2 Cube Views stored procedure API 673
API architecture overview 674
Purposes and functionality of the API 675
The stored procedure interface 677
Example 678
Error logging and tracing 679
db2mdapiclient utility 680

Appendix E. The case study: retail datamart 685
The cube model 686
The cube 687
Tables in the star schema 687
MQT 694

Related publications 697
IBM Redbooks 697
Other publications 697
Online resources 698
How to get IBM Redbooks 699

Index 701

Figures

1-1 Simple multidimensional model representation 5
1-2 Database slices 6
1-3 DB2 Cube Views and MQTs 10
1-4 DB2 Cube Views metadata 14
1-5 MQT optimization 16
1-6 Web services applications 18
1-7 OLAP metadata bridges 19
2-1 Your star schema database 22
2-2 Mapping your star schema to OLAP objects 23
2-3 Sharing OLAP metadata with reporting tools 24
2-4 Using aggregates 25
2-5 The Optimization Advisor gathering its aggregate intelligence 27
2-6 The big picture 28
2-7 Metadata flows 29
2-8 Relational to OLAP metadata mappings 30
2-9 DB2 Cube Views metadata mappings 30
2-10 Metadata exchange with back-end tools 33
2-11 Metadata exchange with front-end tools 35
2-12 Entering metadata from scratch 37
2-13 Data and metadata exchange 39
2-14 MOLAP with higher grain from that of star schema 41
2-15 MOLAP with different dimensionality from that of star schema 42
2-16 Data and metadata exchange with MOLAP tools 43
2-17 MOLAP scenario 44
2-18 Data and metadata exchange with ROLAP tools 47
2-19 ROLAP scenario: drill-down 48
2-20 ROLAP scenario: report 49
2-21 HOLAP scenario 51
2-22 Data and metadata exchange with HOLAP tools 52
2-23 Data and metadata exchange with bridgeless ROLAP tools 55
2-24 Data and metadata exchange as a Web Service 58
3-1 Star schema 65
3-2 Snowflake schema 66
3-3 Relational data schema in DB2 68
3-4 Facts object 69
3-5 Facts and measures 70
3-6 Attributes on dimension tables 71
3-7 Dimension and attributes 72

3-8 Hierarchy in a Time dimension 73
3-9 Balanced hierarchy 74
3-10 Unbalanced hierarchy 74
3-11 Ragged hierarchy. 75
3-12 Dimension, attributes, hierarchies, and attribute-relationships 77
3-13 Joins. 78
3-14 Complete layered architecture of a cube model 79
3-15 Cube model and cubes 81
3-16 Launching OLAP Center 82
3-17 OLAP Center architecture 83
3-18 Cube model building methods 84
3-19 Connecting to database 87
3-20 OLAP Center import. 89
3-21 Choose Metadata Source File 89
3-22 Import options 90
3-23 Imported cube model and cube 91
3-24 Create Cube Model wizard 92
3-25 Create Cube Model wizard 93
3-26 Provide cube model name 94
3-27 Create the facts object 95
3-28 Facts object’s name 96
3-29 Select the Facts table 97
3-30 Select Measures 98
3-31 Create calculated measure 99
3-32 Create Dimension 100
3-33 Provide name of dimension object 101
3-34 Select Dimension Table 102
3-35 Select Dimension Attributes 103
3-36 Select Dimension Type 104
3-37 Specify Fact-Dimension Join or create a new join 104
3-38 Create Time_Fact join 105
3-39 Dimension created 106
3-40 Attributes for facts objected created implicitly 107
3-41 Create the hierarchy 108
3-42 Name the hierarchy 109
3-43 Select elements of the hierarchy 110
3-44 Sample hierarchy deployment 110
3-45 Specify related attributes. 111
3-46 Related attributes selection 112
3-47 Finish creating hierarchy 113
3-48 Complete cube model 114
3-49 Create the cube 115
3-50 Name and schema for the cube. 115

3-51 Select from available measures . . . 116
3-52 Select the cube dimensions . . . 117
3-53 Select the cube hierarchy . . . 118
3-54 Ragged hierarchy . . . 121
3-55 Hierarchy flowchart . . . 121
4-1 The Optimization Advisor . . . 129
4-2 Star schema example . . . 139
4-3 Create the MQT . . . 139
4-4 Run the query . . . 140
4-5 Sample star schema . . . 142
4-6 Drill down . . . 144
4-7 Report . . . 145
4-8 Extract . . . 146
4-9 Drill through . . . 148
4-10 Optimization Advisor wizard . . . 153
4-11 Sales cube model . . . 155
4-12 Sales Cube . . . 156
4-13 Menu selection to access Optimization Advisor . . . 158
4-14 Query types . . . 161
4-15 Specify Limitations . . . 163
4-16 Summary tables . . . 165
4-17 Summary . . . 166
4-18 Expected disk space usage . . . 167
4-19 Expected disk space usage and refresh DEFERRED . . . 167
4-20 MQTs implementation steps process . . . 169
4-21 Explain SQL . . . 174
4-22 Explain SQL statement dialog . . . 175
4-23 Explain SQL statement result . . . 176
4-24 Explain SQL statement result without MQT . . . 177
4-25 A cube model hierarchy . . . 180
4-26 The top five most profitable consumer groups without MQTs . . . 190
4-27 The top five most profitable consumer groups with MQTs . . . 191
4-28 Sales amount and quantity by region area without MQTs . . . 192
4-29 Sales amount and quantity by region area with MQTs . . . 193
4-30 Sales through coupons campaign access graph . . . 195
4-31 Sales through Coupon campaigns with MQTs . . . 196
4-32 MQTs: FULL refresh for DEFERRED and IMMEDIATE . . . 204
4-33 Incremental refresh on MQts IMMEDIATE . . . 209
4-34 Incremental refresh on MQTs DEFERRED . . . 211
6-1 Architecture of Office Connect environment . . . 227
6-2 Process flow chart for Office Connect . . . 228
6-3 Office Connect . . . 229
6-4 Office Connect Add-in . . . 229

6-5 Provide connection information . . . 230
6-6 Cube Import Wizard . . . 231
6-7 Select cube(s) for import . . . 232
6-8 Imported cube metadata . . . 232
6-9 Export data to Microsoft Excel . . . 233
6-10 Select Excel sheet . . . 233
6-11 View data in Excel spreadsheet . . . 234
6-12 Show Pivot table field list . . . 235
6-13 Drag and drop . . . 236
6-14 Member selection or filtering . . . 237
6-15 PivotTable wizard . . . 238
6-16 Layout wizard . . . 239
6-17 PivotChart . . . 239
6-18 SQLDebug . . . 242
6-19 Access plan graph . . . 243
6-20 Customized report . . . 244
6-21 Access plan graph - STORE . . . 245
6-22 Access plan graph - PRODUCT . . . 246
7-1 Components required for QMF for Windows with DB2 Cube Views . . 250
7-2 New object window for QMF for Windows . . . 251
7-3 List of queries saved at the server . . . 251
7-4 OLAP Query wizard server . . . 252
7-5 OLAP Query wizard sort . . . 253
7-6 OLAP Query wizard cube schema . . . 253
7-7 OLAP Query wizard cube . . . 254
7-8 View of the cube in Object Explorer . . . 255
7-9 Hierarchy levels in Object Explorer . . . 255
7-10 Default Layout Designer toolbar . . . 256
7-11 Layout Designer without enable online mode option . . . 256
7-12 Default filter window . . . 258
7-13 Filter window with options . . . 259
7-14 Formatting options . . . 260
7-15 Drill down operation . . . 261
7-16 Drill up operations . . . 262
7-17 Roll up operations . . . 262
7-18 Slices of the product dimension . . . 263
7-19 Portion of CONSUMER table from a relational view . . . 263
7-20 Sales cube example in DB2 OLAP Center . . . 265
7-21 OLAP report 1: most profitable consumer groups in the West region . 266
7-22 OLAP report 2: most profitable sales . . . 267
7-23 OLAP report 3: consumer buying trends . . . 268
7-24 Resource Limits Group in QMF for Windows Administrator. . . . 269
8-1 Ascential Enterprise Integration Suite . . . 272

8-2 Metadata flow. 276
8-3 Recommended metadata flow 277
8-4 Metadata flow with DB2 Cube Views 279
8-5 Detailed metadata flow 280
8-6 ERwin 4.1 sales star schema. 282
8-7 ERwin 4.1 dimensional dialog 283
8-8 Summary of using ERwin 4.1 dimensional metadata 284
8-9 MetaStage ERwin import dialog. 284
8-10 ERwin import parameters dialog 285
8-11 ERwin sales model imported into MetaStage 286
8-12 MetaStage new subscription dialog 287
8-13 DB2 Cube Views export parameters 288
8-14 DB2 Cube Views Sales model from ERwin 289
8-15 Hyperion Essbase Integration Server cube model. 290
8-16 Summary metadata flow from Hyperion Essbase to DB2 Cube Views 291
8-17 MetaStage Hyperion import dialog. 291
8-18 Hyperion import parameters dialog 292
8-19 Hyperion metadata in MetaStage 293
8-20 DB2 Cube Views subscription to Hyperion metadata 294
8-21 DB2 Cube Views MetaBroker parameters 295
8-22 ERwin Sales data model 296
8-23 OLAP Center export dialog 297
8-24 MetaStage import selection dialog. 298
8-25 DB2 Cube Views MetaBroker parameters 299
8-26 MetaStage after ERwin and DB2 Cube Views metadata import 300
8-27 Object connector dialog 300
8-28 Impact analysis report showing Connected_To relationships 301
8-29 Switch from sourcing ERwin metadata view 302
8-30 Select IBM DB2 Cube Views of ERwin metadata 303
8-31 ERwin import category. 304
8-32 Browse from ERwin 305
8-33 ERwin Where Used impact analysis menu 306
8-34 Impact analysis without connected objects via creation view 307
8-35 Impact analysis path viewer with creation view context. 308
8-36 MetaStage data lineage example 309
8-37 Process analysis example 311
8-38 DataStage job flow. 312
8-39 Operational metadata components 313
8-40 MetaStage: new import category 314
8-41 Importing multiple DataStage job designs from a DataStage project . 315
8-42 MetaStage: DataStage import 315
8-43 DataStage login 316
8-44 MetaStage: computer instances 318

8-45 DataStage Director 319
8-46 DataStage run results 319
8-47 RunImport output 321
8-48 MetaStage category browser 322
8-49 Data lineage menu 323
8-50 Data lineage path 323
8-51 MetaStage category browser 324
8-52 Browse from menu 325
8-53 Process analysis menu 326
8-54 Process analysis path 326
8-55 Inspect event 327
9-1 A sample of typical metadata movement solutions 330
9-2 Meta Integration functionality 331
9-3 Meta Integration architecture 332
9-4 Meta Integration supported tools 333
9-5 A metadata integration solution example 334
9-6 Business cases for metadata movement solutions 335
9-7 Possible metadata movement solutions for DB2 Cube Views 336
9-8 Metadata movement scenarios illustrated in this chapter 336
9-9 The cube model used 339
9-10 Logical view of the ERwinv4 model 340
9-11 Enabling the ERwin v4 dimensional features. 341
9-12 Specifying the table dimensional roles. 342
9-13 DB2 schema generation 343
9-14 Importing the ERwin v4 model into MIMB 345
9-15 Specifying the export bridge parameters 345
9-16 Exporting the model to DB2 Cube Views. 346
9-17 Specifying the XML file to import into OLAP Center 347
9-18 Controlling how the metadata is imported into OLAP Center 347
9-19 The ERwin v4 business names and description are also converted . . 348
9-20 Exporting from the DB2 cube model as XML. 349
9-21 Converting the cube model XML file to an ERwin v4 XML file. 350
9-22 Cube model converted to ERwin v4 with business names 351
9-23 Logical view of the ERWin model 352
9-24 Logical view of the ERwin model 354
9-25 Enabling the ERwin dimensional features 355
9-26 Specifying the tables dimensional roles. 356
9-27 Saving the model as ERX 357
9-28 ERwin names expansion feature 358
9-29 Importing the ERwin model into MIMB. 358
9-30 Specifying the export bridge parameters 359
9-31 Exporting the model to DB2 Cube Views. 360
9-32 Specifying the XML file to import into OLAP Center 360

9-33 Controlling how the metadata is imported into OLAP CEnter 361
9-34 The ERwin business names and descriptions are also converted. . . . 362
9-35 Converting the DB2 cube model XML to an ERwin ERX file 363
9-36 DB2 cube model converted to ERwin 364
9-37 The cube model reversed engineered to ERwin 3.x 365
9-38 Logical view of the PowerDesigner PDM model 367
9-39 Specifying the tables' dimensional type 368
9-40 Adding documentation to the PowerDesigner model. 369
9-41 DB2 schema generation 370
9-42 Importing the PowerDesigner model into MIMB 371
9-43 Specifying the export bridge parameters 372
9-44 Exporting the model to DB2 Cube Views. 373
9-45 Specifying the XML file to import into OLAP Center 373
9-46 Controlling how the metadata is imported into OLAP Center 374
9-47 The PowerDesigner business names and descriptions are converted 375
9-48 Converting the cube model XML file to an PowerDesigner XML file . . 376
9-49 The cube model reverse engineered to PowerDesigner 377
9-50 PowerDesigner business names and descriptions 378
9-51 The Rose object model 380
9-52 The Rose data model. 381
9-53 Create a new database 382
9-54 Define the database properties 382
9-55 Transform to data model option 383
9-56 Transforming an object model into a data model in Rose 383
9-57 Generation of the SQL DDL in Rose 384
9-58 Specifying the import parameters 385
9-59 Importing the Rose model into MIMB. 386
9-60 Specifying the export bridge parameters 386
9-61 Exporting the model to DB2 Cube Views. 387
9-62 Specifying the XML file to import into OLAP Center 388
9-63 Controlling how the metadata is imported into OLAP Center 389
9-64 The Rose objects' business name and description are also converted 390
9-65 Converting the cube model XML file to a Rose MDL file 391
9-66 The cube model converted to Rose Data Modeler 392
9-67 The cube model converted to Rose Object Modeler 393
9-68 The OMG standards 394
9-69 CWM Enablement Showcase 395
9-70 MIMB: Importing the cube model XML file 396
9-71 Specifying the export options: model 397
9-72 Specifying the export options: encoding 398
9-73 MIMB: exporting to CWM 398
9-74 Sample CWM XMI file reverse engineered from a cube model 399
9-75 The Beverage company model in Data Warehouse Center 400

9-76 This is the fact table of the star schema 401
9-77 Starting the CWM export wizard from DB2 Data Warehouse Center . 402
9-78 Selecting the database to be exported to CWM 402
9-79 The CWM XMI file rendered in a browser 403
9-80 MIMB: importing the DB2 Data Warehouse Center CWM XMI file . . . 404
9-81 The sample warehouse Beverage Company imported from CWM . . . 404
9-82 Specifying the export parameters 405
9-83 Choosing a subsetting mode 406
9-84 Subsetting the star schema model. 406
9-85 Exporting the cube model 407
9-86 The Beverage Company star schema imported into DB2 Cube Views 408
9-87 The sample Informatica XML model 409
9-88 Importing the Informatica model 410
9-89 Specifying the export parameters 410
9-90 Exporting the model to DB2 Cube Views. 411
9-91 The cube model as imported in DB2 OLAP Center 412
10-1 OLAP Server architecture 419
10-2 Integration Server 420
10-3 Metadata flow through the Integration Server bridge. 424
10-4 Reverse metadata flow through the Integration Server bridge 425
10-5 Export from DB2 Cube Views 428
10-6 Integration Server Bridge window 430
10-7 The IS bridge from DB2 Cube Views to Integration Server 430
10-8 Use of bridge from DB2 Cube Views to Integration Server 431
10-9 Import model into Integration Server 432
10-10 Import metaoutline into Integration Server. 433
10-11 Result of import into Integration Server model. 434
10-12 Integration Server column renaming in metadata 436
10-13 Integration Server column properties. 437
10-14 Integration Server measure hierarchy 437
10-15 Add back missing columns in Integration Server 439
10-16 Integration Server export 440
10-17 The IS bridge from Integration Server to DB2 Cube Views 441
10-18 Import wizard screen 1. 442
10-19 Import wizard screen 2. 443
10-20 DB2 Cube Views cube 452
10-21 Integration Server metaoutline. 453
10-22 The measure in DB2 Cube Views 454
10-23 The measure in Integration Server 454
10-24 MQT script FROM clause 454
10-25 MQT script GROUP BY clause 455
10-26 Integration Server data load SQL 456
10-27 Load explain with MQT 457

10-28 MQT script GROUP BY GROUPING SETS clause 459
10-29 Hybrid 1 461
10-30 H1_Query 1 462
10-31 H1_Query 2 463
10-32 Hybrid 2 463
10-33 H2_Query 2a 464
10-34 Hybrid 3 465
10-35 H3_Query 1a 466
10-36 H3_Query1b. 466
10-37 Hybrid 4 467
10-38 Hybrid 5 468
10-39 H5_Query 2c 469
10-40 H5_Query 2d 469
10-41 H1_Query 1 without MQT 471
10-42 H1_Query 1 with MQT 472
10-43 Integration Server drill through report definition. 478
10-44 Integration Server drill through report sample 479
10-45 Integration Server drill through report without MQT. 480
10-46 Integration Server drill through report with MQT 481
11-1 Cognos Business Intelligence 485
11-2 Architecture with DB2 Cube Views 488
11-3 Impromptu window 488
11-4 Cognos Metrics Manager, Visualizer and Cognos Query 491
11-5 DB2 Dimensional Metadata wizard implementation steps. 493
11-6 import metadata 494
11-7 Logon to DB2 494
11-8 Cognos file options 495
11-9 Metadata bridge log file 496
11-10 Transformer model default. 496
11-11 Impromptu default view 497
11-12 Default DB2 Cube Views cube model 499
11-13 Transformer model. 500
11-14 Impromptu default view 501
11-15 Transformer model measure drill through setup 503
11-16 Drill through to campaigns by customer and region. 504
11-17 Drill through result 505
11-18 Impromptu drill through reports 506
11-19 the drill through SQL 506
11-20 Drill through query DB2 explain: using MQTs 507
11-21 Drill through DB2 explain: without MQT (lower level access graph) . . 508
11-22 Drill through DB2 explain: without MQT (upper level access graph) . . 509
11-23 Impromptu Report wizard. 510
11-24 Transaction detail rows from the fact table 511

11-25 Query properties 511
11-26 Data definition 512
11-27 Results on aggregate data 512
11-28 Calculated measure in Db2 Cube Views 513
11-29 Calculated measure in Impromptu 514
11-30 Alternate hierarchies in DB2 Cube Views 515
11-31 Alternate hierarchies in PowerPlay Transformer 516
11-32 Reproduce the alternate hierarchies in DB2 Cube Views 517
11-33 Impromptu query 518
11-34 transformer: adding alternate hierarchy 519
11-35 Create alternate drill down 519
11-36 Transformer Model relative time 520
11-37 Transformer model Day of Week 521
11-38 PowerPlay seasonality 521
11-39 PowerPlay seasonality: another example 522
11-40 Transformer Model measure formatting 523
11-41 Scenario 1: report example 1 525
11-42 Scenario 1: report example 2 525
11-43 Scenario 1: report example 3 526
11-44 Scenario 1: report example 4 526
11-45 Scenario 1: report example 5 527
11-46 Scenario 1: report example 6 528
11-47 Scenario 1: report example 7 528
11-48 Scenario 1: report example 8 529
11-49 Scenario 1: report example 9 530
11-50 Scenario 2: report example 1 531
11-51 Scenario 2: report example 2 531
11-52 Scenario 2: report example 3 532
11-53 Scenario 3: add a new calculation 533
11-54 Scenario 3: the graph 533
11-55 Financial scenario: report example 1 534
11-56 Financial scenario: report example 2 535
11-57 Financial scenario: report example 3 535
11-58 Forecasting scenario: Forecast option 536
11-59 Forecasting scenario: result 537
11-60 Create a mobile sub-cube 538
11-61 Open the PowerPlay sub-cube saved 539
12-1 BusinessObjects Enterprise 6 product family 544
12-2 Metadata flow 547
12-3 Metadata mapping 548
12-4 Additional metadata mapping 549
12-5 Cube model to universes mapping 550
12-6 Hierarchies mapping 551

12-7 Joins mapping 552
12-8 Descriptive values mapping 553
12-9 Complex measure with multiple aggregations 554
12-10 DB2 Cube Views XML file for multiple aggregations 555
12-11 AVG aggregation example 556
12-12 MAX aggregation example 556
12-13 SUM aggregation example 557
12-14 Universe result 557
12-15 Data types 558
12-16 Application mode process 559
12-17 API mode process 559
12-18 Batch mode process 560
12-19 Export metadata from DB2 OLAP Center 561
12-20 Export metadata to XML file 562
12-21 Universal Metadata Bridge panel 563
12-22 Choose the XML file to import 563
12-23 Browse the XML file 564
12-24 Specify the cube model and/or cube 565
12-25 Object tree 566
12-26 The universe created 567
12-27 Batch file arguments 569
12-28 Warning messages 570
12-29 Additional warning messages 570
12-30 Get the SQL statement from SQL Viewer 571
12-31 Launch the explain 572
12-32 Check the response time 573
12-33 Report 1 574
12-34 Without MQTS: DB2 explain result 575
12-35 With MQTS: DB2 explain result 576
12-36 Report 2 577
12-37 Report 3 578
12-38 Without MQTS: DB2 explain result 580
12-39 With MQTS: DB2 explain result 581
12-40 Iterative process 582
13-1 MicroStrategy product suite overview 586
13-2 Import process information flow 588
13-3 Import Assistant Dialog 590
13-4 Import Assistant Feedback window 592
13-5 Question 1: report grid 599
13-6 SQL View option 600
13-7 Scroll down to SQL Statements section 601
13-8 DB2 explain for Question 1: without MQT 602
13-9 DB2 explain for question 1: with MQT 603

13-10 Drilling to campaign 604
13-11 Question 2: report grid 605
13-12 DB2 explain for question 2: without MQT 606
13-13 DB2 explain for question 2: with MQT 607
13-14 Question 3: report grid 608
13-15 Question 4: report grid 609
13-16 Question 5: report grid 611
14-1 Web services layered architecture 616
14-2 Web services for DB2 Cube Views Architecture 620
14-3 Web services for DB2 Cube Views 621
14-4 Sales cube 622
14-5 XML Representation of Sales Cube (Part 1 of 2) 623
14-6 XML Representation of Sales Cube (Part 2 of 2) 624
14-7 Metadata for STORE dimension 626
14-8 Dimensions in Sales cube 626
14-9 XML Representation of Dimension members in Sales Cube (1 of 2) . 627
14-10 XML Representation of Dimension members in Sales Cube (2 of 2) . 628
14-11 Dimension Members - STORE dimension 630
14-12 Top level members in DATE dimension. 631
14-13 Slice of Sales Cube 632
A-1 DataStage Administration 641
A-2 Project properties dialog 642
A-3 Services manager under Windows 643
A-4 Directory Administrator 645
A-5 Select data source 645
A-6 MetaStage attach. 648
A-7 ERwin import category. 648
A-8 New import category 649
A-9 New import. 649
A-10 Import selection 650
A-11 ERwin import parameters 651
A-12 ERwin saved as XML. 652
A-13 DB2 Configuration Assistant 652
A-14 New user-defined category 653
A-15 ERwin Sales model User Category 654
A-16 Add Selection to Category 654
A-17 Select Category 655
A-18 Request Publication 655
A-19 MetaStage Subscribe. 656
A-20 New subscription 656
A-21 Subscription options. 657
A-22 DataStage MetaBroker subscription parameters 657
A-23 DataStage client login 658

A-24 DataStage Manager 658
A-25 DataStage Designer: load dimensions. 659
A-26 DataStage Designer: load fact 660
A-27 DataStage Designer: load columns 660
D-1 API architecture 674
D-2 Metadata operations and its parameters 676
D-3 Syntax of md_message stored procedure 677
D-4 How the db2mdapiclient utility works 680
D-5 Usage of the db2mdapiclient utility 681
E-1 The cube model 686
E-2 The cube 687

Tables

1-1 DB2 Cube Views interface options . . . 17
3-1 Deployment of a balanced hierarchy . . . 76
3-2 Overview of cube model building tasks . . . 84
3-3 Creating a cube model from scratch . . . 93
4-1 DBA major work needed - various System Maintained MQT options . 133
4-2 Optimization Advisor query types . . . 150
4-3 Parameters for Optimization Advisor wizard . . . 159
4-4 Percentage disk space used - expected performance improvements. 162
4-5 Implementation tools . . . 171
4-6 Special register names vs. database configuration keywords. . . . 184
4-7 The Optimization Advisor query type behavior . . . 186
4-8 Suggestions for Optimization Advisor query specifications . . . 187
4-9 Initial FULL refresh on refresh DEFERRED and IMMEDIATE MQTs . 205
4-10 FULL refresh on refresh DEFERRED/ IMMEDIATE MQTs . . . 208
4-11 Steps for INCREMENTAL refresh on refresh IMMEDIATE MQTs. . . . 210
4-12 INCREMENTAL refresh on refresh DEFERRED MQT . . . 212
5-1 IBM business partner bridge implementations . . . 223
6-1 Drill down times . . . 246
10-1 Object mapping between DB2 Cube Views and Integration Server . . 422
10-2 Data load performance costs . . . 457
10-3 H1_Query1. . . . 461
10-4 H1_Query 2 members . . . 462
10-5 H3_Query 1 . . . 465
10-6 H1_Query 1 without MQT . . . 473
10-7 H1_Query 1 with MQT . . . 473
10-8 Hybrid Analysis performance results . . . 473
10-9 DB2 OLAP Server calculation times . . . 476
10-10 Integration Server drill through report performance cost . . . 481
11-1 Metadata bridge mapping . . . 498
11-2 Drill-through performance result . . . 509
11-3 Impromptu report performance result . . . 513
11-4 Impromptu report performance result . . . 540
12-1 Query performance result . . . 575
12-2 Query performance result . . . 579
13-1 Mapping summary . . . 593
13-2 Query performance result . . . 603
13-3 Query performance result . . . 607
13-4 Query performance result . . . 609

13-5 Query performance result 610
13-6 Query performance result 612
14-1 Parameters for Describe 625
14-2 Parameters for Members 629
14-3 Parameters for Execute 634
A-1 Client and server summary 640
A-2 Default Process MetaBroker variables. 642
A-3 Listener configuration variables 644
A-4 RunImport configuration parameters 646
B-1 Detailed query performance results 661
D-1 db2mdapiclient utility tasks 680
E-1 Consumer dimension 688
E-2 Campaign dimension 689
E-3 Product dimension 690
E-4 Date dimension 691
E-5 Store Dimension 692
E-6 Fact Table 693

Examples

3-1 Derived measure 69
3-2 Additive, semi-additive measures 119
3-3 Calculated measure: Profit = Sales - Cost 120
3-4 Complex measure 120
3-5 Two parameter function 120
3-6 Calculated attribute 120
3-7 Gender flag and gender description attribute relationship 122
4-1 MQT example 132
4-2 MQT creation 137
4-3 Query that requires referential integrity 137
4-4 Query that does not requires referential integrity 138
4-5 Checking the validity on an informational constraint 140
4-6 The MOLAP outline 146
4-7 An abbreviated MQT 178
4-8 db2advis options 182
4-9 db2advis input file 182
4-10 Query to the top of the cube 189
4-11 Querying down the cube 192
4-12 Query moving towards the middle of the cube. 194
6-1 SQL for a retrieval in Office Connect 242
6-2 Drill down to West region SQL 244
6-3 Drill down to SKINCARE products 244
7-1 Unsupported cube error message 252
8-1 Sample run XML 320
10-1 Profit: calculated measure 437
10-2 Sales per unit: calculated measure 437
10-3 Profit %: calculated measure 438
10-4 Lookup member names for H1_Query 1 470
10-5 Fetch data for H1_Query 1 470
11-1 SQL example 502
12-1 SQL 1 574
12-2 SQL 2 577
12-3 SQL 3 579
13-1 Measure Profit edited 596
13-2 Asymmetric measure example. 596
14-1 XML Representation of the Sales Cube slice 632
14-2 XML ouput for Execute 635
C-1 db2md_config.xml 671

C-2 Olap center trace command 671
D-1 Structure of a retrieval operation (describe) 678
D-2 db2md_config.xml 679
D-3 Using db2mdapiclient utility to import metadata 682
D-4 Using db2mdapiclient utility to export metadata 682
D-5 Using db2mdapiclient utility to validate metadata 682
D-6 Validate.xml 683
E-1 Script to create/refresh summary tables 694

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX 5L™	eserver™	Rational®
AIX®	ibm.com®	Rational Rose®
alphaWorks®	IBM®	Redbooks (logo) ™
CICS®	Illustra™	Redbooks™
DB2 OLAP Server™	iSeries™	SP1®
DB2 Universal Database™	MVS™	TME®
DB2®	Notes®	WebSphere®
eserver ™	QMF™	XDE™

The following terms are trademarks of International Business Machines Corporation and Rational Software Corporation, in the United States, other countries or both:

Rational Rose®	Rational®	XDE™

The following terms are trademarks of other companies:

Ascential™	Ascential Software™	Ascential™ Enterprise Integration Suite
Ascential AuditStage™	DataStage®	
Ascential QualityStage?	MetaBroker®	
Ascential ProfileStage™	MetaStage®	

Meta Integration is a registered trademark of Meta Integration Technology, Inc.

The following terms are trademarks of other companies:

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Preface

Multidimensionality is the primary requirement for an OLAP system, and the cube always refers to the collections of the data that an OLAP system implements.

Business Intelligence and OLAP systems are no longer limited to the privileged few business analysts: they are being democratized by being shared with the rank and file employee demanding Relational Database Management Systems (RDBMS) that are more OLAP-aware.

IBM DB2® Cube Views V8.1 (DB2 Cube Views through the redbook) and its cube model provide DB2 Universal Database™ (DB2 through the redbook) the ability to address multidimensional analysis and become a key player in the OLAP world.

This redbook focuses on the innovative technical functionalities of DB2 Cube Views:

- To define and store multidimensional data in DB2
- To recommend model-based summary tables to speed up query performance and help in building them automatically
- To provide an advanced API to allow other Business Intelligence partners' tools to benefit both metadata exchange

This redbook positions the new functionalities, so DB2 database administrators and Business Intelligence architects can understand and evaluate their applicability in their own Business Intelligence and OLAP system environment. It provides information and examples to help to get started planning and implementing the new functionalities.

This redbook also documents, within Part 3, some front-end tools and metadata bridges to DB2 Cube Views provided by IBM and different business partners through their own products. The business partners' metadata bridge chapters, also delivered as Redpapers, are:

- MetaStage® metadata bridge from Ascential™ (REDP3712)
- Universal metadata bridge from Business Objects (REDP3711)
- Cognos metadata bridge from Cognos, Inc. (REDP3713)
- QMF™ for Windows® front-end tool from IBM and Rocket Software (REDP3702
- MetaIntegration metadata bridge from MetaIntegration Technologies, Inc. (REDP3714)
- MicroStrategy metadata bridge from MicroStrategy, Inc. (REDP3715)

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Corinne Baragoin is a Business Intelligence Project Leader at the International Technical Support Organization, San Jose Center. She has over 17 years of experience as an IT specialist on DB2 UDB and related solutions. Before joining the ITSO in 2000, she worked as an IT Specialist for IBM France, supporting Business Intelligence technical presales activities and assisting customers on DB2 UDB, data warehouse, and OLAP solutions.

Geetha Balasubramaniam is a Senior Software Engineer in IBM Global Services in India. She has over 5 years of experience in IT. She holds a Masters degree in Computer Applications from Bharathiar University in Coimbatore, India. Her areas of expertise include developing client/server applications, data modeling, OLAP, and ETL.

Bhuvaneshwari Chandrasekharan is a Staff Software Engineer in the U.S.A. at the IBM Silicon Valley Laboratory, San Jose, California. She holds a Masters degree in Mathematics from the University of Cincinnati, Ohio. She is an IBM Certified Database Associate and is a member of the Business Intelligence Technical Support team, providing advanced technical support for the DB2 OLAP Server™ products.

Landon DelSordo is a Certified Senior IT Specialist in the US. She has over 25 years experience in IT. She has a degree in mathematics from the College of William and Mary. Her areas of expertise include business intelligence, OLAP and large data warehouses. She began working with DB2 in 1982 prior to the availability of Version 1 on MVS™.

Jan B. Lillelund is a Senior IT Specialist in Denmark. He has 20 years of experience in the IT field and 12 years experience with DB2 on various platforms including pervasive OSs. He holds a degree in Electrical Engineering from The Technical University of Denmark. His areas of expertise include data modeling, database architecture design and implementation, performance tuning and database maintenance.

Julie Maw is a Senior IT specialist in the United Kingdom. She has 19 years of experience in IBM, mostly working with iSeries™ customers. She has been working in the business intelligence field for six years, initially as a member of the the Business Intelligence Offering Team within IBM Global Services. She is currently a member of the EMEA Business Intelligence Technical Sales team specializing in DB2 OLAP Server.

Annie Neroda is a Senior Consulting Software IT Specialist in the USA. She has 35 years of experience in the IT field. She holds a degree in Mathematics from Trinity College in Washington, DC. Her areas of expertise include Business Intelligence, especially OLAP, ETL, and Data Mining. She has taught extensively on DB2, Business Intelligence, and Data Warehousing.

Paulo Pereira is Business Intelligence Specialist at the Business Intelligence Solutions Center (BISC) in Dallas, Texas. He has over 8 years of experience with customer projects in the Business Intelligence arena, in consulting, architecture design, data modeling, and implementation of large data warehouse systems. He has worked with the majority of the Business Intelligence IBM Data Management and partners portfolio, specializing in parallel UNIX® solutions. He holds a master's degree in electrical engineering from the Catholic University of Rio de Janeiro (PUC-RJ), Brazil.

Jo A. Ramos has 16 years of experience in Information Technology and practical experience in implementing Data Management and Business Intelligence solutions. He works for the Advanced Technical Support Organization (ATS) in Dallas, in the Business Intelligence Solution Center (BISC) group. He provides presales support for Data Management products, developing customized demonstrations and proof of concepts for ETL and OLAP applications.

Thanks to the following business partners or IBMers who came onsite in San Jose during one week to test and document their metadata bridge from/to DB2 Cube Views:

Marlene Coates, IBM on QMF for Windows front-end tool

Ricardo Arriaga and **Cuong Bui, MicroStrategy, Inc.** on MicroStrategy metadata bridge

John Ellis, Ascential on MetaStage metadata bridge

Nina Sandy and **Patrick Spedding, Cognos, Inc.** on Cognos metadata bridge

Olivier Schmitt, MetaIntegration Technology, Inc. on the MetaIntegration metadata bridges

Wuzhen Xiong, Business Objects on BusinessObjects Universal Metadata bridge

Thanks to the following people for their help in planning and preparing this project and their involvement and input all along the project:

Nathan Colossi
Daniel De Kimpe

John Poelman
Gary Robinson
Christopher Yao
IBM Silicon Valley Lab

Thanks to the following people for their input and contributions during the project:

William Sterling
IBM WW Technical Sales Support

Richard Sawa
Hyperion

Mike Alcorn
Upendra Chitnis
Jason Dere
Bruno Fischel
Suzanna Khatchatrian
Jeff Gibson
Gregor Meyer
Benjamin Nguyen
Joyce Taylor
Craig Tomlyn
Tamuyen Phung
Jennifer Xia
IBM Silicon Valley Lab

John Medicke
Stephen Rutledge
IBM Software Solutions and Strategy Division

Wenbin Ma
Calisto Zuzarte
IBM Toronto Lab

Thanks to the following people for their reviews on this redbook:

Jon Rubin
IBM Silicon Valley Lab

Doreen Fogle
IBM WW Technical Sales Support

Ian Allen
IBM EMEA Technical Sales Support

Matt Kelley
Rocket Software

Yvonne Lyon, for technical editing
International Technical Support Organization, San Jose Center

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an Internet note to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM® Corporation, International Technical Support Organization
  Dept. QXXE Building 80-E2
  650 Harry Road
  San Jose, California 95120-6099

Part 1

Understand DB2 Cube Views

In this part of the book we introduce DB2 Cube Views and illustrate its benefits through generic scenarios.

1

An OLAP-aware DB2

This chapter introduces the terms and concepts that are the subject of this book.

Following a brief introduction to the concept of OLAP, we provide an overview of the functionalities of DB2 Cube Views.

1.1 Business Intelligence and OLAP introduction

The ability to collect, organize, and effectively exploit the mass of data that is available to an organization has long been a goal of those that deploy information systems. Over the years technologies have evolved from simple reporting systems through to fully integrated Business Intelligence systems, as organizations have strived to make effective use of their business information.

During the course of this evolution we have seen the development of sophisticated tools to Extract data from source systems, Transform data, and Load data into target systems — known as ETL tools. The front-end tools that are used to query the data have likewise evolved to handle the different data structures, the emergence of Web based technologies, and the ever increasing demands of the information analysts. Database technologies have similarly undergone a series of enhancements in order to try to satisfy the information analysts requirements.

Relational database management systems have to handle the often conflicting requirements of holding large amounts of data and yet providing users with fast query response times. The larger the tables and the more complex the data model, the more challenging it is to provide acceptable response times. In addition, exposing a complex data model to a business analyst introduces other issues. An information analyst knows the business question they need to ask of the database. However, if the structure of the database is such that they do not understand how to formulate their query in a way that is understood by the database, they are going to have difficulties in defining their queries, and therefore their productivity will be low.

The growth of multidimensional data models has seen an attempt by data modelers to structure data in a way that is more easily understood by the information analyst. A multidimensional data model is typically oriented towards a specific business area, for example a sales model or a finance model. Central to the multidimensional model is the fact table. The fact table holds the business metrics such as unit amounts, monetary values, and business ratios, that are applicable to that business subject area. The fact table is joined to a number of dimension tables. These dimension tables reflect the different ways in which a user needs to analyze the business metrics within the fact table for example sales by customer by month by region. Figure 1-1 provides a simple example of such a model. For more information on dimensional schemas and dimensional modeling, please refer to *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*, Ralph Kimball, John Wiley, 1996 and to Ralph Kimball's work, in general.

The multidimensional model in DB2 Cube Views simply reflects the physical layout of the tables.

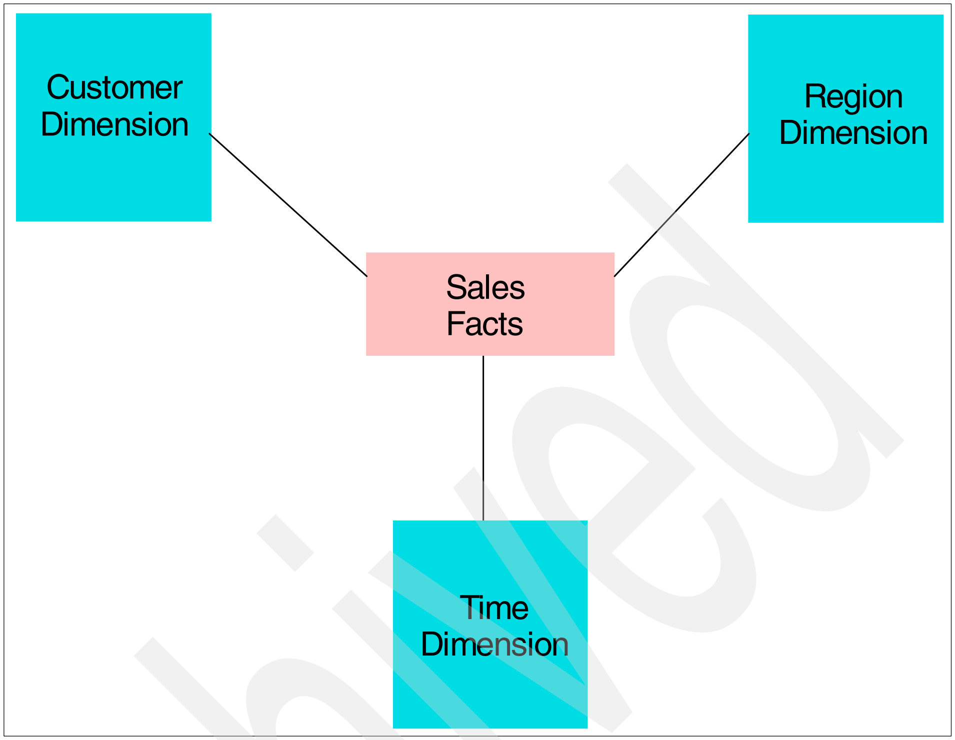


Figure 1-1 Simple multidimensional model representation

A further objective of the multidimensional model is to reduce the joins required to be performed by the database. By requiring fewer joins, the query should perform faster.

1.1.1 Online Analytical Processing

This concept of being able to analyze related business facts by multiple business dimensions is the concept that is exploited with Online Analytical Processing (OLAP) technology. Using OLAP technologies, related business metrics can be analyzed by dimensions.

Each dimension is typically expressed as a hierarchy. For example, the Time dimension could be expressed as a hierarchy of Year, Quarter, Month, Week. Queries then represent an expression of the business metrics (or facts) for a given slice of the multidimensional database. The term slice is used to depict the domain of facts that all possible queries can access at a given level per dimension, for the full set of dimensions.

An example is shown in Figure 1-2.

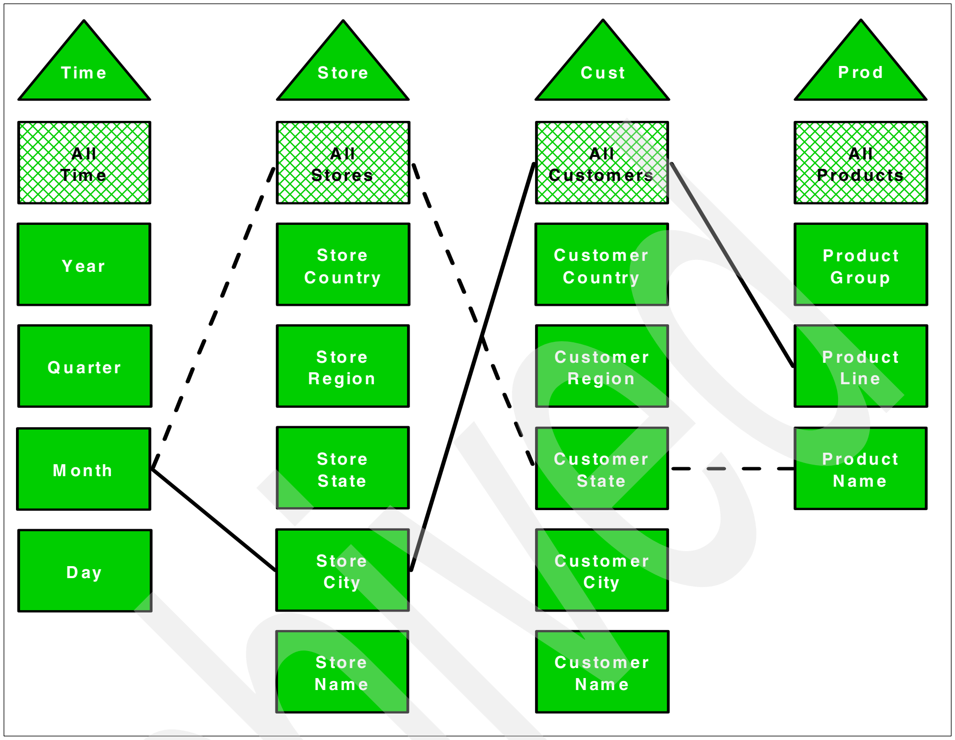


Figure 1-2 Database slices

In Figure 1-2 we see four dimensions: Time, Store, Customer, and Product. The fact table itself is not represented in this diagram. The solid line and the dashed line represent two different slices of the data. The solid line query is a slice across the database for months, store cities, all customers, at the product line level for one or more business metrics in the fact table. For example, the query could be for sales in May of a specific product line for all customers consolidated by store city. The slice of data represented by the dashed line represents months, all stores, customer states and product name level data.

OLAP implementations

The term OLAP is a general term that encompasses a number of different technologies that have been developed to implement an OLAP database. The most common server implementations that are available currently are MOLAP, ROLAP, and HOLAP.

MOLAP stands for Multidimensional OLAP. The database is stored in a special, usually proprietary, structure that is optimized for multidimensional analysis.

ROLAP stands for Relational OLAP. The database is a standard relational database and the database model is a multidimensional model, often referred to as a star or snowflake model or schema.

HOLAP stands for Hybrid OLAP and, as the name infers, is a hybrid of ROLAP and MOLAP. In a MOLAP database the data is mostly pre-calculated which has the advantage that it offers very fast query response time, but the disadvantages include the time taken to calculate the database and the space required to hold these pre-calculated values. There is therefore a practical limit on the size of a MOLAP database. In a ROLAP database the performance of the queries will be largely governed by the complexity of the SQL and the number and size of the tables being joined in the query. However, within these constraints, a ROLAP solution is generally a more scalable solution.

A HOLAP database can be thought of as a virtual database whereby the higher levels of the database are implemented as MOLAP and the lower levels of the database as ROLAP. For example in Figure 1-2 on page 6, one or more of product name, customer name, store name, or day might be stored in relational, while the rest of the database would be stored in MOLAP. This is not to say that only the very lowest level of any one dimension could be stored in relational. For example, for the customer dimension it might be that both customer name and customer city are stored in relational.

From a user perspective the line between what is stored in MOLAP and what is stored in relational should be seamless. A HOLAP environment therefore attempts to combine the benefits of both MOLAP and ROLAP technologies. By storing the lower levels of a dimension in relational instead of in MOLAP, the MOLAP database size is reduced and therefore the time required to perform the pre-calculation of the aggregate data is reduced. Queries that request data from the MOLAP section of the database will benefit from the fast performance that is expected from having pre-calculated data. Moreover, by storing the lower levels of the database in relational, the database as a whole (MOLAP and relational combined) can be extended to take advantage of the scalability benefits in the relational database.

The above terms are used to refer to server based OLAP technologies. There is also another acronym, DOLAP, which refers to Desktop OLAP. DOLAP enables users to quickly pull together small cubes that run on their desktops or laptops.

All of these OLAP technologies can be implemented using IBM DB2's family of products. MOLAP is provided with IBM DB2 OLAP Server that is an OEM version of Hyperion Essbase, and is separate from the DB2 database engine. HOLAP is available using both IBM DB2 OLAP Server and IBM DB2 itself. ROLAP and DOLAP are available with DB2, and various front end tools provide a dimensional representation to the end user of the ROLAP database using terminology with which the business analysts are familiar.

1.1.2 Metadata

While DB2 has successfully accommodated each of these OLAP technologies, it is not until now (with DB2 Cube Views) that DB2 has actually been aware of the multidimensional structure of the OLAP database and the business entities within that structure.

Information about the OLAP environment is stored as metadata. Metadata is data about data. The nature of the metadata is dependent upon the context within which it is used. So, for example, within an ETL environment the metadata may well contain information about the source and target data layouts, the transformation rules, and the actual data load jobs themselves (number of rows inserted, number not inserted, time taken, for example). Within an OLAP context the metadata will contain information about the structure of the underlying database (usually made available to the user visually as a graphical representation), the business metrics that are available, and the hierarchies that exist in each of the dimensions.

The presence of metadata enhances productivity because it explains to the user of the metadata the nature of the subject they are looking at. As an analogy consider arriving in a town for the first time and trying to find your way around. You could either find your way around by exploring and discovering where everything is, or you could purchase a map and a guide book and use these to navigate yourself around the town. The guide book will tell you what is in the town and the map will enable you to find places swiftly, and enable you to understand very quickly what you need to do to get from the library to the post office, for example. The map offers a visual representation of the town, which gives the reader an immediate level of understanding of the layout of the town. These same concepts can be applied to metadata.

Furthermore the structured metadata can be utilized by applications and hence improve productivity. If the structure of the metadata is known then applications can be written to make use of the metadata. If the metadata is not made available to applications then the application has first to discover the nature of the underlying database before it can use it.

In a Business Intelligence implementation, therefore, there is a wealth of metadata already defined and available. The ETL tools may have their own metadata, the MOLAP tool may have its metadata describing the source data that is used to load the MOLAP database and any transformations that are required to load into the MOLAP database; the MOLAP, ROLAP, DOLAP tools will also have their own metadata to describe the business metrics and the dimensions and hierarchies that are represented in the database. In a HOLAP environment there will also be metadata describing where the cut-off point is between what is in the MOLAP database and what is in the relational database. In a HOLAP environment metadata is required to enable the hybrid engine to know what SQL to generate in order to respond to queries that are outside of the MOLAP database.

All styles of OLAP need the same basic metadata: the cube model and a mapping of that model to the relational source. A common factor in most of these scenarios is that at some point, a query against the relational database is going to be generated. However, up till now the relational database, unlike the other tools described above, has had no metadata at all describing the nature of the OLAP structure within the database.

1.2 DB2 UDB V8.1 becomes OLAP-aware

This chapter started by discussing the evolution of technologies in order to meet the ever increasing demands of the business intelligence market place. This book is intended to introduce yet another evolution, DB2 Cube Views. With DB2 Cube Views DB2 UDB V8.1 for the first time is made aware of the OLAP structure within the database. DB2 Cube Views enables DB2 UDB V8.1 to store metadata about the OLAP model, for example the measures, the dimensions and the hierarchies within those dimensions.

This does not mean however that this is yet another isolated store of metadata. To develop DB2 Cube Views, IBM involved its business partners with the result that these partners have developed associations between their products and DB2 Cube Views. Most Business Intelligence tools are able to exchange metadata between their products and DB2 Cube Views via a bridge. For example, a ROLAP front end tool requires metadata about the OLAP structure within the underlying database. DB2 Cube Views has that information and is able to share that information by passing the metadata across the bridge to the front end tool. Some products, namely QMF for Windows and Office Connect to date, have been enhanced such that they access the DB2 Cube Views metadata directly. Then DB2 Cube Views allows to share metadata with all partner tools but also to define the OLAP model and its mapping and export this metadata as well. Defining the OLAP model and mappings is performed in DB2 once, and to avoid replicating effort, exported to the tool.

Now that DB2 UDB V8.1 is OLAP-aware it is able to make use of the metadata it has available to it to optimize the database for OLAP. As we have discussed, an OLAP database is a subject area specific database that holds business metrics and dimensions. Any business metric can be queried according to numerous slices of the database as governed by the hierarchies that are available within the dimensions. A query that is based on the lowest level available in each dimension is a query based at the level of granularity of the fact table itself. Often, however, a query is going to be expressed at a higher level in one or more of the dimensions and as such represents an aggregate or summary of the base data.

Aggregates or summaries of the base data can be created in advance and stored in the database as Materialized Query Tables (MQTs). The DB2 optimizer is then able to recognize that a specific query requires an aggregation and if it has a relevant MQT available for it to use, can attempt to rewrite the query to run against the MQT instead of the base data. As the MQT is a precomputed summary and/or filtered subset of the base data it tends to be much smaller in size than the base tables from which it was derived, and as such significant performance gains can be made from using the MQT. Another performance benefit from MQT is pre-joined as joining tables can be more costly than aggregating rows within a single table.

DB2 Cube Views provides an advisor that is able to recommend scripts to generate MQTs, based on the DB2 Cube Views metadata that describes the OLAP structure and its proposed usage and environment. This relieves the database administrator from having to spend time and effort trying to find out the most effective MQTs to create and to understand and write complex SQL syntax involving super-aggregate operators (`CUBE`, `ROLLUP`, `GROUPING SETS`).

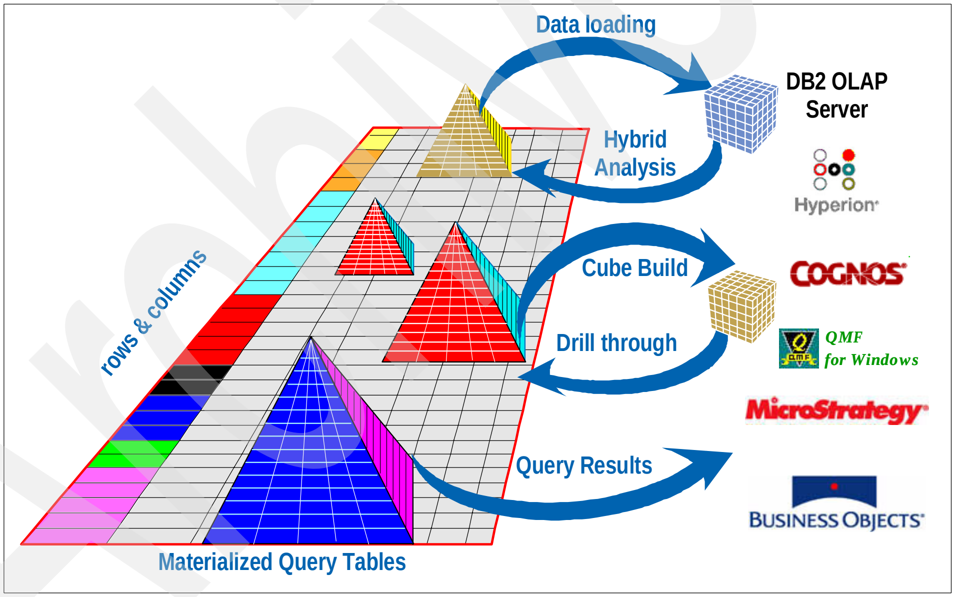


Figure 1-3 DB2 Cube Views and MQTs

Figure 1-3 illustrates some examples of how the presence of MQTs can aid performance in on OLAP environment. At the top of the figure we see DB2 OLAP Server. Where an aggregate of the base data is required to load into DB2 OLAP Server, then the use of an MQT should improve the performance of extract for the data load. More critically perhaps, the use of an MQT may significantly improve the performance of the relational queries that are generated in a hybrid environment. Elsewhere in the figure we see examples of other reporting tools where again the use of MQTs can improve performance. DB2 Cube Views will address efficient aggregate management for a range of OLAP styles including loading MOLAP and DOLAP cubes, drill through and hybrid analysis as well as simple reporting and ROLAP style interactive slice, dice and drill directly against DB2 UDB V8.1.

1.3 Challenges faced by DBA’s in an OLAP environment

When implementing an OLAP environment there are a number of on-going challenges that a DBA has to meet after the initial challenge of designing the OLAP database has been completed. In this section we will look at some of those challenges.

1.3.1 Manage the flow of metadata

This title pre-supposes that there is in fact a flow of metadata through the product set that may be installed in the business intelligence environment. However, part of the problem may be that there is no flow of metadata at all, and that isolated pockets of metadata exist to support different parts of the solution that have been implemented.

The challenge here is that at each part of the implementation where metadata is required, it is necessary to perform steps to re-create the metadata. This process of losing information and rediscovering it is expensive and error prone. Nor is this a one time problem. As the schema changes, all tools and applications will have to be updated.

Even where there is an element of metadata interchange between products and applications, if DB2 does not have the same understanding of the higher level constructs such as dimensions and hierarchies, then the task of optimizing the database will require significant DBA resources.

1.3.2 Optimize and manage custom summary tables

Managing multidimensional data efficiently requires the ability to create and manage aggregates and index data across multiple dimensions.

Aggregate management is particularly challenging because storage requirements explode when you combine measures across multiple dimensions. Let us assume you want to model sales by product by customer by channel over time. To aggregate all combinations across all dimensions to provide weekly data for two years across 1,000 products sold to 500,000 customers through four channels, you'd create a multidimensional space of 104x1000x500,000x4=208,000,000,000 sales figures.

OLAP models are rarely this simple; they usually include more dimensions and require additional storage for aggregates such as months, quarters, years, product groups, and sales regions. The space and time required to build such aggregates can make aggregation impractical when it is equivalent to preaggregate everything.

The alternative to aggregating everything is to choose where you build aggregates within the multidimensional space. Although the complete multidimensional space still contains the same number of total values, only some of the values are pre-aggregated and stored; the rest are aggregated on demand. However the task of deciding which values to pre-aggregate is a major challenge for the DBA, and will typically involve the DBA in analysis work to determine whether the pre-aggregated summary tables are being chosen by the optimizer in the way intended by the DBA.

Having decided on which summary tables to create the challenge is then one of managing those summary tables. Summary tables occupy space and take time to refresh. The DBA will need to determine a balance between creating more summary tables and operating within the space and time limitations that exist in their particular environment. Moreover there is also a balance to be struck between creating more summary tables and overloading the optimizer.

1.3.3 Optimize MOLAP database loading

Where the OLAP database resides in a non-relational data structure, the first challenge the DBA faces is to produce scripts to load the data from relational into the MOLAP database. For example, the DBA needs to develop data load rules that specify how the data should be loaded into the MOLAP database. The DBA has to code the SQL that is required to access the data, and the data load rules will additionally specify how to map the incoming relational data to the OLAP structure, and what transformations of the data should take place.

Using the same example, the DBA would have to generate the metadata to describe the relational source data and the target MOLAP database structure, and then generate the scripts required to load the data as it has a full understanding of the source and target databases and any transformations that are required.

Whatever method is available to the DBA, having specified how the data should be loaded into any MOLAP database, the next step is to optimize that data load. The DBA will need to determine the indexes that should be built and may well need to analyze the query to determine how best to improve the performance of the data load query. A decision may need to be made regarding summary tables. Would a summary table improve the performance of the data load query, and if so, what should the summary table look like?

1.3.4 Enhance OLAP queries performance in the relational database

The challenge for the DBA in enhancing query performance involves making best use of the many facilities available to the DBA, both in terms of DB2 functionality and DB2 performance analysis tools.

As discussed in the previous section, managing multidimensional data efficiently will inevitably involve the creation of selected aggregate tables. The challenge is to work out which ones to build.

A DBA needs to understand the dimensional model, the nature of the data, and the access patterns. Cost/benefit calculations that consider the cost to build, the space the aggregates will consume, and the benefit they will yield may help. The cost/benefit analysis will help determine which slices of the multidimensional model will be pre-aggregated and stored and which will be computed on demand. Some incoming queries will directly correspond to pre-aggregated stored values; others can be quickly derived from existing partial aggregates. In both cases, faster queries result. However, getting to the point where the DBA is confident in their choice of aggregate tables takes time.

1.4 How DB2 can help

DB2 Cube Views enables DB2 UDB V8.1 to act as an OLAP accelerator as DB2 UDB V8.1 is now aware of the multidimensional structure of the database design. By having direct access to OLAP metadata DB2 UDB V8.1 is able to advise on the most efficient Materialized Query Tables (MQTs) to be built. DB2 itself has the ability to intercept queries written against base tables and rewrite them to use these MQTs, so resulting in faster response times.

Through the use of bridges, DB2 Cube Views is able to share its OLAP metadata (import/export) with other partner Business Intelligence tools, offering users of those tools a fast start option and assisting in reducing the maintenance involved in changing metadata that may be stored repeatedly in different formats in different products.

DB2 Cube Views is available as part of both DB2 UDB V8.1 Data Warehouse Standard Edition and DB2 UDB V8.1 Data Warehouse Enterprise Edition, in addition to being available separately.

1.4.1 Efficient multidimensional model: cube model

The DB2 Cube Views metadata model is a highly comprehensive model capable of modelling a wide range of schemas from simple to complex. Various OLAP tools place a relative importance on different object types within the OLAP model, and as such the DB2 Cube Views metadata model has been developed with an inbuilt flexibility to accommodate these differences.

The DB2 Cube Views metadata model takes a layered approach, an overview of which is shown in Figure 1-4.

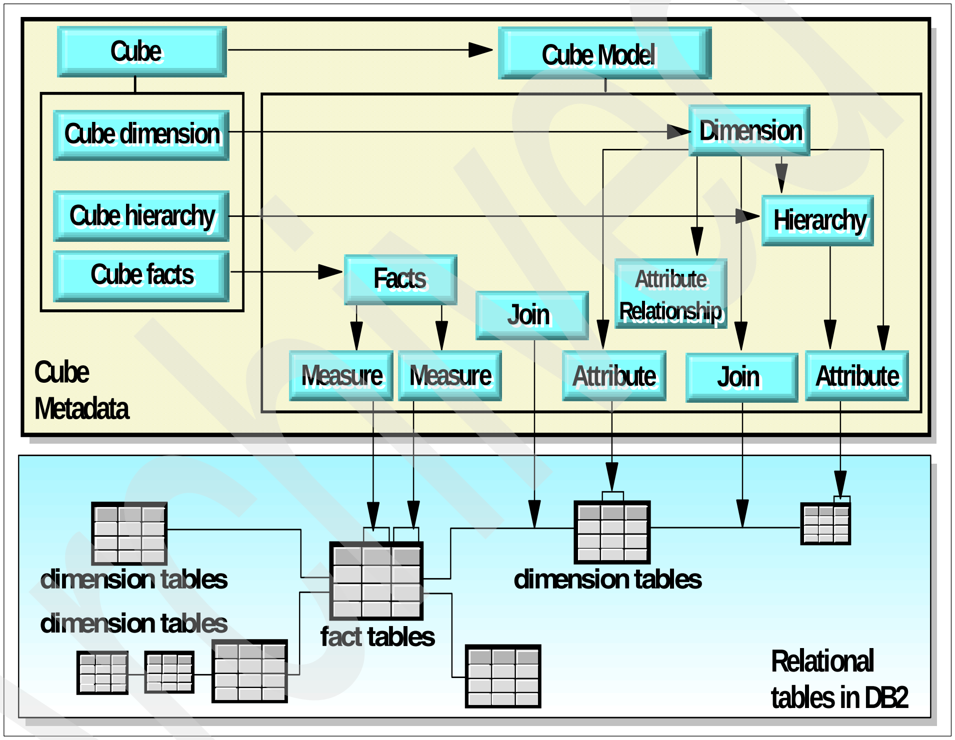


Figure 1-4 DB2 Cube Views metadata

Figure 1-4 demonstrates how the cube metadata, shown in the top part of the diagram, maps to the relational table constructs in DB2 UDB V8.1, shown in the bottom part of the diagram.

The cube metadata defines two major structures, the *cube model* and the *cube:*

- The *cube model* can be compared to a conceptual OLAP database. The cube model can be constructed in many ways. It maps OLAP metadata objects to the relational structures in DB2 UDB V8.1. The metadata objects that are stored within the cube model are facts objects, dimensions, hierarchies, measures attributes and attribute relationships. A full definition of these objects is documented in Chapter 3, “Building a cube model in DB2” on page 63.

- The *cube* is an extrapolation of the overall cube model. It is possible to have one cube per cube model, or multiple cubes per cube model. A cube is the closest object to a MOLAP database. The metadata objects that are stored within the cube are cube facts, cube dimensions and cube hierarchies. Again, please see Chapter 3, “Building a cube model in DB2” on page 63 for further details on cubes and cube models.

Some query tools are able to connect directly to the DB2 Cube Views metadata via the DB2 Cube Views API and provide the end user with the cube definition that they require in order to navigate the cube and query the data. Other tools will make use of the DB2 Cube Views metadata via a bridge as is discussed in “Metadata bridges to back-end and front-end tools” on page 19.

The user interface to the DB2 Cube Views metadata is via a client workstation graphical user interface called OLAP Center. OLAP Center is a Java™ based utility that uses available DB2 UDB V8.1 common classes and maintains the same look and feel as the other DB2 GUI tools. OLAP Center can launch and can be launched by other DB2 UDB V8.1 tools. The the architecture of OLAP Center is depicted in Figure 3-17 on page 83.

The use of OLAP Center to manipulate OLAP metadata is described in detail in Chapter 3, “Building a cube model in DB2” on page 63.

1.4.2 Summary tables optimization: Optimization Advisor

An analyst using an OLAP query tool requires that they are able to slice the data in many different ways at different combinations of levels within the hierarchy of each dimension. For example sales by product line by month in north region, or sales by product group by week in east region. Queries that are aggregates of the base data will perform better if the data for these queries is sourced from summary tables rather than being sourced from the base data itself. The issue for the DBA is to work out which summary tables should be created.

DB2 Cube Views provides an Optimization Advisor. The interface to the Optimization Advisor is through a wizard in OLAP Center. The goal of the Optimization Advisor is to build an ideal set of Materialized Query Tables (MQTs) and indexes for a given cube(s) and class of query. All references to aggregate tables or summary tables with DB2 Cube Views are meant to imply MQTs.

The Optimization Advisor takes as its input the metadata, the input values that are entered in OLAP Center (disk space limit, time limit and MQT maintenance preference) and the statistics held in the DB2 UDB V8.1 catalog tables. Furthermore the Optimization Advisor makes use of new functionality within DB2 UDB V8.1 (`TABLESAMPLE`) and actually samples the data in order to size how big a specific grouping would be. Optional, sampling, when used analyzes column values appearing in the cube, thereby accounting for cardinality and skew in the underlying data that impacts size and shape of resulting MQT.

The Optimization Advisor may also make use of the recently introduced super-aggregates operators in order to create MQTs that can potentially be used by a greater number of queries.

As the Optimization Advisor is optimizing at the cube model level, and is able to take advantage of recently developed assists within DB2 UDB V8.1, it is in a good position to meet its objective of maximizing efficiency by being able to determine a smaller number of MQTs than might otherwise have been determined manually. This is illustrated in Figure 1-5.

Queries that could take advantage of MQTs would include:

- ROLAP type queries
- Queries used to load a MOLAP database from DB2 UDB V8.1
- Drill-through queries from a MOLAP database to DB2 UDB V8.1
- Queries generated against DB2 UDB V8.1 in a HOLAP environment

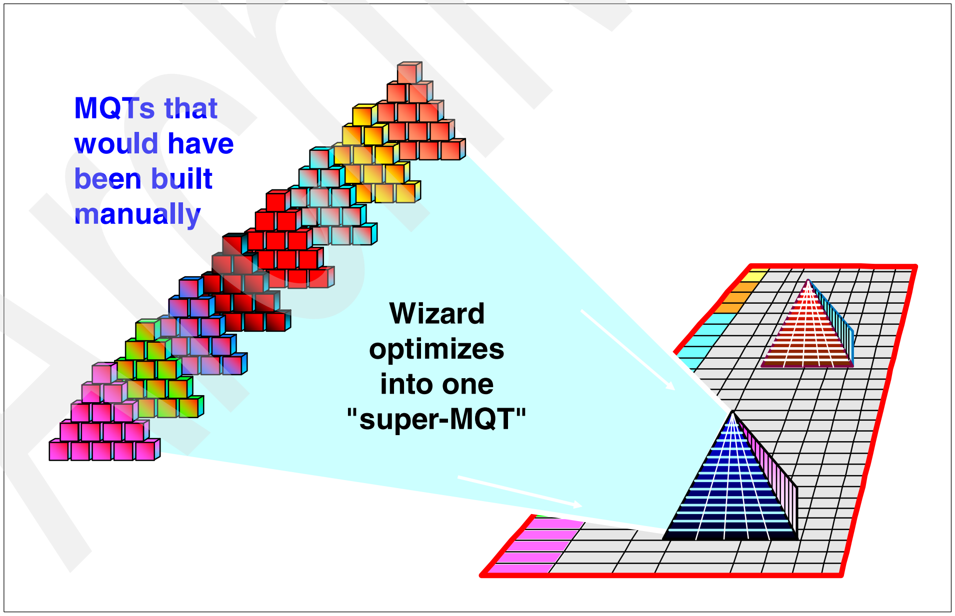


Figure 1-5 MQT optimization

“Using the cube model for summary tables optimization” on page 125 provides a more detailed discussion on the use of MQTs within DB2 Cube Views and also details of the Optimization Advisor.

1.4.3 Interfaces

The interface options that are available with release one of DB2 Cube Views have already been discussed in earlier sections of this chapter. The purpose of this subsection is to summarize those interface options, and also to introduce an additional interface that is not actually part of release one of the product, but is available as a Technology Preview.

The interface options that are available in release one of the product to access DB2 Cube Views metadata are listed in Table 1-1.

Table 1-1 DB2 Cube Views interface options

Type of interface	Product/ IBM Business Partner
Direct access to metadata through DB2 Cube Views API	IBM QMF for Windows 7.2f, Office Connect 4.0 Analytic Edition
Access via a bridge	Initial IBM Business Partners include Ascential, Business Objects, Cognos, Hyperion, Meta Integration, MicroStrategy
Administration interface	OLAP Center

Note: Cognos accesses DB2 Cube Views metadata using both bridge and API.

A Web services interface is available as a Technology Preview. As a Technology Preview it is not part of the supported product offering, but is available as a preview of what is intended to be in a future release of the product.

Figure 1-6 illustrates some possible future application scenarios using DB2 Cube Views Web services.

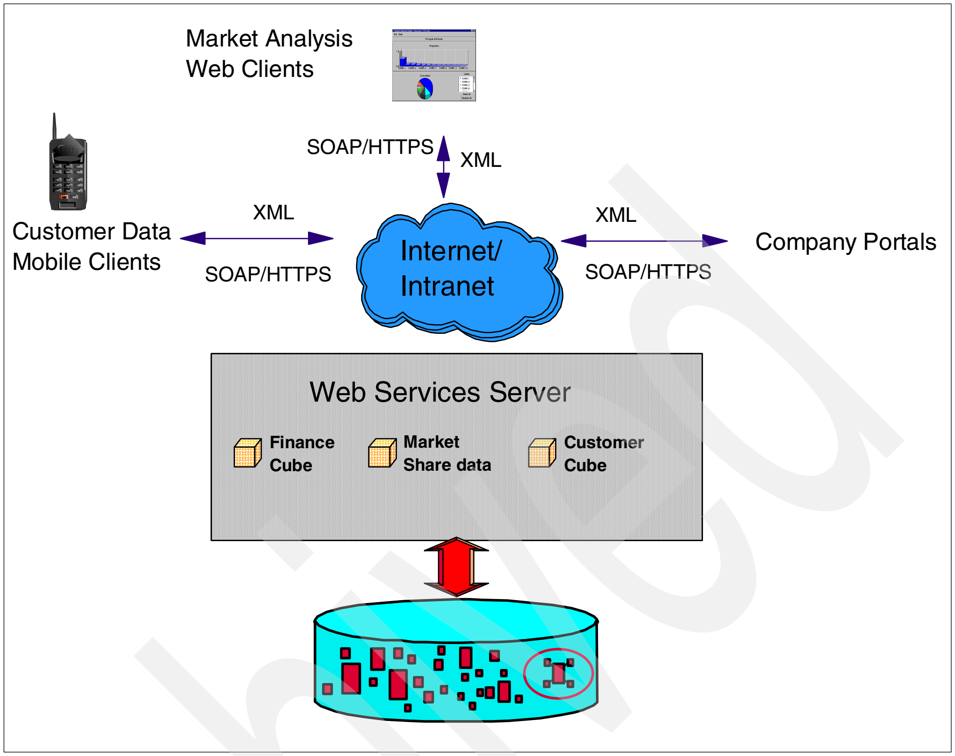


Figure 1-6 Web services applications

The intention for DB2 Cube Views Web services is that it will provide access for Web services developers to OLAP analytical data. It is not the intention for DB2 Cube Views Web services to become a new slice, dice, and drill interface, but more that DB2 Cube Views Web services will allow developers to quickly find sources of dimensional information for their applications; determine the slices they need, and retrieve the data using an XPath-based execute method. Without learning OLAP interfaces and query languages, Web services developers will be able to call on their existing knowledge of XML and XPath to add analytic information to their applications.

DB2 Cubes Views Web services is available from the alphaWorks® IBM Web site:

`http://www.alphaworks.ibm.com.`

1.5 Metadata bridges to back-end and front-end tools

An important aspect of DB2 Cube Views is the ability to share metadata with other Business Intelligence tools to avoid repetition and errors, and to reduce the maintenance effort involved in managing multiple metadata repositories. The way in which the DB2 Cube Views metadata is shared is via a bridge.

Some tools will *push* metadata into DB2 Cube Views, some will *pull* metadata from DB2 Cube Views into their own tool metadata structure, and some will offer a two-way bridge which can both push and pull metadata to and from DB2 Cube Views. This is illustrated in Figure 1-7. Typically design and ETL tools will be pushing metadata into DB2 Cube Views; and query and reporting and OLAP tools will be pulling metadata from DB2 Cube Views.

The bridge is implemented as a DB2 stored procedure that passes XML documents both in and out for all of its arguments.

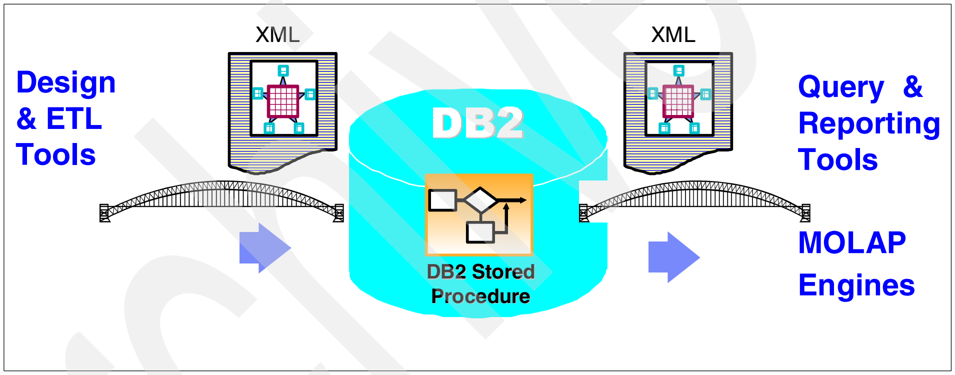


Figure 1-7 OLAP metadata bridges

Using this simple XML based interface instead of working directly against the new DB2 Cube Views, catalog tables protect developers of these bridges from changes to the underlying tables. For further information on front-end tools and metadata bridges, please refer to Part 3, on page 219.

2

DB2 Cube Views: scenarios and benefits

In this chapter, we show how DB2 Cube Views delivers a large return on your investment, whether it is used alone or in combination with back-end and/or front-end tools.

In each scenario, we refer to the back-end tools and front-end tools in a generic way rather than naming any specific products.

2.1 What can DB2 Cube Views do for you?

Let's say your organization has decided to deliver first rate analytical capabilities to its end users, and after reading all the latest books and articles on Business Intelligence systems, they have decided to build a star schema database like the one in Figure 2-1 as the heart of this new system. They have probably done this because star schemas offer such rich, business oriented analytical options, such as slicing and dicing, trending, comparisons, rollups and drill-downs.

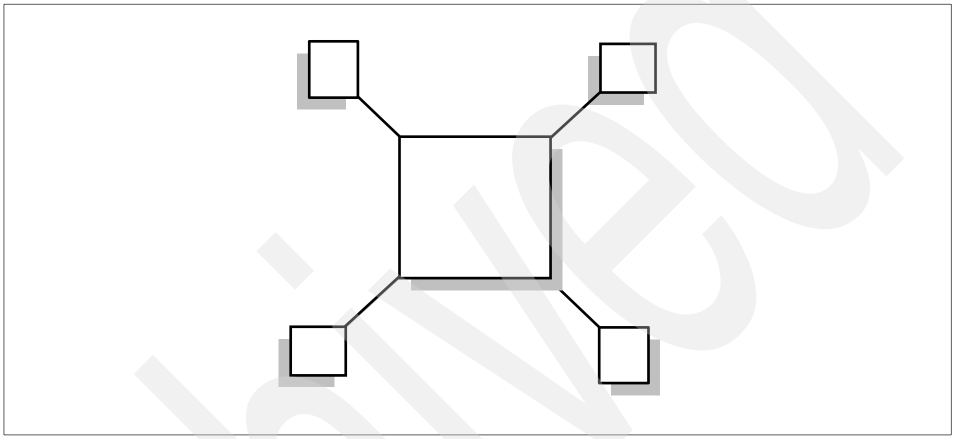


Figure 2-1 Your star schema database

In addition, they will most likely be using one of today's premier data delivery platforms as a front-end for the database because it provides ease of use and because it works so well when coupled with a star schema database. To integrate your front-end tool, the star schema that you have built as tables, columns, primary keys, foreign keys will need to be mapped to the tool as a collection of OLAP objects like measures, derivations, dimensions, hierarchies, attributes and joins. DB2 Cube Views gives you a new GUI called the OLAP Center where you can map these OLAP objects directly to your relational objects and hold these mappings in DB2, as shown in Figure 2-2.

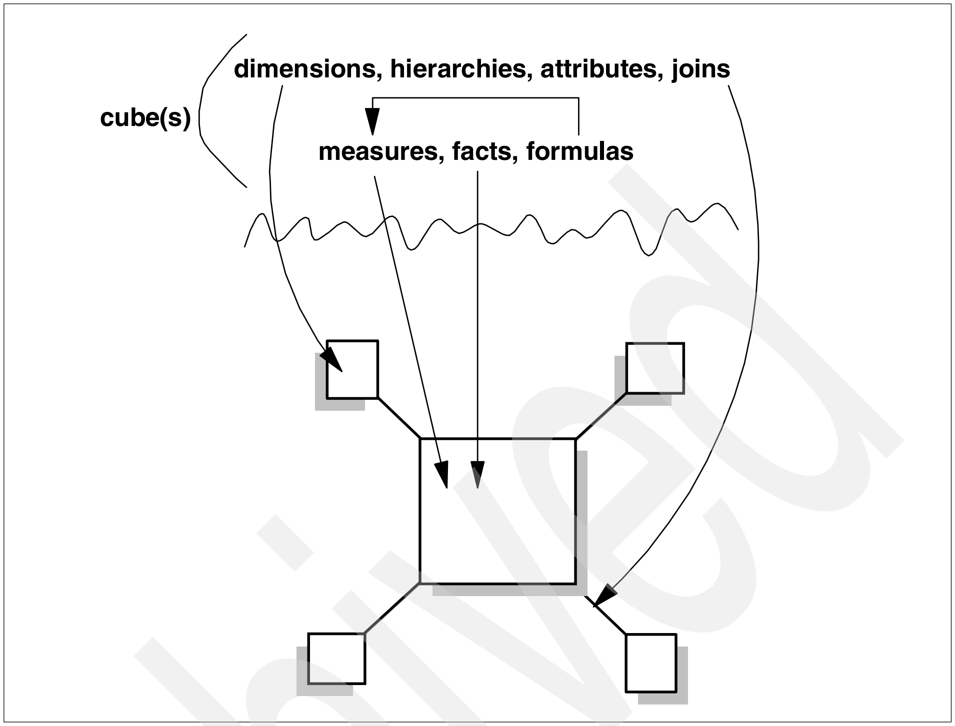


Figure 2-2 Mapping your star schema to OLAP objects

Using the OLAP Center, you can pinpoint the columns in the fact table that actually contain the measures and capture formulas for deriving additional measures that are not physically stored in the star. Further, you can describe the dimensions and their various hierarchies, even multiple hierarchies if that applies. You can also indicate the proper joins to use when accessing the star. Once you have these OLAP objects described, you can group them into cubes, even into multiple cubes, each of which represents a subset of your full cube model based on the star schema. If you have already captured this information in a back-end data modeling or ETL (Extract, Transform, Load) tool, you can skip the data entry and just import the metadata directly via a metadata bridge.

Once the OLAP metadata is stored in DB2 Cube Views, you can use another metadata bridge to send it over to your favorite front-end data delivery tool, automatically, to populate its metadata layer. This way, if a different person is responsible for the database from the one who is responsible for the data delivery tool, then the metadata layer will be consistent. Also, if you will be using multiple tools, the metadata only needs to be captured once, in DB2 Cube Views, and then shared with all the other tools in your solution. Figure 2-3 below illustrates this metadata transfer.

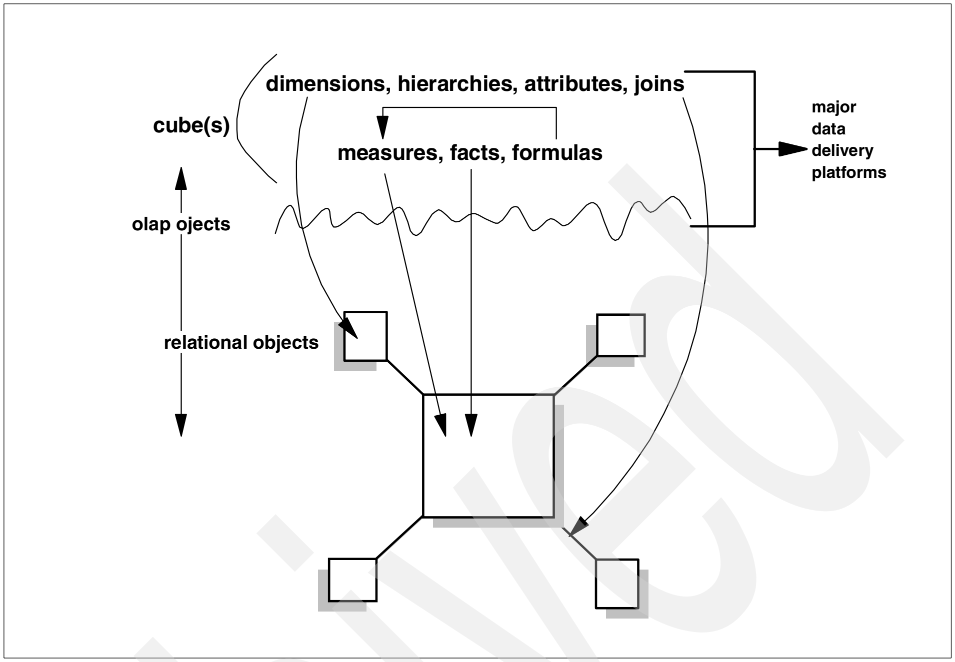


Figure 2-3 Sharing OLAP metadata with reporting tools

Once the metadata layer in your reporting tool has been populated, the tool will soon be sending SQL queries to your star schema. If the SQL requires aggregation and joins, and it probably does, the user's response time could possibly be slow. That is a problem.

But let us say you have a good DBA who knows what to do. He pre-builds an aggregate table and adds it to the database where your star schema is located. The really nice thing about pre-built aggregates in DB2 is that the tool writing the SQL doesn't have to know about them. The DB2 optimizer will automatically use them if the query matches up to them well enough. This makes for very much faster query response times. Figure 2-4 shows a query being satisfied by a pre-built aggregate.

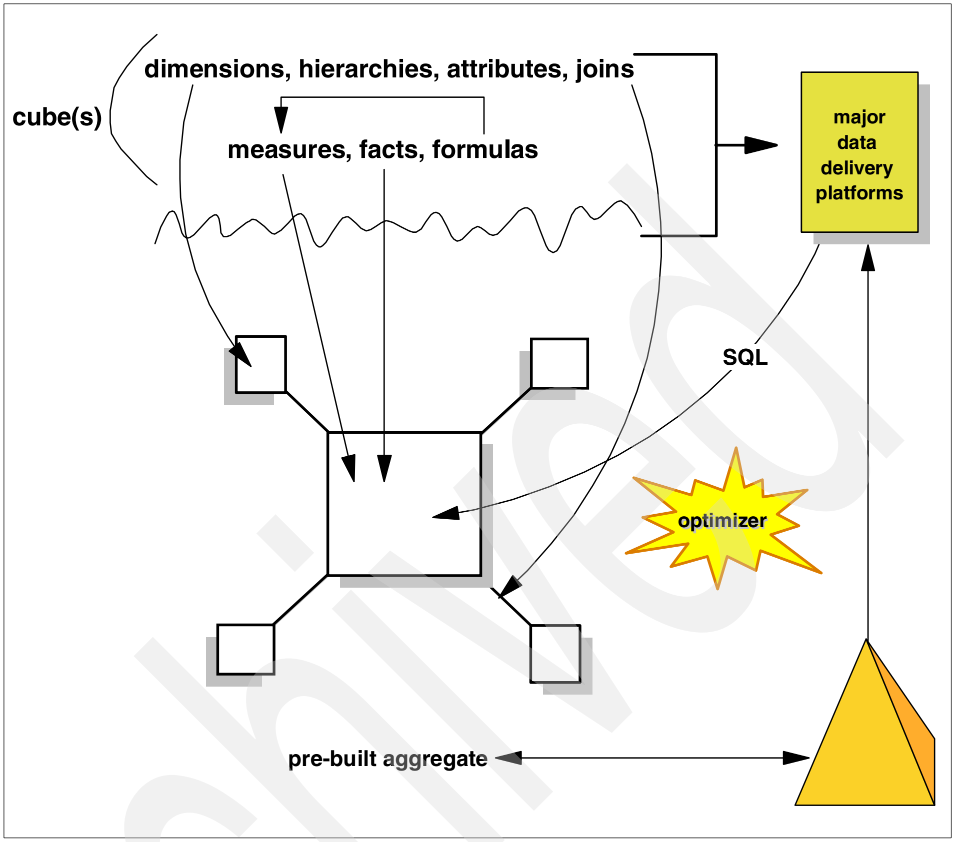


Figure 2-4 Using aggregates

The not-so-nice thing about pre-built aggregates is that the optimizer might not choose to use them every time if the SQL doesn't quite match up. In that case, your DBA may have wasted his time building the wrong aggregates. Perhaps he could solve this problem by building more aggregates, maybe even one for every possible situation. The trouble with that approach is he might end up using as much disk space on aggregates as he did on the star schema itself, not to mention the time he'll have to spend designing the aggregates and refreshing them with data periodically. DB2 Cube Views can help. It can build the ideal set of aggregates or MQTs for him the first time and find out the best compromise between space, time and query performance.

In Figure 2-5, you can see the DB2 Cube Views Optimization Advisor, a very smart expert system on performance that is going to ask your DBA a few questions before it gets to work on building the aggregates. Questions like these:

1. What kinds of queries do you plan to use against this star schema?

– Extracts? For instance, are you going to load multidimensional *(MOLAP)* databases from this star and need a pre-built aggregate that corresponds to the base level or “bottom” of the DB2 Cube Views logical cube?

– Drill-downs? For instance, are your users going to start-at-the-top (spreadsheet-style), and then drill down from there, typically as ROLAP tools do when they emulate cube-drilling, originating at the top levels of the dimensional hierarchies? If yes, you are going to need aggregations that are at or near the “top” of the logical cube.

– Drill-through? (also known as Hybrid OLAP or *HOLAP*). For instance, are your users going to drill-down beyond the base level of the MOLAP database, back to the relational database?

– Reporting? For instance, will your users be making any of the ad-hoc combinations of dimensions and levels, hitting various levels of aggregation through the “center” of the logical cube?

2. How much space are you willing to spend on aggregates?

   – Clearly, if you give the Optimization Advisor lots of space, it will build bigger, more inclusive aggregates.

   – If you give it less space to work with, it will prioritize and build very useful aggregates that will fit.

3. Next, it will look at your DB2 Cube Model metadata to understand your aggregations and dimensions and hierarchies to improve its decisions.

4. Next, it is going to look at the DB2 catalog statistics on your star schema tables, just as your DBA would do.

5. Next, using a data sampling technique, the Optimization Advisor will examine the data in your star schema. This affects the aggregate decisions because while it is sampling, it will actually do the star joins so it can understand the *sparsity* of your data — this gives a very accurate estimate of aggregate size.

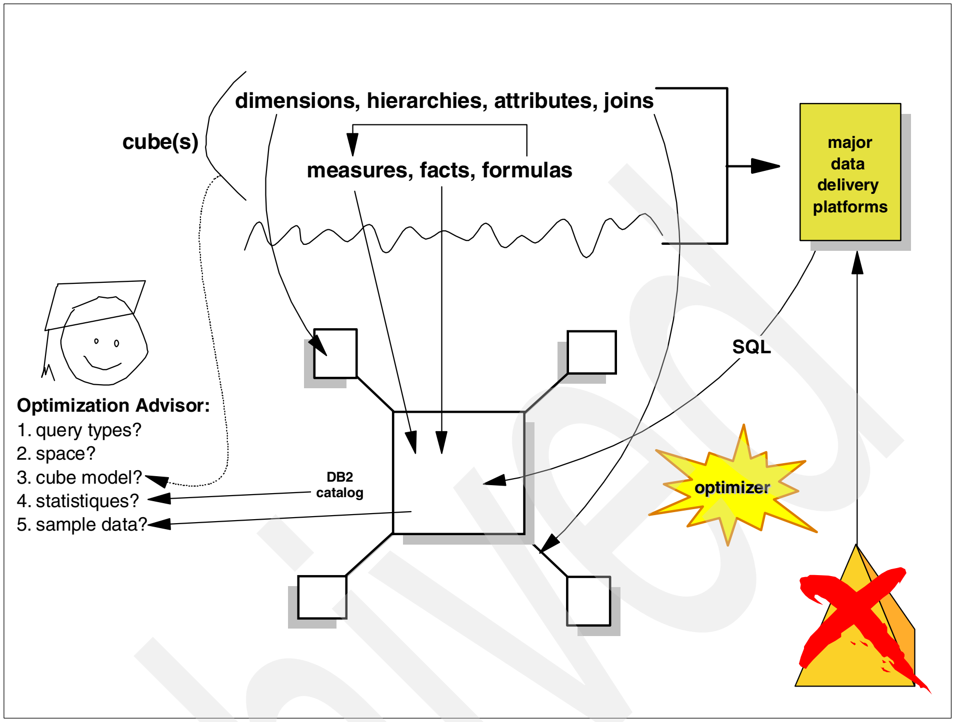


Figure 2-5 The Optimization Advisor gathering its aggregate intelligence

Now, the Optimization Advisor has what it needs to recommend one or more aggregates for your database. In Figure 2-6 you can see that it has generated an aggregate table, in some ways similar to the aggregates your DBA might have built by himself, but it is probably much more than that. By using very sophisticated rules and techniques, the aggregates recommended by the Optimization Advisor will very likely be *super aggregates* with multiple aggregations across multiple combinations of hierarchical levels of multiple dimensions defined within the cube model. In a way, some aggregate tables become a little bit like cubes, but not complete ones because of the space restrictions placed on it by your DBA and by the Optimization Advisor itself. Best of all, the aggregates will be recommended in such a way that they are highly likely to be chosen by the DB2 optimizer at query time.

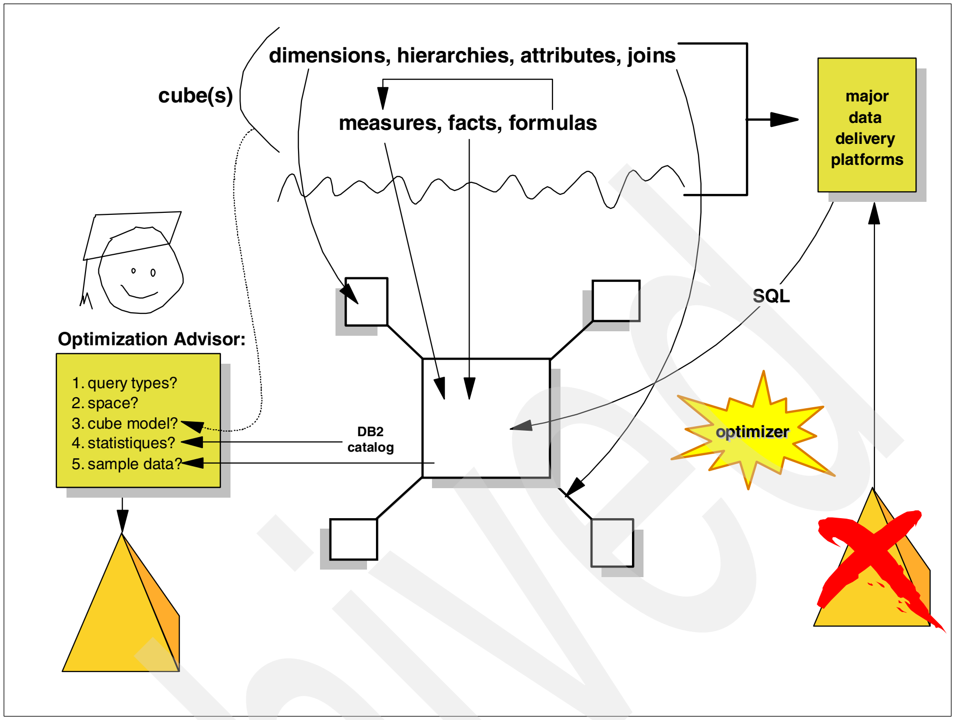


Figure 2-6   The big picture

That's the big picture!

Now, let's gain a deeper understanding of the benefits of DB2 Cube Views by examining a series of scenarios one by one:

- Feeding metadata into DB2 Cube Views
- Feeding front-end tools from DB2 Cube Views
  - Supporting Multidimensional OLAP *(MOLAP)* tools with DB2 Cube Views
  - Supporting Relational OLAP *(ROLAP)* tools with DB2 Cube Views
  - Supporting Hybrid OLAP *(HOLAP)* tools with DB2 Cube Views
  - Supporting bridgeless ROLAP tools with DB2 Cube Views
- Feeding Web services from DB2 Cube Views

These scenarios will help the reader understand the metadata flows in and out of DB2 Cube Views, as well as the performance and administrative benefits of using DB2 Cube Views in each case.

2.2 Feeding metadata into DB2 Cube Views

In order to derive any benefits from DB2 Cube Views, the job of mapping your relational star schema objects into OLAP metadata objects must be accomplished. There are basically three ways to do this, and each one carries its own unique benefits. You can feed the metadata mappings into DB2 Cube Views from back-end tools, such as data modeling or ETL tools, whose metadata already contains all or part (fact and dimensions only) of the needed information. Alternatively, you can feed it in from front-end tools, such as MOLAP or reporting tools, whose metadata is also rich with this type of information. A third approach is to enter the metadata mappings directly into DB2 Cube Views through its own graphical interface called the OLAP Center. These three paths for metadata are shown in Figure 2-7.

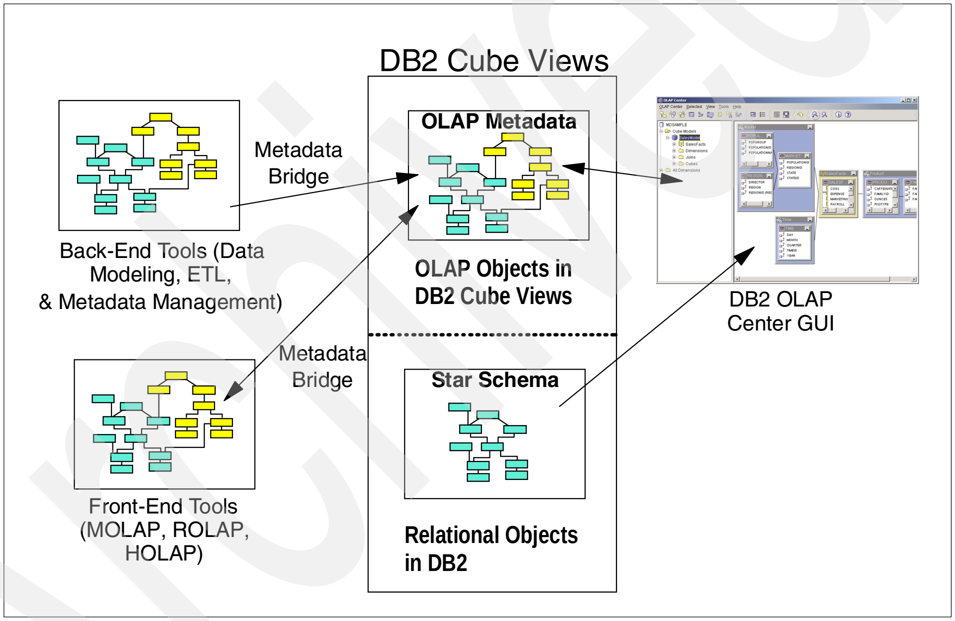


Figure 2-7 Metadata flows

Whichever of the three approaches you choose to use, the result will be a mapping between your relational objects and your OLAP objects that will lie at the heart of DB2 Cube Views. Figure 2-8 shows an example of some relational objects you might have and the OLAP objects to which they might map.

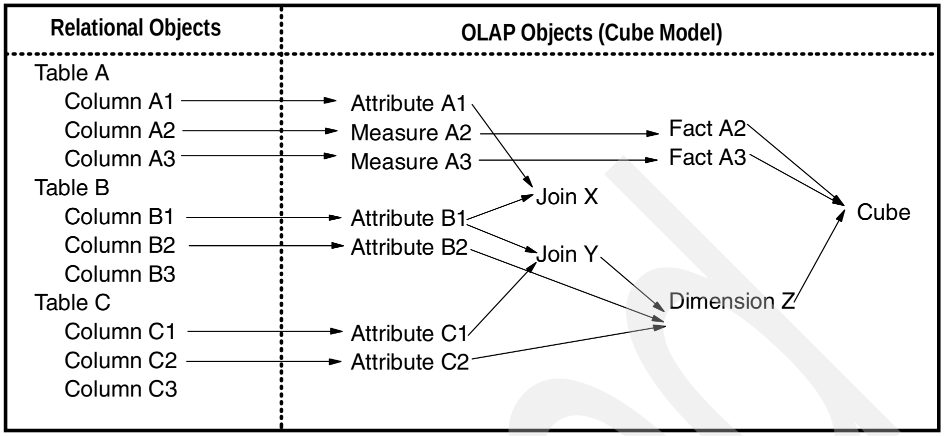


Figure 2-8 Relational to OLAP metadata mappings

Figure 2-9 represents the same example but this time showing the relational objects as a star schema.

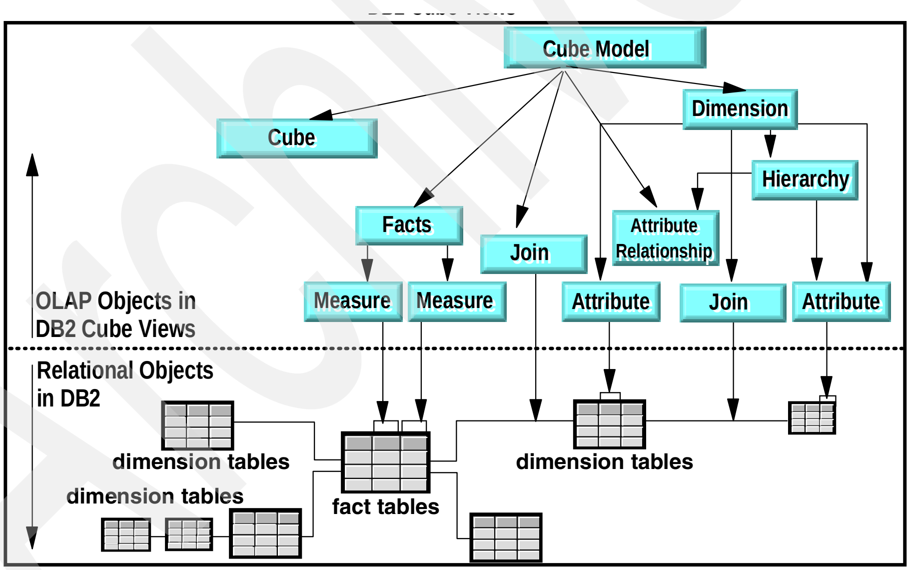


Figure 2-9 DB2 Cube Views metadata mappings

Let's explore 3 approaches to feed metadata into DB2 Cube Views:

- From back-end tools
- From front-end tools
- From scratch

2.2.1 Feeding DB2 Cube Views from back-end tools

Let's look at scenarios involving the three types of back-end tools:

- Data modeling tools
- Extract, Transform and Load or *ETL* tools
- Metadata management tools

Data modeling tools add a lot of value to database implementation projects. They greatly increase understanding through graphic representations of data relationships and data meaning, while they dramatically decrease the time it takes to develop a new database from its inception to its implementation. They capture information and store it as metadata to be used as future reference and as the basis of further development. Also, they typically generate database commands capable of creating all the physical objects for the new database.

ETL tools are also rich sources of metadata related to the star schema, since they are used to populate it. Their metadata includes detailed information about the target star schema tables and columns, as well as information about the source system databases and the transformations that have been performed on each data element on its way from source to target. This transformation history information makes *data lineage* reporting possible. For example, an end user might find it useful to know that the net sales figure he is looking at on a report is actually the result of a complex calculation involving two separate fields each of which was originally extracted from a different operational database.

Metadata management tools offer very special advantages, too, since they interact with multiple tools and exchange and integrate the metadata from multiple tools into one centralized, consolidated resource. These powerful metadata resources offer valuable assistance to the enterprise in the form of cross-tool data lineage reporting as well as cross-tool *impact analysis* reporting. An impact analysis report would alert a data analyst that a change made in one tool, for example a data modeling tool, will have an impact on another tool, such as an ETL tool or a reporting tool.

Note: When feeding the metadata mappings into DB2 Cube Views from back-end tools, the imported metadata may contain all or part of the needed information. For example, the metadata may describe only the relational schema (the star schema) and not the complete metamodel. So while importing that star schema metadata helps jump start the DBA's work, there may still remain the tasks of defining a most complete cube model using DB2 Cube Views OLAP Center and mapping it back to the star schema.

A scenario

Let's say you have built your star schema database and you are ready to populate the metadata layer of DB2 Cube Views. If you have already created similar metadata in a data modeling tool, you can feed it into DB2 Cube Views because the data modeling tool stores so many metadata objects in common with DB2 Cube Views' metadata, such as mappings from relational objects (tables, columns, primary keys, and foreign keys) to OLAP objects (facts, dimensions, hierarchies, joins, and attributes). Metadata bridges make the translations from modeling tool metadata formats to DB2 Cube Views metadata formats.

Also, if you are using an ETL tool that offers data lineage reporting, then the objects in DB2 Cube Views can show up in the data lineage reports because metadata bridges exist to share the DB2 Cube Views metadata with the ETL tool repositories. Lastly, if you are using a metadata management tool that offers cross-tool impact analysis and shares metadata with DB2 Cube Views, then its reports can show users how a change in a data model can affect the DB2 Cube Views objects, or how a change in a DB2 Cube Views object can affect an existing report on your data delivery platform.

Flow and components

The flow of metadata between back-end tools and DB2 Cube Views can be in either direction. In Figure 2-10, the metadata bridge carries descriptions of relational objects that are members of the star schema (for example tables and columns), OLAP objects that make up the dimensional model (for example, facts and dimensions), and the mappings between them. Each of these objects is represented differently in each tool and is handily translated by the bridge either though a direct call to DB2 Cube Views application programming interface or *API* or by exchanging XML files using export and import techniques. If a data modeling tool that produces DDL is being used, the DB2 relational objects can be created in DB2 using the generated DDL from these tools. The DDL exchange creates the relational objects, and the metadata exchange creates the mappings from the relational objects to the OLAP objects to populate the DB2 Cube Views metadata.

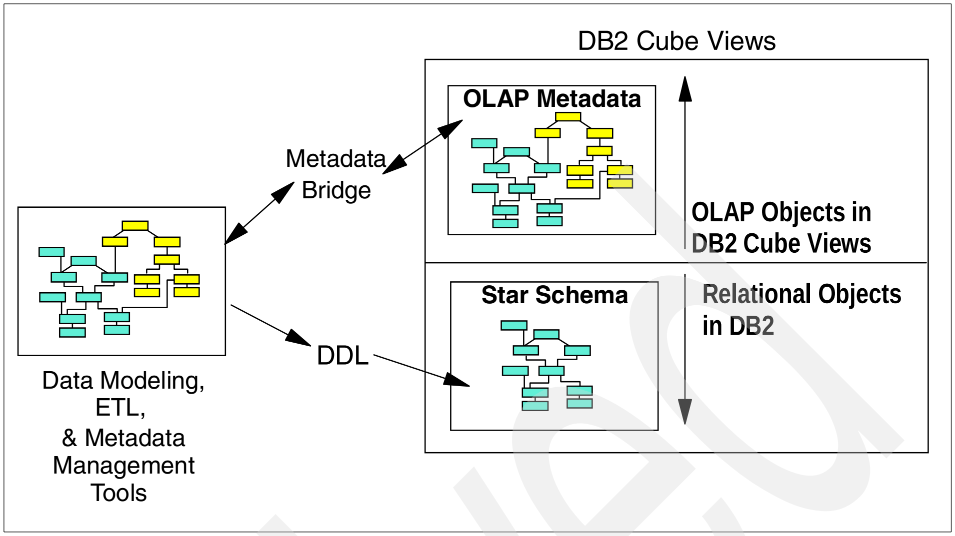


Figure 2-10 Metadata exchange with back-end tools

Benefits

The benefits in this approach are:

- Low administrative effort
- Better cross-tool data understanding
- Data model enrichment

Low administrative effort

By importing the metadata for OLAP objects into DB2 Cube Views via a metadata bridge instead of entering it manually, the data analyst or DBA will save time and guesswork. The tables, columns, and primary and foreign keys have already been created in another tool, so there is no need to re-enter them and possibly introduce errors. Many data modeling tools also capture dimensional objects and map them to the relational objects, so importing their metadata into DB2 Cube Views can save the time that would be spent creating dimensions manually.

Better cross-tool data understanding

Since metadata management tools accept metadata from multiple tools and maintain meaningful relationships between the diverse metadata objects, they can offer their users significant advantages in the form of cross-tool data lineage analysis and cross-tool impact analysis. Using DB2 Cube Views in conjunction with these tools extends their benefits to the DB2 Cube Views objects. For example, a measure in a fact table can be traced back to its origins in an operational database using cross-tool data lineage reporting, and a DBA can see that a change made in a data modeling tool will affect an OLAP object in DB2 Cube Views using cross-tool impact analysis reporting. This reduces errors that might otherwise arise due to false assumptions about data lineage or the possible impacts of changes in design.

Data model enrichment

Let’s say you modeled your basic star schema in a data modeling tool, but you did not model the dimensional hierarchies or derived attributes there. Let’s say further that after you imported this metadata into DB2 Cube Views, you added the dimensional hierarchy and derived attribute information using the OLAP Center. If your data modeling tool has a two-way metadata bridge, then you can send the enhanced model back to the data modeling tool from DB2 Cube Views to share the enriched model metadata without having to enter it in two places. This saves time and reduces the possibility of introducing errors.

2.2.2 Feeding DB2 Cube Views from front-end tools

There are a number of sophisticated data delivery platforms that can serve as excellent front-ends to your warehouse or datamart. They add tremendous value by masking the complexities of the underlying data and they greatly simplify the process of turning data into information and delivering it to the decision makers who need it. These tools accomplish these feats by offering intuitive interfaces that present the data in business terms and organize the data into dimensions to make navigation and drill-down easy for the users. These tools maintain a rich collection of metadata describing the ROLAP structure of the star schema as well as the OLAP objects that translate them into business-friendly terms and the mappings between the relational objects and the OLAP objects. Their metadata is used to formulate queries against the relational tables based on user selections among the OLAP objects. In some cases, the tools use this metadata to create and populate materialized MOLAP databases in order to give their users extra speed or unique MOLAP features like dynamic time series analysis.

A scenario

Let’s say you had already created a rich layer of metadata in your favorite front-end tool before you installed DB2 Cube Views. Since that metadata already contains descriptions of your star schema database and of the OLAP objects related to it, it makes sense to save time and re-work by exporting the metadata from the front-end tool and importing it into DB2 Cube Views using a metadata bridge and get the most complete meta-model.

Flow and components

This metadata flow from front-end tools to DB2 Cube Views is shown in Figure 2-11.

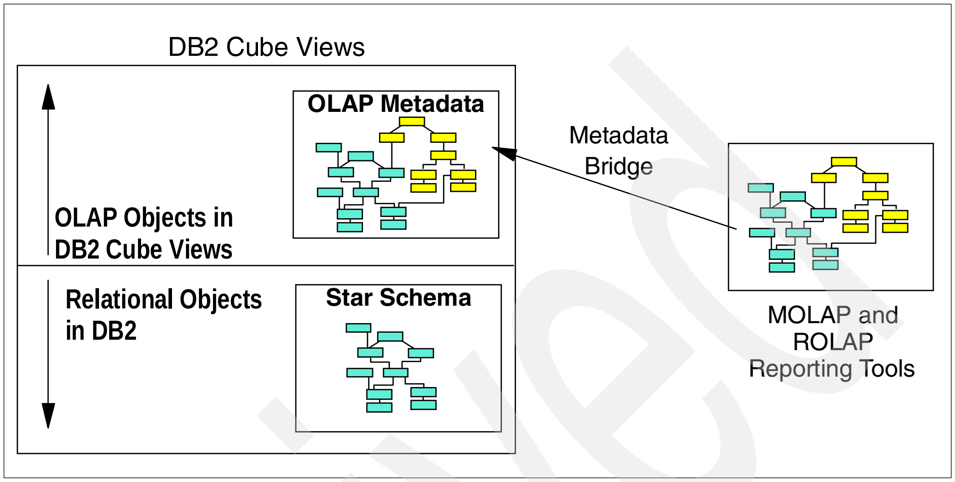


Figure 2-11 Metadata exchange with front-end tools

Benefits

The main benefit will be to speed-up your start-up with DB2 Cube Views.

Speedy start-up

Clearly, any star schema reporting system that was implemented without DB2 Cube Views stands to improve its performance by adding DB2 Cube Views, and you can get started building and using the automatic high performance aggregates (also known in DB2 as Materialized Query Tables, or *MQTs*), as soon you have completed the job of defining your OLAP objects and mapping them to your relational objects. Your data delivery platform already contains the relational and OLAP metadata objects and the mappings that you need, so all you have to do to get the DB2 Cube Views model populated is to import them from the front-end tool via a metadata bridge. This type of metadata exchange is possible with any reporting tool that supports a two-way metadata bridge with DB2 Cube Views, and it will speed you on your way to reaping the performance benefits of DB2 Cube Views.

2.2.3 Feeding DB2 Cube Views from scratch

If importing metadata into DB2 Cube Views is not an option for you, then the cube model can also be built manually. After the relational star schema objects have been created in DB2, and after Referential Integrity (RI) has been defined between them, then, it is time to create the OLAP-related metadata for DB2 Cube Views through a graphical user interface called the OLAP Center. These metadata objects can be entered manually, or they can be *Quick-Started* by a powerful wizard that can logically infer them from your schema. The Quick Start wizard creates the cube model and the corresponding facts, measures, dimensions, attributes, and joins all at once based on your relational schema. You specify the fact table and measure columns, and the Quick Start wizard will detect the corresponding facts, dimensions, joins, and attributes. After you complete the Quick Start wizard, you can add, drop, and modify the metadata objects as needed.

A scenario

Let’s say none of your back-end or front-end tools offers any bridges to DB2 Cube Views. In that case, you will use the OLAP Center to create your OLAP metadata from scratch, using a GUI built especially for that purpose.

Flow and components

We can assume your relational star schema tables already exist, and that you have already set up referential integrity between them. You will start out in OLAP Center by invoking the Quick Start wizard and telling it which table in the database is your fact table. The wizard will then detect the rest of the tables in your star schema and automatically create as many of your OLAP objects from them as it can. Figure 2-12 shows the metadata flows involved.

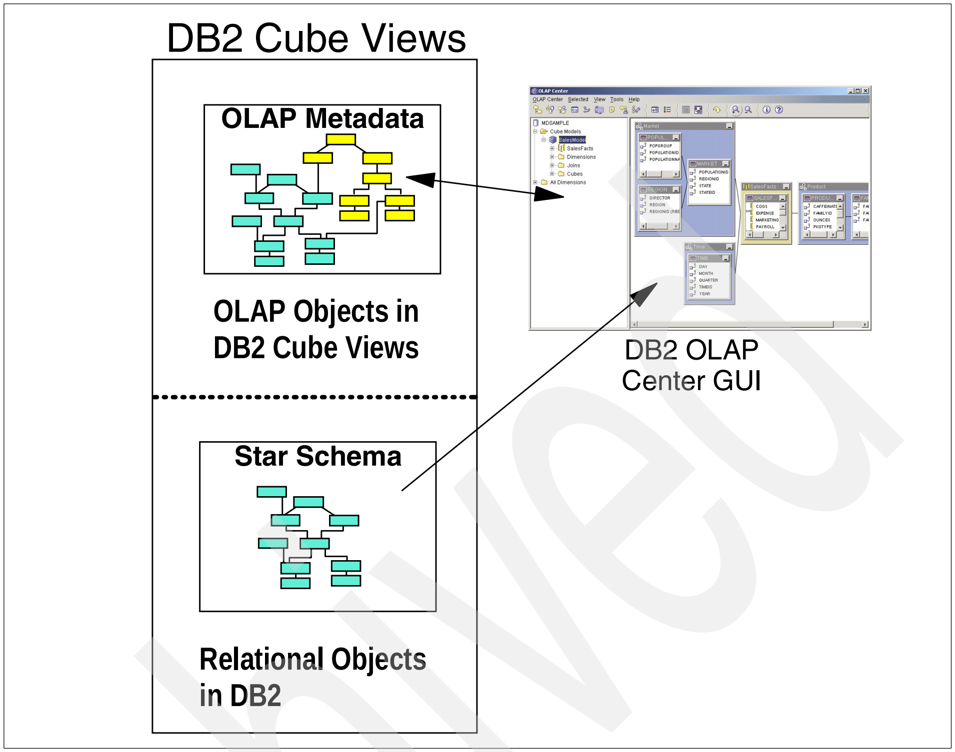


Figure 2-12 Entering metadata from scratch

Benefits

The benefits in this approach are:

- Speedy start-up
- Highly refined OLAP object definitions

Speedy start-up

The Quick Start wizard in the OLAP Center is truly a time saver. By detecting the OLAP objects instead of requiring the user to enter each one manually, the OLAP model is quickly built and the user can spend his time doing further refinements, rather than basic tasks. The Quick Start wizard can detect and create the following objects:

- A cube model that contains all of the other metadata objects.
- A facts object that corresponds to the fact table you specified.
- Measures that correspond to the fact table columns you specified.

- Dimensions that correspond to each dimension table joined to the facts table. Outrigger tables that are joined to a dimension table are included in the appropriate dimension object.
- Attributes that correspond to each column in the dimension and outrigger tables, and to any foreign keys in the facts table.
- Join objects that serve as facts-dimension joins and joins within a dimension object that join the dimension table and any corresponding outrigger tables.

Highly refined OLAP object definitions

In addition to the OLAP objects that can be created with the Quick Start wizard, the OLAP Center offers the capability to create more highly refined OLAP metadata objects in order to capture the exact meanings and intended usages of each object. It also allows you to refine your model by entering object types that do not exist in the other tools you are using to populate DB2 Cube Views.

For example, you may have a model that contains facts and dimensions, but not hierarchies. The OLAP Center has a Hierarchy wizard you can use to create hierarchies for each dimension. A hierarchy can be defined using only one attribute, or it can define relationships between two or more attributes within a given dimension of a cube model. Defining these relationships provides a navigational and computational means of traversing the specified dimension. You can define multiple hierarchies for a dimension in a cube model. This wizard allows you to specify other advanced OLAP objects, such as:

- Hierarchy type (for example., balanced, unbalanced, standard, ragged, network, recursive)
- Hierarchy level
- Attributes associated with each hierarchy level
- Attribute type (for example, associated or descriptive)

Other wizards in the OLAP Center enable the creation of still more metadata objects:

- Dimension type (for example, regular or time)
- Attributes associated with each dimension
- Calculated attributes
- Calculated measures
- New tables
- New measures
- New attributes
- New joins
- Aggregation rules for each measure (for example, SUM, COUNT, MIN, MAX, AVG, STDDEV, script, none)

These advanced objects can be created to complete the cube model after a metadata import or after a Quick Start. Once these advanced objects are created, the cube model is ready for export to other tools or for providing important information to the DB2 Cube Views Optimization Advisor for query performance optimization using aggregates.

2.3 Feeding front-end tools from DB2 Cube Views

When we try to describe the many benefits of using DB2 Cube Views with front-end reporting tools, we have a metadata story to tell and a data story to tell. The metadata story is one of sharing understanding, and Figure 2-13 shows it in the upper set of arrows pointing from left to right. Once the mappings from your relational objects to their OLAP counterparts have been entered or captured, the metadata exists in DB2 Cube Views and is ready to be shared with the front-end tools. Via a custom-tailored metadata bridge, these mappings are sent across to the front-end tools and translated into native metadata understandable to the specific tool. Each time an end user goes to build a new query, this is the metadata that populates the tool's navigation interface screens.

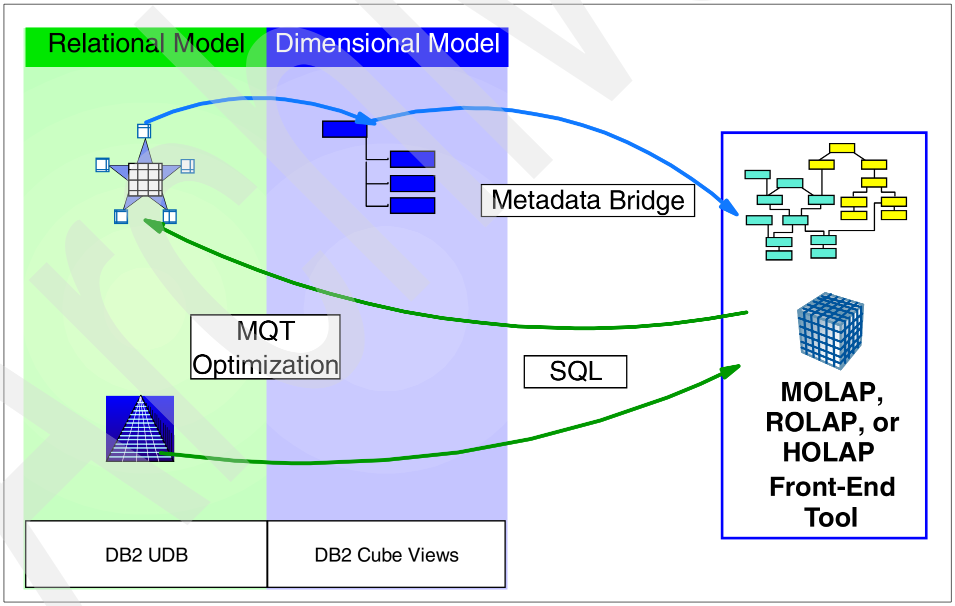


Figure 2-13 Data and metadata exchange

The data story is one of speed and efficiency, owing to the superiority of DB2 Cube Views' automatically-built aggregate tables over manually-built aggregates. Since the aggregates are built directly by the Optimization Advisor, they offer speed because they are so much more likely to be used by the DB2 optimizer to satisfy your SQL queries. Also, since they will often contain multiple slices of aggregated data rather than a single slice, your relational environment will operate more efficiently with fewer aggregate tables to be refreshed. Fewer aggregate tables means shorter overall aggregate refresh time, and less overhead for the optimizer at query time. Figure 2-13 traces the data story with the lower set of arrows that point both left and right. While the end user navigates through the OLAP metadata presented to him in business terms, he chooses the objects he wants to see on reports. Next, the tool internally translates this request into SQL and sends it to DB2. DB2 optimizes the query and uses the pre-built aggregate to satisfy the query quickly and efficiently.

Next, we will take a look at specific types of front-end tools:

- Multidimensional OLAP or *MOLAP* tools.
- Relational OLAP or *ROLAP* tools.
- Hybrid OLAP or *HOLAP* tools

Each scenario is a little different from the others, and each will offer the user some unique benefits. It is entirely possible that your plans will include implementing more than one of these types of tools. If so, then DB2 Cube Views will offer you the additional benefit of allowing you to collect your OLAP metadata in one central place, namely in DB2, and then share it many times with all your reporting tools via metadata bridges.

2.3.1 Supporting MOLAP tools with DB2 Cube Views

A Multidimensional OLAP or *MOLAP* tool is one that offers pre-built multidimensional databases or *MOLAP cubes* to its users. These proprietary non-relational database tools provide extremely fast query response times that enable speed-of-thought analysis capabilities to their users. Their exceptional data retrieval speed is made possible by the underlying array structure of the database, as well as by the presence in the database of pre-built aggregates for every measure at every level of hierarchy of every dimension. These databases perform by anticipating and pre-optimizing every possible query that could be thrown at them.

A scenario

Let's say you decide to build a MOLAP cube for your users, using the data in your star schema as the source of the cube. If you were to compare the data in your MOLAP cube to the data in your relational star schema, you would probably notice a few striking differences. The first difference you are likely to notice is one of *grain*. Grain refers to the lowest level detail data in terms of dimensional hierarchy that is stored in the database.

You can see an example in Figure 2-13 on page 39, where a row of data in your relational fact table, might represent all sales of one size and variety of soup (in other words, one Stock Keeping Unit or *SKU*), sold on one given day in one given store to one given customer. In that case, the grain of that fact table would be the intersection of SKU + day + store + customer. Your MOLAP cube, on the other hand, is customized for a certain group of users and has a different grain.

For example, a leaf-level cell in your MOLAP database might represent all the GrandMa's soup sales (that is, all the GrandMa's Soup SKU's combined) for one month for one state for all customers from the same zip code. In that case the grain of the MOLAP database could be said to be the intersection of product group + month + state + customer zip. Clearly, the MOLAP database in this case would represent an aggregation of the fact table data and it would have a different grain from that of the fact table. Figure 2-14 shows the cube as a subset of the star schema.

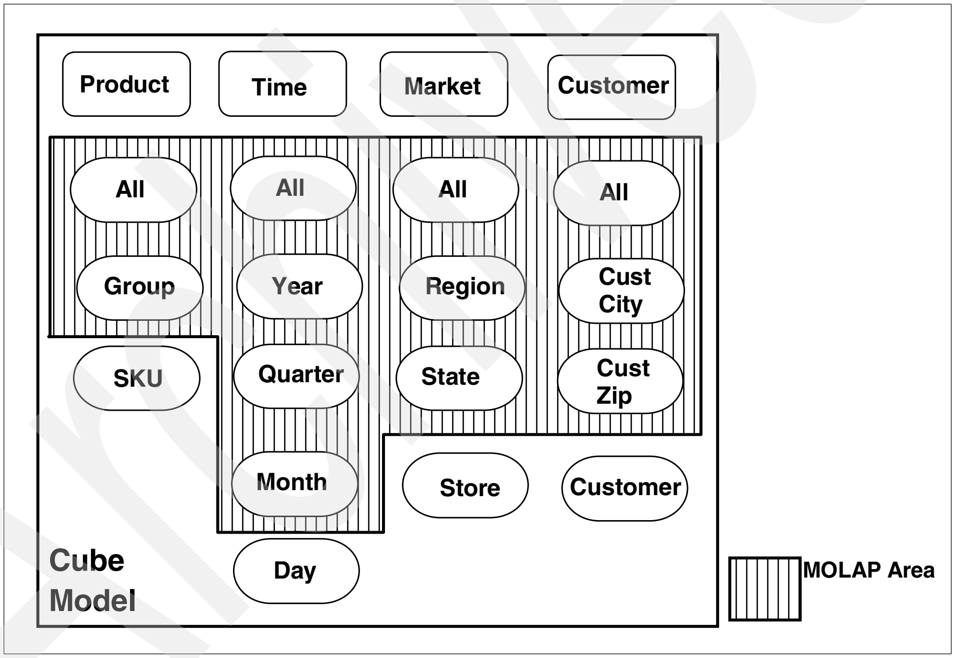


Figure 2-14 MOLAP with higher grain from that of star schema

Another difference that might exist between your relational star schema and a MOLAP database built from it is one of *dimensionality*. Building on the example above, let's assume you want to build a second MOLAP database from the same fact table data. This time, you want each leaf-level cell in the cube to represent all the GrandMa's soup sales for one month for one state for all customers. In other words, you do not want to keep of track of the sales by customer at all in this cube, just the sales by product, store, and date.

Similar to the last example, the MOLAP database in this example is also an aggregate of the fact table, and it also has a different grain from the fact table. This time, it also has one less dimension, and all the data is aggregated to the "All Customers" level. Figure 2-15 shows this second cube as a different subset of the star schema, with different dimensionality from the star schema.

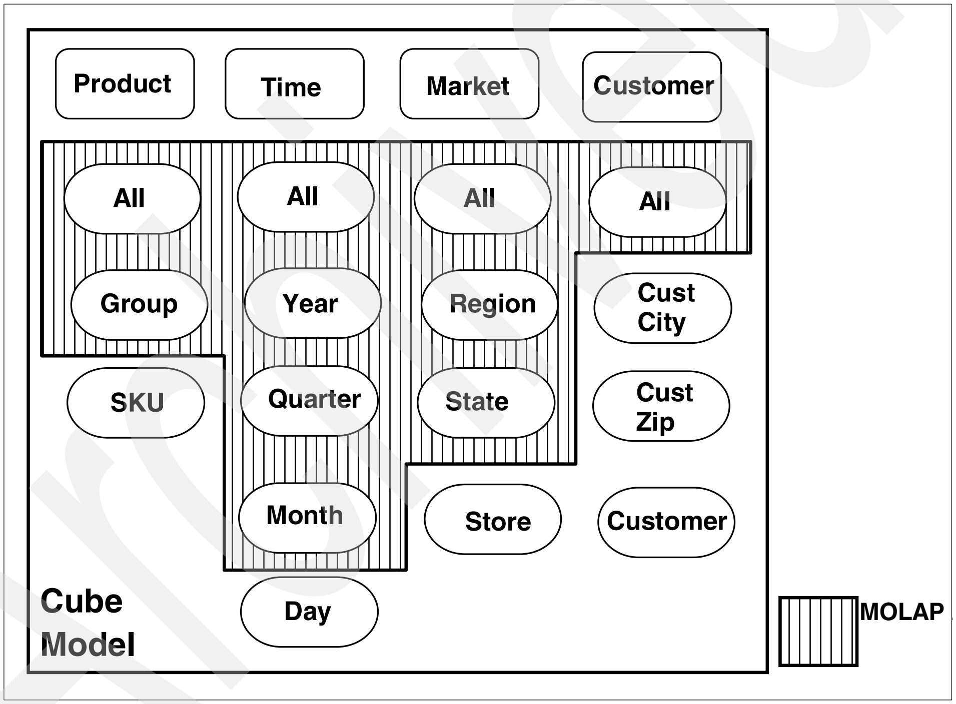


Figure 2-15 MOLAP with different dimensionality from that of star schema

Differences in grain and dimensionality between the fact table and the MOLAP database make it necessary to aggregate the fact table data in order to load it efficiently into the lowest-level cells of the MOLAP database. The MOLAP tools understand this and generate SQL containing appropriate aggregation grouping constructs when it loads the relational data into the MOLAP databases.

Flow and components

Figure 2-16 shows both the metadata flow and the data flow between DB2 Cube Views and a front-end MOLAP tool. The metadata flow, represented by the upper set of arrows pointing from left to right, contains the mappings from relational objects to OLAP objects that exist in DB2 Cube Views. These OLAP objects contain enough information for the front-end tool to build its cube structures, consisting of measures, dimensions, and hierarchies. The mappings from relational objects to OLAP objects provide the front-end tool with sufficient information so that it can construct the SQL statements it will need to extract data from the relational tables and populate the MOLAP databases.

The data flow, represented by the lower set of arrows pointing both directions, carries the SQL extract request from the MOLAP tool to DB2 and the result set data from the relational tables to the MOLAP database at load time. Notice that the pre-built aggregate is being used as the source of this load rather than the relational base tables, even though the front-end tool created its extract SQL based on the base tables.

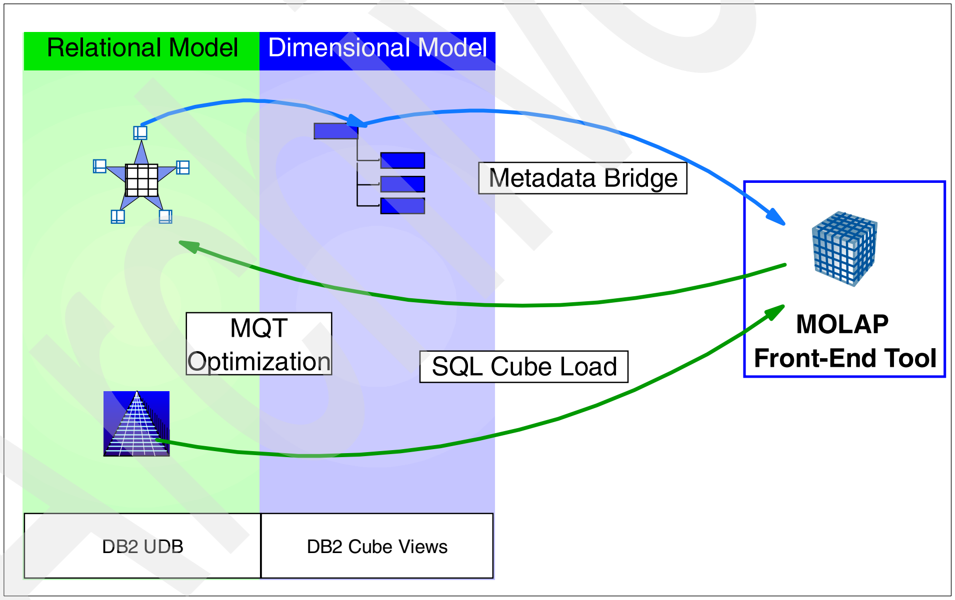


Figure 2-16 Data and metadata exchange with MOLAP tools

Benefits for MOLAP users

The benefits in this approach are:

- Low administrative effort
- Fast MOLAP cube loads and refresh
- Economies of scale

Low administrative effort

Once your DB2 Cube Views metadata is populated, you can realize an administrative benefit by passing it over to your MOLAP tool via a metadata bridge. Rather than having to use the MOLAP tool's GUI to describe your relational environment as a collection of facts and dimensions, and then define one or more hierarchies over each dimension and then map one or more MOLAP database definitions to your relational model, all this work will have been done for you as an automatic by-product of running the bridge, saving considerable time.

Fast MOLAP database loads and refresh

DB2 Cube Views can automatically build aggregates that are tailored specifically to optimize your MOLAP data loads and refreshes. If want to optimize for an extract to MOLAP, you first create a cube definition within your cube model that corresponds to your MOLAP database in terms of grain and dimensionality. Figure 2-17 shows an example of such a cube definition.

Based on this knowledge, the Optimization Advisor can create an MQT that contains a slice of fact table data aggregated to match the lowest level of data to be loaded to your MOLAP database. Once this is done, your loads and refreshes will be run considerably faster, since there is no additional aggregation that needs to be done at load time.

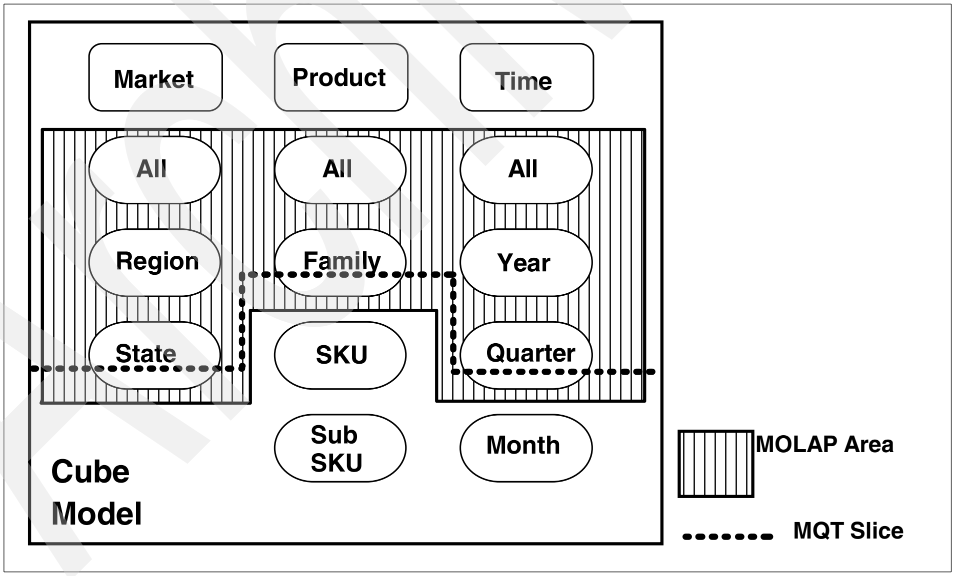


Figure 2-17 MOLAP scenario

These are the assumptions in Figure 2-17:

- Your cube model has three dimensions: Market, Product, and Time.
- Your cube will hold data by market down to the state level, by product down to the family level, and by time down to the quarter level (this is the MOLAP area in Figure 2-17).

It could be argued rightly that the time it takes to refresh an MQT takes away some of the benefit described just above, but by no means all of it. Since the MOLAP database will be unavailable to the end users during the MOLAP load, this downtime can be considerably reduced by moving the aggregation work from the MOLAP tool to the relational database in the form of the MQT refresh. Also, if your schedule is such that the periodic fact table refresh can be scheduled well ahead of the periodic MOLAP load, then the MQT refreshes can be done early, along with the relational update, when there is less pressure on the load window. Another key inherent advantage of MQTs is that they are available to all users of the data warehouse, not just a particular MOLAP tool. This is a general benefit of MQTs overall vis-à-vis MOLAP databases.

Economies of scale

The benefit of fast MOLAP database load is greatly increased in certain situations, reaping many times the basic benefit. Time savings are multiplied when you are building several MOLAP databases from the same star schema data. Grain and dimensionality will vary from MOLAP database to MOLAP database, but DB2 Cube Views can build an MQT that will be shared by multiple MOLAP database loads. If you want this benefit, then you will define one cube within your cube model that represents a combination of all your MOLAP databases, or a *superset,* of all the MOLAP databases you want to load, in terms of grain and dimensionality. In these cases, the time saved during the MOLAP loads is many times the time spent refreshing the MQTs because there are multiple loads, but only one MQT.

Another multiplied benefit can be realized because of the MQT's ability to accept incremental refreshes. Every time the fact table is updated with new data, the MQTs associated with it will need to be refreshed as well, so that they will stay synchronized and usable by the DB2 optimizer. If the MQT is built using SUM and COUNT column functions, then it is capable of being refreshed incrementally, rather than having to be rebuilt from scratch each time the fact table is updated. The multiplied benefit is realized in shops where the star schema data is updated more frequently than the cubes. For example, if the relational fact table is updated once a day and the MOLAP databases are refreshed once a week, then only a fraction of the aggregation work for the week has to be done on the day of the MOLAP load. This can add up to a tremendous benefit in time saved.

2.3.2 Supporting ROLAP tools with DB2 Cube Views

A Relational OLAP or *ROLAP* front-end tool is one that offers its end users a reporting platform based on business facts and dimensions that are logically mapped to the physical objects in a relational star schema database. They differ from MOLAP tools in that their users are not limited to the boundaries of a cube for their reporting. Rather, they can drill down all the way to the lowest grain of data in the star schema. Often, this means a great deal more detail is available to the end users. These tools can offer standard columnar reporting, but they also go beyond that to offer analytical slice-and-dice and drill-down reporting as well. This analytical reporting necessitates a high degree of aggregation activity within the relational database.

A scenario

Let's assume you have built the same star schema as the one described in the MOLAP scenario. This time, however, you are not going to offer any MOLAP cubes. Instead, you want to allow your users to access the entire star schema directly, in ROLAP fashion, producing reports at any and all levels of grain and dimensionality. Let's assume you want to allow them to be able to do both *Drill-down* queries and general *Report* queries. Drill-down queries produce reports that typically present the end user with a view of the data that corresponds with the very highest level of aggregation of your data, and then allows the user to drill-down, dimension by dimension, until he can see the data that is most interesting to him. By contrast, report queries can start with a query that is equally likely to access any part of the cube model, and possibly offer drill-down from there.

Flow and components

Figure 2-18 shows both the metadata flow and the data flow between DB2 Cube Views and a front-end ROLAP tool. The metadata flow, represented by the upper set of arrows pointing from left to right, contains the mappings from relational objects to OLAP objects that exist in DB2 Cube Views. These objects and mappings populate the metadata layer of the ROLAP tool automatically and provide it with sufficient information so that it can construct the SQL statements it will need to extract data from the relational tables and produce reports.

The data flow, represented by the lower set of arrows pointing both directions, carries the SQL data retrieval request from the ROLAP tool to DB2 and the result set data from the relational tables to the ROLAP tool at report query time. Notice that the pre-built aggregate is being used to satisfy this query rather than the relational base tables, even though the front-end tool constructed its SQL to read the base tables.

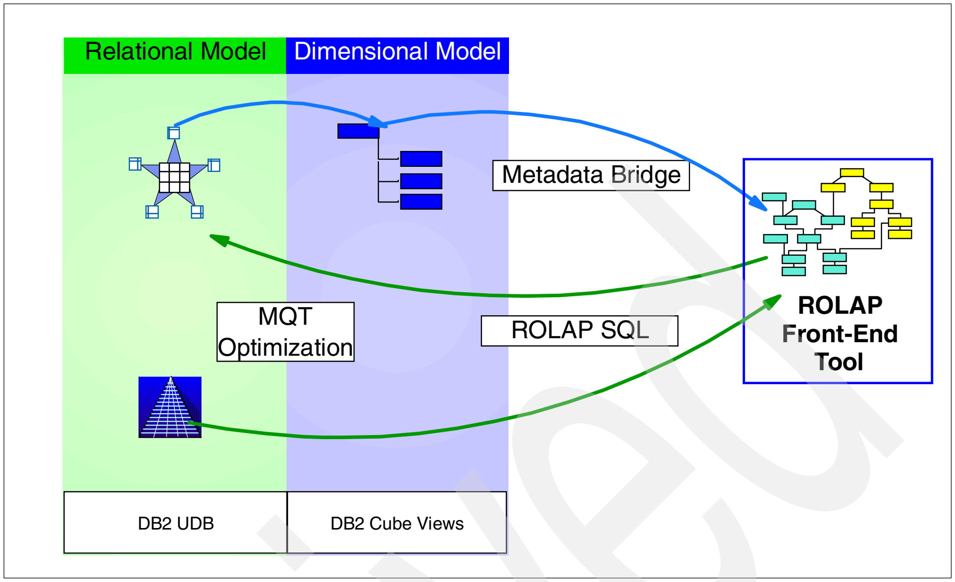


Figure 2-18 Data and metadata exchange with ROLAP tools

Benefits for ROLAP users

The benefits in this approach are:

- Fast query response
- Low administrative effort
- Economies of scale

Fast query response

DB2 Cube Views will automatically build aggregates that are optimized for your ROLAP queries. There are two types of ROLAP queries that can be optimized, drill-down and report, and you can tell DB2 Cube Views whether you want either one of these, or both. For ROLAP tools, which do not use cubes, drill-down reporting can quickly create performance problems starting with the very first display, since it could very well consist of an aggregation of the entire star schema!

If you tell DB2 Cube Views that you want to optimize for drill-down, then it will build an MQT for you that has a concentration of aggregations at the middle levels of your dimensions as well as dense aggregations at the top levels of your cube model. Then at query time, your end users will get fast response times from start to finish, without your having to build or maintain cubes for them. See Figure 2-19 for a graphical look at the dense rollup plus the additional aggregation slices that would be built into your MQT in this scenario.

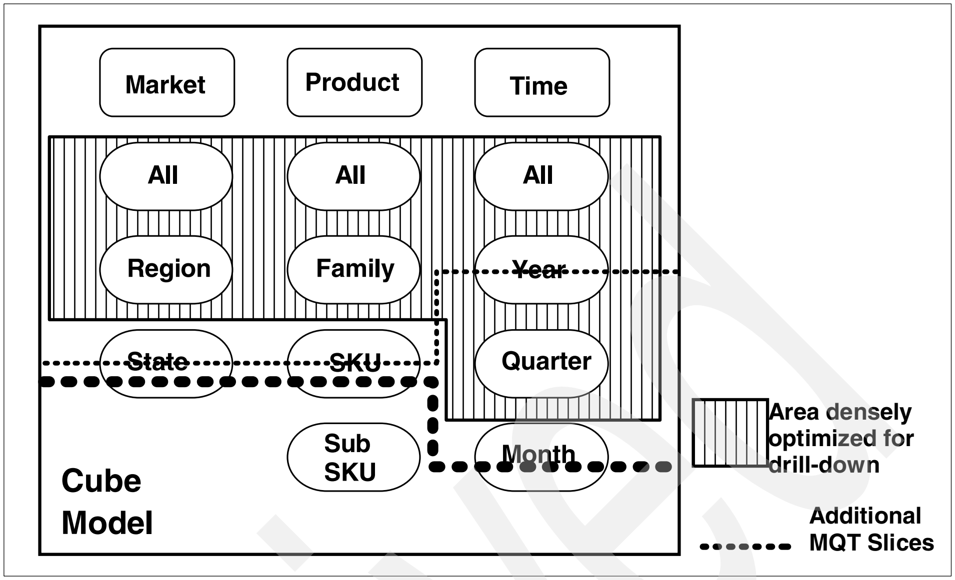


Figure 2-19 ROLAP scenario: drill-down

The second optimization option, Report, is the most generalized of all query types. If you indicate through the OLAP Center that you want this type of optimization, then the Optimization Advisor will build you an MQT similar to the one built for Drill-down, but without the dense rollup at the top. The MQTs that are built to support ROLAP reporting contain multiple slices of aggregated data, so that as many queries as possible can be re-routed to the MQT by the DB2 optimizer. See Figure 2-20 for a depiction of the multiple-slice MQT that DB2 Cube Views might build for you in this situation.

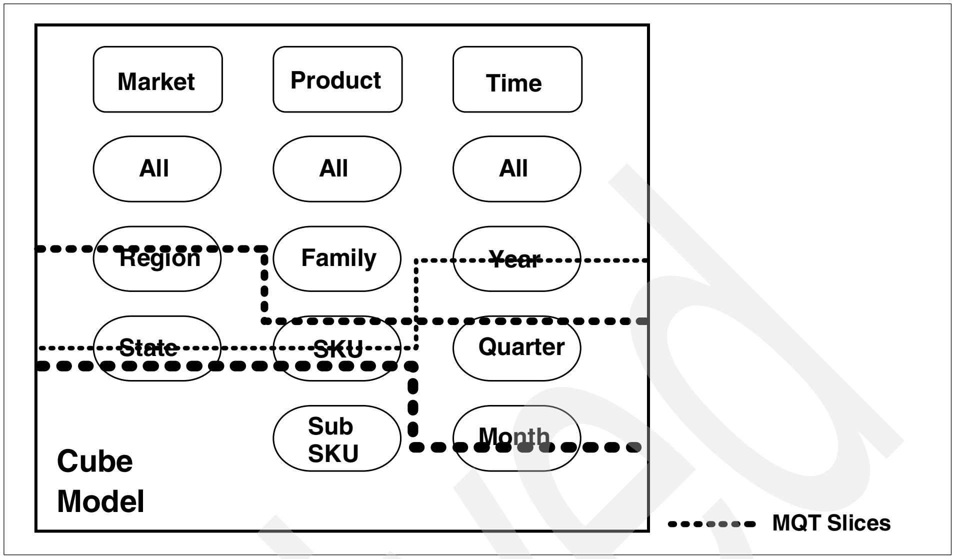


Figure 2-20 ROLAP scenario: report

Low administrative effort

Administrative benefits can be achieved when DB2 Cube Views is used in conjunction with ROLAP tools, both on the metadata side and on the data side. On the metadata side, once the DB2 Cube Views metadata has been populated, the task of building all or most of the metadata layer of the ROLAP tool is reduced to a metadata transfer via a bridge. After the metadata transfer, your tool's semantic layer of tables, columns, joins, facts, dimensions and hierarchies has been automatically built to match the DB2 Cube Views environment and is ready for use, saving your administrator considerable time and reducing the possibility of introducing errors into the process. If you are using multiple ROLAP tools, this benefit only gets more pronounced.

The largest benefits are on the data side. Since ROLAP tools access very large tables and do not use cubes, their analytical queries must depend on pre-built aggregates in order to achieve outstanding performance. The aggregate tables built by the DB2 Cube Views expert system satisfy this requirement extremely well because they are chosen by the DB2 optimizer in a high percentage of cases. Although most of these tools offer their own aggregate awareness features that access pre-built aggregate tables instead of using the base star schema tables when they can, using aggregate-awareness features outside of DB2 adds a considerable amount of administrative overhead to the configuration and use of the ROLAP tool. By relying instead on DB2 and DB2 Cube Views to create and maintain the aggregate-awareness in the overall system, administration is greatly simplified and much more efficient.

Economies of scale

If you are using multiple ROLAP tools in your enterprise, then the benefits of using centralized aggregate tables maintained by DB2 instead of maintaining aggregate-awareness components within the ROLAP tool are even greater. By keeping all the aggregate-awareness within DB2, the administration of the overall system is kept centralized, and every time a new aggregate is added to the solution or an existing aggregate is changed or deleted, there is no further maintenance needed in any of the ROLAP tools that use the relational database. The ROLAP tools only need to be aware of the base tables, not of the aggregates.

2.3.3 Supporting HOLAP tools with DB2 Cube Views

A Hybrid OLAP or *HOLAP* tool is one that combines the features of a MOLAP tool with those of a ROLAP tool. They can be seen as providing the best of both worlds because their users can exploit the extremely fast retrieval times achieved through the use of MOLAP database, as well as having access to the lowest level of data grain stored in the relational database, like ROLAP tools. For example, administrators might choose to define the base level of the cube one or two layers higher, and allow drill-through to MQTs provide the missing lower level aggregates. A higher base level provides a smaller MOLAP database, less disk, faster load or refresh. It basically raises the “hybrid line”, implements more of the (original) cube design in DB2 as MQTs, and increases utilization of the warehouse within the overall OLAP continuum (the conceptual spectrum of analysis encompassing both the MOLAP database and the DB2 warehouse).

A scenario

Let’s assume, once again that you have built the same star schema as you did for the MOLAP and ROLAP scenarios, but this time you want it all. You want the speed of MOLAP for those queries that can be resolved within your cubes, and you want optimized ROLAP for those queries that stray outside your MOLAP boundaries. Also, you want all the slicing and dicing to appear seamless to your end users, regardless of which database is used to satisfy their queries (see Figure 2-21).

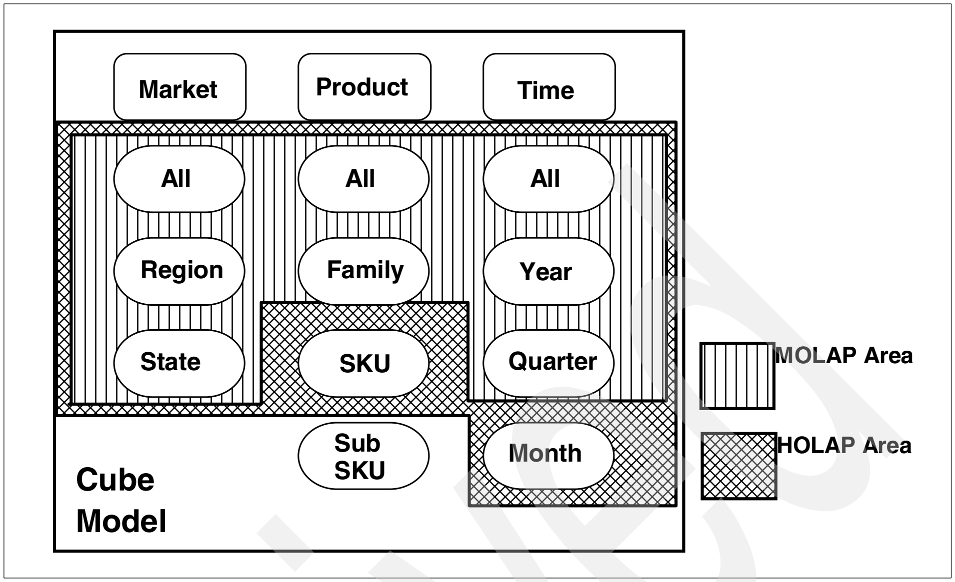


Figure 2-21 HOLAP scenario

Figure 2-21 assumptions are:

- Your cube model has three dimensions: Market, Product, and Time.
- Your cube will hold data by market down to the state level, by product down to the family level, and by time down to the quarter level (this is the MOLAP area in Figure 2-21).
- The HOLAP drill-through area will include data by product below the family level down to the SKU level (but not to the sub-SKU level). This dimension has the most values of any dimension by far.
- The HOLAP drill-through area will also include data by time down to month.

DB2 Cube Views can automatically build aggregates that are tailored specifically to optimize your HOLAP drill-throughs. If you want these queries optimized, you first create a cube definition within your cube model that corresponds in terms of grain and dimensionality to the HOLAP area you expect will be hit by drill-through queries. In this case, what you define in the OLAP Center as your cube actually represents more data than will be held in your MOLAP database because you are specifying the HOLAP area rather than the MOLAP area with your cube definition.

Flow and components

Figure 2-22 shows both the metadata flow and the data flow between DB2 Cube Views and a front-end HOLAP tool. The metadata flow, represented by the upper set of arrows pointing from left to right, contains the mappings from relational objects to OLAP objects that exist in DB2 Cube Views. These objects and mappings populate the metadata layer of the HOLAP tool automatically and provide it with sufficient information so that it can construct the SQL statements it will need to extract data from the relational tables and produce reports.

The data flow, represented by the lower set of arrows pointing in both directions, carries the SQL data retrieval request from the HOLAP tool to DB2 and the result set data from the relational tables to the HOLAP tool at MOLAP load time or at report query time. Notice that the pre-built aggregate is being used to satisfy these queries rather than the relational base tables, even though the front-end tool constructed its SQL to read the base tables.

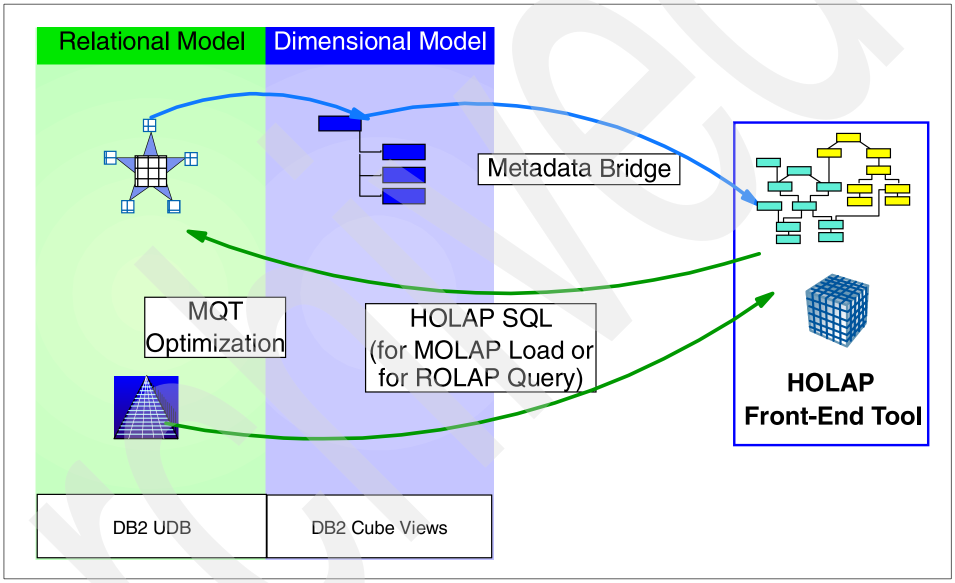


Figure 2-22 Data and metadata exchange with HOLAP tools

Benefits for HOLAP users

The benefits in this approach are:

- Low administrative effort
- Fast drill-through reporting
- Potential to design smaller MOLAP databases
- Fast MOLAP database loads

Low administrative effort

Similar to the MOLAP and ROLAP scenarios, considerable administrative time and trouble can be saved by populating your HOLAP tools' metadata layer directly from DB2 Cube Views by using a metadata bridge. This process carries information from DB2 Cube Views about all the tables and columns in your relational star schema and their relationships to the OLAP facts, dimensions, hierarchies, joins, and cubes that your HOLAP tool will use to construct and load its MOLAP databases and to create drill-through queries.

Fast drill-through reporting

When a HOLAP tool user is slicing and drilling within the MOLAP database in the tool, he is experiencing fast response times because of the optimized array structure of the MOLAP database. But the story changes as soon as the user attempts to drill below the boundary of the MOLAP database. This action is called *Drill-through* and it is at this point that the HOLAP tool will have to begin generating SQL queries to deliver the report data, and the user will be subject to relational query response times.

Based on this knowledge, the Optimization Advisor can create an MQT that contains multiple slices of fact table data aggregated to optimize both the load of your MOLAP database and your drill-through queries that go below the MOLAP database. When the advisor is choosing which slices to build, it gives highest priority to the dimension with the greatest number of distinct values, in other words, the dimension with the greatest *cardinality*, because drill-through queries in this dimension will benefit the most from using an MQT. Once the MQT is built, your MOLAP loads will run considerably faster, since there is no additional aggregation that needs to be done at MOLAP database load time, and your drill-through queries will also be optimized.

In our scenario, DB2 Cube Views will understand that since the cardinality of the product dimension is very high, then the cost to aggregate in that dimension will be expensive, and give high priority to building an MQT that has a slice of data that aggregates the product dimension lower than the boundary of the cube. On the other hand, since there are only 3 months per quarter, pre-building an aggregate by month would get lower priority. Consequently, the slices of data just below the cube grain that are likely to be hit at drill-through time have been optimized.

Potential to design smaller MOLAP databases

Given the acceleration of drill-through to low-level aggregates, administrators might choose to define the base level of the cube one or two layers higher, and allow drill-through to MQTs provide the missing lower level aggregates. A higher base level provides a smaller MOLAP database, less disk, faster load or refresh. It basically raises the “hybrid line”, implements more of the (original) cube design in DB2 as MQTs, and increases utilization of the warehouse within the overall OLAP continuum (the conceptual spectrum of analysis encompassing both the MOLAP database and the DB2 warehouse).

Fast MOLAP database loads and refreshes

The aggregate that DB2 Cube Views builds to optimize your HOLAP drill-through queries described in the preceding paragraphs will also be used to optimize the MOLAP load piece of your HOLAP solution. In this case, you will specify in OLAP Center that you want Drill-Through, and not MOLAP Extract. If you want this optimization, you will have first created a cube definition within your logical cube model that corresponds in terms of grain and dimensionality to the HOLAP area that you expect will be hit by drill-through queries, even though this actually represents more data than will be held in your MOLAP. Refer the HOLAP area in Figure 2-21 on page 51.

As stated above, these actions will produce an MQT that optimizes both your MOLAP load and your drill-through queries and the MOLAP load and refresh will get optimized by the drill-through optimization. Your loads and refreshes will be run considerably faster, since there is no additional aggregation that needs to be done at load time.

2.3.4 Supporting bridgeless ROLAP tools with DB2 Cube Views

These tools have no metadata bridges because they have no need to store any OLAP metadata locally.

In the past, reporting tools that were not equipped to store any OLAP metadata were not able to offer their users any OLAP slice and dice or drill-down reporting at all, even if they were connected to star schema relational databases. Since DB2 Cube Views stores the OLAP metadata centrally in the database, that inability is now gone. DB2 Cube Views has an application programming interface or *API* that allows any program or tool to access the OLAP metadata directly to retrieve OLAP-aware information about the underlying star schema. This way, the end user sees only OLAP objects to choose from, and he never has to see anything about tables, columns, joins or SQL. Several tools have already incorporated this API and by doing so they have transformed themselves into bridgeless ROLAP tools.

A scenario

Let’s assume that you have the same star schema as was built for all the previous scenarios, but the difference this time is your front-end tool. Instead of deploying a sophisticated (and expensive) enterprise-wide data delivery platform rich with metadata and report distribution options, your organization has opted for a less expensive tool that will offer your users OLAP-style navigation through your star schema data using only the resources contained immediately in your DB2 database.

Flow and components

Figure 2-23 shows both the metadata flow and the data flow between DB2 Cube Views and a front-end bridgeless ROLAP tool. The metadata flow, represented by the upper set of arrows pointing from left to right, contains the mappings from relational objects to OLAP objects that exist in DB2 Cube Views. Since the bridgeless ROLAP tool does not have a metadata layer of its own, the DB2 Cube Views metadata is read on the fly via the DB2 Cube Views API. Once this metadata has been read and understood, the tool can display the OLAP facts, dimensions and hierarchies graphically to its end users who can then choose which facts and dimensions they want to see.

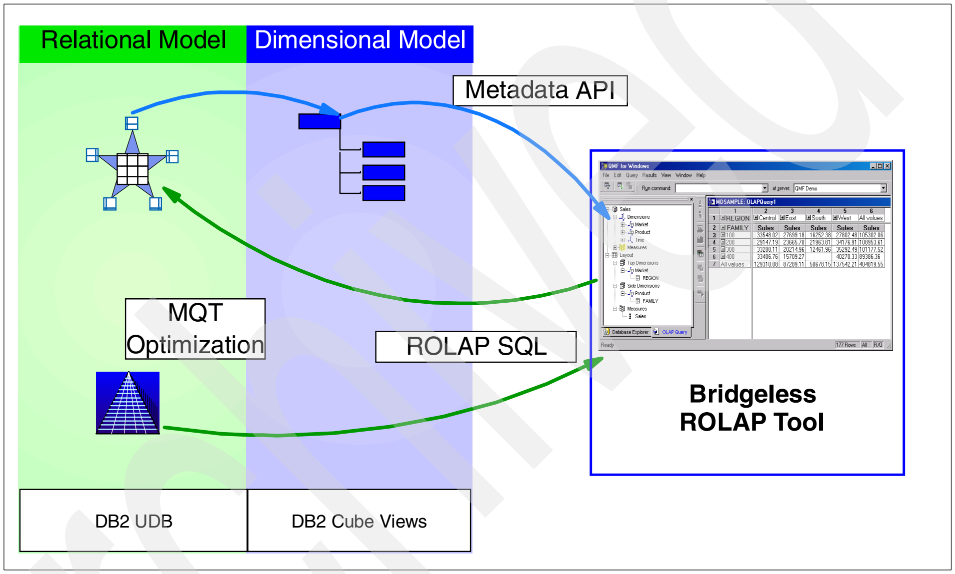


Figure 2-23 Data and metadata exchange with bridgeless ROLAP tools

The data flow begins as the user clicks on the displayed OLAP metadata objects. The bridgeless ROLAP tool uses the information in the relational mapping metadata retrieved earlier via the API to construct the SQL statements needed to extract the corresponding data from the relational tables and produce reports. Notice that the pre-built aggregate is being used to satisfy the query rather than the relational base tables, even though the bridgeless ROLAP tool constructed its SQL to read the base tables.

Benefits for bridgeless ROLAP users

The benefits in this approach are:

- Fast query response
- Low administrative effort

Fast query response

DB2 Cube Views will automatically build aggregates that are optimized for your bridgeless ROLAP queries in exactly same way it would construct them for any other ROLAP tool. The two types of query optimization are the same, Drill-down and Report, and the dense rollup and multiple slice MQTs that will be generated are also the same. An example of this type of MQT is shown in Figure 2-19 on page 48.

Low administrative effort

This scenario, using DB2 Cube Views with bridgeless ROLAP tools, has the lowest level of administration of any scenario in this chapter. In fact, administration in the tool can be said to be zero, because all of the administration is accomplished completely within DB2 Cube Views, and there is no administration left for the bridgeless ROLAP tool to do at all. All the mappings of relational objects to OLAP objects are already done and stored in the DB2 database, and all the performance-optimizing MQTs have already been built via the OLAP Center. The bridgeless ROLAP tool only needs to read the OLAP metadata and construct SQL statements. The bridgeless ROLAP tool administrator can work on something else altogether.

2.4 Feeding Web services from DB2 Cube Views

More and more, businesses are looking at Web services to satisfy their ever increasing need to answer business questions in real time. Web services offer the necessary infrastructure to link the applications that need information to the applications that have it. All kinds of information will be provided as services, including OLAP information. Applications will be able to retrieve dimensional information, determine the slices or cells that they need, and retrieve the data without having to learn any OLAP interfaces or query languages. Web services developers will be able to call on their existing knowledge of XML and XPath to quickly add analytic information to their applications.

> **Note:** DB2 Cube Views Web services, available from the alphaWorks IBM Web site:
>
> `http://www.alphaworks.ibm.com.`
>
> This is provided as a Technology Preview, and the available code should be seen as examples rather than a complete library for building custom Web applications.

2.4.1 A scenario

A company called Grocery Max has 100 grocery stores located in California and Washington State. In order to keep track of its customer habits, Grocery Max offers its customers a rewards card that captures customer information (names, items, totals) at check-out time. However, the data captured is not sufficient to explain an increasing demand for bread, cheese, and fine wines in several stores in California.

In its search to find external information on its customers, the Grocery Max folks discovered a Company called Cross References that specializes in demographic data collection and is also a Web services provider. After finding the description of Cross References' Web services on a public UDDI registry, Grocery Max decided to use Cross References' OLAP data to augment their own. By doing so, they learned that 30% of the locals and potential customers of the selected stores had French origins. In the light of this new information Grocery Max executives decided to increase supply of French products for the selected stores.

2.4.2 Flow and components

Figure 2-24 shows both the metadata flow and the data flow when OLAP is offered as a Web Service using DB2 Cube Views. An OLAP provider might offer one Web service to retrieve cube model metadata such as fact names and dimension names and another Web service to retrieve a list of possible values for a dimension. This information, returned to the caller in an XML document, will give the calling application enough information to format a data query using XPath.

In addition to the metadata services described above, the OLAP provider might offer a Web service to retrieve the OLAP data based on the fact names and dimension values that were previously retrieved in the metadata and to translate the caller's XPath statements into SQL. This data flow is represented by the lower set of arrows pointing in both directions, which carry the SQL data retrieval request from the Web service to DB2 and the resulting dataset from the relational tables back to the Web service. Notice that the pre-built aggregate is being used to satisfy this query rather than the relational base tables, even though the SQL was constructed to read the base tables.

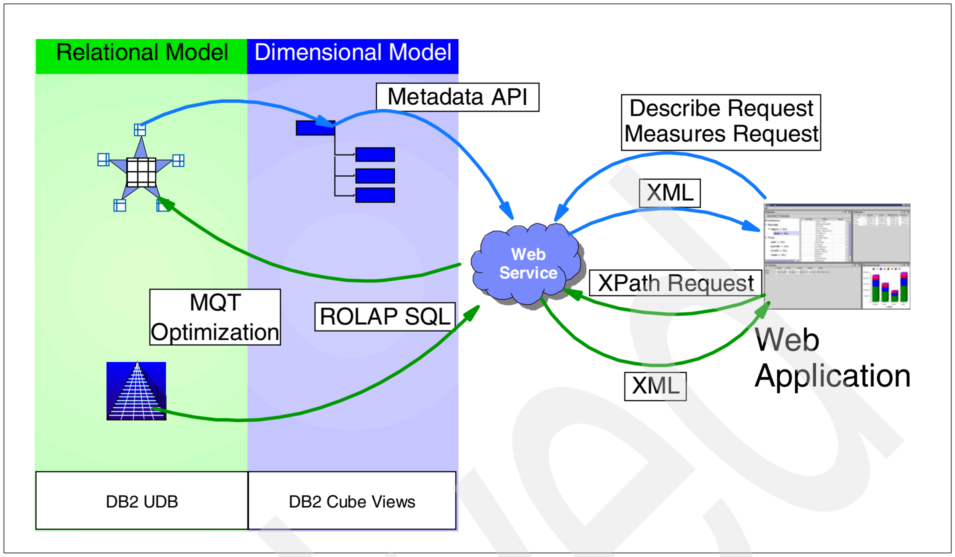


Figure 2-24 Data and metadata exchange as a Web Service

2.4.3 Benefits

The benefits in this approach are:

- Easy integration of information
- Easy access to remote information

Easy integration of information

Let's say someone in your organization needs to access data that is stored in multiple OLAP systems throughout your enterprise. Without the Web services infrastructure, he would need to learn how to use the appropriate front-end OLAP tool, possibly multiple tools, gain permission to use them, and probably manually paste together the OLAP data retrieved from each one. By designing and using an application to retrieve this OLAP data through Web services, he can get the information in an automated way, with each piece of information precisely defined, much the same way that users get information by browsing through Web sites.

Easy access to remote information

Using Web services in this way to access OLAP data makes the process easier on every level. No longer does a user of OLAP data need to have specialized tools and specialized skills. If he has access to an application created using OLAP Web services, he needs only a browser with a secure Web connection. Administrators in this environment need only to configure Web servers instead of maintaining multiple protocols. Also, these benefits are greatly increased if multiple applications that use OLAP Web services are integrated together into portals.

Part 2

Build and optimize the DB2 Cube Model

In this part of the book we describe how to start building the DB2 cube model using OLAP Center and how to start practicing with the Optimization Advisor from DB2 Cube Views.

3

Building a cube model in DB2

The DB2 Cube Views metadata model is a key functionality in DB2 which enables the RDBMS become OLAP aware through multidimensional metadata management in the database catalogs.

Database management systems have always played a key role in the deployment of OLAP solutions as the source and support for dynamic data queries and drill through reports.

Cube modeling in DB2 UDB V8.1 now makes the database aware of the higher level of organization that OLAP requires by building metadata objects in the DB2 catalog in the form of dimensions, hierarchies, attributes, measures, joins, and so on. This metadata model is very strong and complete and capable of modeling a wide range of schemas from simple to complex. It adds value not only to DB2 but also to the tools and applications that access such dimensional data through simple DB2 interfaces. It allows access of dimensional intelligence in the form of facts, dimensions, hierarchies, attributes to be exchanged from back-end tools through the database to front-end tools. Metadata needs to be defined only once and is then available to all tools and applications that need such metadata.

3.1 What are the data schemas that can be modeled?

Multidimensional databases have gained widespread acceptance in the market for supporting OLAP applications. The DB2 Cube Views model provides semantic foundation to multidimensional databases and extend their current functionality. This is a strong, flexible and complete model that is capable of modeling a wide range of schemas. Please refer to Ralph Kimball's definitions of dimensional schemas in *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling (Second Edition)* by Ralph Kimball and Margy Ross, April 2002, ISBN 0471-200247.

OLAP tools and applications that interface with DB2 have each a different view of OLAP. It ranges from well-defined (rigid) dimensional models to flexible ones. With all these requirements as input, the DB2 Cube Views model has taken a layered approach to creating objects in the DB2 catalog. This approach allows the tools to derive maximum benefit from the cube model.

The DB2 Cube Views model is designed to handle star/snowflake schemas due to simple yet compelling advantages that these type of schemas possess:

- Industry standard:

  Such designs are widely implemented and easily understood for OLAP-type solutions. Understanding allows more useful navigation to be performed by users/tools accessing the database and allows meaningful data to be retrieved more easily.

- Performance:

  Star schema databases deliver high performance in data retrieval by minimizing the number of joins required (relative to a normalized relational model) and generally by simplifying access to the data. Performance is enhanced for queries that join many tables (instead of spanning many records) and a single table scan could span many records.

These have such significance that star schemas (or snowflake designs) are recommended for performance reasons when building cube models in DB2.

3.1.1 Star schemas

Star schema has become a common term used to connote a dimensional model. It typically has a number of small tables (known as dimensions or snowflakes) with descriptive data surrounding a centralized large table (known as the fact table) whose columns contain measures such as sales for a given product, for a given store, for a given time period.

What defines the dimension tables is that they have a parent primary key relationship to a child foreign key in the fact table. The star schema is a subset of the database schema.

This model is named star schema due to the dimension tables appearing as points of a star surrounding the central fact table, as shown in Figure 3-1.

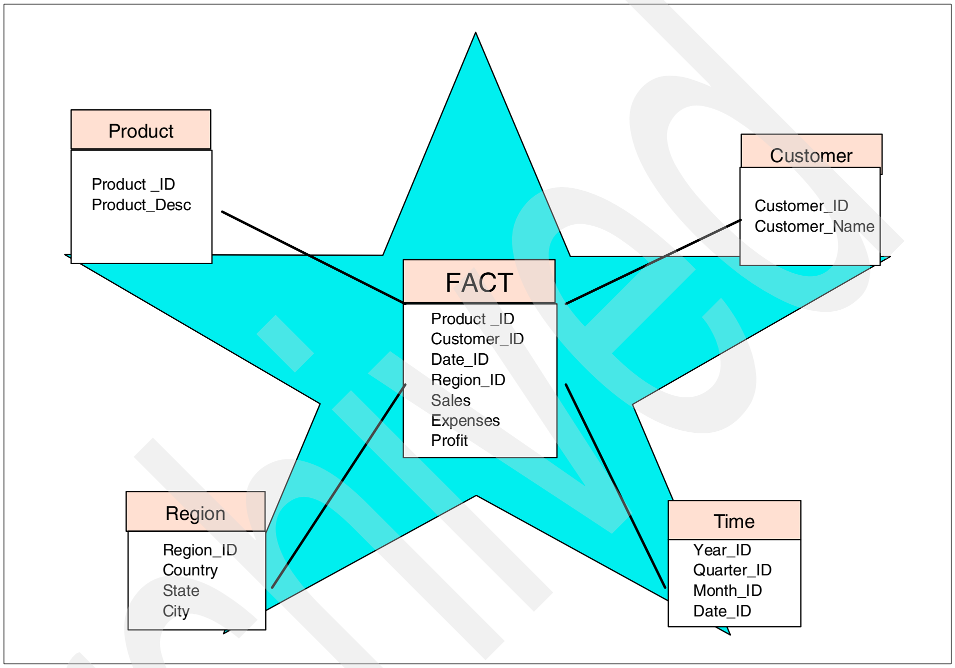


Figure 3-1 Star schema

We refer throughout to star schema generically because it is such a standard practice in the OLAP/warehousing world, and we make the assumption that DB2 Cube Views will be sitting on top of a star schema.

How you came to the decision to build a star schema, whether it is a star schema data warehouse or a star schema datamart drawn from a 3NF data warehouse, is another debate not addressed directly in this book.

With DB2 Cube Views the database designers can define logically all dimensions, measures, and hierarchies from the same transaction data in the form of cube models and then deploy as many cubes they feel necessary, maintaining consistency among applications.

3.1.2 Snowflakes

Further normalization of the dimension tables in a star schema results in the implementation of a snowflake design (see Figure 3-2) where dimensions are snowflaking and dimension tables are subsets of rows.

For example, the *Product* dimension table generates subsets of rows. First, all rows from the table (where level=*Family* in the star schema) are extracted and only those attributes that refer to that level (*Family_Intro_Date*, for example) and the keys of the hierarchy (*Family_Family_ID*) are included in the table.

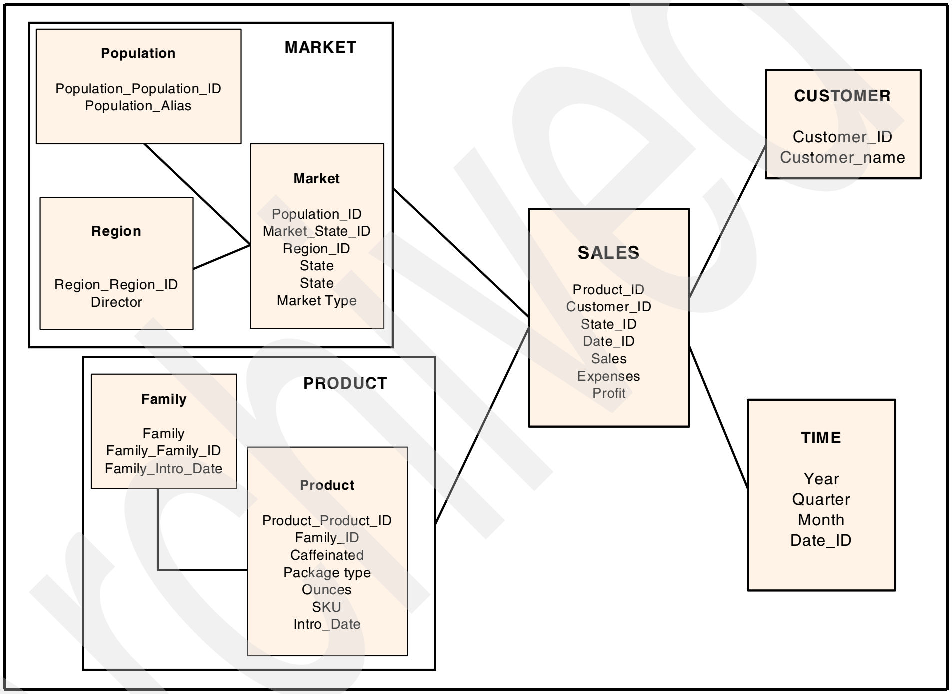


Figure 3-2 Snowflake schema

3.1.3 Star and snowflakes characteristics

Whatever the layout is, here are the most important characteristics for a star or snowflake design:

- A large fact table that can be in the order of millions of data rows. It contains the atomic or lowest level of detail, which may be a sales transaction, a phone call, a reservation, a customer service interaction – whatever represents the most granular fact of business operation which is meaningful for analysis.
- Small dimension tables containing a finite number of descriptions and detail information for codes stored in the fact table.
- Use of primary and foreign keys
- Measures in the fact table
- Fact table with multiple joins connecting to other tables

In this chapter, we will use star schema to also represent snowflake unless explicitly required to distinguish between them.

3.2 Cube model notion and terminology

A cube model stores metadata to represent a relational star schema, including information on tables, columns, joins, and OLAP objects, and the relationship between each of these objects.

It can be visualized as a collection of high-level objects that are obtained by compounding entities and grouped as dimensional entities such as facts, dimensions, hierarchies, attributes that relate directly to OLAP-type solutions.

To better understand the notion of a cube model in DB2, we will pursue a layered approach to this concept in the context of the following scenario.

Imagine that over time, we are tracking sales data of a retail company selling cosmetics that has stores spread over several states. Information is stored about each customer, line of products that the company sells, the stores, campaign details that the company adopts. These questions now arise when the company wants to decide on a new campaign:

- Which is the best geographic location that they start with, based on stores making consistent profits?
- Which time period is the best to start the campaign?
- Who is the target market for the new product?

The analyst needs to understand the complex table structures and their relationships, and without this understanding, data retrieved may well prove meaningless. Building a cube model makes this effort easier and faster than it would be without DB2 Cube Views.

For our business case study, we used a retail database with the tables *CONSUMER_SALES*, *CONSUMER*, *PRODUCT*, *STORE*, *DATE*, *CAMPAIGN*.

Note: Refer to Appendix E, "The case study: retail datamart" on page 685 for a complete description of the star schema for the retail database case study.

We start with an examination of the relational data (star or snowflake schema) design in DB2 (see Figure 3-3) and will present the different concepts (measures facts, dimensions, joins).

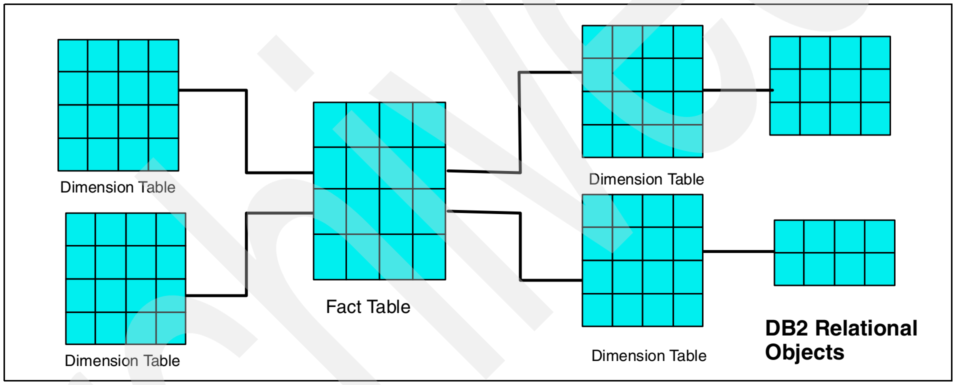


Figure 3-3 Relational data schema in DB2

3.2.1 Measures and facts

A measure object defines a measurement entity and populates each fact table row. They are usually numeric and additive and common examples of measured facts or measures are *Revenue*, *Cost*, and *Profit*. Consider the set of measures defined in the fact table *CONSUMER_SALES* in Figure 3-4.

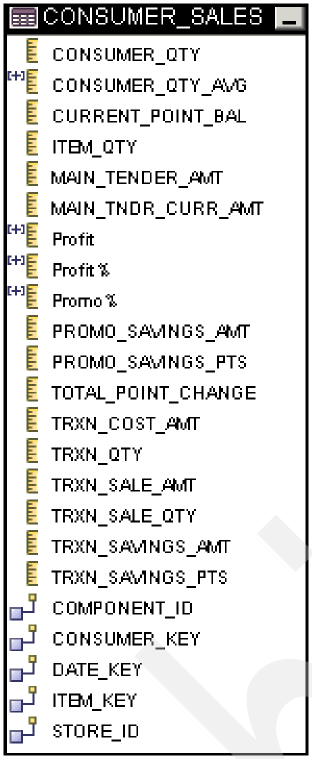


Figure 3-4 Facts object

Measures can be derived from the columns of the *CONSUMER_SALES* table: for example Transaction Sale Amount (*TRXN_SALE_AMT*), Transaction Cost Amount (*TRXN_COST_AMT*), *Profit* as in Example 3-1.

Example 3-1 Derived measure

```
Profit =@Column(STAR.CONSUMER_SALES.TRXN_SALE_AMT) -
@Column(STAR.CONSUMER_SALES.TRXN_COST_AMT
```

Some of the measures described in the facts object can be actual columns from the relational table or aggregated measures (measure that have been calculated using the aggregation functions as **`SUM`**, **`AVG`**). For example, using the *Profit* SQL expression as input for the SUM aggregation function, an aggregation on the Profit measure would be: **`SUM(Revenue - Cost)`**. For further information on the options available when creating advanced measures in DB2 Cube Views, please refer to 3.4, “Enhancing a cube model” on page 118.

Related measures are grouped together in the fact table to represent facts which are interesting when performing analytics on a specific subject area. The Fact table, at the center of the star schema, contains the grain of the business process and has the fundamental atomic level of data represented by the Fact table. These are typically individual transaction level data or snapshots taken on a daily or monthly basis.

In DB2 Cube Views, we define a metadata object called Fact based on one fact table from the relational star schema (see Figure 3-5).

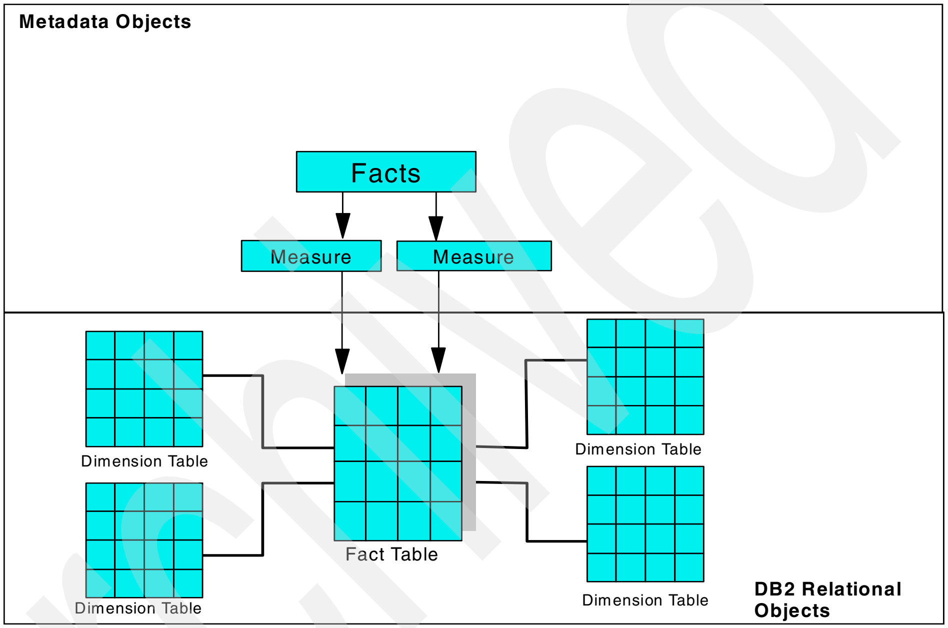


Figure 3-5 Facts and measures

3.2.2 Attributes

Simply stated, an attribute represents a database table column.

For example, attributes for the facts object are derived from the key columns: *DATE_KEY*, *CONSUMER_KEY*, *STORE_ID*, *ITEM_KEY*, *COMPONENT_ID*.

Attributes for a dimension object DATE, are CAL_YEAR_ID (Calendar Year Identifier), *CAL_MONTH_ID* (Calendar Month Identifier), to name a few (see Figure 3-6).

An attribute is defined by a SQL expression that can be a simple mapping to a table column, can involve multiple columns and other attributes, and can involve all functionality of the underlying database such as user-defined functions.

Note 2: When other attributes are used in the defining SQL expression, the other attributes cannot form attribute reference loops. For example, if Attribute A references Attribute B, then Attribute B cannot reference Attribute A.

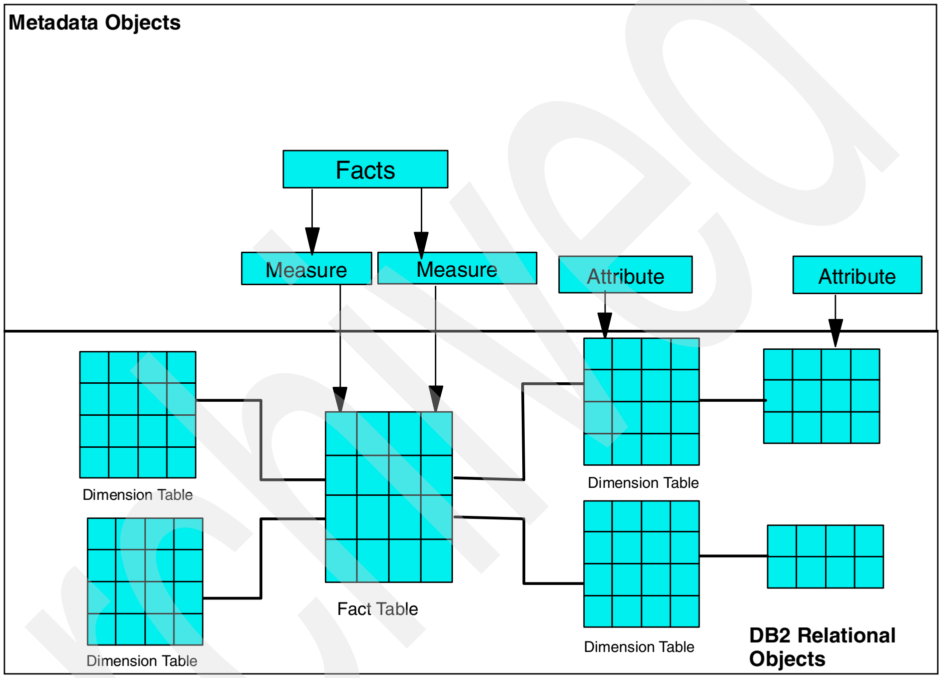


Figure 3-6 Attributes on dimension tables

3.2.3 Dimensions

The dimension object provides a way to categorize a set of related attributes that together describe one aspect of a measure (see Figure 3-7). A dimension is a collection of data of the same type.

Information for these objects is abstracted from the relational tables of the star schema constituting the dimensions.

Dimensions are used in cube models to organize the data in the facts object according to logical categories like Region, Product, or Time.

For example, in our business case study, *CONSUMER*, *PRODUCT*, *STORE*, *DATE* and *CAMPAIGN* are the dimensions.

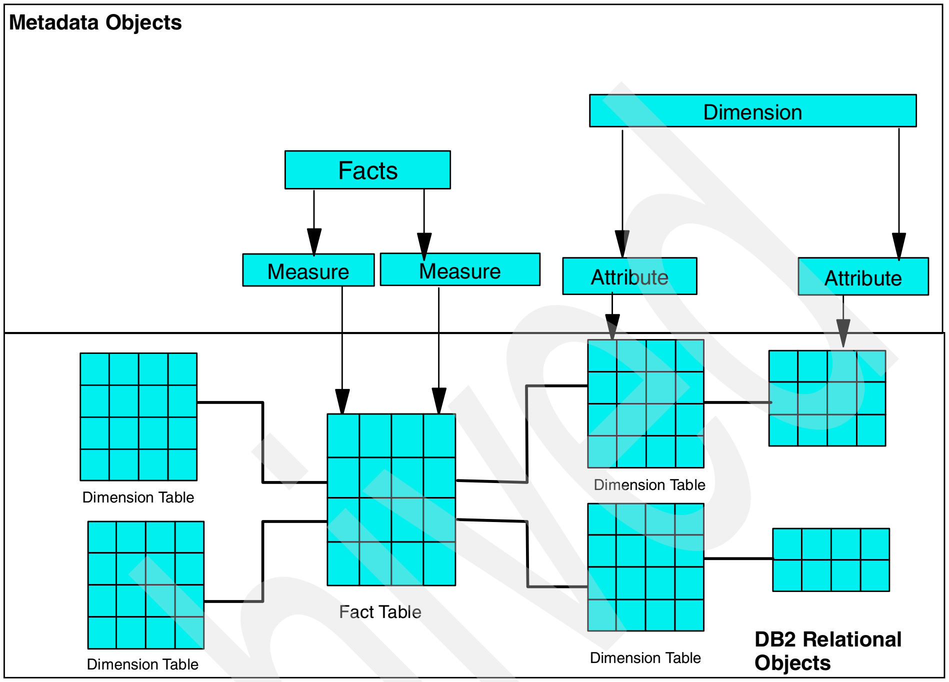


Figure 3-7 Dimension and attributes

Dimensions help make a meaningful interpretation of measures in the facts object. For example, a profit of 300 has no meaning by itself. When described in the context of a dimension, say *STORE* (which has information on the stores worldwide that sell the product, broken down in terms of country, state, city and so on) or Date (information related to *TIME* in terms of *Year*, *Quarter*, *Month* typically) then such a measure becomes meaningful. It is now easier to understand “A profit of 300 for Quarter-2 in Florida or Loss of 100 in Seattle year-to-date for the shampoo line of products”.

Dimension can have a type of regular or time as described in “Create the dimension objects” on page 100.

Dimension objects also reference hierarchies and attribute relationships.

3.2.4 Hierarchies

A hierarchy defines relationships among a set of one or more attributes within a given dimension of a cube model.

For example, Year-Quarter-Month is an example of a hierarchy naturally occurring within a *Time* dimension.

Defining these relationships provides a navigational and computational means of traversing a given dimension.

Figure 3-8 is a graphical representation of hierarchy for the *Time* dimension.

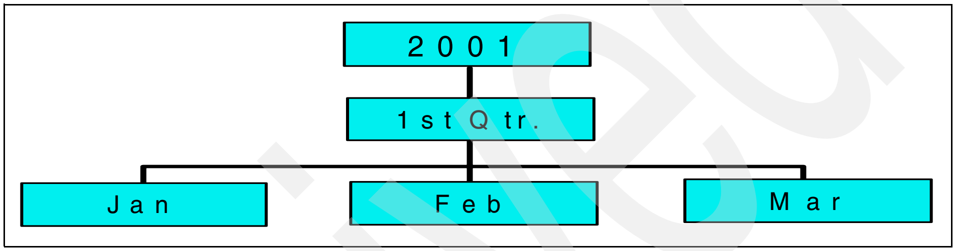


Figure 3-8 Hierarchy in a Time dimension

Hierarchies are defined by the different levels in the dimension and the parentage.

The type of hierarchy objects defined in DB2 Cube Views can be:

- Balanced:

  A hierarchy is a *balanced* hierarchy if children have one parent and levels have associated meaning or semantics and the parent of any member in a level is found in the level above. For example, see the Time dimension in Figure 3-9.

Year	Quarter	Month	Day
2003	Q1	Jan	1
2003	Q1	Jan	2
2003	Q1	Jan	3

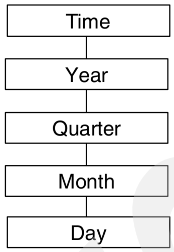


Figure 3-9 Balanced hierarchy

- Unbalanced:

  A hierarchy is *unbalanced* if children have one parent and levels do not have associated meaning or semantics, as in Figure 3-10.

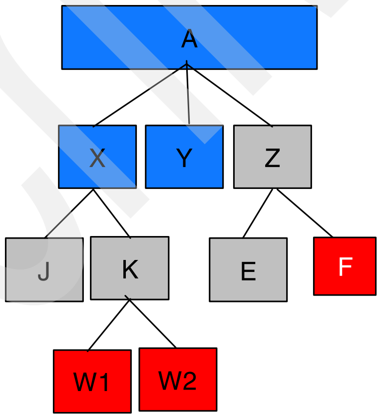


Figure 3-10 Unbalanced hierarchy

The semantics are in the relationships between levels rather than in the level itself, as in the example:

```
Product A 'is composed of' products X and Y and component Z
Component Z 'is composed of' Component E and part F
Product X is composed of Components J and K
Component K is composed of parts W1 and W2
```

- Ragged:

  In a *ragged* hierarchy, children have one parent and levels have associated meaning or semantics and the parent of any member may not be found in the level above. For example, see Figure 3-11.

Region	Country	State	City
Americas	USA	California	San Jose
Europe	Greece	-	Athens
-	Iceland	-	Reykjavik

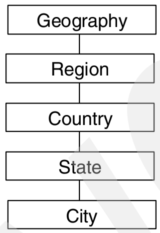


Figure 3-11 Ragged hierarchy

- Network:

  A hierarchy is a *network* hierarchy if children can have more than one parent, such as a family tree, for example.

The implementation in the DB2 tables defines the deployment mode: it can be standard or recursive:

- In a standard deployment, the attributes in the dimension table define each level in the hierarchy. All types of hierarchies are supported.
- In a recursive deployment, the levels in a dimension hierarchy is defined by the parent-child relationship:
  - One attribute defines parent.
  - One attribute defines children.

  Only the unbalanced hierarchy type is supported.

  Thus hierarchy can be recursively deployed (using the OLAP Center GUI) when the hierarchy has only two attributes.

For a more detailed description for type and deployment of hierarchy object, see the *IBM DB2 Cube Views Setup and User's Guide*, SC18-7298.

As an illustration, Table 3-1 shows a standard deployment of a balanced hierarchy in the Time dimension.

Table 3-1 Deployment of a balanced hierarchy

Year	Quarter	Month
2001	1st Quarter	Jan
2001	1st Quarter	Feb
2001	1st Quarter	Mar
2002	1st Quarter	Jan
2002	1st Quarter	Feb
2002	1st Quarter	Mar

Multiple hierarchies can be defined for a dimension of a cube model. Conversely, a dimension should have at least one hierarchy defined.

3.2.5 Attribute relationships

An attribute relationship describes relationships of attributes in general. The relationships are described by a left and a right attribute, a type, a cardinality, and whether or not they determine a functional dependency.

Suppose that we are defining a relationship between *ProductCode* (left attribute) and *ProductName* (right attribute). A *ProductName* describes a *ProductCode*. This relationship is of *descriptive* type.

The *associated* type specifies that the right attribute is associated with the left attribute, but is not a descriptor of the left attribute. For example, a *CityPopulation* right attribute is associated with, but not a descriptor of *CityID*.

Cardinality defines type of relationship between the left and right attributes. For example, in a 1:1 cardinality, there is at most one left attribute instance for each right attribute instance, and at most one right attribute instance for each left attribute instance. Other possible values for cardinality are 1:Many, Many:1, and Many:Many.

A functional dependency defines a functional relationship between two attributes. For example, a functional dependency can be defined between attributes like *City* and *Mayor*, or *Product* and *Color*. The functional dependency tells that every *City* value determines a *Mayor* value or that every *Product* value determines a *Color* value.

When functional dependency is defined between attributes, it means that the cardinality of the of the relationship is guaranteed by the designer and this becomes of great use in performing query optimizations.

Attribute relationships are mainly used within the context of a hierarchy. Attributes that are directly related to attributes in a hierarchy can become part of the query. For example, you can include *CityPopulation* in a query that retrieves *CityID*.

Figure 3-12 provides a graphical representation of attributes, attribute relationships, and hierarchies within the context of a dimension.

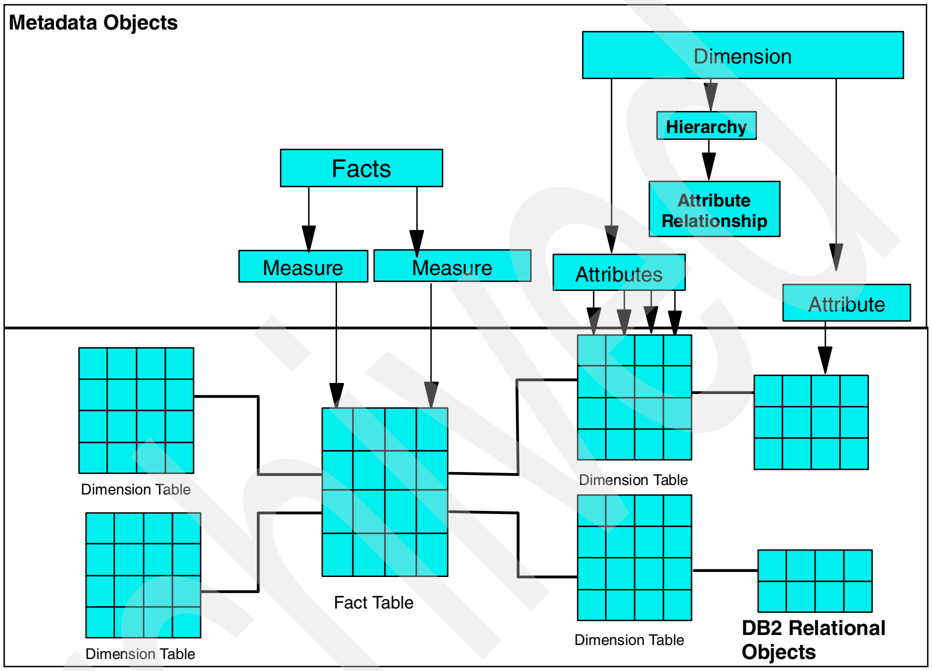


Figure 3-12 Dimension, attributes, hierarchies, and attribute-relationships

3.2.6 Joins

DB2 Cube Views stores metadata objects called joins, representing joins in the star schema (see Figure 3-13).

In the case of a star schema, the join objects in the cube model are those that exist between the facts object and each dimension object.

In the case of a snowflake design, joins can be defined between dimension tables (when there is more than one relational table from which the dimension object is derived) and between the fact and dimension tables.

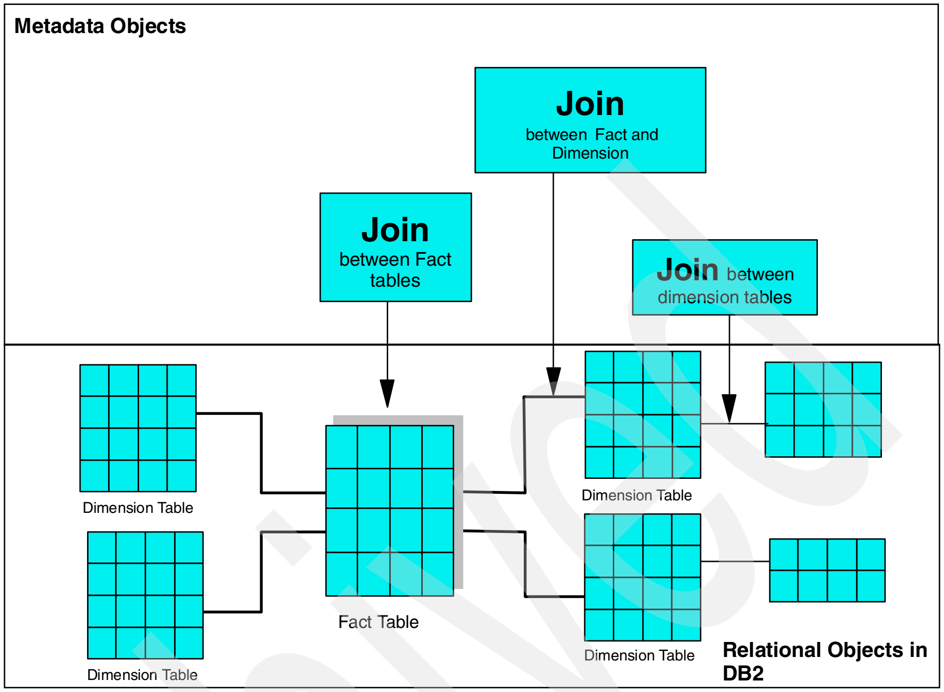


Figure 3-13 Joins

A join object joins two relational tables together. The simplest form of a join maps a column in the first table to a column in the second table, along with an operator to indicate how the columns should be compared.

If any type of join can be selected for modeling purposes, inner joins based on the optimization rules as described in *IBM DB2 Cube Views Setup and User's Guide,* SC18-7298 are required for the Optimization Advisor.

The cube model now has the objects that are depicted in Figure 3-14.

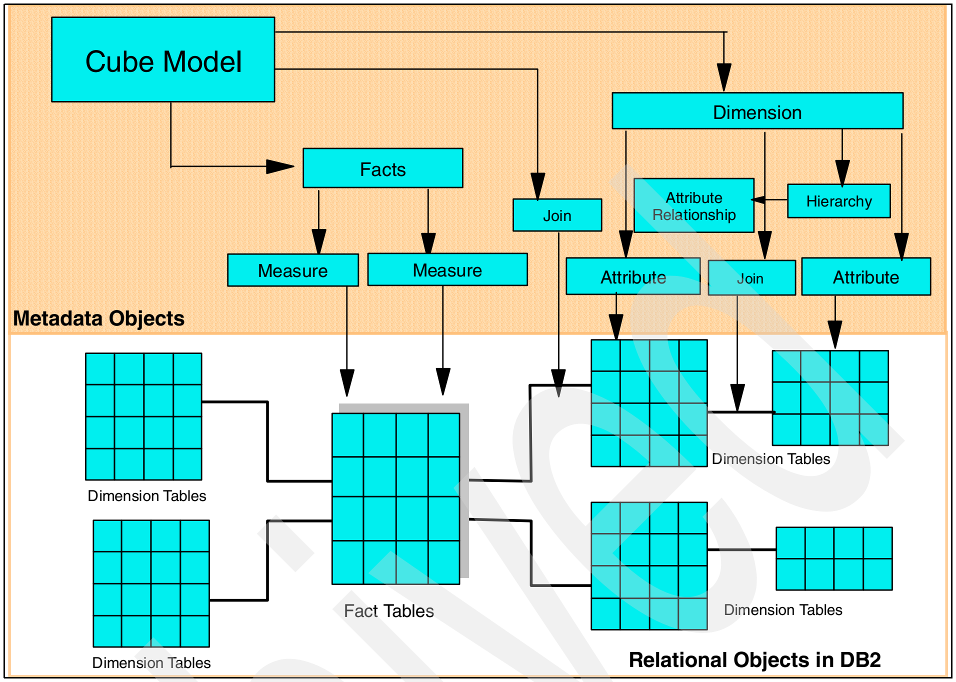


Figure 3-14 Complete layered architecture of a cube model

3.2.7 In a nutshell: cube model and cubes

A cube model is a complete description of a complete star or snowflake schema that model designers can make as rich and as complete as possible.

A cube is derived from a cube model.

Cube model

A cube model in DB2 Cube Views is a logical representation of the underlying physical tables in DB2 and is itself a metadata object in DB2 Cube Views. A cube model and all its related metadata objects are stored in the DB2 catalog within the DB2 database prepared for DB2 Cube Views.

From the perspective of a BI tool, ultimately importing the DB2 Cube Views metadata model, the cube model is the virtual multidimensional environment or universe within which users navigate through their graphical interface. Tool users are unaware of the underlying mapping to DB2 relational objects, and tend not to think of their environment as a logical abstraction of a DB2 star schema but rather as a pure conceptual representation of the business.

A cube model is a grouping of relevant dimension objects around a central facts object. You can also think of a cube model object as described by its properties — Facts, Dimensions, and Joins. The facts object is a grouping of relevant measures. The dimension objects contain a set of related attributes to describe one aspect of the measure. The dimensions each contain one or more hierarchies. Hierarchies reference attribute-relationships. Joins between the facts object and the dimension objects are also stored in the cube model. Cube objects are scoped-down version of the cube model and the corresponding objects.

Cube, cube fact, cube dimensions, cube hierarchies

The notion of a cube comes from scaling down on a cube model.

A cube is a very precise definition of an OLAP cube that can be delivered using a single SQL statement. The cube facts and list of cube dimensions are subsets of those in the referenced cube model. Cubes are appropriate for tools and applications that do not use multiple hierarchies because cube dimensions only allow one cube hierarchy per cube dimension. You can use the Cube wizard in OLAP Center to create a cube. You must have a complete cube model to create an associated cube.

One or more cubes can be derived from a cube model. A cube has a cube facts as the central object and cube dimensions. The cube facts (measures) and cube dimensions are again subsets of the corresponding objects referenced in the cube model. Cube hierarchies are scoped down to the *cube* and each can be a subset of the parent hierarchy that it references. Each cube dimension can have only one hierarchy defined. This structural difference between cube model and a cube allows a slice of data (the cube) to be retrieved by a single SQL query.

Using cubes is appropriate in the case of tools and applications that do not require multiple hierarchies. Cube metadata can also be used by the Optimization Advisor when optimizing query performance or a specific business subject (refer to Chapter 4, “Using the cube model for summary tables optimization” on page 125).

See Figure 3-15 for the complete picture of the layered architecture.

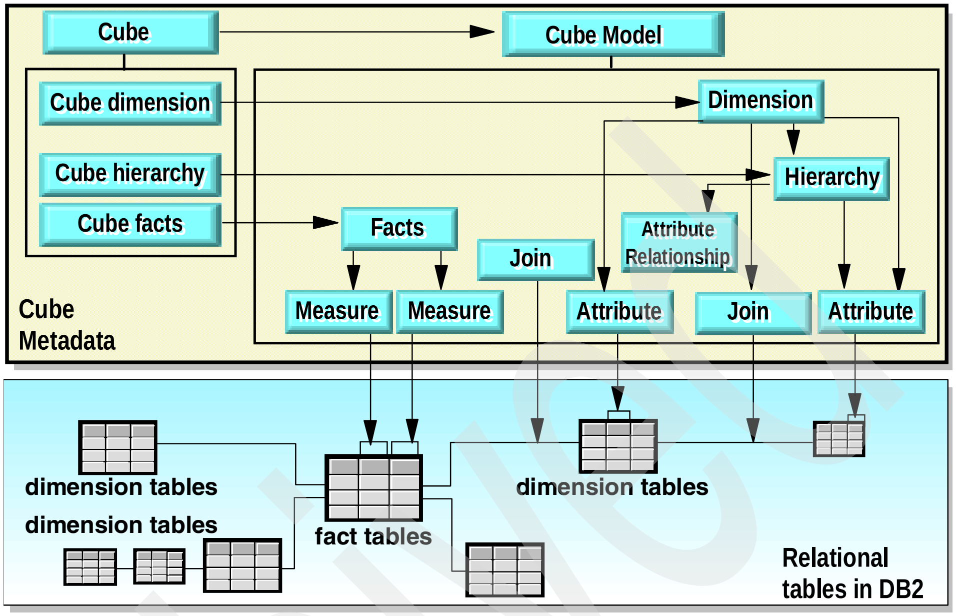


Figure 3-15 Cube model and cubes

A basic complete cube model based on a star schema should have a facts object joined to two or more dimension objects. At least one hierarchy should be defined for *each* dimension.

3.3 Building cube models using the OLAP Center

Using the OLAP Center as your central point for metadata maintenance with its simple interface and wizards helps to minimize the administrative tasks, especially when you have different technologies to deploy.

OLAP Center is a Graphical User Interface (GUI) that allows users of warehousing and business intelligence tools to view, create, modify, import, export, and optimize cube models, cubes, and other OLAP-related metadata objects in the DB2 catalog.

Subsequent to installation of DB2 Cube Views, it can be launched from any of the DB2 GUI tools. For example, it can be launched from the Control Center for DB2 UDB V8.1 (see Figure 3-16).

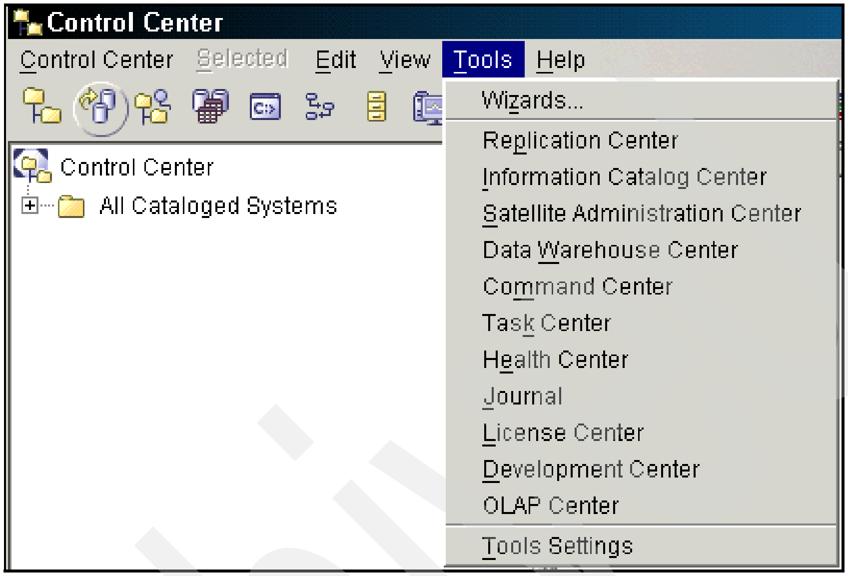


Figure 3-16 Launching OLAP Center

The OLAP Center has the same look and feel of other DB2 GUI tools. It is a Java based program using available DB2 common classes.

On the Windows platform, the OLAP Center can also be started from **Start -> Programs -> IBM DB2 -> Business Intelligence Tools -> OLAP Center**.

Figure 3-17 provides a diagrammatic overview of the OLAP Center architecture.

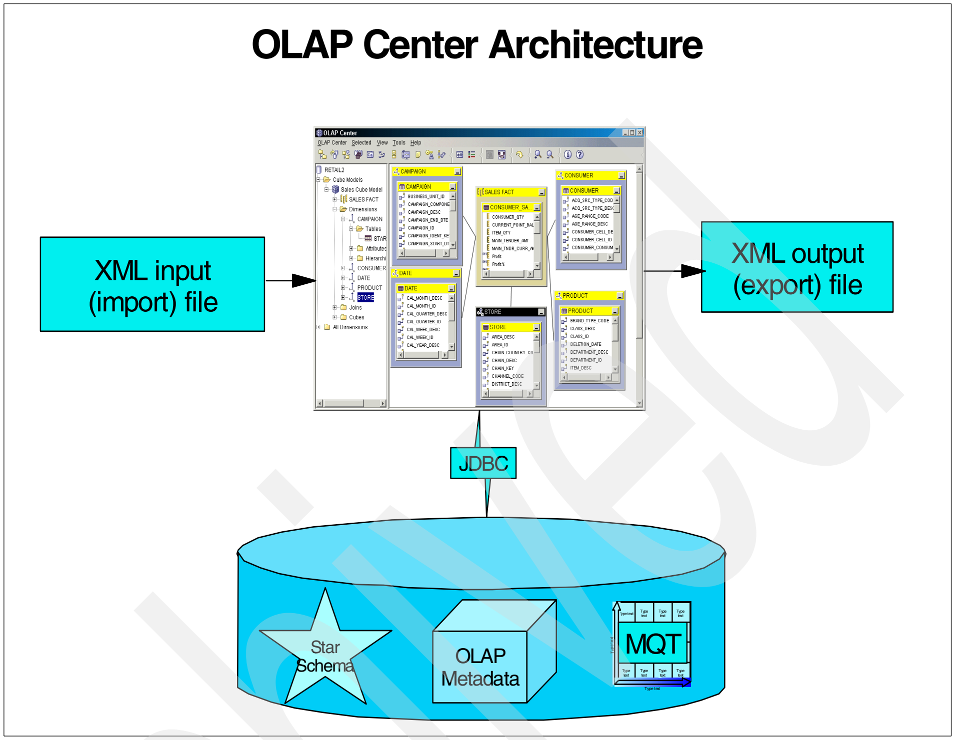


Figure 3-17   OLAP Center architecture

These are the main tasks that are performed from the OLAP Center:

- Import of OLAP partner metadata in the form of eXtensible Markup Language (XML) files into DB2. This is done using the Import wizard available through the OLAP Center menu. The XML files can be imported from partners' tools through their bridge. See Chapter 5, “Metadata bridges overview” on page 221 on OLAP partner bridges.
- Export of OLAP metadata in DB2 that can also be exported as XML files and made available to other OLAP solutions. The XML files can be exported and passed through a bridge to partners' tools. See Chapter 5, “Metadata bridges overview” on page 221 on OLAP partner bridges.
- Creation and manipulation of metadata objects in DB2. The GUI helps view metadata objects using detailed and graphic views and manipulation of metadata objects through **Object Properties**. The Quick Start wizard and Object Creation wizards help in creating the OLAP metadata objects in DB2.

- Cube model optimization. The OLAP Center GUI provides the performance for the Optimization Advisor wizard to help improve performance of queries executed against a cube model based on query type (drill down, extract, report, or drill though). See Chapter 4, “Using the cube model for summary tables optimization” on page 125 for a detailed discussion of this topic.

Now, let’s get started on creating and building metadata objects in DB2 Cube Views. We will focus on the different methods to build a cube model using the OLAP Center, as depicted in Figure 3-18.

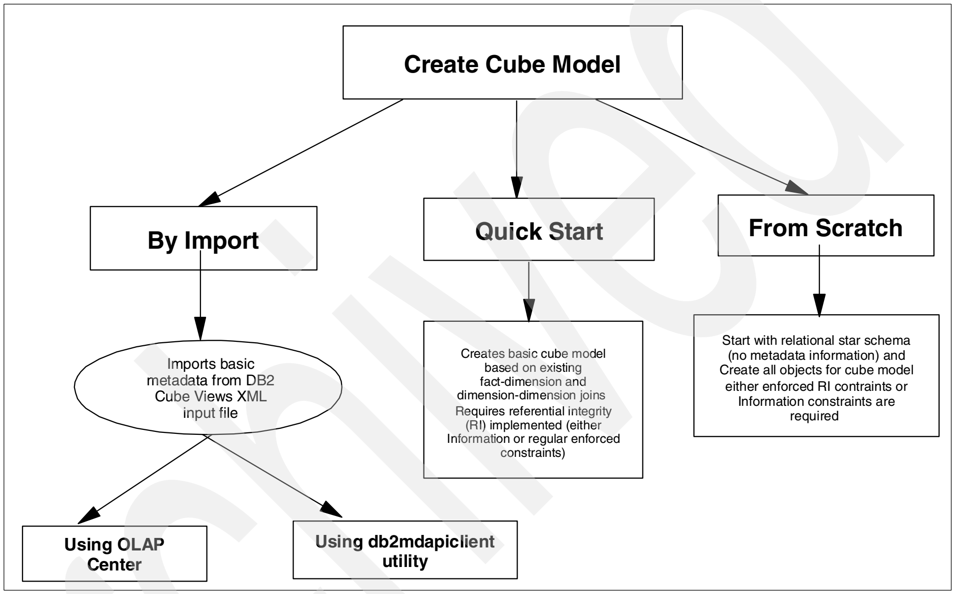


Figure 3-18 Cube model building methods

The different tasks involved in building a cube model (based on a star schema) are presented as a broad overview in Table 3-2.

Table 3-2 Overview of cube model building tasks

Phase	Steps/Objectives	Section Reference
Planning for building a cube model	Understand the star schema in place, analytics and usage requirements	Section 3.3.1, “Planning for building a cube model” on page 85

Phase	Steps/Objectives	Section Reference
Preparing the relational database	Register DB2 Cube Views stored procedure with the database Create metadata catalog tables	Section 3.3.2, “Preparing the DB2 relational database for DB2 Cube Views” on page 86
Building a cube model	With Quick Start wizard	Section 3.3.4, “Building a cube model with Quick Start wizard” on page 91
	By Import	Section 3.3.3, “Building the cube model by import” on page 87
	From scratch	Section 3.3.5, “Creating a basic complete cube model from scratch” on page 92

3.3.1 Planning for building a cube model

Before creating a cube model with DB2 Cube Views, a complete dimensional modeling exercise should be carried out first or should exist.

When done and before starting using DB2 Cube Views:

- Identify the methodology:
  - For example, consider if the cube model metadata is going to be imported from partner tools using the appropriate bridges, or built from scratch.
- Study the data:
  - Understand the data of the relational star schema.
  - Understand the type of analytics to be performed based of business requirements and subject area; identify fact table(s) and dimension tables.
  - Make sure that the star schema database is well formed with primary keys and foreign keys pairs and Referential Integrity in place, either enforced or informational.
  - Identify types of measures, and if there are measures that the designer cannot directly retrieve from the data, then create derived measures.
  - Identify hierarchies for each dimension, since a dimension should have at least one hierarchy.

- Study the usage:
  – Types of accesses to the cube model, like drill through, extraction, ad-hoc reports, and drill down.
  – Type of dimensional model: MOLAP, ROLAP, or HOLAP. This may be used to determine if you require to have a single cube model with one or several cubes.
  – Identify business names for the columns from the relational tables for better understanding of data for the users.

The designer can then proceed to build the cube model.

3.3.2 Preparing the DB2 relational database for DB2 Cube Views

Setting up the database to be used with DB2 Cube Views includes:

- Registering the DB2 Cube Views stored procedure with the database
- Creating metadata catalog tables for DB2 Cube Views

This preparation is done manually using DB2 commands or from the OLAP Center GUI

Using DB2 command

1. Open a DB2 command window and connect to the database:

   ```
   db2 connect to dbname user username using password
   ```

2. Change to the SQLLIB\misc directory and enter the following command:

   ```
   db2 -tvf db2mdapi.sql
   ```

> **Attention:** Do **NOT** modify the db2mdapi.sql script. Results will be unpredictable if you try to do so.

Using OLAP Center

This option can be pursued, if configuration using the DB2 command has not been done.

On successful connection to a database, the user is prompted to specify whether he wants to configure the database or not:

1. Connect to the relational database as shown in Figure 3-19.

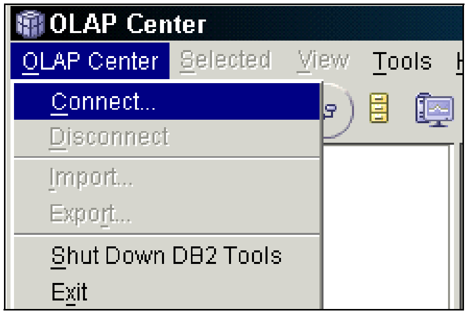


Figure 3-19 Connecting to database

2. You will see a message pop-up window, on connecting to the unconfigured database, reporting that the database has not been configured for DB2 Cube Views.

Click **Yes** to allow preparation of the database to take place.

3.3.3 Building the cube model by import

This section describes the process of building a basic (but not necessarily complete) cube model in DB2 Cube Views based on metadata that may already exist in your Business Intelligence environment.

Typically, in a Business Intelligence environment, design tools have probably been used to design the multidimensional schema, and ETL tools have probably been used to populate the schema.

The import scenario is used in such BI environments that already have OLAP metadata (dimensional models) captured by various tools which can be reused and/or consolidated by DB2 Cube Views. This can help gain a head start in the development of cube models by reducing development effort. This can also make OLAP definitions across the enterprise consistent.

The following subsections discuss the principle and actual implementation of building a cube model by importing dimensional model metadata into DB2 Cube Views.

The principle

Building a cube model by import is based on the principle that a dimensional model metadata is available from sources outside of DB2 Cube Views. This metadata information has been converted into a form that is compatible with DB2 Cube Views. Metadata to be imported into DB2 Cube Views is always in the form of an XML file. The import XML file is the output received from passing source metadata through an appropriate bridge. The type of bridge used depends on where the metadata is pulled from. The import in to DB2 Cube Views can be done using the OLAP Center or using the db2mdapiclient utility that is shipped with the product.

More detailed implementation of various bridges and how to convert available OLAP definitions and metadata into DB2 Cube Views import file is discussed in detail in Part 3, “Access dimensional data in DB2” on page 219.

For a practical and successful implementation of this principle, it is imperative that the source and repository (or directory) of metadata, to which these other OLAP tools connect, is also in DB2 UDB, so that the meaning of the metadata (in terms of the relational table names and columns) that is imported into DB2 Cube Views is still meaningful.

Using the OLAP Center

We start with a DB2 Cube Views import XML file (one that has already been transported through an appropriate bridge depending on the source of the metadata).

For example, if we are importing metadata coming from a partner tool bridge that uses XML files, then the partner tool metadata is first exported as an XML file from the partner tool and its bridge. The output from the bridge is then used as the import XML for the following steps which guide you through the steps to import metadata into DB2 Cube Views using the OLAP Center:

1. Launch OLAP Center.
2. Connect to the relational database in DB2 from OLAP Center (see Figure 3-19 on page 87) in which you wish to store the cube model metadata. This should be a relational database that has already been prepared for DB2 Cube Views (if not, preparation will take place on connecting to the database for the first time, as explained earlier).
3. Once connected to the database, choose from **OLAP Center** --> **Import** (see Figure 3-20).

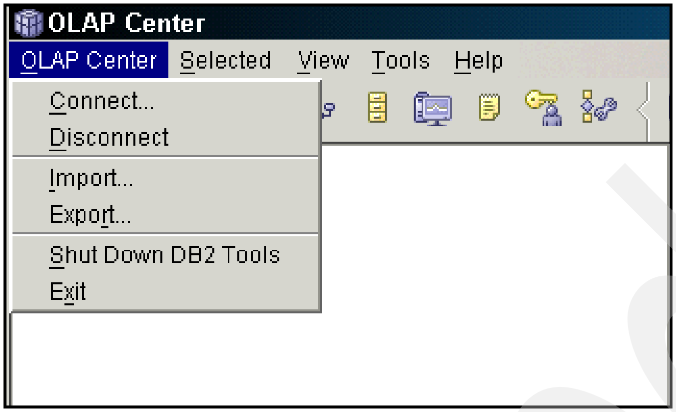


Figure 3-20 OLAP Center import

4. Choosing to import metadata into DB2 Cube Views launches an Import wizard (see Figure 3-21). Select the file to be imported into DB2 Cube Views either by directly typing the name and location of the file or by using the browser button.

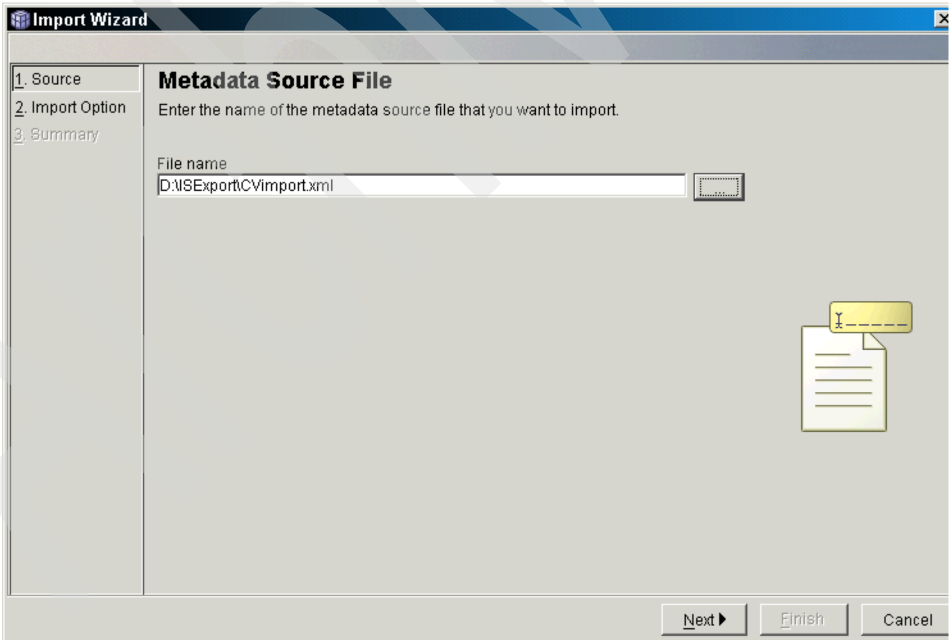


Figure 3-21 Choose Metadata Source File

5. Click **Next** to choose the import options (see Figure 3-22).

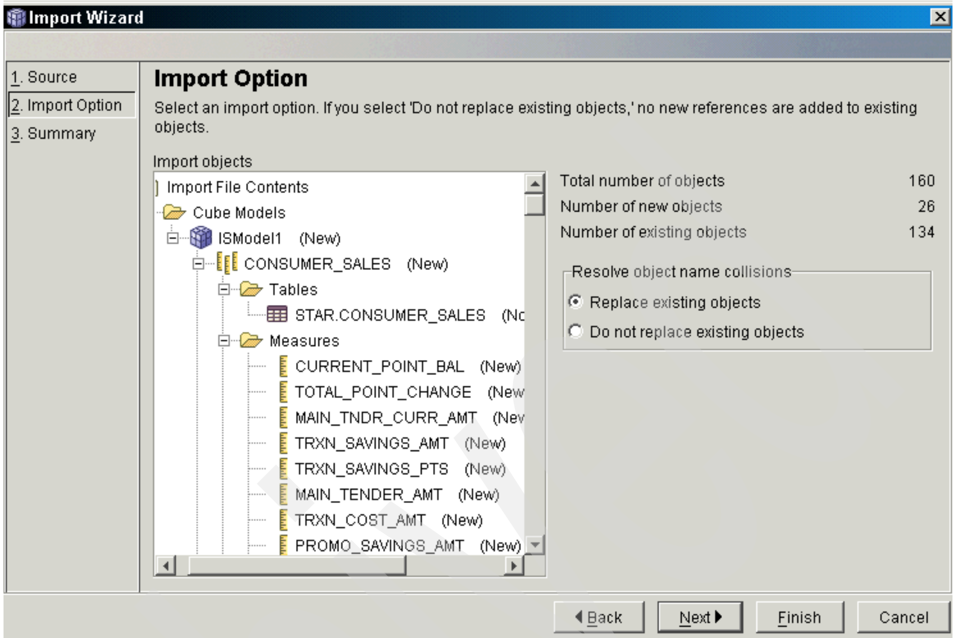


Figure 3-22 Import options

At this stage, the import XML file is read, and information is displayed about the objects that it contains (cube model, facts, dimensions, cube and so on). If these objects that are being imported are brand new definitions to be added to DB2, then they have a (**New**) tag associated with the name. If an object with that name already exists in DB2, then an (**Existing**) tag is displayed next to the object. Apart from this graphical description, the window also displays a textual description of the number of new objects, number of existing objects, and total number of objects being imported.

Here you have the option to replace existing objects or create new objects in to the DB2 catalog.

6. Click **Next** to see the summary of options that you have chosen and click **Finish** to actually import the metadata.
7. On successful import, the detailed view of OLAP objects in OLAP Center shows the cube model and cubes (if any) imported (see Figure 3-23).

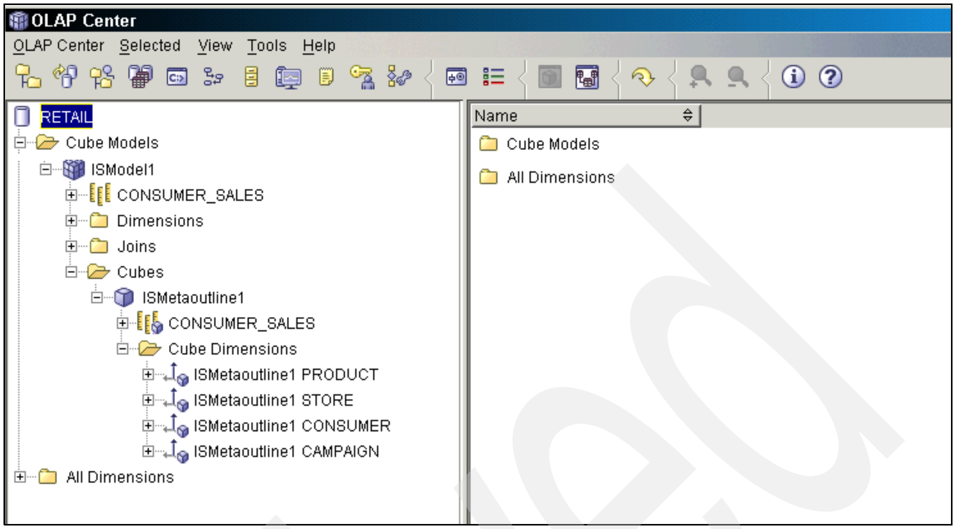


Figure 3-23 Imported cube model and cube

Using the db2mdapiclient utility

The db2mdapiclient utility is provided as sample source code for coding an application for DB2 Cube Views. This utility is a thin wrapper to the DB2 Cube Views stored procedure interface. The utility is provided as sample source code to show how to code an application against the API. The source code is located in \SQLLIB\samples\olap\client\db2mdapiclient.cpp.

For importing, the db2mdapiclient utility typically uses an XML file that is produced by a DB2 Cube Views bridge or that was exported from the OLAP Center.

For example, to import DB2 Cube Views metadata into the relational database in DB2, say, MDSAMPLE, change to the ..\SQLLIB\samples\olap\xml\input directory and enter the following command:

```
db2mdapiclient -d MDSAMPLE -u db2admin -p mypasswrd -i create.xml -o
myresponse.xml -m MDSampleMetadata.xml
```

3.3.4 Building a cube model with Quick Start wizard

The option in OLAP Center **Selected > Create Cube Model - Quick Start** launches a wizard that helps you create a cube model based on your relational tables.

However, Referential Integrity (RI) needs to be implemented for the fact and dimension tables before using this option. Once the facts object and measures have been specified, the wizard completes the cube model by creating the dimensions, attributes, and joins using the RI constraints. After creating the cube model using the wizard, the properties of the metadata objects can be modified later.

When creating empty cube models, Quick Wizard should always be used if possible, as it includes additional features as join auto-detection that does not exist when creating the cube model manually.

To start using the Quick Start wizard, right-click **Cube Models** as shown in Figure 3-25 and choose **Create Cube Model - Quick Start**.

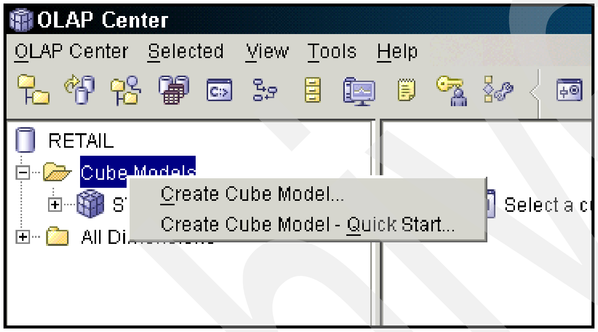


Figure 3-24 Create Cube Model wizard

The Quick Start wizard allows user to specify the fact object and consequently its measures. Once the fact objects and measures have been specified, the wizard creates a basic cube model with a fact object and other dimension objects.

Basic objects (dimensions, attributes and joins) are created using the RI constraints and the primary key/ foreign key pairs information. Once a basic cube model has been created, the properties of the metadata objects that have been created can be modified at a later time. Other metadata objects like hierarchies and cubes, which do not get created while using the Quick Start wizard, need to be added manually, from scratch.

3.3.5 Creating a basic complete cube model from scratch

Creating a cube model from scratch with OLAP Center means that you start with the relational database in DB2 containing the star schema (and prepared for DB2 Cube Views). The metadata objects (fact, dimensions, and related objects) are then defined to form a complete cube model.

Table 3-3 describes the steps for creating a basic cube model with fact and dimension objects.

Table 3-3 Creating a cube model from scratch

Step	Section Reference
Create an empty cube model	"Create an empty cube model" on page 93
Create the facts object	"Create the facts object" on page 94
Create the dimension objects	"Create the dimension objects" on page 100
Create the hierarchies	"Create the hierarchies" on page 107
Create the cube	"Create the cube" on page 114

Create an empty cube model

Creating an empty cube model means creating a cube model without any of the objects in it. In this step, we only provide a name for the cube model. The fact and dimension objects are created/added after completion of this step.

To create an empty cube model, right-click the cube models object and select **Create Cube Model** (see Figure 3-25)

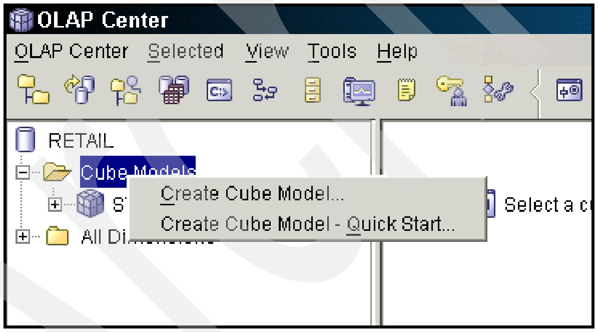


Figure 3-25 Create Cube Model wizard

Type in the name of the model and select **Finish** (see Figure 3-26).

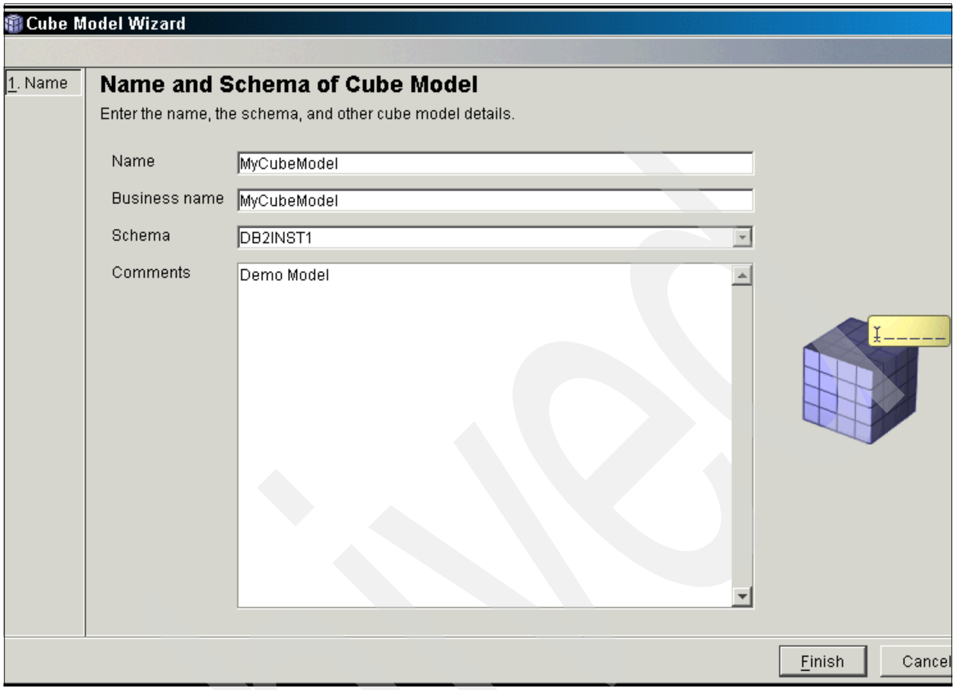


Figure 3-26 Provide cube model name

Create the facts object

The process of creating the facts objects also includes the following:

- Providing a name for the facts object
- Measures
- Aggregations
- Attributes

Right-click the model name and select the **Create Facts** option to launch the wizard (See Figure 3-27).

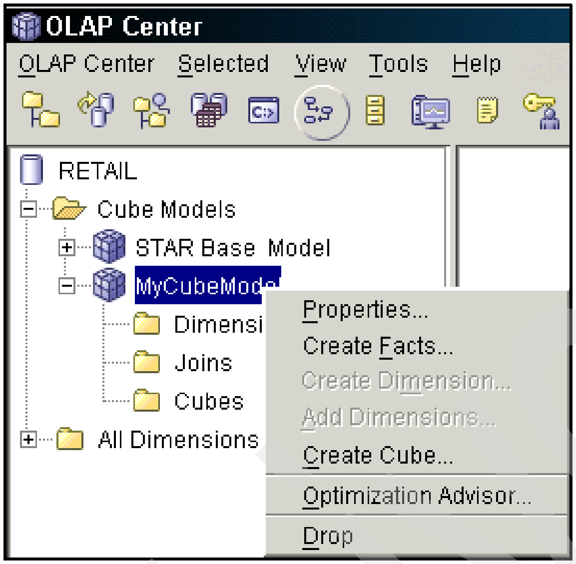


Figure 3-27 Create the facts object

Provide the name for the facts object and select the schema (see Figure 3-28).

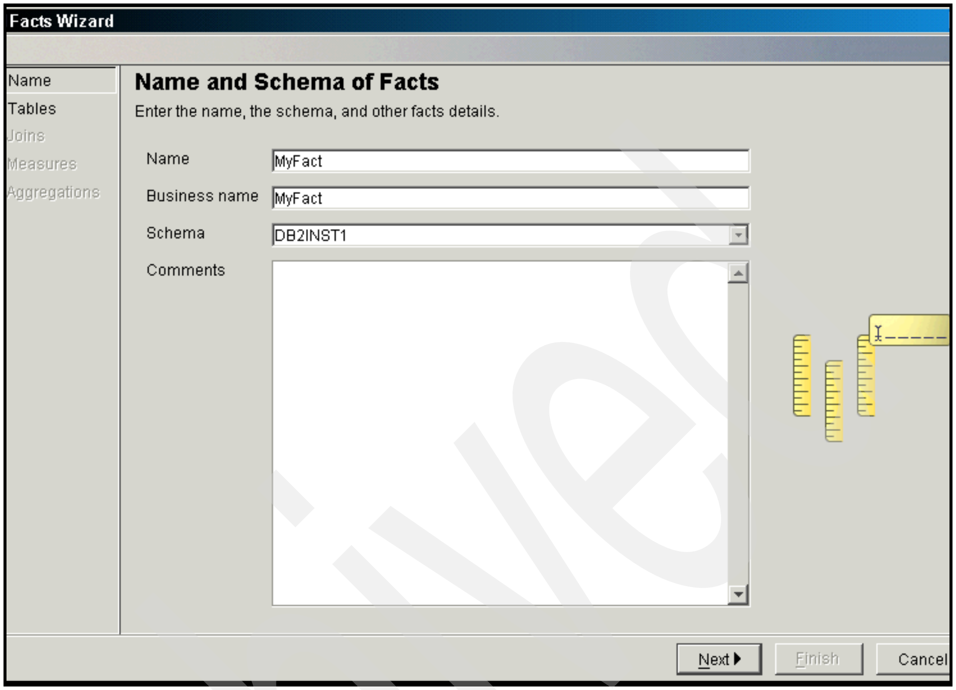


Figure 3-28 Facts object's name

Click **Next** and select the table that is required for the facts object (see Figure 3-29).

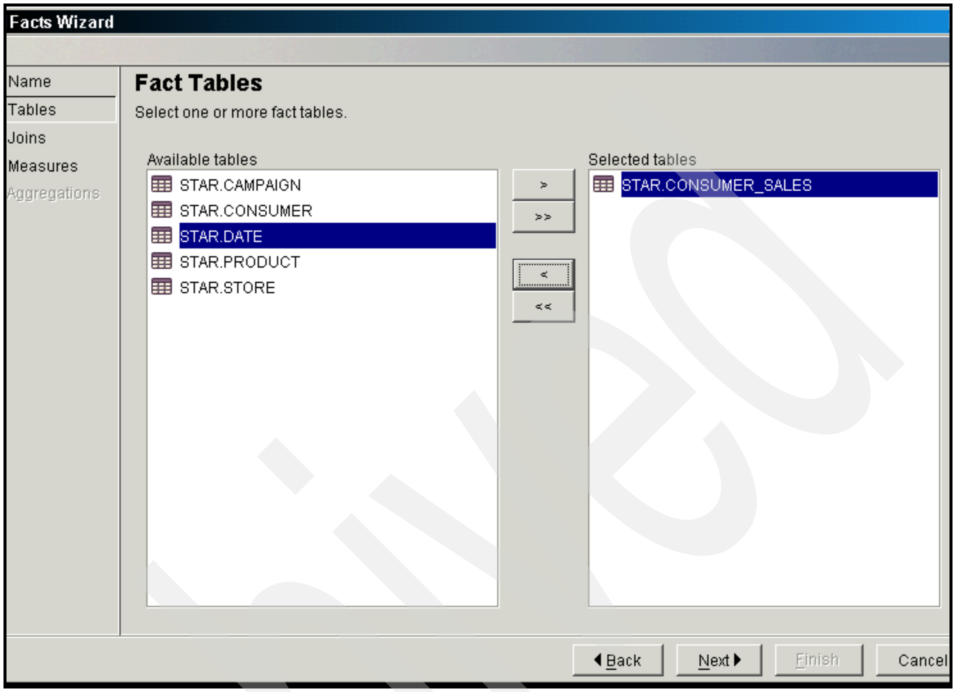


Figure 3-29 Select the Facts table

Note: If more than one table is needed to build the facts object, then you will be prompted to specify the join. In our illustrative star schema, there is only one table for the facts object.

Select **Next** to select the Measures (see Figure 3-30).

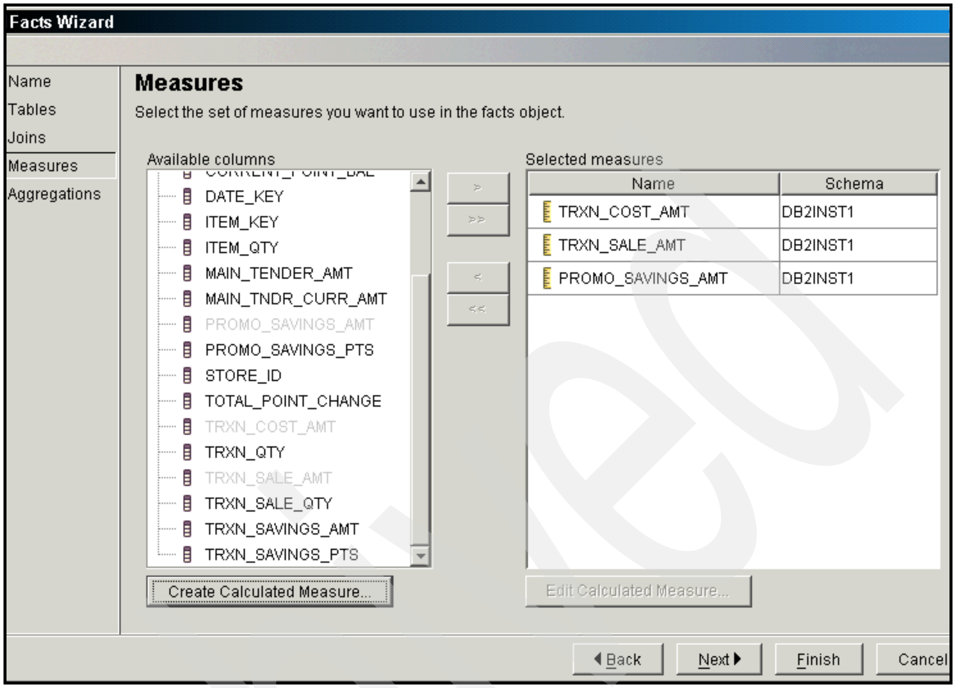


Figure 3-30 Select Measures

Available measure columns from the facts table are listed on the left hand side panel from which the user is allowed to select measures. Measures are selected with a mouse left-click action and then clicking **>** to move the measure to the panel on the right hand side. Figure 3-30 shows *TRXN_COST_AMT, TRXN_SALE_AMT* and *PROMO_SAVINGS_AMT* as the selected measures.

Additionally, you can create calculated measures that need to be calculated based on existing measures from the fact table. For example, *Profit* is calculated as *sale - cost.*

Click **Create Calculated Measure** (see) to launch the SQL expression builder as shown in Figure 3-31 and create a calculated measure, for example:

```
Profit =@Column(STAR.CONSUMER_SALES.TRXN_SALE_AMT) -
@Column(STAR.CONSUMER_SALES.TRXN_COST_AMT)
```

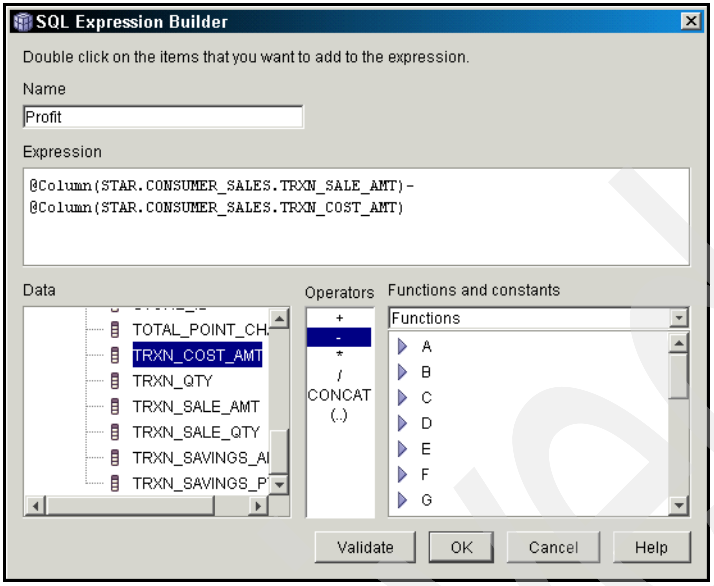


Figure 3-31 Create calculated measure

Type in the name of the calculated measure (example *Profit*) in the **Name** field. The actual expression is built by selecting the measures used in the calculation with the mouse (from the list in **Data**) and by choosing the appropriate operator (for example, the operator is '-' in the case of calculating *Profit*).

Thus, to calculate profit, select *TRXN_SALE_AMT* first, select the operator (-) and then select the measure *TRXN_COST_AMT*.

Click **Validate** to check if the expression built is valid and click **OK** to create the calculated measure and return to the **Facts Wizard**.

Click **Next** to see the list of measures (which includes the calculated measures that you created). You can change the type of aggregation to SUM or any of the types listed in the drop down box according to the requirement. This action can also be performed after creating the Facts object from its context menu option **Properties...**

> **Tip:** You can create your own aggregation scripts at a later time when you have built at least one dimension for the cube model. Typically, the basic cube model is built and then further enhancements are made depending on business requirements. Building an aggregation script is done by choosing **Edit aggregations...** from context menu option on selecting a measure.

Click **Finish** to complete creation of the Facts object.

Create the dimension objects

Creating dimension objects includes the following:

- Provide a name for the dimension
- Select relational table(s) to represent the dimension
- Select attributes; create calculated attributes, if applicable
- Define dimension type (Time or Regular)
- Select/Create Fact-Dimension join

Dimension objects are created from the OLAP Center GUI by selecting **Create Dimension...** from the context menu option on Dimensions (see Figure 3-32)

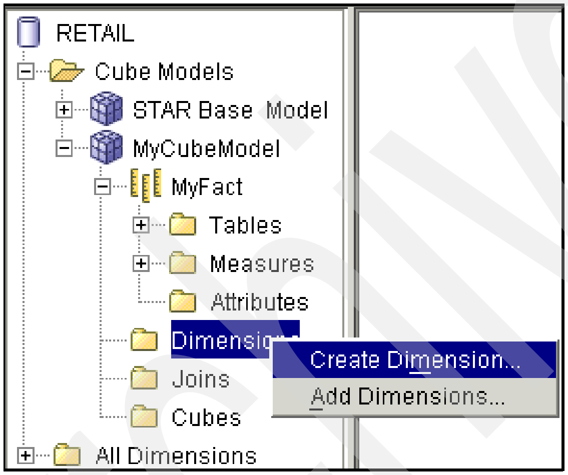


Figure 3-32 Create Dimension

> **Note:** Alternatively, dimensions can also be created from the context menu option on “All Dimensions” from the OLAP Center. Pursuing this option does not require the user to specify information about the join with the facts objects. Dimensions once created in this manner can be added to the cube model at a later time. Adding such dimensions to the cube model, will then be done from the context menu option **Add Dimension** on Dimensions in the cube model. The join to the facts objects needs to be specified only at the time of adding the dimension to the cube model.

Note: The context menu option on a dimension named **Remove** removes the dimension information from only the cube model. However, the dimension object still remains available in “All Dimensions”. The “Drop” option removes the dimension information, including hierarchies created, from both the cube model and “All Dimensions”.

Specify the name of the Dimension object, including the schema name under which it needs to be created (see Figure 3-33).

As an example, we will create a *Time* dimension based on the relational table *DATE.*

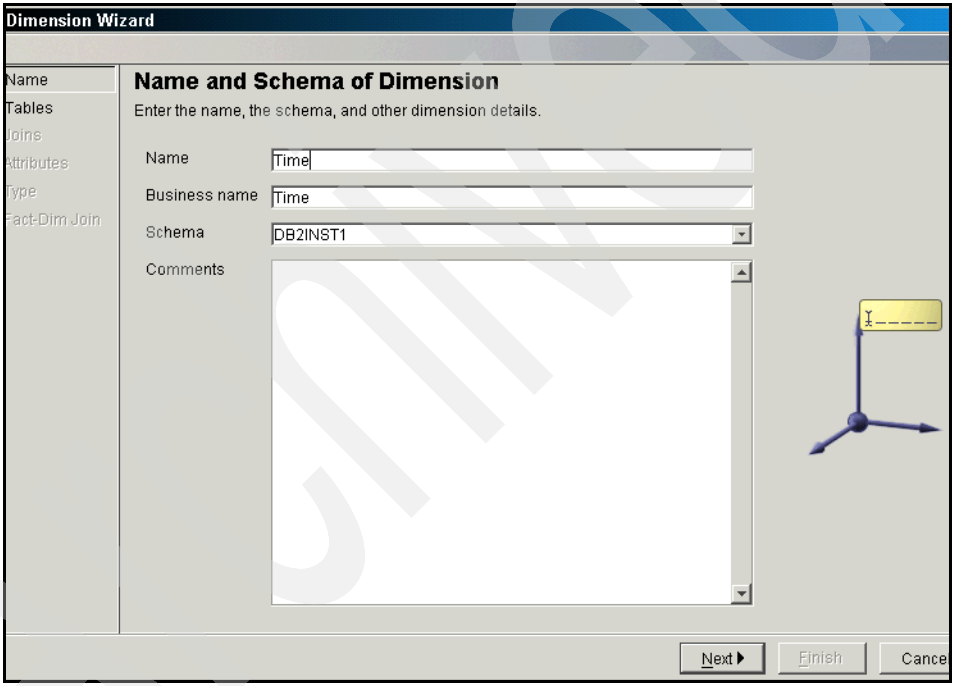


Figure 3-33   Provide name of dimension object

Click **Next** to move to the next screen.

Select the relational table needed to build the dimension. In the case of more than one table, you will need to specify the join between those tables by selecting the dimension table to be joined (see Figure 3-34).

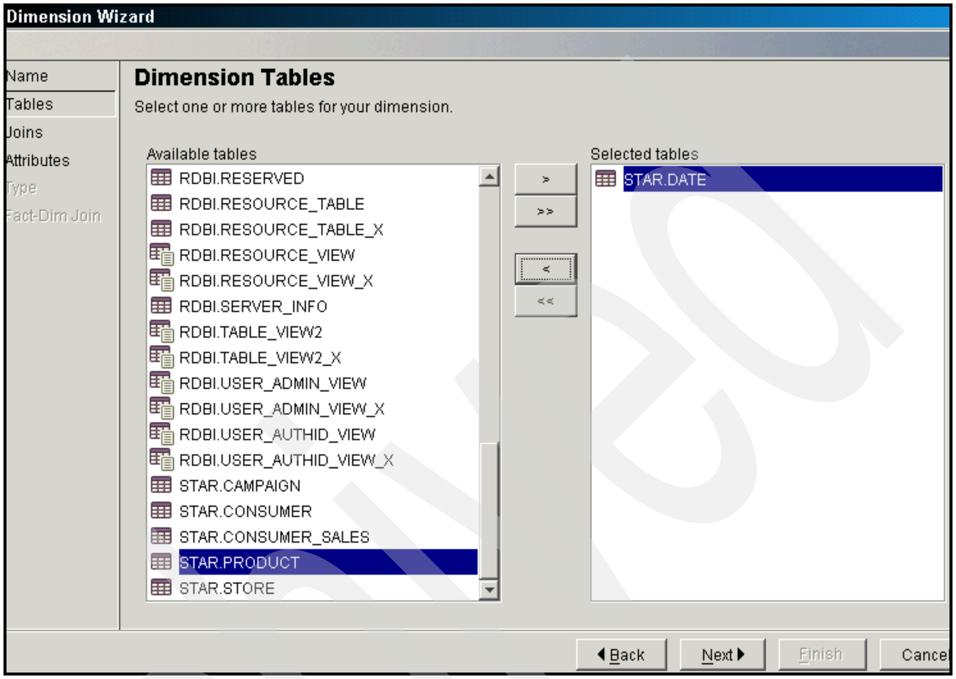


Figure 3-34 Select Dimension Table

The selected table appears in the window on the right by clicking the > button.

Click **Next** to specify the joins if using more than one table. In our example, there is only one candidate table to represent the Time dimension.

Click **Next** to specify the attributes for this dimension.

Select the attributes to describe the dimension object (see Figure 3-35).

Select and click > to identify specific attributes or click >> to select the entire list of existing attributes from the relational table.

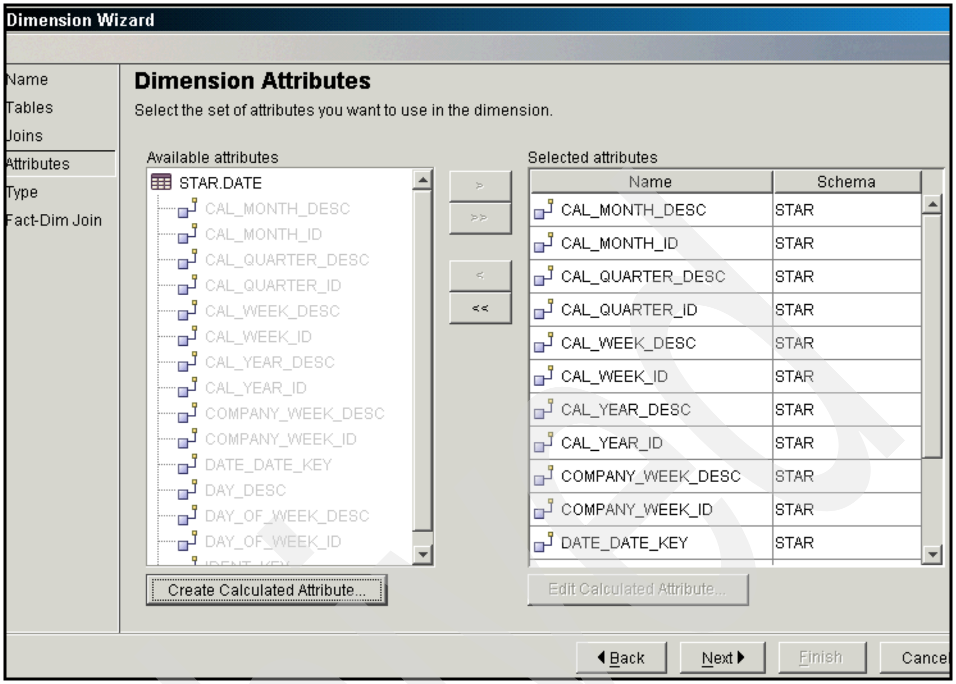


Figure 3-35 Select Dimension Attributes

Note: You can also create calculated attributes at the time of creating the dimension using the **Create Calculated Attribute...** or at a later time, by editing the attribute properties from the context menu option on the attribute.

Click **Next** to specify the Dimension Type.

Specify whether the type of the dimension is *Regular* or *Time* — those are the two dimension types that you can have — by selecting the appropriate radio button (see Figure 3-36).

In our example, we have selected type as *Time*. For other dimensions such as *Product*, *Region*, and so on, you should choose the type to be *Regular*.

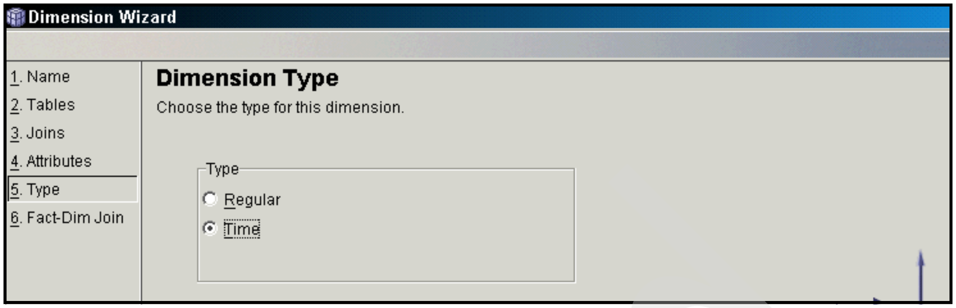


Figure 3-36 Select Dimension Type

Click **Next** to specify the join between the dimension and the facts object.

> **Note:** When using Quick Start, where Referential Integrity has already been defined, Quick Start will automatically detect the joins.

You can select an existing candidate join or use **Create Join** to define new joins (see Figure 3-37).

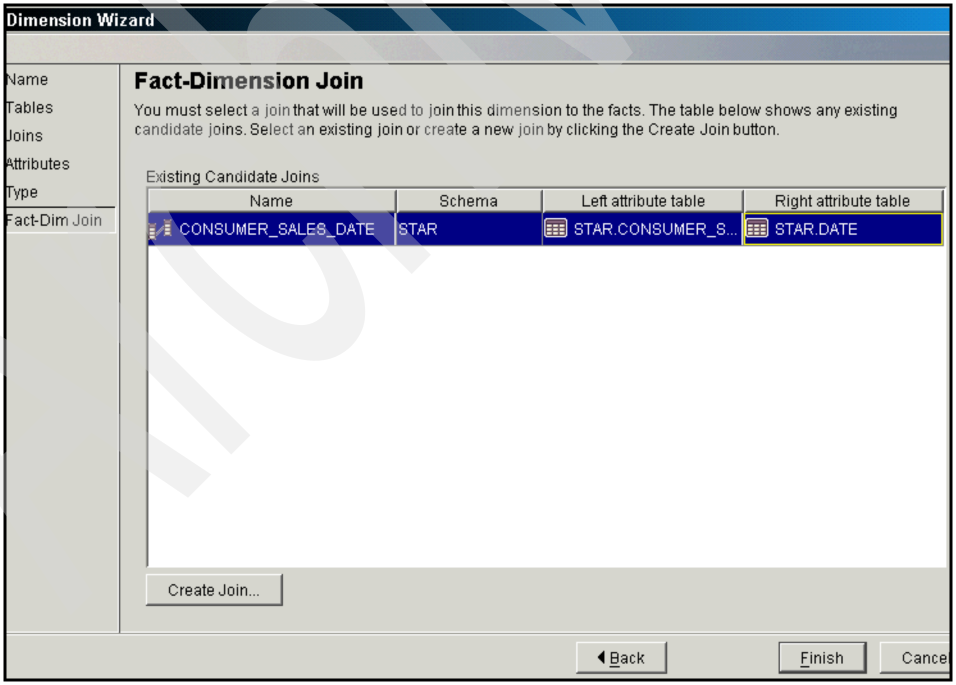


Figure 3-37 Specify Fact-Dimension Join or create a new join

Using **Create Join** will launch a window that shows columns from the relational table for the fact and dimension objects. You can select a key from each table to specify the join, type of join, and cardinality (see Figure 3-38).

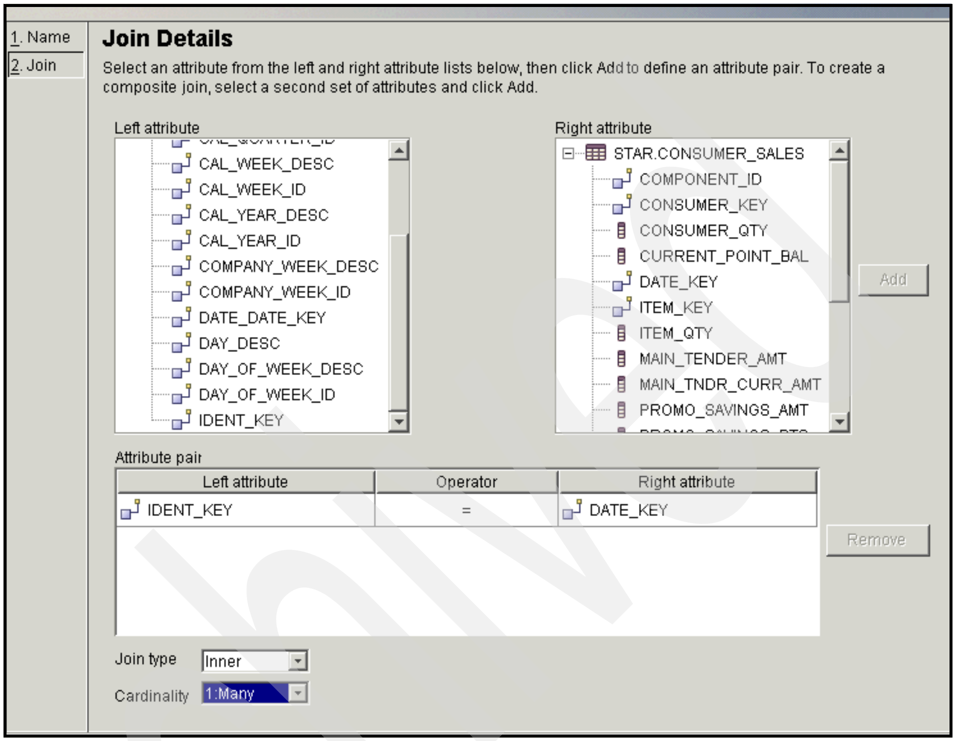


Figure 3-38 Create Time_Fact join

Click **OK** to return to the Dimension wizard and select the requisite join.

> **Note:** Two tables can be joined using more than one attribute pair or, in other words, specifying more than one attribute pair in the join information while creating/modifying a join object. To do this, select the attribute from each column to form the attribute pair and click **Add**. Repeat this to add another attribute pair.

Click **Finish** to complete creation of the dimension (see Figure 3-39).

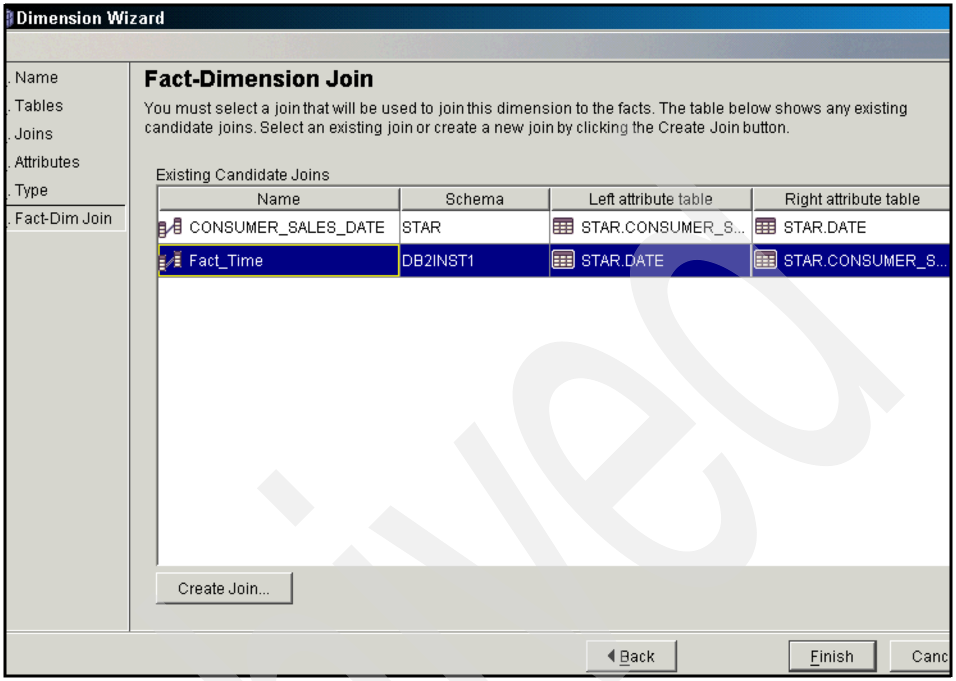


Figure 3-39 Dimension created

From the OLAP Center GUI, you can now see the dimension created under the cube model. Set the view to **Show OLAP objects --> Graphical view** (see Figure 3-40).

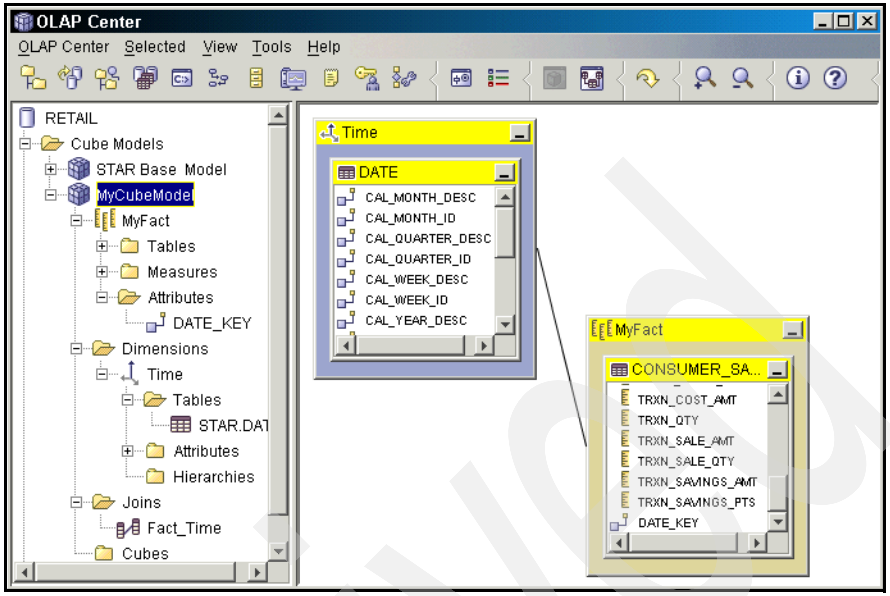


Figure 3-40 Attributes for facts objected created implicitly

On the panel on the right hand side, you will see the graphical view of the objects created with a line between the Fact and Dimension object denoting the join.

Expanding the object-tree list on the left hand side, you will see that the facts object has an implicitly created attribute once the fact-dimension join has been specified. For example, *DATE_KEY* (the foreign key in the fact table) is an attribute of the facts object *MyFact*.

Proceed in the same manner, by launching the Dimension wizard to create the other dimension objects for the cube model.

Create the hierarchies

Creating hierarchies is a important step in the creation of a cube model. Although hierarchies are a part of a dimension, they are created after the dimension object has been created.

At least one hierarchy must be defined for each dimension object in a cube model, if you want to use the dimension as part of a cube.

To create a hierarchy, select **Create Hierarchy...** from the context menu option on the dimension you want to create a hierarchy for, as shown in Figure 3-41.

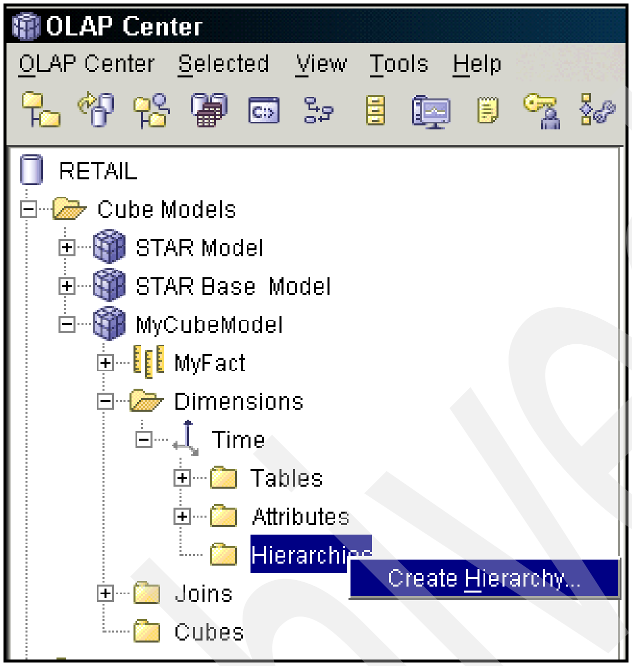


Figure 3-41 Create the hierarchy

This action launches the Hierarchy Wizard (see Figure 3-42). Provide the name and schema for the hierarchy and click **Next**.

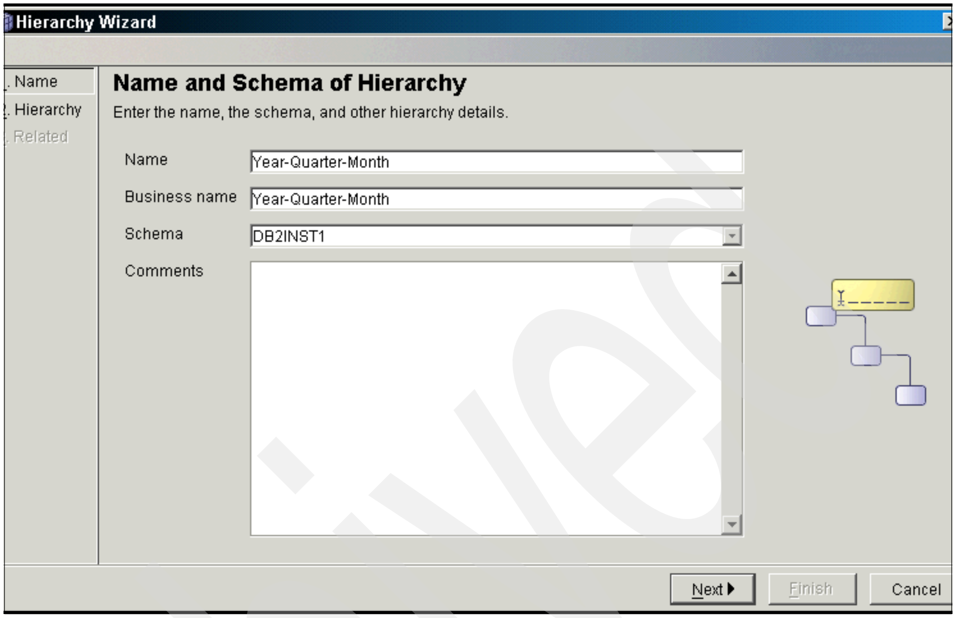


Figure 3-42 Name the hierarchy

Select the elements or attributes that form the hierarchy (see Figure 3-43). For example, to build a Year-Quarter-Month hierarchy in the Time dimension, we can choose CAL_YEAR_ID, CAL_QUARTER_ID and CAL_MONTH_ID.

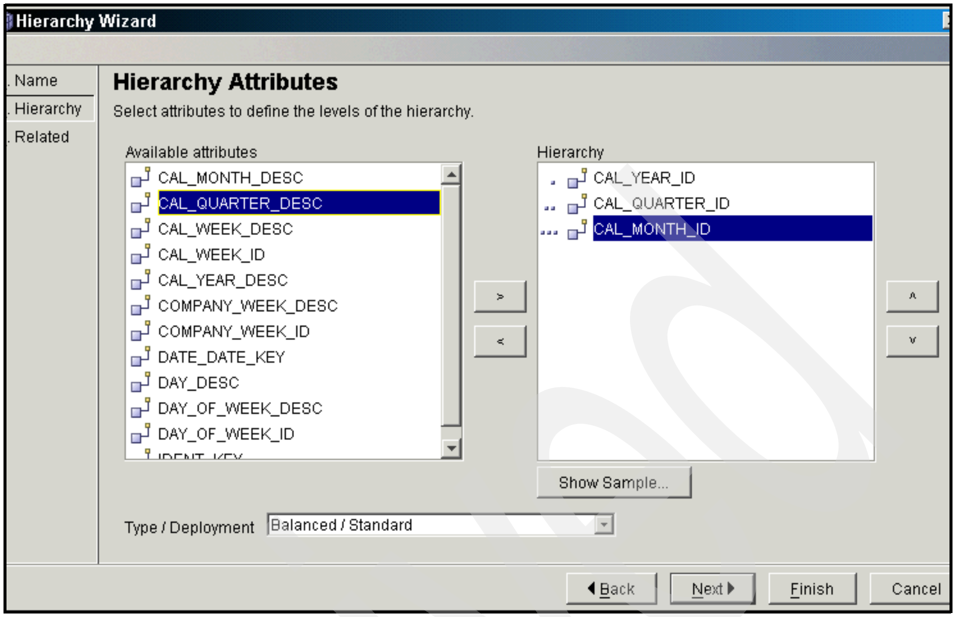


Figure 3-43 Select elements of the hierarchy

Click **Show Sample...** to display a sample hierarchy, as shown in Figure 3-44.

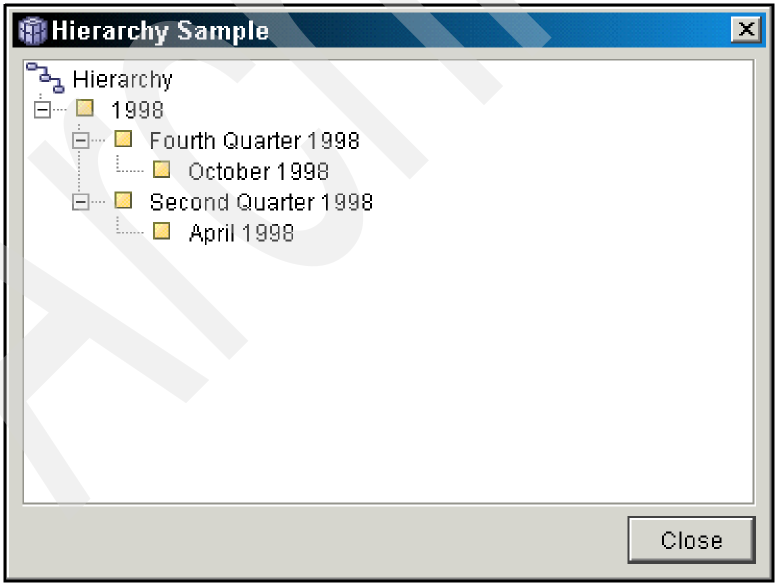


Figure 3-44 Sample hierarchy deployment

Click **Close** to return to the Hierarchy Wizard.

Select the type of hierarchy from the type/deployment drop-down list (see Figure 3-43).

> **Note:** Recursive deployment is only valid if you have two members in your hierarchy and will be shown in the pull down option if you have selected exactly two hierarchy level attributes for your hierarchy. If you have more or less than two, you won't see that option, since it won't be valid to choose.

Click **Next** to specify related attributes for the hierarchy attributes (see Figure 3-45).

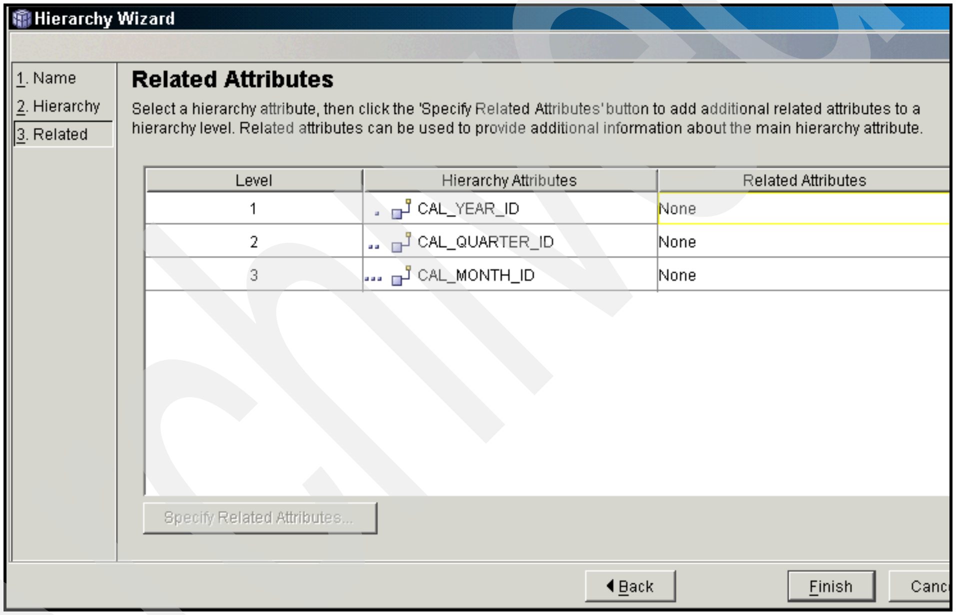


Figure 3-45 Specify related attributes.

Select an attribute in the hierarchy and click **Specify Related Attributes...** to launch a Related Attributes Selection window (see Figure 3-46). By specifying related attributes, we implicitly create attribute-relationships for the cube model.

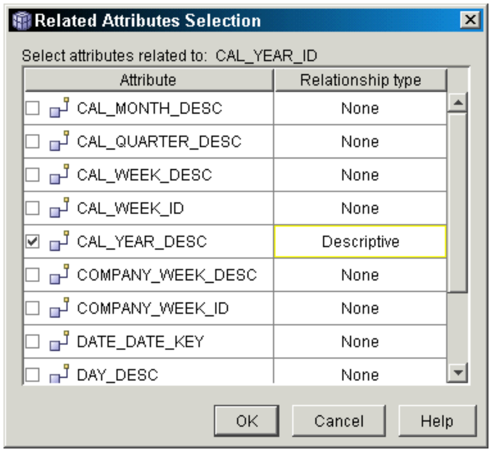


Figure 3-46 Related attributes selection

For example, we could specify that *CAL_YEAR_DESC* is related to *CAL_YEAR_ID* as a descriptive relational attribute.

Define all of the relational attributes that you need to define and then click **OK** to return to the Hierarchy Wizard. Select **Finish** to complete the definition of the hierarchy (see Figure 3-47).

You can create more than one hierarchy for a dimension in a cube model.

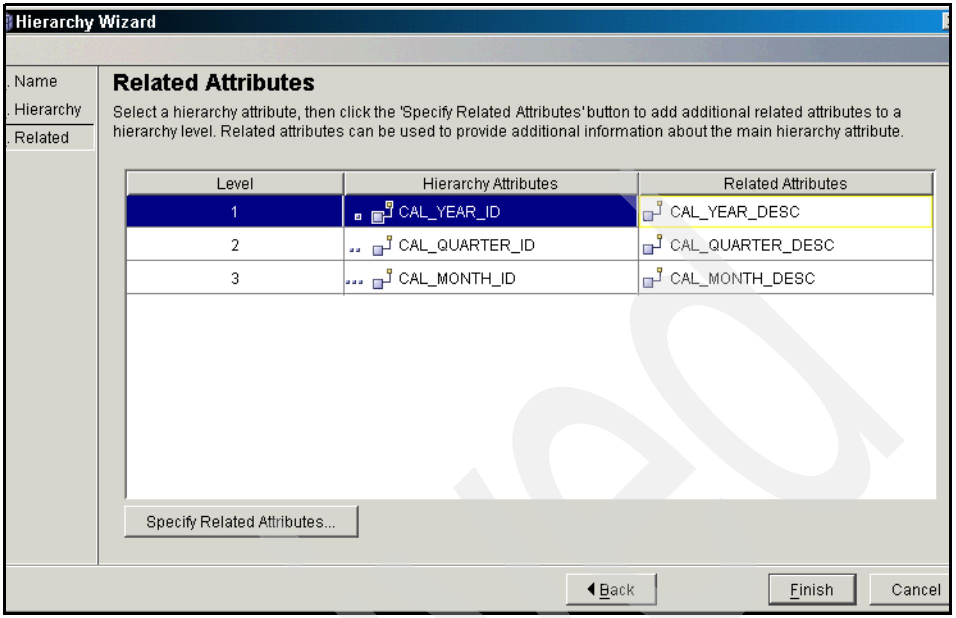


Figure 3-47 Finish creating hierarchy

Proceed to create all the dimension objects (for example, *STORE, PRODUCT, CONSUMER, CAMPAIGN* for the retail model depicted in Appendix E, “The case study: retail datamart” on page 685) for the cube model and the hierarchies for each dimension.

This completes the process of creating a basic cube model with fact and dimensions and related objects.

See Figure 3-48 for a representation of a complete cube model created using the OLAP Center.

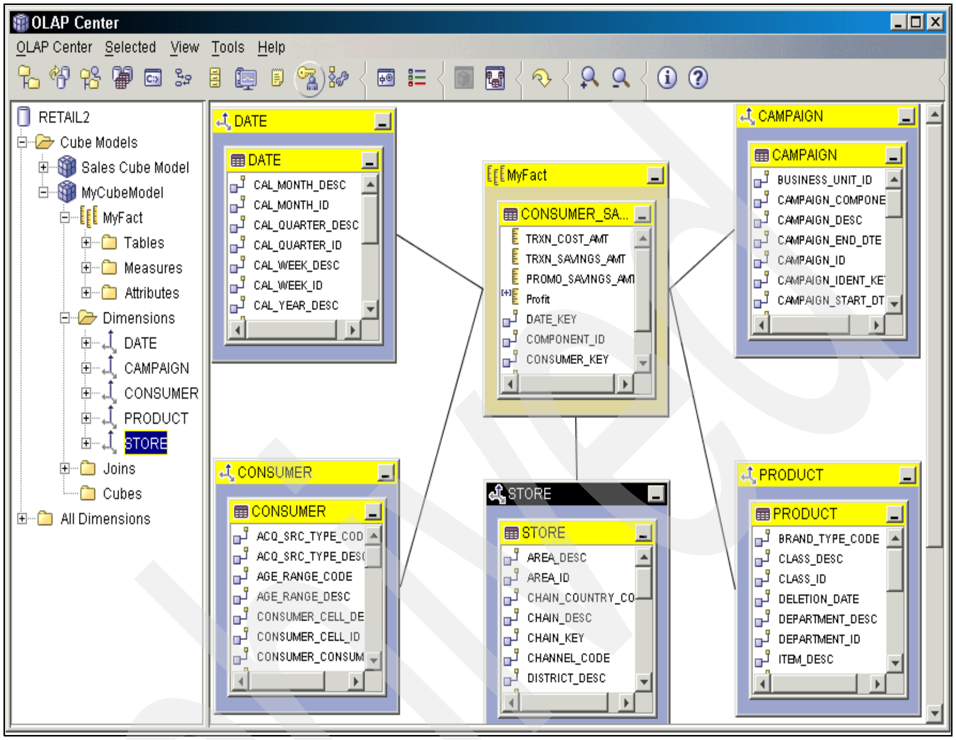


Figure 3-48 Complete cube model

Create the cube

Even if you can create a cube model without any cubes and validate it (when you cannot create a cube without a cube model), a basic recommendation will be to always define a cube model and a cube, especially to benefit from the Optimization Advisor on query performance (refer to 4.4.1, “Get at least a cube model and one cube defined” on page 136). The cube model corresponds to the subject area or star schema, and the cube satisfies the requirements of a particular project.

1. To build a cube, from the OLAP Center, expand the object tree list for the cube model. Right-click **Cubes** and select **Create Cube...** (see Figure 3-49) to launch the Cube Wizard.

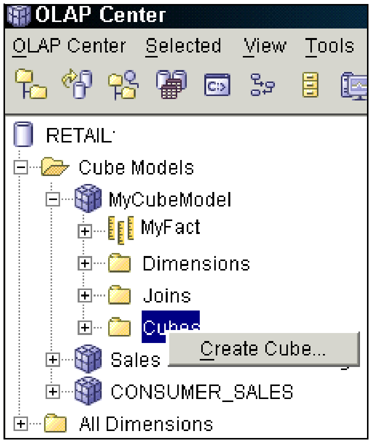


Figure 3-49 Create the cube

2. Provide the Name and Schema for the cube (see Figure 3-50).

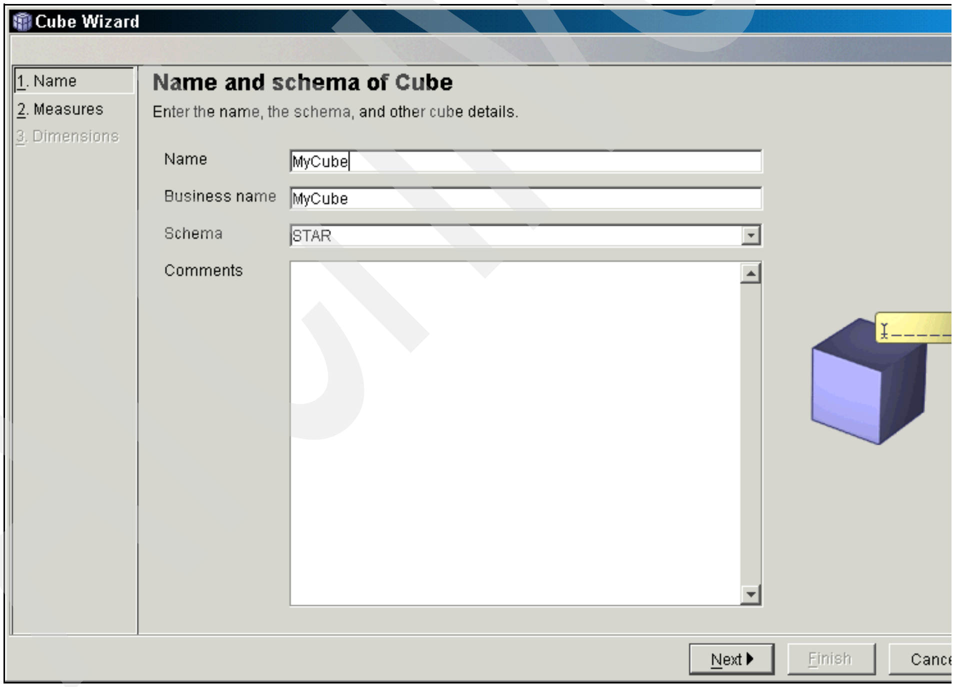


Figure 3-50 Name and schema for the cube

3. Select from available measures, a subset that you want to include for the cube for your specific project business requirements. The available measures are the ones that were defined for the cube model (see Figure 3-51).

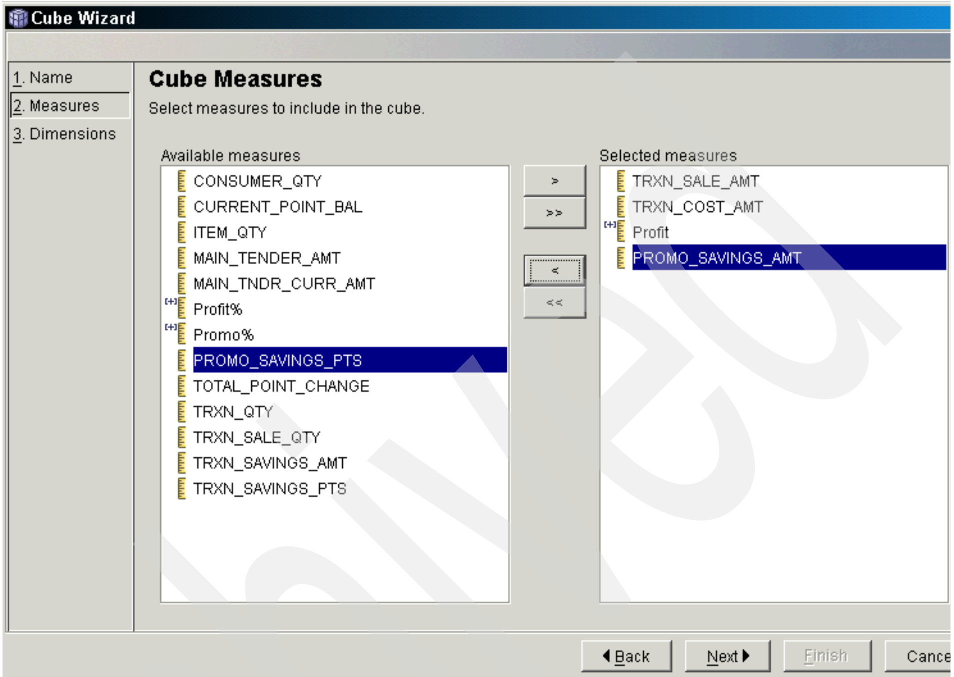


Figure 3-51 Select from available measures

4. Select the dimensions for the cube (see Figure 3-52). The available dimensions are those defined for the cube model.

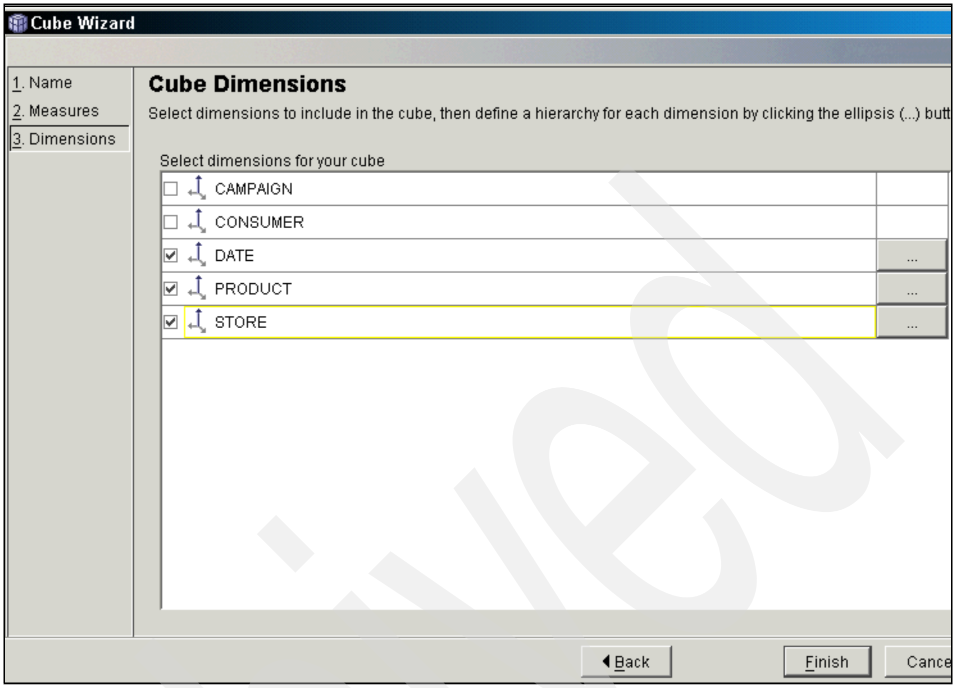


Figure 3-52   Select the cube dimensions

5. Select the hierarchy for the cube by clicking the push button. Remember that a cube can have only one hierarchy defined. Select the hierarchy for the cube from the drop down list (see Figure 3-53).

   By default, all the levels in a chosen hierarchy are selected. You can further deselect members from the hierarchy list.

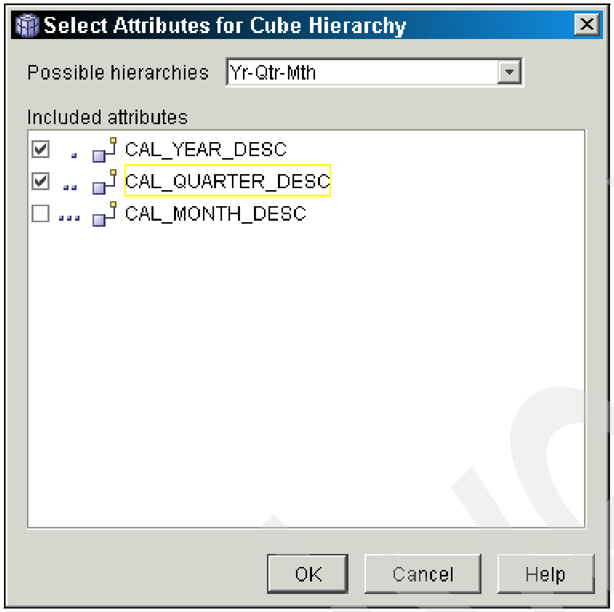


Figure 3-53 Select the cube hierarchy

6. Click **OK** to return to the Cube Wizard and click **Finish** to complete creating the cube.

To ensure cube model completeness, OLAP Center will validate cube models and cubes once created and will enforce a set of rules. The rules are listed and documented in *IBM DB2 Cube Views Setup and User's Guide,* SC18-7298 in the Metadata Object Rules section as:

- Base rules
- Completeness rules
- Optimization rules

3.4 Enhancing a cube model

Irrespective of whether the cube model has been imported or created, it may need additional updates. Updates to a cube model are essentially driven by the requirement for enhanced functionality in order to satisfy certain analytical reporting requirements.

3.4.1 Based on end-user analytics requirements

The source from which metadata is imported may not have the same level of detail that DB2 Cube Views requires, or there may be limitations, due to the bridges used, on the level of detail that can be imported into DB2 Cube Views. For example, metadata pulled from an ErWin model has only information of fact and dimensions. Other information like hierarchies and attribute-relationships have no meaning within the context of an ErWin model and consequently, the imported cube model will not contain hierarchy related information. Therefore, the imported cube model will have to undergo enhancements by the designer to match business requirements.

Next, we discuss some of the scenarios where enhancements to a cube model may be applicable:

- ***Business Names for metadata objects in DB2 Cube Views:***

  Modifying the Business Name of an object makes more sense to the end user performing the analytics. If not given, the tools and applications that access metadata from DB2 Cube Views will simply retrieve the column names from the relational table. Cryptic names for columns make it very hard for a user to make sense of the data retrieved. For example, Transaction Sale Amount can be the business name for the column *TRXN_SALE_AMT*.

  To change the Business Name of any object in DB2 Cube Views, right-click the object and select **Properties...**

- ***Create Advanced Measures:***

  Instead of having measures that come directly from relational table columns, DB2 Cube Views supports complex measures. These measures can be additive (symmetrical or distributive) or semi-additive (asymmetrical or non-distributive). See Example 3-2.

Example 3-2 Additive, semi-additive measures

```
Profit is an example of an additive measure
Inventory is an example of a asymmetric measure (semi-additive)
   Inventory is an example of a measure that does not SUM across Time.
   Inventory = function (Time), SUM (Stores)
   Functions that can be used are AVG, MAX, MIN. So, value of Inventory can be
   averaged over Time and summed over Stores.
```

You can define how the measures are derived from the database columns when you create a calculated measure. See Example 3-3.

Example 3-3 Calculated measure: Profit = Sales - Cost

```
@Column(STAR.CONSUMER_SALES.TRXN_SALE_AMT) -
@Column(STAR.CONSUMER_SALES.TRXN_COST_AMT)
```

You can also write an aggregation script to specify different types of aggregation across different dimensions.

When defining complex measures as in Example 3-4, you can control the order of aggregation.

Example 3-4 Complex measure

```
Profit Margin = SUM (Profit)/ SUM (Revenue) and NOT SUM (Profit/Revenue)
```

You can create a measure as a function of two parameters (see Example 3-5), to correlate Sales and Marketing to support analysis of the effectiveness of your marketing.

Example 3-5 Two parameter function

```
CORRELATION (Sales,Marketing)
```

All these functions can be performed from the OLAP Center by right-clicking the Measures object and selecting either **Edit Measures** or **Edit Aggregations**

- ***Create calculated attributes:***

  You can create attributes derived from base data (see Example 3-6). To create a calculated attribute, from the OLAP Center, right-click the **Attributes** tree object in a dimension and select **Edit Attributes** and then click **Create Calculated Attribute**

  You can also perform the same function by right-clicking a dimension object and select **Properties...** or **Edit Attributes...**

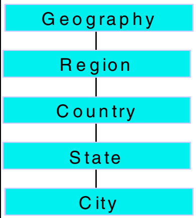
Example 3-6 Calculated attribute

```
To build an attribute that has Day, Month and Year (for example, 1Jan2003), you
can create a calculated attribute called Day_Month_Year which maps to the
expression
CAST(@Column(STAR.DATE.DATE_KEY) AS CHARACTER) CONCAT
@Column(STAR.DATE.CAL_MONTH_DESC)
```

Note: You have to perform a `CAST` to ensure that the strings which are concatenated are of the same data type.

- ***Create hierarchies other than just balanced hierarchy:***

  Data in the relational table may not always allow implementation of a balanced hierarchy like the Year-Quarter-Month hierarchy in a Time dimension. Figure 3-54 is an example of a ragged hierarchy.



Region	Country	State	City
Americas	USA	California	San Jose
Europe	Greece	-	Athens
-	Iceland	-	Reykjavik

Figure 3-54 Ragged hierarchy

Some countries in this hierarchy do not have *State* and similarly some countries do not have any associated semantics for Region. For this type of data, you can implement a ragged hierarchy for the cube model using the OLAP Center. To do this, right-click the Hierarchies tree object or the dimension object that you wish to create a hierarchy for and select **Create Hierarchy...**

The top-bottom flow chart in Figure 3-55 gives an idea of how to decide the type of hierarchy to be deployed.

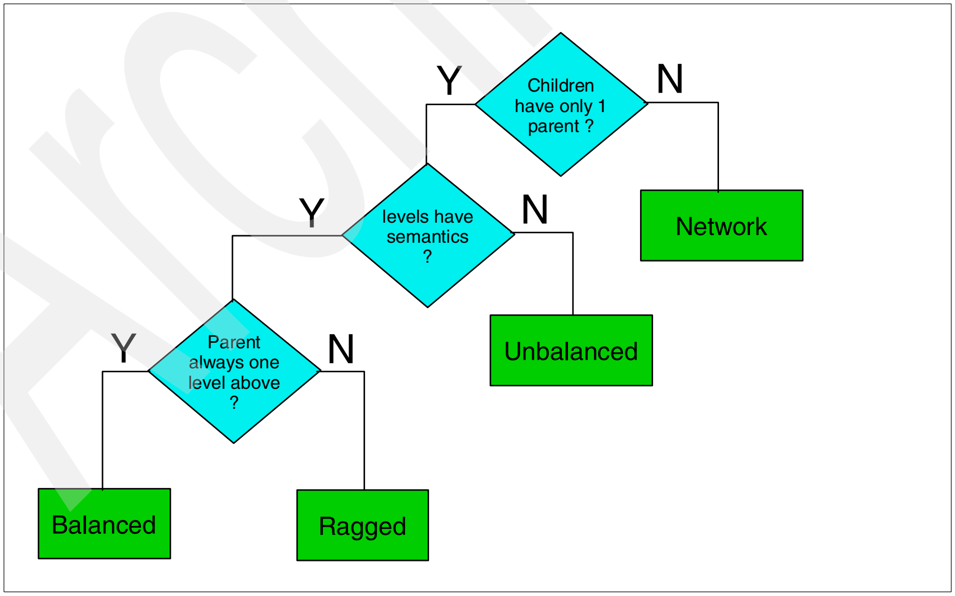


Figure 3-55 Hierarchy flowchart

- ***Create attribute relationships explicitly:***

  Attribute relationships can be created explicitly (see Example 3-7). To do this, **Show Relational Objects** from **View** in the OLAP Center. Right-click **All Attribute Relationships** and select **Create Attribute Relationship**

Example 3-7 Gender flag and gender description attribute relationship

```
You can create an attribute relationship between Gender_Flag and Gender_Desc.
The type of relationship is Descriptive - Gender_desc describes Gender_Flag.
The cardinality is 1:1 i.e. a value of Gender_Flag determines only one value of
Gender_Desc and vice versa.
```

3.4.2 Based on Optimization Advisor and MQT usage

There are four different types of query patterns that can be performed against the relational database. They are extract, drill down, drill through, and report. Please refer to 4.4.3, "Do you know or have an idea of the query type?" on page 143 for further information on the query patterns.

The cube model may need to be enhanced based on the type of queries that are run against the star schema in order that those actions have better performance. This may involve building cubes within the cube model to accommodate better optimization. We need to remember here that cubes should be built as proper subsets of the cube model.

Whether to build a cube or not is also relevant from the end user tool or application tool accessing the metadata. Using Office Connect, for example, requires that many cubes be built based on the slice of data that the user frequently retrieves. If extracts are regularly performed, then again a cube must be built to mimic the SQL query behind the extract. Cubes can also be used to act as filters against the cube model, thus letting the user access only a slice of the data that he is interested in.

These concepts are discussed in detail in Chapter 4, "Using the cube model for summary tables optimization" on page 125.

3.5 Backup and recovery

Regular DB2 backup and restore procedures should be in place for the relational database in DB2 storing the star schema. This is the recommended approach to back up and recover DB2 Cube Views metadata objects to be sure to keep synchronization between the metadata objects and the data itself.

In other words, avoid to backup the DB2 Cube Views metadata objects only, for backup purposes.

For example, the OLAP Center export and import features for backing up and recovering DB2 Cube Views metadata only is not a recommended approach and should be used with caution as an issue may be to lose synchronization between metadata objects in DB2 Cube Views catalog tables and data in the DB2 database. For the same raisons, use **db2move** utility to move the DB2 Cube Views catalog tables only is not recommended either.

When you need to move the DB2 Cube Views catalog tables only from one server to another one for example from development to test environment, prefer using OLAP Center export and import features or the **db2mdapiclient** utility instead of using the DB2 utility **db2move**. The **db2mdapiclient** utility, provides a way to export all objects within a cube model (cube model and *all* its cubes and other objects). This does not allow the user to select which cubes within a cube model should be exported.

3.6 Summary

In this chapter, we looked at the basic concepts and terminologies involved when describing a cube model in DB2 Cube Views. Cube modeling with DB2 Cube Views is designed for star (or snowflake) schemas. The objects that describe a cube model are the fact, dimensions, hierarchies, joins, attributes and attribute-relationships.

This chapter also demonstrates the different methods of building a cube model in DB2 Cube Views. A cube model can be built by import, with Quick Start wizard or from scratch. When building a cube model by import, you start with OLAP metadata that is already available, which has been passed through a suitable bridge to transform it into DB2 Cube Views format. When building a cube model from scratch, you can either use the Quick Start wizard or build the metadata objects yourself. Using the Quick Start wizard builds a cube model using existing joins between fact and dimension tables and this requires RI (referential integrity) implemented for the star schema.

You can also choose to build a cube model by sequentially defining the objects (facts, dimensions, joins) yourself.

> **Note:** Even if Referential Integrity is highly recommended for DB2 Cube Views and pre-requisite for Quick Start wizard, it is not mandatory when building the cube model manually and informational constraints may be used (see4.4.2, “Define referential integrity or informational constraints” on page 136).

> **Important:** It is RI alone (along with the variation introduced in DB2 V8.1 called Informational Constraints) that informs the DB2 optimizer of the relationships that guide query rewrite and MQT routing.

4

Using the cube model for summary tables optimization

This chapter describes what Materialized Query Tables (MQTs) are, how they are used, and how you can improve on or optimize their use.

4.1 Summary tables and optimization requirements

Data warehouses and datamarts generally contain large amounts of information, often exceeding terabytes in size. Decision support functions in a data warehouse or datamart, such as OnLine Analytical Processing (OLAP), involve hundreds of complex aggregate queries over these large volumes of data.

Since many of these queries are run frequently, they may cause a significant workload on the systems supporting the data warehouse or data mart. Other queries may aggregate so much information that they impede or exclude other work scheduled to run on the system. Taken as a whole, available system resources prohibit the repeated aggregation of the base tables every time one of these queries are run, even when appropriate indexes exist.

Therefore, as a solution to this problem, the decision support DBAs can build a large number of summary tables, or materialized aggregate views, that pre-aggregate and store the results of these queries to help them increase the system performance.

In modeling terms, the summary tables group the data along various dimensions, corresponding to specified levels of hierarchy, and compute various aggregate functions or measures.

As an example, some of the kinds of aggregate requests we might expect could include:

- Sales of a specific product category in all stores by month
- Sales of a specific product category by store type relative to campaigns
- Sales data for a specific time period, product, and district
- Sales data by consumer demographics

These types of requests involve data scans, joins, aggregations, and sorts, and if they are performed repeatedly against the base fact table and dimension tables, they will result in poor query performance. Instead, when a DBA creates summary tables that have already performed this work and stored the results so that they are available for subsequent query users, the result can be dramatically improved response times for the query workload.

Support for summary tables usually requires the following activities:

- In generic terms, a DBA must define and precompute an aggregate query and materialize the results in a table. This aggregate will contain a superset of the data requested by a large number of queries. In DB2 Cube Views, however, a wizard is provided which does most this work for the DBA. This wizard, the Optimization Advisor, will be described in detail in this chapter.

- The optimizer must recognize the existence and applicability of the summary tables to these queries and automatically rewrite the queries to access the pre-aggregated data.

Summary tables are a powerful performance feature. They are typically considerably smaller than the cross product of the base tables on which they are defined. Because of this and the fact that they contain pre-aggregated data, queries requesting aggregates may experience dramatically improved performance through their use.

DB2 provides support for summary tables through its Materialized Query Tables, or MQTs. Its implementation of MQTs is more generalized than just summary data. DB2 permits a materialized query table to be created without aggregation, providing the benefits of pre-joins or caching of data from distributed remote sources. In the case of analytical data in general and of DB2 Cube Views in particular, the materialized data is always summarized and aggregated. A summary table, therefore, is a specialized type of MQT and the only type we're considering in this book.

4.2 How cube model influences summary tables and query performance

As we discussed in Chapter 3, "Building a cube model in DB2" on page 63, a cube model is usually built to represent some form of star schema containing dimensions, facts, hierarchies and aggregates. This aggregated, dimensional data and the analytical workload described in the previous section are well-suited to benefit significantly from MQTs. While MQTs do offer significant performance advantages, they must be carefully planned in order to maximize their effectiveness.

OLAP aggregates are developed from combinations of measures across dimensions. Storage requirements quickly balloon as dimensions and levels of hierarchy in those dimensions expand. To give a very simple example, let's assume a DBA wants to aggregate sales of products by store by campaign by week for two years. Let us further assume that there are 5,000 products, 1,000 stores, and 20 campaigns. That would give us 10,400,000,000 sales values (5,000 x 1,000 x 20 x 104). But realistically, there will be more dimensions and a requirement to aggregate to a number of higher levels of hierarchy in each dimension, for example, months and quarter-years, in the time dimension.

In addition to the concern about the potential for disk consumption, there are other challenges with containing aggregates. We must consider the time required to create and maintain the aggregates. Particularly in the cases where you require that the MQTs remain current with the base data, we must factor in the amount of time required to update all MQTs affected by base data changes. We'll discuss the various types of MQTs and maintenance options in the next section.

As the number of MQTs grows, we also must recognize that this places an additional burden on the DB2 optimizer to evaluate each one as a candidate for the target of a query rewrite. A very large number of MQTs can potentially slow down the performance of a query by significantly extending the amount of time required to optimize it.

One solution to these issues is to pre-aggregate only some of the data, allowing the rest of the data to be aggregated on demand. Obviously, this is most effective when the most frequently requested slices of data are pre-aggregated and stored. Making that determination can be quite challenging for DBAs.

DB2 Cube Views has introduced a very sophisticated Optimization Advisor (see Figure 4-1) that performs a cost/benefit analysis of a number of potential MQTs based on the multidimensional model, the anticipated workload type, catalog statistics, and block-level sampling of the data, as described in Figure x below. The extent of this cost/benefit analysis is governed by the administrator's specification of the amount of disk space available to store MQTs and the maximum amount of time to spend on the sampling process.

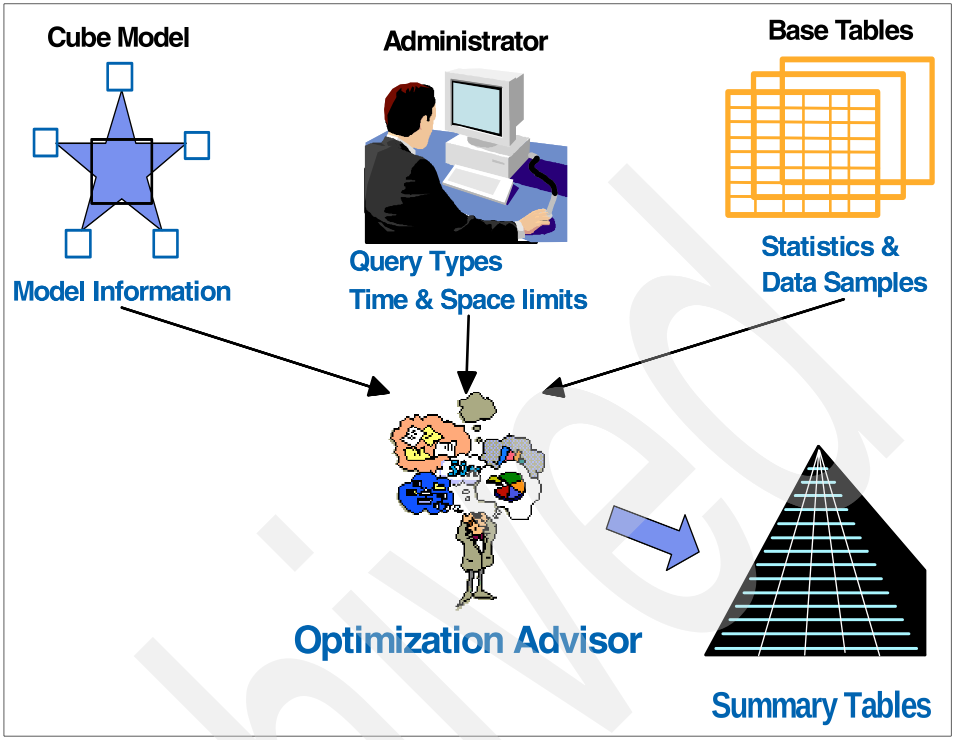


Figure 4-1 The Optimization Advisor

Based on this analysis, the Optimization Advisor makes recommendations about the particular slices of the multidimensional model that will be most beneficial to pre-aggregate and store while minimizing the total number of MQTs that have to be stored, maintained, and considered for optimization. This is very important to the DBA who wants to maximize the effectiveness of the MQTs while minimizing storage utilization and controlling maintenance. DB2's optimizer can take advantage of both exact and partial matches with existing MQTs. The number of MQTs recommended by DB2 Cube Views will vary based on the number of hierarchies defined in the model, the types of query workloads anticipated, and the existence of non-distributive measures.

Distributive measures use simple aggregation functions such as **`SUM`** and **`COUNT`** that can be aggregated from intermediate values. Non-distributive measures use more complex aggregation functions, such as **`STDDEV`** and **`AVG`**, which must be aggregated each time from the base tables.

> **Note:** If **AVG** is needed in an MQT and will be aggregated further in a number of queries, you may consider including **SUM** and **COUNT** as measures and derive the **AVG** function from these values (**SUM**(**SUM**)/**SUM**(**COUNT**)) to avoid pushing the query to the base tables.

DB2 supports several complex **GROUP BY** expressions that offer significant benefit with MQTs. DB2 can view these complex aggregations as separate groupings, thus allowing queries to use MQTs that are defined with groupings that are supersets of those requested in the query. DB2 Cube Views exploits this capability to reduce the total number of MQTs it recommends while still accommodating a large number of potential queries. The two complex groupings which DB2 Cube Views supports are **GROUPING SETS** and **ROLLUP**.

A grouping set allows multiple grouping clauses to be specified in a single statement. It can be thought of as the union of two or more groups of rows into a single result set. Its efficiency results from reducing the requirement to scan the data. For example:

```
SELECT … GROUP BY GROUPING SETS ((store_id, product_group_id),(date_year,
date_month))
```

is equivalent to:

```
SELECT … GROUP BY store_id, product_id
UNION ALL
SELECT … GROUP BY date_year, date_month
```

With the grouping set, the data is scanned only once whereas with the union, it is scanned twice. This is very significant with MQTs in that we’re not only reducing the requirement for scanning the data but also reducing the number of MQTs being built.

ROLLUP is an aggregation over a dimension hierarchy, sub-totaling at every level of the hierarchy.

For example:

```
SELECT …
GROUP BY ROLLUP (region_id,district_id,store_id)
```

is equivalent to:

```
SELECT …
GROUP BY (store_id, district_id, region_id)
UNION ALL
SELECT …
GROUP BY (district_id, region_id)
UNION ALL
```

```
SELECT …
GROUP BY (region_id)
UNION ALL
SELECT …
GROUP BY ()
```

As a rule, DB2 Cube Views recommends the smallest number of MQTs possible to enhance the performance of the largest number of queries. The number of MQTs can be minimized by using these complex constructs because DB2 is able to understand the separate sub-groupings that exist as part of the superset and optimize a large number of queries that are not an exact match with the MQT SELECT statement. DB2 Cube Views provides three significant advantages by using these complex constructs:

- Reduction in number of MQTs
- Reduction in total disk space
- Reduction in refresh times by minimizing base table scans

DB2 can use an MQT when the grouping is different from the requested grouping if the MQT has a **`GROUP BY`** on a finer granularity than the request. For example, if an MQT aggregates at the month level, DB2 can use that MQT to support requests for data grouped by quarter or year. Therefore, it is unnecessary to define multiple MQTs or slices for each different level.

Thus, the DB2 Cube Views Optimization Advisor and the DB2 optimizer combine to provide a very powerful summary table capability.

4.3 MQTs: a quick overview

MQTs offer quite a bit of flexibility in terms of whether their data remains current with regard to the base data, whether the system automatically maintains them, how they are maintained, and which of these types is available to the optimizer for query rewrite.

MQTs is a very comprehensive subject. In this section we only attempt to provide a quick overview of the different options available with MQTs and to focus on the features supported by DB2 Cube Views.

We will also discuss additional MQT options available in DB2 and considerations for using them.

First, there are options you specify when you create MQTs, and then there are options for maintaining them. We'll begin by discussing the options for creating them, and then cover maintenance.

4.3.1 MQTs in general

An MQT is created with a **CREATE TABLE** statement with an AS fullselect clause and the **REFRESH IMMEDIATE** or **REFRESH DEFERRED** option. A summary table additionally includes aggregation.

Example 4-1 is a simple example of the SQL to create and populate an MQT aggregating data between the STORE and SALES tables.

Example 4-1 MQT example

```
CREATE TABLE MQT1 AS (
SELECT S.SID, SUM(SALES) AS SALES
FROM STORE S, FACT F
WHERE S.SID = F.SID
GROUP BY S.SID)
DATA INITIALLY DEFERRED
REFRESH DEFERRED
```

You may optionally identify the names of the columns of the MQT. The column names are required if the result of the full select has duplicate names or unnamed columns.

After the table has been created, the MQT has to be synchronized with the base tables. This is controlled via the refresh option. The System Maintained MQT refresh options are summarized in Table 4-1.

> **Note:** Since DB2 Cube views only generate System Maintained MQTs, we do not cover User Maintained MQTs in this book.

Table 4-1 DBA major work needed - various System Maintained MQT options

	REFRESH IMMEDIATE		REFRESH DEFERRED	
	FULL REFRESH	**INCREMENTAL REFRESH**	**FULL REFRESH**	**INCREMENTAL REFRESH**
Maintenance after SQL INSERT, UPDATE, DELETE to base tables	-	Not Applicable	-	REFRESH <MQT>;
Maintenance after LOAD INSERT to base tables	DROP <MQT>; CREATE <MQT>; REFRESH <MQT>;	CREATE<index> on <MQT>; SET INTEGRITY <base table>; REFRESH <MQT>;	DROP <MQT>; CREATE <MQT>; REFRESH <MQT>;	CREATE<index> on <MQT>; SET INTEGRITY <base table>; SET INTEGRITY <staging_table>; REFRESH <MQT>;

Table 4-1 basically specifies what major work the DBA needs to do with the MQTs after changes are made to the base tables. There are two cases:

- One where changes are made with SQL
- One where the base tables are loaded

There is, however, an important difference between the two options when a **`LOAD INSERT`** is performed on the base tables:

> **Important:** The MQTs can still be used in query rewrites when they are created using the **`REFRESH DEFERRED`** option even though the base tables have been reloaded with new and most likely changed data. This can produce wrong results from queries as long as the MQTs are not in sync with the base tables.
>
> However, if the **`REFRESH IMMEDIATE`** option is used, the MQTs will be set to check pending and thus can not be considered by DB2 as candidates for query rewrites. This ensures that queries will only use the base tables after a load as long as the MQTs are not synchronized with them. Unless you perform an online **`LOAD (ALLOW READ ACCESS)`** on the base tables, then the MQT still can be considered as a candidate for query rewrite.

It should also be noted that because MQTs are instantiated as *real tables*, the same guidelines apply to MQTs as for ordinary tables with regard to optimizing access using tablespace definitions, creating indexes, reorganizing and collecting statistics.

4.3.2 MQTs in DB2 Cube Views

Since the Optimization Advisor will generate the MQTs for you, you do not have to worry about the creating the DDL yourself. The most important thing to consider, however, is the summary table update option and what to do when a cube model is updated or removed.

Refreshing the MQTs

The Optimization Advisor creates summary table refresh scripts to synchronize the summary tables with any change in the base table data.

The two options are **REFRESH DEFERRED** or **REFRESH IMMEDIATE**:

- If the summary table (MQT) has been created with the **REFRESH IMMEDIATE** option and if the base tables are updated using regular SQL statements, such an **INSERT**, **UPDATE**, **DELETE**, and **IMPORT**, DB2 automatically synchronizes the affected summary tables.
- However, in all cases, if the base tables are updated using the DB2 **LOAD** command, or the summary table has been created with the **REFRESH DEFERRED** option, the synchronization needs to be manually triggered by running the refresh script.

What this basically means is that your summary tables have to be refreshed manually with a **REFRESH TABLE <MQT tablename>** statement in all cases except where the summary table has been created with the **REFRESH IMMEDIATE** option and the base table(s) are changed using regular SQL statements.

Since it is likely that the summary tables will be out of synchronization with the base tables after the base tables are updated or changed, it is important to plan for the maintenance of the summary tables in advance prior to putting them in production. There are several reasons for this:

- The main reason is that the base tables oftentimes are loaded and not altered with SQL statements.
- Only the **SUM**, **COUNT**, **COUNT_BIG** and **GROUPING** aggregation functions are usable in **REFRESH IMMEDIATE** MQT. Otherwise the Optimization Advisor might change the **CREATE TABLE** statement of the MQT from **REFRESH IMMEDIATE** to **REFRESH DEFERRED**.

- MQTs can be in **CHECK PENDING** state (see “Is the MQT accessible?” on page 185 for further information). Whenever an MQT table is in **CHECK PENDING**, the table cannot be used for query rewrite, and system performance is thereby impaired.

As a basic starting rule, we recommend that you select **REFRESH IMMEDIATE** whenever possible to insure that the summary tables are automatically kept synchronized with the base tables, but keep in mind that **REFRESH DEFERRED** may be useful in some cases that we will detail later on.

Further steps will be to plan for synchronization of summary tables with the base tables as early as possible in the development cycle and to include it in the normal database maintenance schedule. Detailed scripts are provided in “Further steps in MQT maintenance” on page 198.

> **Note:** DB2 does not permit an MQT to be built on another MQT. Therefore, if your base data contains MQTs, they will not be considered by OLAP Center to be part of the model.

Dropping a summary table

DB2 Cube Views does not drop the associated summary tables when a cube model is dropped. If you do not use the summary tables for any other purpose, you can drop the tables to free up disk space. Summary tables are a type of table, and can be dropped using the normal DB2 procedures using the Control Center or the command line. Any associated indexes are also dropped with the summary table.

Here are the steps to drop a summary table from a command line:

1. Connect to the database of the cube model that you dropped. For example, enter: **db2 connect to RETAIL**.
2. Enter the following command: **DROP TABLE <table_name>**, where table_name is the name of the summary table that you want to drop.

> **Note:** When rerunning the Optimization Advisor, you should manually drop any old MQTs from the previous optimization not being recreated during the new optimization. They will not be dropped automatically.

Over time, the number of MQTs suggested by the Optimization Advisor may change. This means that if the number of MQTs decreases compared to an earlier optimization, you need to manually drop the extra MQTs. The reason for this is that the Optimization Advisor only creates **DROP** statements for those MQTs that are being created.

4.4 What you need to know before optimizing

There are basically four things that you need to have or know before using Optimization Advisor:

1. Get at least a cube model and one cube defined.
2. Make sure that referential integrity and informational constraints are in place on the base tables.
3. Know or have an idea of the type of queries that will be used on the OLAP database.
4. Understand how Optimization Advisor uses the cube model/cube definitions and how they interact together to leverage query optimization.

4.4.1 Get at least a cube model and one cube defined

The first and foremost design recommendation is that there should only be one cube model created per star schema. There are several reasons for this, but among them is the difficulty of synchronizing cube models based on the same star schema, especially when changes are made to the base tables. Metadata inconsistency can cause performance problems as well as problems for other tools depending on this data to produce correct results.

As the Optimization Advisor is using the cube definition to optimize for some of the query types, for example, drill through, a good practice would be to create at least one cube as a subset of the cube model.

A cube model can be easily created using the OLAP Center graphical interface. It can also be imported via bridges from different partners tools (such as MetaStage, Meta Integration, DB2 OLAP Integration Server, and so on).

4.4.2 Define referential integrity or informational constraints

The purpose of defining referential integrity constraints is to guarantee that the table relationships are maintained, and the data entry rules are followed.

In addition to the foregoing reasons, this also allows the DB2 optimizer to exploit knowledge of these special relationships to process queries more efficiently. If the Referential Integrity constraints can be guaranteed by the application and you do not want to incur the overhead of maintaining the constraints, consider using informational constraints. Informational constraints are constraint rules that can be used by the DB2 optimizer but are not enforced by the database manager. This permits queries to benefit from improved performance without incurring the overhead of referential constraints during data maintenance.

The DB2 Cube View Optimization Advisor requires:

1. Primary key definition on dimension tables.
2. Foreign key definition on the Fact table.
3. Referential integrity constraint (can be either ENFORCED or NOT ENFORCED).

The DB2 Cube Views Quick Start wizard used to create the cube model also requires referential integrity.

The DB2 Optimizer (specifically for MQTs) only requires referential integrity for query rewrite purposes in the following situation:

- When the SQL statement being processed has less tables than the SQL statement used to create the MQT.

For example, consider that the MQT was created using the SQL statement in Example 4-2.

Example 4-2 MQT creation

```
CREATE mymqt  as (
SELECT SUM(f.sales_amount) as sales_amount, s.store_name, p.product_name,
t.quarter_desc, t.month_name, y.scenario_name
FROM FACT_TABLE f,
        PRODUCT p,
 STORE s,
 TIME t,
SCENARIO y
WHERE f.product_id=p.product_id and
f.store_id=s.store_id and
f.time_id=t.time_id and
f.scenario_id=y.scenario_id
GROUP BY s.store_name, p.product_name, t.quarter_desc, t.month_name,
y.scenario_name)
DATA INITIALLY DEFERRED
REFRESH DEFERRED
ENABLE QUERY OPTIMIZATION
MAINTAINED BY SYSTEM
NOT LOGGED INITIALLY;
```

The DB2 optimizer for the query in Example 4-3 does require referential integrity, because it is not using *all* tables defined on the MQT.

Example 4-3 Query that requires referential integrity

```
SELECT     SUM(f.sales_amount) as sales_amount, s.store_name,
       t.quarter_desc, t.month_name, y.scenario_name
```

```
FROM FACT_TABLE f,
STORE s,
TIME t,
SCENARIO y
WHERE
f.store_id=s.store_id and
f.time_id=t.time_id and
f.scenario_id=y.scenario_id
GROUP BY s.store_name, t.quarter_desc, t.month_name, y.scenario_name
```

The DB2 optimizer for the query in Example 4-4 does not require referential Integrity, because it is using all tables defined on the MQT.

Example 4-4 Query that does not requires referential integrity

```
SELECT  SUM(f.sales_amount) as sales_amount, s.store_name, p.product_name,
       t.quarter_desc, t.month_name, y.scenario_name
FROMFACT_TABLE f,
PRODUCT p,
STORE s,
TIME t,
SCENARIO y
WHERE f.product_id=p.product_id and
   f.store_id=s.store_id and
   f.time_id=t.time_id and
f.scenario_id=y.scenario_id
GROUP BY s.store_name, p.product_name, t.quarter_desc, t.month_name,
y.scenario_name
```

Since referential integrity is required by DB2 Cube Views as well it is recommended as a best practice implementation on star schema and snowflake model, you certainly need to implement it in your application in order to take benefit of using the DB2 Cube Views Optimization Advisor.

DB2 V8.1 introduced informational constraints, and they may be used as well.

> **Attention:** If you use informational constraints, you must ensure that the data is in fact accurately adhering to the constraints you have described. Otherwise, you could get different results from the base tables than from the MQT.

In addition to the foregoing examples, also consider the star schema example in Figure 4-2.

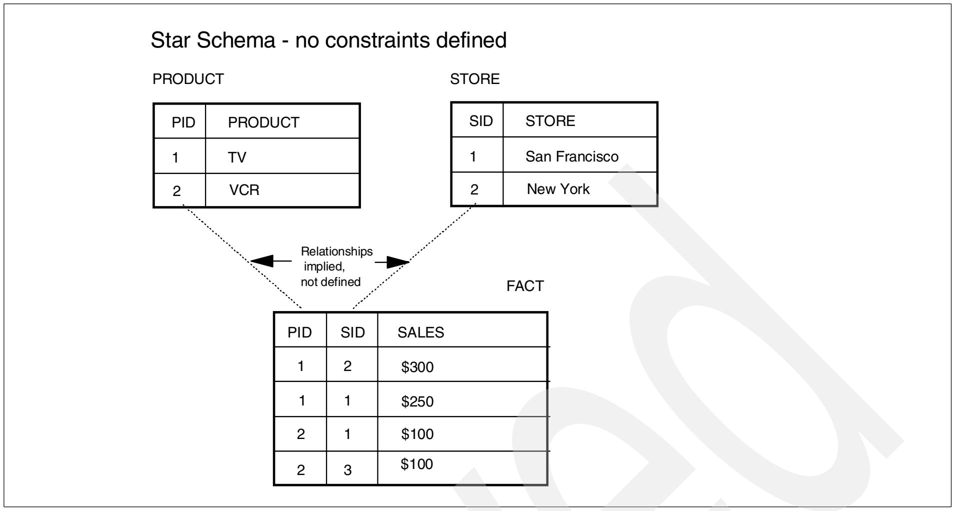


Figure 4-2 Star schema example

Primary keys and foreign keys have not been defined for referential integrity, and we create the MQT in Figure 4-3.

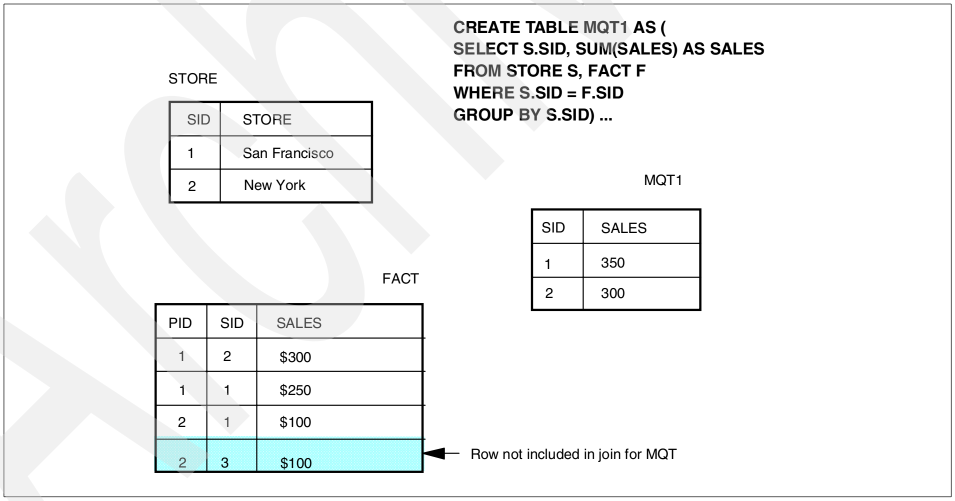


Figure 4-3 Create the MQT

As you can see on Figure 4-4, DB2 will not include the non-matching rows on the MQT and different queries can generate different results such as **`Select SUM (sales) from Fact`** result in a different result compared to a **`Select SUM (sales) from (MQT1)`**, even if the two queries are semantically different.

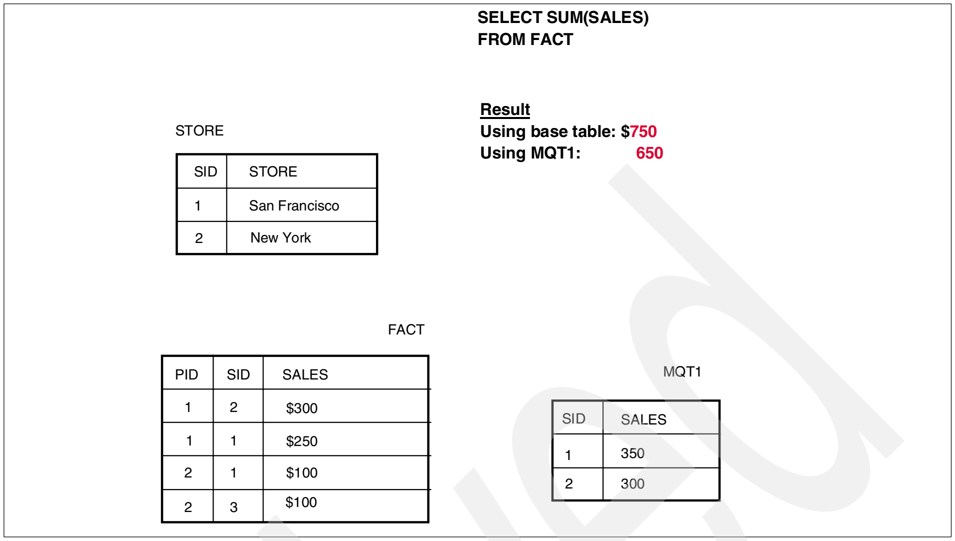


Figure 4-4 Run the query

Without referential integrity defined on the base tables or informational constraints, DB2 cannot guarantee the results are the same. In order to avoid creating MQTs that will not be used or will only be used in specific, infrequent cases, DB2 Cube Views enforces the requirement that you create constraints.

Example 4-5 is an example of SQL to check the validity of an informational constraint in our previous example.

Example 4-5 Checking the validity on an informational constraint

```
SELECT LID, SALES FROM FACT WHERE LID NOT IN
(SELECT LID FROM LOCATION)
```

Similar issues exist with NULLS as with constraints, and for that reason DB2 Cube Views requires that foreign keys be created as non-nullable. If a foreign key is nullable, DB2 assumes that it could contain NULLS. If all foreign keys are nullable, an MQT will only be used if the joins in the MQT exactly match with the joins in the query. In this case, many MQTs would be created in order to optimize the model. Therefore, DB2 Cube Views requires non-nullable foreign keys to avoid an explosion of the number of MQTs.

You must define referential integrity or informational constraints on the base tables before you can use the Optimization Advisor. These constraints need to enforce the base rules as well as cube model completeness and optimization rules.

In this section, however, we will expand on the optimization rules and, as an example, apply these rules on a basic star schema. Note that optimization rules further extend the base rules and cube model completeness rules and ensure that the SQL queries created for the metadata can be optimized successfully.

These are the optimization rules for DB2 Cube Views:

- **Join optimization rules:**
  - A referential integrity constraint must be defined for the columns that participate in a join. For example, columns involved in facts-to-dimension joins and if applicable dimension-to-dimension joins used in a snowflake schema need constraints. A primary key constraint must be defined on one side and a foreign key which references the primary key must be defined on the other side of the join.
  - You can use informational constraints for the foreign key constraints. Informational constraints provide a way to improve query performance without increasing maintenance costs. These constraints can be used by the DB2 SQL compiler but are not enforced by the database manager. This type of constraint allows DB2 to know about the relationships in the data without requiring the relationship to be enforced.

    Note: If you use informational constraints, you must ensure that the data is in fact accurately adhering to the constraints you have described.

  - The join cardinality must be 1:1, Many:1, or 1:Many.
  - All attributes used in the join must resolve to non nullable SQL expressions. For example, if the foreign keys are nullable, a summary table will only be used if the joins in the summary table exactly match the joins in the query. This is because DB2 will not use a summary table if it determines based on the constraint that it is possible for the results in the summary table to differ from the results from the base tables. So, having a nullable foreign key would result in creating inefficient summary tables.
  - The join type must be an inner join for summary tables optimization purposes.
- **Dimension optimization rule:**
  - A dimension must have one primary table to which joins attach with a 1:1 or Many:1 cardinality.
- **Cube model optimization rule:**
  - The join used to join the facts and dimension must have a cardinality of 1:1 or Many:1 and must join a facts table to a dimension's primary table.

In order to enforce the optimization rules on this star schema, you need to define constraints on each of the facts-to-dimension joins as shown in Figure 4-5. Several rules define each of these joins. You can use informational constraints only for foreign key constraints.

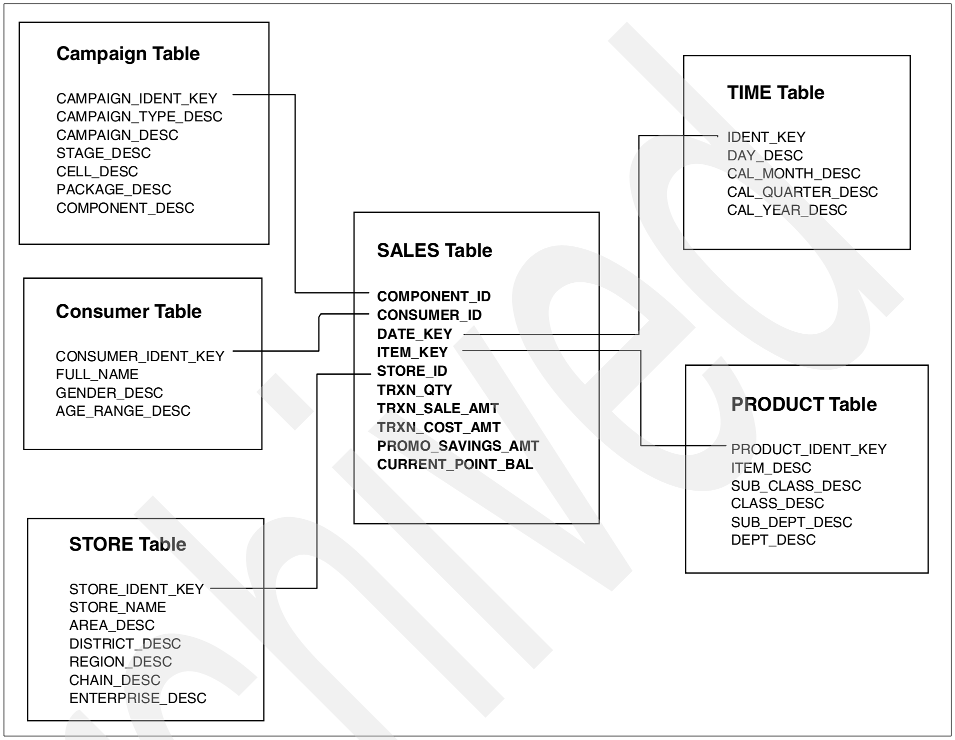


Figure 4-5 Sample star schema

For example, for the join between the *Product* and *SALES* tables, you must define constraints for the following rules:

- *Product_Ident_Key* is the primary key in the *Product* table.
- *Product.Product_Ident_Key* and *SALES.Item_Key* are both non-nullable columns.
- *SALES.Item_Key* is a foreign key referencing *Product.Product_Ident_Key*.
- The join cardinality is 1:Many (*Product.Product_Ident_Key*: *SALES.Item_Key*).
- The join type is inner join if summary tables optimization needed.

If you have a snowflake schema, you need to define additional constraints between the dimension tables. In a snowflake schema, each dimension has a primary table, to which one or more additional dimension tables join. The primary dimension table is the only table that can join to the fact table.

4.4.3 Do you know or have an idea of the query type?

In the following we will take a tour of the four query types for which the Optimization Advisor optimizes:

- Drill down
- Report
- Extract
- Drill through

The examples supplied are from the *Sales* cube model and the lines denote the portions of a cube model that each query type accesses.

Drill down queries

Drill down queries usually access a subset of data starting at the top of the cube model and then further down to the lower levels. These queries mostly concentrate on the top of a cube model but they can go down to any level in the cube model. When users drill down deeper in one dimension, they are typically at a higher level in other dimensions.

For example, in Figure 4-6, a user might start by accessing the sale value for all stores of all products for the year 2002. Then the user can move deeper into the data by querying for sales by quarter in all stores for all products. Performance is usually very important for these types of queries because they are issued real-time.

For the drill down query type, the Optimization Advisor optimizes based on the cube model and not on the cubes defined on the model. The Optimization Advisor recommends summary tables that aggregate data at the top of the cube model. Using the Optimization Advisor for optimizing for the drill down query type will benefit queries that access the top levels of the cube model.

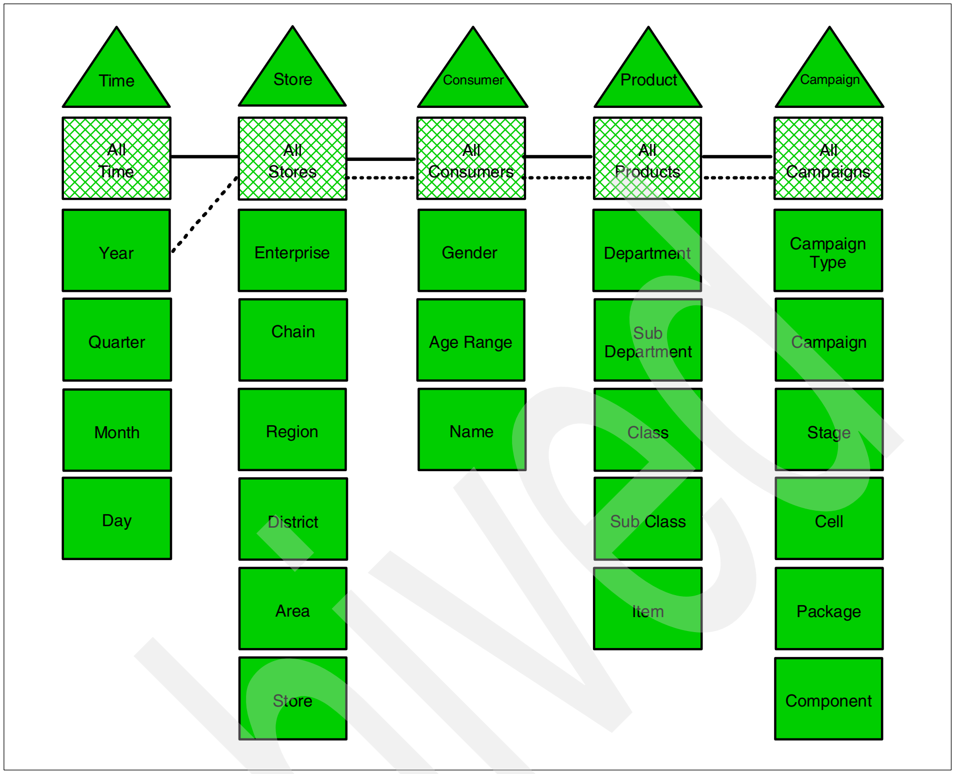


Figure 4-6 Drill down

Accessing the top level data without these summary tables in place will require repeated queries and numerous computations to be done on the base data. With the summary tables that pre-compute the aggregations at the top level, there will be considerable performance improvement.

Report queries

Report queries can hit anywhere in the cube model, but they usually tend to favor the top and middle of the hierarchies. For example, as depicted in Figure 4-7, a user might access the sale value of each item for all stores for the month January 2002. Then the user might access the sale value for each store area by product class for each month in the year 2002.

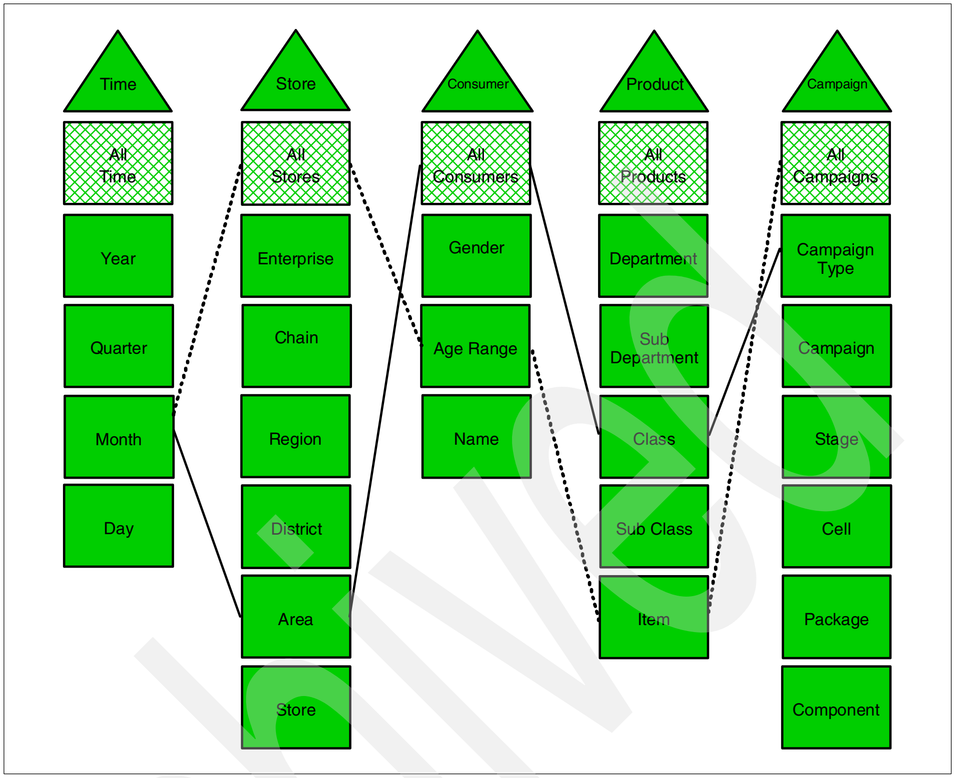


Figure 4-7 Report

For the report query type, the Optimization Advisor optimizes based on the cube model and not on the cubes defined under the model. The Optimization Advisor recommends summary tables that aggregate data from the top, down towards the middle of the cube model. Query performance is usually not as critical for report queries as for drill down queries because a user is less likely to be waiting for an immediate response to each individual query. If optimization for many query types will be required and space is at a premium, you should consider the inclusion of report optimization last.

Extract queries

Extract queries access only the base level of a cube defined for the cube model and are used to load data into a Multidimensional OLAP (MOLAP) data store. Data aggregated to the base level of the cube is loaded into a MOLAP application for further processing. For example, the Quarter-Chain-Age Range-Sub Department-Campaign Type in Figure 4-8 represents the base level of a cube defined for the cube model.

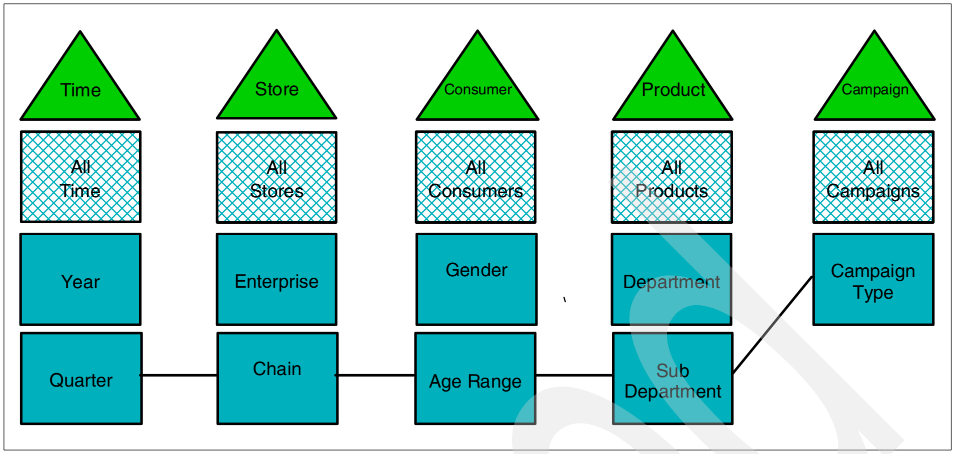


Figure 4-8 Extract

Extract query optimization is based on the bottom slice of the cubes defined for the cube model. Performance improvements will vary depending on how close the base level of the cube is to the bottom of the cube model. The higher the slice is on the cube model, the higher the expected performance improvements are. Accessing the higher level data without these summary tables in place will require repeated and costly queries to get the base data for the cube. With the summary tables that pre-compute the aggregations at the base level of the cube, there will be a lot of performance improvement.

The cube defined on the cube model should logically map to the MOLAP cube to which you want to load the data. Theoretically, there will be an MQT generated for each cube defined against the cube model.

Consider having a MOLAP outline (Example 4-6) which maps to the cube in Figure 4-8.

Example 4-6 The MOLAP outline

```
Accounts (Accounts Dimension)
   Profit (+)
   CURRENT_POINT_BAL (+)
   MAIN_TENDER_AMT (+)
   MAIN_TNDR_CURR_AMT (+)
   PROMO_SAVINGS_AMT (+)
   PROMO_SAVINGS_PTS (+)
   TOTAL_POINT_CHANGE (+)
   TRXN_COST_AMT (+)
   TRXN_SALE_AMT (+)
   TRXN_SALE_QTY (+)
```

```
   TRXN_SAVINGS_AMT (+)
   TRXN_SAVINGS_PTS (+)
   CONSUMER_QTY (+)
   ITEM_QTY (+)
   TRXN_QTY (+)
   Ratios (~)
      Profit% (~) "Profit" % "TRXN_SALE_AMT";
      Promo% (~) "PROMO_SAVINGS_AMT" % "TRXN_SALE_AMT";
DATE (+)
   1998 (+)
      Fourth Quarter 1998 (+)
      Second Quarter 1998 (+)
   1999 (+)
CAMPAIGN (+)
   New Product Introduction (+)
   New Store Opening (+)
CONSUMER (+)
   Female (+)
      less than 19 (+)
      19-25 (+)
   Male (+)
PRODUCT (+)
   BODYCARE (+)
      HAIRCARE (+)
      SKINCARE (+)
   HOMECARE (+)
STORE (+)
Enterprise (+)
   Chain Retail Market (+)
```

The extract query for the MOLAP cube in Example 4-6 requires the data at the base level of the cube as given in Figure 4-8. The aggregation for the higher levels in the MOLAP cube will be performed by the MOLAP application based on the base level data.

Drill through queries

Drill through queries are queries that access the relational database when they go below the MOLAP line. For drill through queries, the cubes defined for a cube model logically map to hybrid cubes that allow a user to access MOLAP data and the lower-level data that remains in the relational database. For example, the Quarter-Chain-Age Range-Sub Department-Campaign Type slice in Figure 4-9 represents the base slice of the MOLAP cube as in Example 4-6. The Year-All Stores-Name-Sub Department-All Campaigns slice in Figure 4-9 illustrates that the query can drill past the bottom of the cube into relational data.

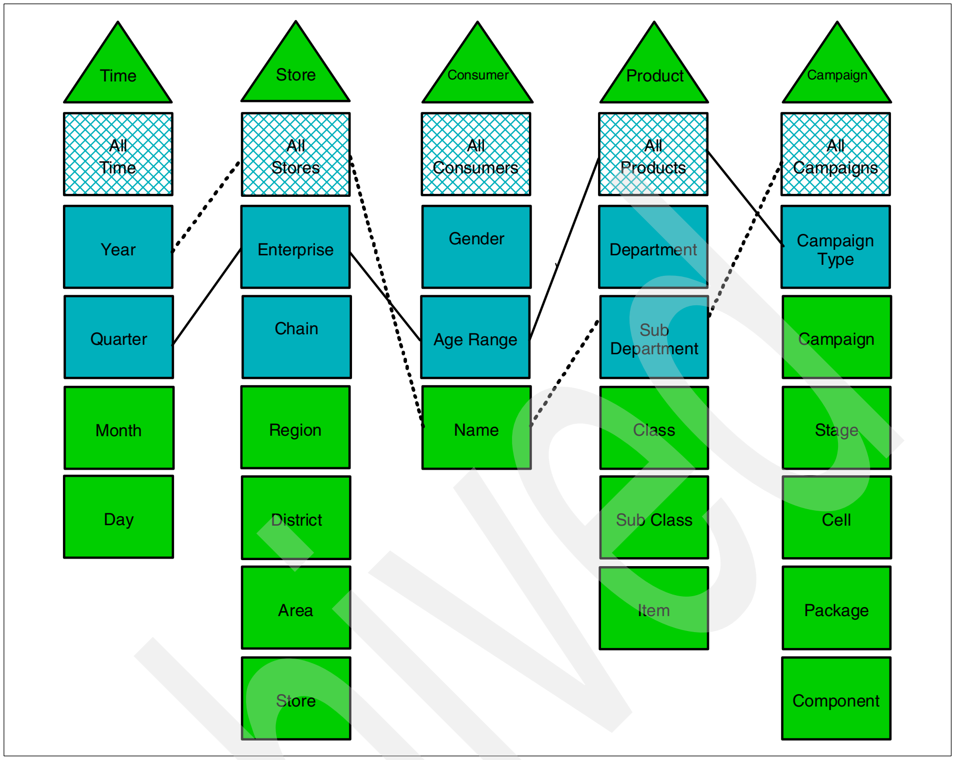


Figure 4-9 Drill through

Drill through query optimization is based on the cubes defined for the cube model. The Optimization Advisor recommends summary tables that aggregate data for a few levels at the bottom of the cube. The level of aggregation is based on the disk space availability. Using the Optimization Advisor for optimizing for drill through query type will benefit queries that frequently access the relational data below the MOLAP cube.

Combining query types

When combining query type specifications, the Optimization Advisor will interpret the specifications by the following rules:

1. The general rule is that, in this order, the earlier type subsumes the later:
   a. Drill down
   b. Report
   c. Drill through
   d. Extract

2. The only exceptions to rule 1 are when you have either:
   a. Drill down + extract, or
   b. Report + extract

   In both cases, both optimizations will take place.

What this means, for example, is that if drill through and extract both are specified, only drill through will be done. If drill down and drill through are selected, only drill down will be done.

> **Important:** Selecting combinations of query types will **NOT** result in MQTs being built for each query type. The query types are presented in the Optimization Advisor in a prioritized list with the highest priority first. The higher priority query type subsumes the lower priority query type. There are some exceptions. See the foregoing text for details on the prioritization rules.

4.4.4 Understand how Optimization Advisor uses cube model/cube

As already said, the first design recommendation is that there should only be one cube model created per star schema.

Overcoming the urge to create multiple cube models can be challenging when one faces the prospect of having to cater to diverse queries, some of which exploit the top, most aggregated, part of the cube while others go to the lowest level of granularity on one or more dimensions. Generally, however, the Optimization Advisor will provide reasonable optimization for any combination of query types specified provided that the defined cube or cubes under the cube model fairly accurately reflect the query types.

For example if you want to optimize for both *extract* and *drill through* it is generally sufficient to create a logical cube in DB2 Cube Views for each HOLAP cube or MOLAP cube and the Optimization Advisor should provide reasonable optimization of both the hybrid queries and the extract queries. The reason is that both *extract* and *drill through* optimizes for the bottom of the defined logical cube in the same way with the small difference that *drill through* includes additional slices near the bottom of the cube to the MQTs, favoring dimensions with high cardinality. However, if measured performance is still not sufficient after optimization, you can consider defining an additional cube to provide a hint to the Optimization Advisor.

Note: Care should be taken if you are making extracts (or drill through) that go to the bottom of the cube model. Creating a cube that basically is a copy of the cube model and running the Optimization Advisor on it will produce an MQT, but since the MQT will be comparable in size to the fact table, it will be most likely that the DB2 optimizer will not reroute the query to the MQT even though the MQT is aggregated on a lower level on all relevant dimensions and thus a candidate for a query rewrite. The reason for this that the cost of going to the base table will be equal to or less than going to the MQT.

Cubes are defined and created for business reasons. They satisfy the business requirements of a particular project when the cube model will represent the whole subject area. They are not required for *drill down* and *report* query patterns. For drill down, the Optimization Advisor builds aggregates of the topmost levels down attempting to reach down as far as the medium levels of the dimensions. The same goes for *report* style optimization. The difference between this optimization and the *drill down* style of optimization is that *report* style optimization does not include rollups at the top of the logical cube.

Table 4-2 summarizes the different optimization styles.

Table 4-2 Optimization Advisor query types

Optimization Advisor query type	Optimizes for
Drill down	Top and medium levels (includes rollups at the top levels)
Report	Top and medium levels
Extract	Queries at the bottom levels of the cube
Drill through	Queries at the bottom levels of the cube (includes additional low level slices)

Cubes *must* be defined when optimizing for drill through and extract query types to reflect the data that these queries will likely access. If you perform both extract and drill through queries for a particular cube model, you should build two cubes: one designed for extract queries, and a second designed for drill through queries. For a further discussion on optimization for different query type, please refer to “Define the query types” on page 160.

When considering the number of cubes to build it should be noted that the more cubes the Optimization Advisor must consider, the more MQTs will be created (when specifying query types that optimizes based on cubes). It may seem than the more MQTs we build, the better chance DB2 will have of doing query rewrites and thus improving performance, but this may not always be the case.

While DB2 will have a wider selection of MQTs to reroute a given query to and thus improve performance, it must be remembered that every dynamic SQL query going to DB2 is considered a candidate for a query rewrite. With many MQTs defined in DB2 and query rewrites being considered, the compilation of a SQL statement may take up to the order of seconds as compared to millisecond compilation times when just going to the base tables.

Therefore, it is important to consider whether the MQTs created are in fact all needed: not only because they consume disk space, but also because they extend the SQL query compilation time of all queries going to the database.

The Optimization Advisor depends in very many ways on the metadata supplied to it. Therefore, it is also important to know that you can improve the recommended summary tables by selecting which measures and attribute relationships you want to include in a particular cube model.

MQTs and measures in the cube model

Each measure defined for the cube model is included as a column in the summary table. This means that the Optimization Advisor does not allow you to choose to optimize for a subset of measures; all measures are automatically included. If you have a large number of measures, your summary table might require a large amount of disk space. If the MQT becomes too wide, the Optimization Advisor will automatically size the MQT to fit the page size.

If you are very limited on disk space, you might choose to include only critical measures in your cube model and drop any measures that you do not expect to use regularly. Excluding measures from the cube model may, however, have an impact on front-end tools, so this action should be chosen judiciously.

It should be noted that distributive measures (like `SUM` or `COUNT`) are handled very well when you do rollups, while non-distributive measures will have to be calculated from the bottom of the cube and up. This means that even though you have an MQT at a lower level in the hierarchy of the cube, it can not be used if the non-distributive measures in the MQT are not aggregated to the exact level needed by the query. From an MQT point of view, what this means is that in a space constrained environment, you should first seek to eliminate non-distributive measures from the cube model before looking at the distributive.

MQTs and attribute relationships in the cube model

Like measures, all attributes involved in attribute relationship in the cube model are included as columns in the summary table. Including attribute relationships provides the benefit of being able to group query results on these items, but at the cost of disk space. Again, if limited disk space is a concern, you might choose to drop all or some of the attribute relationships for your cube model.

4.5 Using the Optimization Advisor

Basically, the Optimization Advisor is designed to create MQTs based on the cube model, the defined cubes, the size and analysis time constraints, and the base tables and their statistics.

Important: The value proposition of the Optimization Advisor is that the DBA will only have to decide whether to use MQTs or not. Once the decision is made, the Optimization Advisor will build the MQTs without further effort.

This contrasts to earlier times where the MQTs would have to be built by hand, often without the expected performance improvement or at the cost of many hours of analysis and reiterative efforts.

In the following sections we'll take you through the necessary steps of using the Optimization Advisor.

4.5.1 How does the wizard work

As discussed earlier, the Optimization Advisor provided with DB2 Cube Views can assist and guide you to improve the performance of queries issued against DB2 by creating appropriate summary tables and summary table indexes. A summary table is a special type of materialized query table (MQT) that specifically includes summary data.

The Optimization Advisor thus analyzes the information that you provide to the wizard along with the metadata of the cube model, the DB2 statistics, and any data sampling of the base tables that you allow (given a time constraint). The result is a recommendation of which summary tables and summary table indexes should be created. The choices that you provide for each parameter in the wizard affect the summary tables the wizard recommends and ultimately the performance that you gain.

In general, the Optimization Advisor will try to limit the recommendation to the smallest number of summary tables possible while seeking to avoid impacting the resulting performance of the queries.

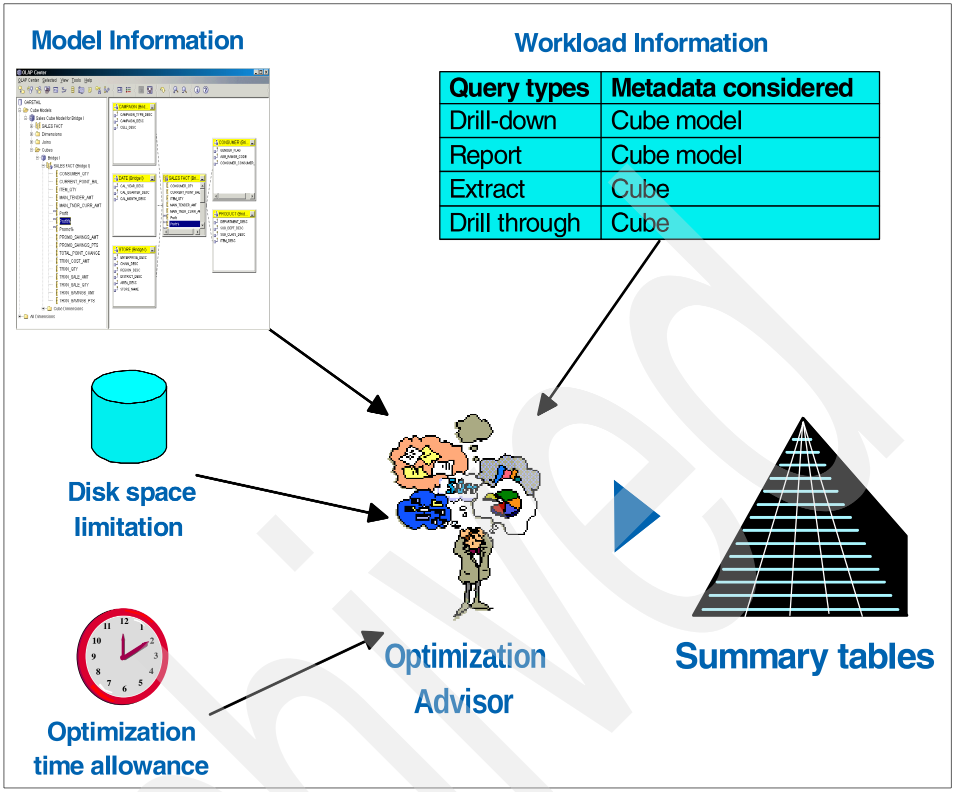


Figure 4-10 Optimization Advisor wizard

The following list describes the information that you must provide to the wizard:

- The type of queries expected on the cube model: This helps the Optimization Advisor understand which portions of the cube model are queried most frequently.
- Disk space limitations: This helps the Optimization Advisor to recommend summary tables that in aggregate do not exceed the maximum allowable disk space.
- Time limitations: This is the maximum amount of time that the Optimization Advisor can use to determine sample the base tables and produce a recommendation.
- If you allow data sampling, the Optimization Advisor examines the base data (fact and dimensions) to get an estimate of how big a given grouping of data would be.

In addition to the information provided to the wizard, the Optimization Advisor analyzes the following information to create recommendations:

- Cube model metadata, which includes the cube model, the cubes defined based on the cube model, the measures, the attribute relationships, etc.
- DB2 statistics, including the number of records, number of pages and average record length.
- Data sampling information (if you allow data sampling, which we recommend highly), which includes overall trends and exceptions in the data. This is also known as data sparsity.

Collectively, the information helps the Optimization Advisor in determining the most appropriate summary tables for any given cube model and related cubes.

As a result, the Optimization Advisor produces two SQL files that can create and refresh the recommended summary tables. If you choose you can change the files, but generally it is recommended to run them unchanged.

4.5.2 Check your cube model

Note: Before optimizing a cube model using the Optimization Advisor, you must have DB2 constraints specified for the base tables used in the cube model. Constraints must be specified between each fact table and dimension table and between each dimension table in a snowflake schema. The constraints must be specified on nonnullable columns.

The Optimization Advisor wizard is launched from the OLAP Center. There are two ways to invoke this; one is illustrated in Figure 4-13. All screens used to perform this task are shown to enable you to see all of the options that are available through the wizard.

Consider the view of the Sales cube model in Figure 4-11 as displayed in the OLAP Center.

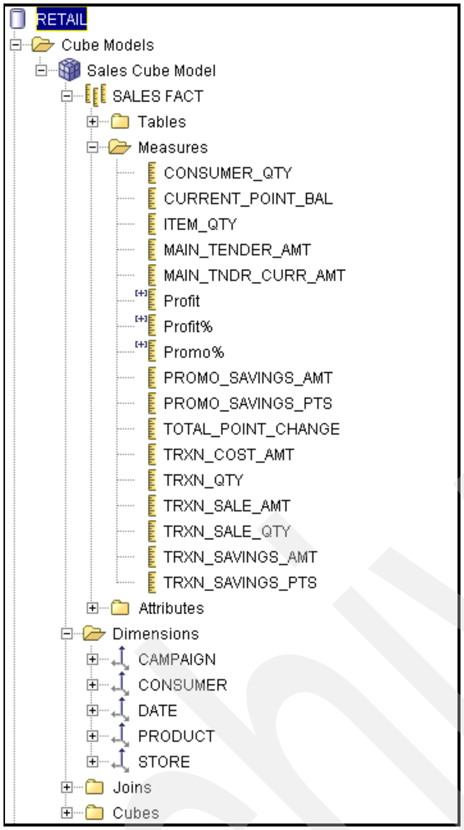


Figure 4-11 Sales cube model

The *Sales* cube model is defined with 5 dimensions: *CAMPAIGN*, *CONSUMER*, *PRODUCT*, *DATE* and *STORE*. *SALES FACT* is the fact table. Refer to the Appendix for the columns in the fact and dimension tables. The joins between the fact table and the dimension tables are defined as in "Define referential integrity or informational constraints" on page 136.

A Sales cube is defined based on the Sales cube model seen in Figure 4-11 with the same 5 dimensions, except that the hierarchies are a subset of the cube model hierarchies. Figure 4-12 shows the view of the Sales Cube as displayed in the OLAP Center.

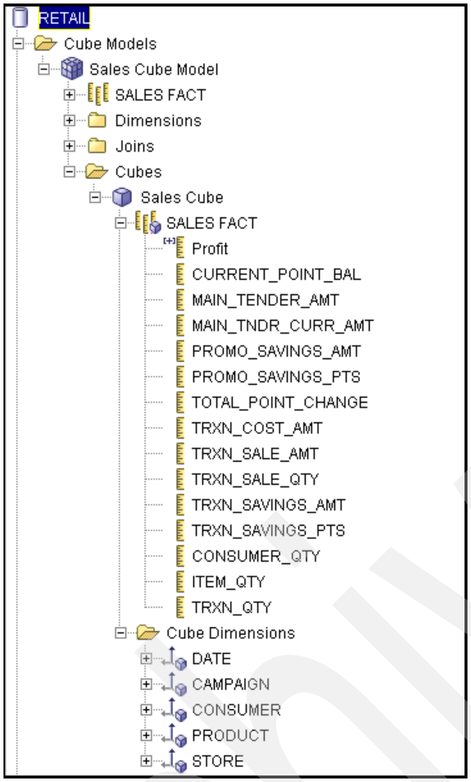


Figure 4-12 Sales Cube

It is not our intention to walk you through the cube model and cubes, but it is important that you have determined that the basic cube model and cubes are in place before you start the optimization process.

4.5.3 Run the Optimization Advisor

For the sample scenario, let us say we want to optimize the Sales Cube model for drill through type of queries.

The next two things we need to plan for before running the Optimization Advisor wizard are:

- How much disk space we have for the summary tables and summary table indexes
- How much time we have to generate the recommendations

These are important because the Optimization Advisor will try to create recommendations that is the most appropriate within the constraints that you have. For our scenario, assume we have no time limit and have a disk space limitation of 8 Gigabytes.

In the following sections we will go through the Optimization Advisor wizard for the sample scenario and analyze the Summary table recommendations.

Here are the steps to optimize a cube model:

1. Launch the OLAP Center.
2. In the OLAP Center object tree, select the cube model that you want to optimize, right-click the cube model, and select **Optimization Advisor.**

   You need to have the following privileges to run the Optimization Advisor:

   – SELECT for system tables and base tables

   The Optimization Advisor checks the validity of the model, and if the cube model is valid, the Optimization Advisor wizard opens (see Figure 4-13).

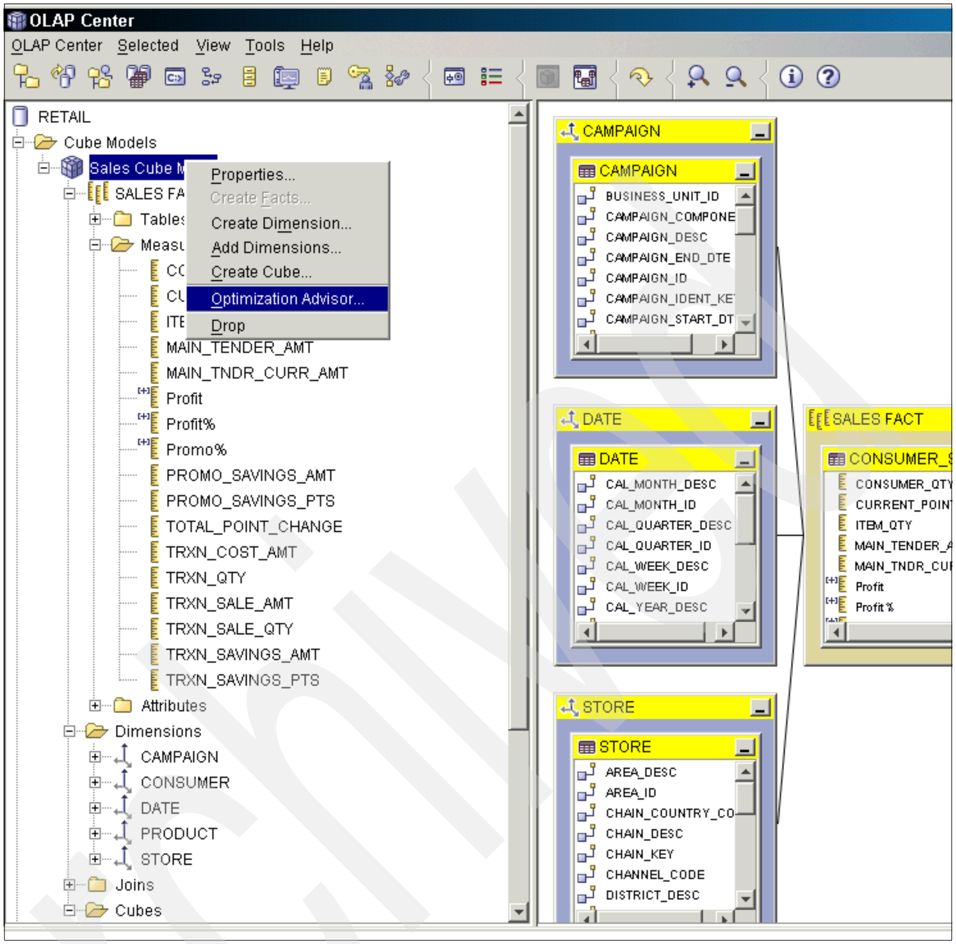


Figure 4-13 Menu selection to access Optimization Advisor

3. Specify the parameters for the Optimization Advisor wizard:

Table 4-3 Parameters for Optimization Advisor wizard

Steps	Objectives	Section Reference
Define query type	Specify the type or types of queries expected to be performed most often on the cube model. The available types of queries are: Drill down, Report, Drill through, and Extract.	Refer to section "Define the query types" on page 160.
Specify limitations	Specify the available disk space for the summary tables and indexes that will be built. Specify if you want to allow data sampling. Also specify the maximum amount of time you want to allow for the Optimization Advisor to determine recommendations. The more space, information, and time that you specify, the more significantly your performance results will improve.	Refer to section "Specify disk space and time limitations" on page 161
Specify summary table creation options	Specify if you want IMMEDIATE or DEFERRED update summary tables. Specify the table spaces to store the summary tables and summary table indexes.	Refer to section "Specify tablespaces and summary table refresh method" on page 164
Specify file names to store the SQL scripts	Enter a unique file name in both the Create summary tables SQL script field and the Refresh summary tables SQL script field.	Refer to section "Specify file names to store the SQL scripts" on page 166

4. Save the recommended SQL scripts into the file names specified and close the Optimization Advisor wizard.
5. Run the SQL scripts. If you are creating large summary tables, building the summary tables might require a substantial amount of time to complete. You can use the DB2 Command Center or Command Window to run the SQL scripts. You need to have the following privileges to run the SQL scripts:
   - CREATEIN, DROPIN on schema DB2INFO
   - SELECT and ALTER (or CONTROL) on base tables

Here are the steps to run the SQL scripts from the DB2 Command Window:

a. Change to the directory where you saved the SQL scripts.

b. Connect to the database of the cube model that you optimized. For example, enter: db2 connect to RETAIL.

c. Enter the following command:

```
db2 -tvf filename
```

where filename is the name of the create summary table SQL script.

You can run the refresh summary table SQL script anytime, depending on how current you want the data in the summary table to be with the base data, to synchronize the summary tables with the change in base data.

4.5.4 Parameters for the Optimization Advisor

The types of input to the Optimization Advisor are described in this section. Each of these inputs plays an important rule in the way Optimization Advisor creates the summary table recommendations.

Define the query types

The query types describe when and how DB2 relational data is typically accessed. Select the most frequently used query types on the cube model by selecting the type or types of queries in the first page of the wizard. This helps Optimization Advisor understand which slices of the cube model are most frequently accessed. The available query types are:

- Drill down
- Report
- Extract
- Drill through

You must choose at least one query type. The Optimization Advisor optimizes based on the cube model or the cubes defined on the cube model based on the query type selected. The options, Drill down and Report, are selected by default.

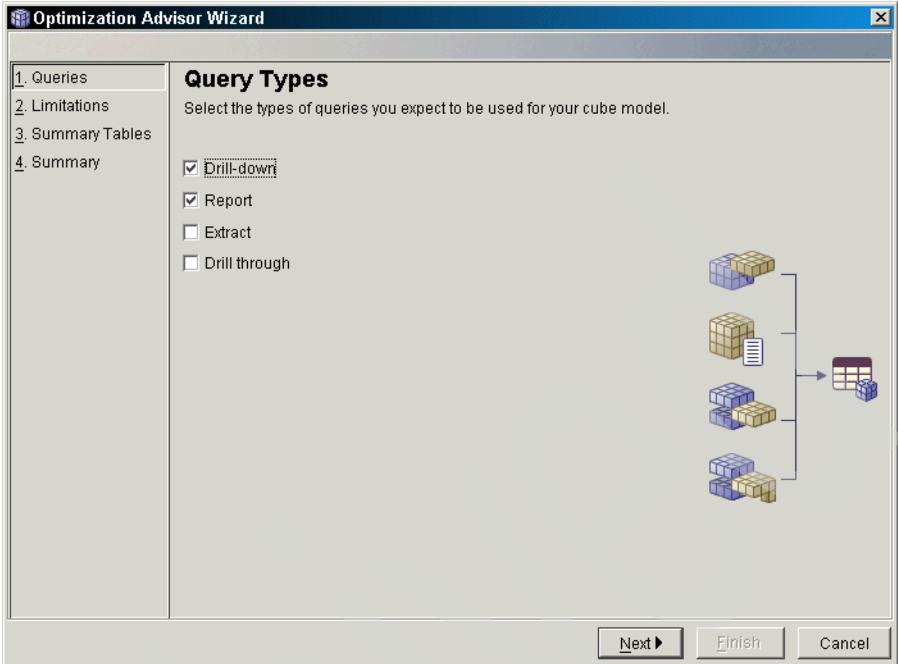


Figure 4-14 Query types

For our sample scenario, since we want to optimize for drill through type of queries, choose the option **Drill through** in this page and click **Next** to continue.

Specify disk space and time limitations

Through the Optimization Advisor, disk space and time limits can also be defined.

Disk space limitations

The amount of disk space available for the summary tables helps the Optimization Advisor in determining the summary table and index recommendations which use more or less of the specified disk space. This is important so that you don't let the Optimization Advisor make recommendations that are not suitable for your system.

The level of aggregation in the recommended summary tables is based the available disk space. The amount of disk space that you specify is directly related to the optimization results. Increasing the disk space can increase both the number of queries with improved performance and the degree of improvement. By default, the Optimization Advisor wizard chooses no limit for the available disk space.

The following factors need to be considered before specifying the available disk space:

- The query performance levels that you want
- The number of cube models that you are optimizing for
- How critical each cube model is
- How frequently each cube model is used
- The availability and cost of the disk space

Typically, you can see significant improvement by allowing a moderate amount of disk space, such as 1% to 10% of the space currently used by the relational tables that are referenced by the cube model. Table 4-4 shows the relationship between the percentage of disk space used and the corresponding expected performance improvements.

Table 4-4 Percentage disk space used - expected performance improvements.

Percentage of base tables disk space used for summary tables	Expected improvement for relevant queries
Less than 1%	Low
5%	Medium
10%	High
Unlimited	Highest

If you want to specify no limit on the disk space for summary tables, select the option **Unlimited disk space available** in the wizard (Figure 4-15). Alternatively, you can specify a disk place limit by choosing the option **Maximum disk space allowed** and specify the available disk space in MB or GB.

For our sample scenario, since we have a disk space limitation of 8 Gigabytes, choose the option Maximum disk space allowed and specify 8 and choose the option GB from the drop down list.

Data sampling and time limitations

DB2 Cube Views employs a very sophisticated sampling technique to obtain statistics on base table intersections in order to make recommendations on optimal MQTs and indexes. It takes advantage of a new function in DB2 V8.1 FixPack2, the `TABLESAMPLE` keyword on the `SELECT` statement. Because DB2's `TABLESAMPLE` only works with base tables and not with views, it is strongly recommended that you avoid using views in source data.

In the case where the Optimization Advisor is not allowed to perform sampling, DB2 Cube Views must rely on the current statistics in the catalog. Because those statistics describe individual tables and not the intersections of values among multiple tables, they will probably be quite high compared to values derived from sampling. This will result in much less efficient recommendations for MQTs.

If you allow data sampling, the Optimization Advisor will examine the data in the cube model to get more information so that it can create the most effective set of recommendations that will match the available disk space. By default, Data Sampling is selected and no limit is set for the time to do the data sampling.

The recommendations by the Optimization Advisor will be more accurate when it has more insight into the data that the cube model and the cube represent. This can be done by allowing data sampling and in particular, allowing sufficient time for sampling. Depending on the size of the base data, the time required to do the data sampling varies.

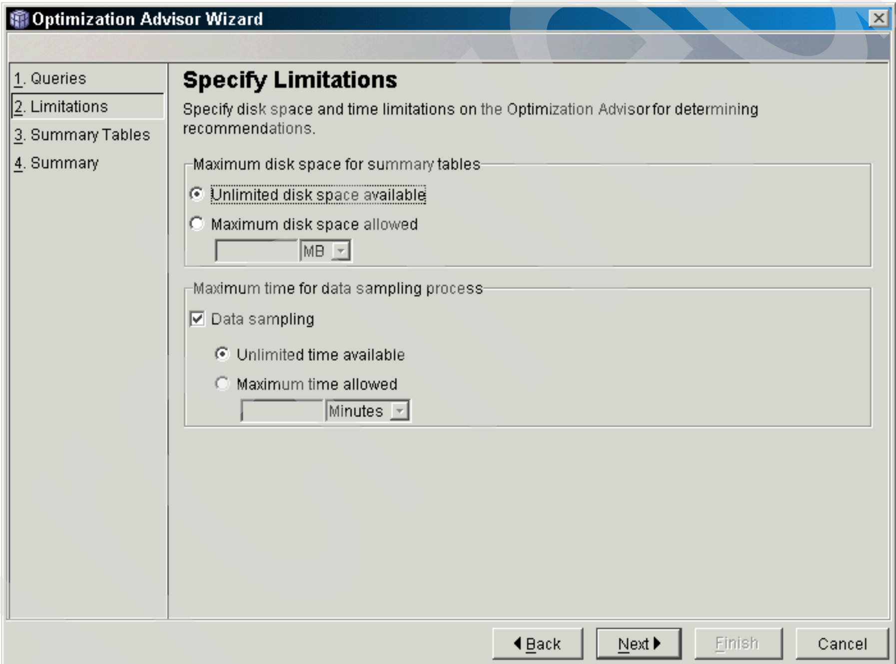


Figure 4-15 Specify Limitations

In the Optimization Advisor wizard (Figure 4-15), you can choose to allow Data Sampling by selecting the option. If you allow Data Sampling, then you can choose to specify unlimited time for the data sampling process by selecting the option **Unlimited Time Available**. Alternatively, if you want to specify a time limit for the data sampling process, select the option **Maximum Time Allowed** and specify a maximum time limit in **Minutes** or **Hours**.

For our sample scenario, since we want to allow Data Sampling with no time limitation, select the **Data Sampling** option and choose the **Unlimited Time Available** option.

The Optimizer recommendation will result in significant performance improvement if more space and time are available.

> **Restriction:** The sampling done by DB2 can only be performed on tables and not views. This means that if a fact table is composed of several tables overlaid with a view, and the view is specified in the cube model as the fact table, the sampling will fail.

Specify tablespaces and summary table refresh method

Specify the tablespaces where you want the summary tables and indexes to be created and the summary table refresh option in the Optimization Advisor wizard as shown in Figure 4-16. You need to:

- Specify the summary table update option: `DEFERRED` or `IMMEDIATE`.
- Specify the tablespace for storing the summary tables.
- Specify the tablespace for storing the summary table indexes.

Specify summary table update option

You can specify how you want the summary tables to be synchronized with the base tables. Summary tables may be refreshed every time the underlying data changes (`IMMEDIATE`) or at intervals controlled by the administrator (`DEFERRED`). It is very important to choose this option carefully, understanding the end users' requirements for currency of data in the summary tables and balancing that against the tradeoffs in terms of overhead. Refer to 4.9, "Further steps in MQT maintenance" on page 198 for more information.

If you are using non-distributive measures or `COUNT(DISTINCT)` in the measures, then the Optimization Advisor creates only summary tables with deferred update. The option `DEFERRED` is selected by default. Having nullable attributes (grouping columns) in the MQT will also cause the Optimization Advisor to recommend `REFRESH DEFERRED` MQT.

For our sample scenario, since we have non-distributive measures, choose the `DEFERRED` option.

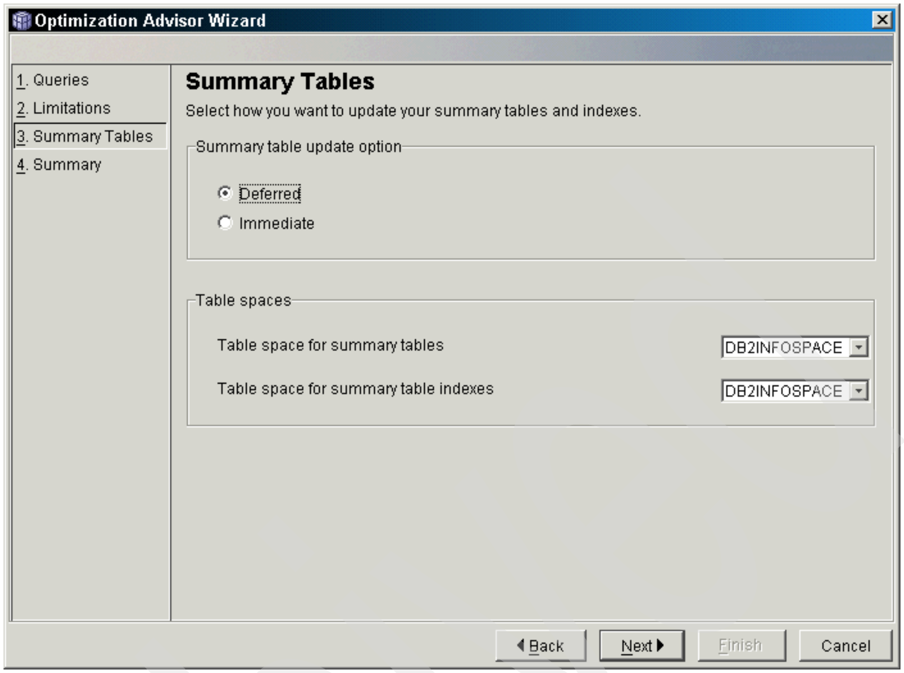


Figure 4-16 Summary tables

Specify tablespaces

You can specify different tablespaces for storing the summary tables and the summary table indexes. The tablespaces defined under the DB2 data source are listed for you to choose from. The SQL for summary tables and the indexes will refer to the selected tablespaces. The summary tables are generally wide, so it is recommended to use a tablespace with a wide page size to store the summary tables.

Click **Next** to have the wizard determine the recommendations for creating and refreshing the summary tables.This might take some time depending on the volume of data we are handling.

For our sample scenario, choose any tablespace from the drop down lists.

Specify file names to store the SQL scripts

In the summary page of the wizard in Figure 4-17, specify unique file names for the create summary table SQL and the refresh summary table SQL scripts. You can view the Create or Refresh SQL that is recommended to optimize the model by clicking the **Show SQL** button.

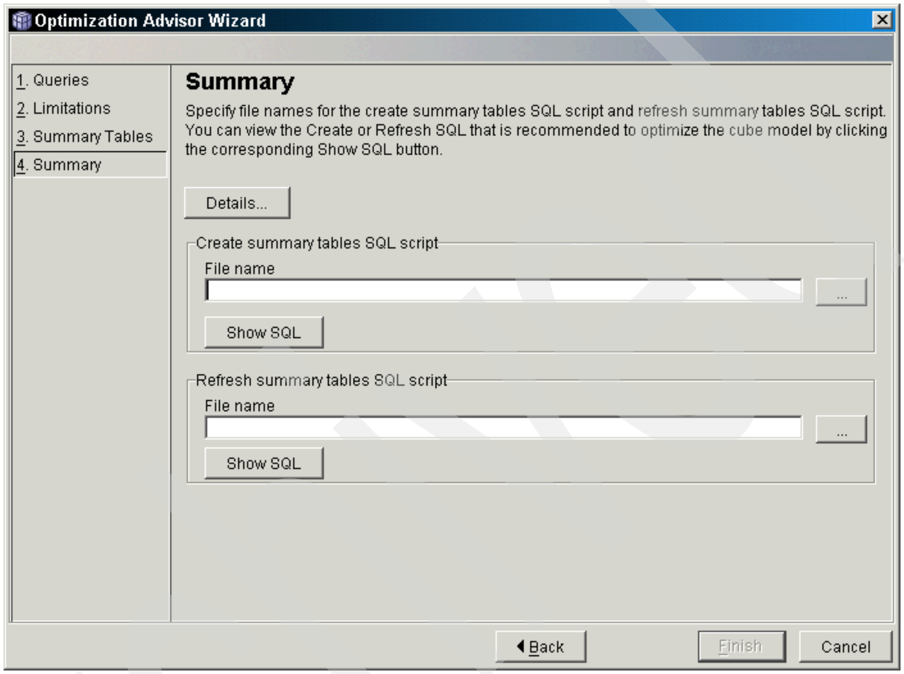


Figure 4-17 Summary

You can see more information about the recommended summary tables that the SQL will create by clicking the **Details** button. The following details will be shown:

- Expected disk space usage by summary tables — see Figure 4-18. See also Table 4-4 on page 162 for some recommendations for summary table disk space usage as a percentage of the fact table.
- The reason to use DEFERRED when IMMEDIATE is specified — see Figure 4-19. The full text of the DEFERRED refresh message is: "[OC7201] The "DB2INFO.MQT0000000002T01" recommended summary table will use DEFERRED refresh because one or more nullable attributes were found as columns in the fullselect of this recommended summary table."

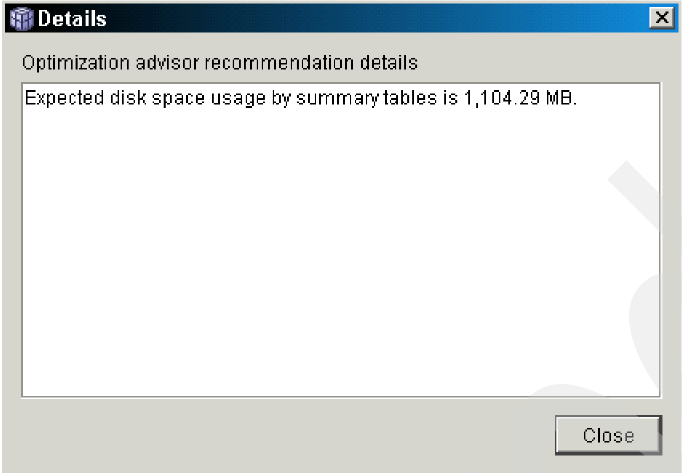


Figure 4-18   Expected disk space usage

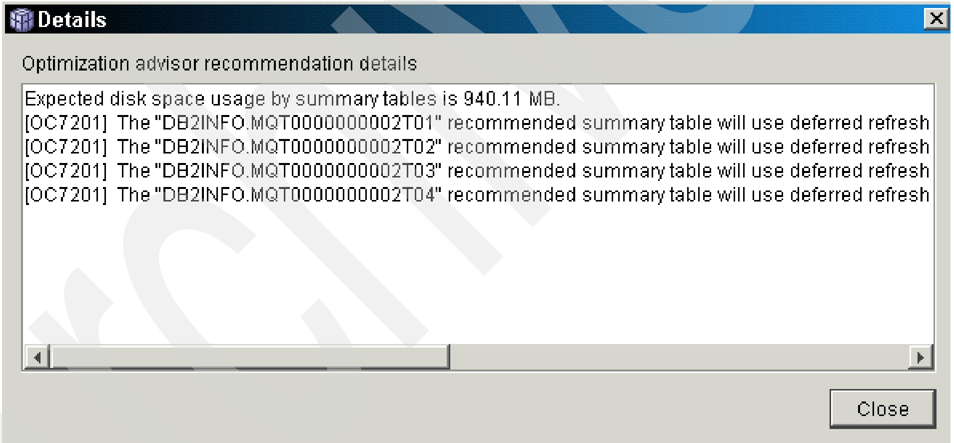


Figure 4-19   Expected disk space usage and refresh DEFERRED

The Optimization Advisor creates one or more summary tables. For summary tables created with the DEFERRED update option, the create summary table SQL and the refresh summary table SQL are the same. The DEFERRED option drops the previously created summary tables instead of applying the delta to the original data. This improves the performance.

To view information, errors, or warnings about the recommendations, click the **Details** push button. To view either SQL script, click the corresponding **Show SQL** push button.

For our sample scenario, specify the file names for the create summary table script and the refresh summary table script as c:\sales_drillthru_createmqt.sql and c:\sales_drillthru_refreshmqt.sql respectively. Click **Finish** to save the scripts and close the Optimization Advisor wizard.

All that remains to be done is to run the resulting create script in a DB2 command window: **DB2 -tvf <DDL script>**.

An example of an MQT creation script is provided in “MQT” on page 694.

It is clear that refreshing the MQTs is a fairly simple operation in this case.

> **Note:** The Optimization Advisor should be re-run periodically when the data source changes significantly in size and the MQT re-optimized.

As can be seen in Example E-1 on page 694, the MQT creation (and sometimes the refresh) scripts are quite large and will take considerable effort to create by hand. With the Optimization Advisor, this is now more or less eliminated. Moreover, since the DBA often has no query workloads to work from — for example, in cases where the datamart is being implemented and not in production yet — the Optimization Advisor will be able to provide a set of MQTs that enhance the performance of the data mart for a number of different queries based on the actual design of the database and the expected workload.

Note that having created and run the MQT creation script, a plan should be established for running the refresh script at appropriate times. In practice, this is often done right after the base tables have been updated or reloaded, for example, in a designated service window or at times with low workload. Deferring the refresh to a later time — especially in cases where the MQTs have been created with refresh DEFERRED — will introduce inconsistencies between the MQTs and the base tables if the base tables have been updated using load insert. This situation should be considered carefully before it becomes practice. Please refer to “MQTs: a quick overview” on page 131 for a further discussion on this subject.

After creating MQTs, we need to consider what activities should follow. This is covered in the next section.

4.6 Deploying Optimization Advisor MQTs

Performance optimization is a large, diverse, and oftentimes complicated subject which the Optimization Advisor in many ways makes easy to manage. It is, however, an ongoing process to adjust your database to changing requirements and workloads, and here we propose a basic methodology to deal with these issues — especially related to MQTs. We also attempt to visit some of the more important technical details related to MQTs.

The steps described in Figure 4-20 propose a method for creating MQTs and keeping them aligned with the workload on the database.

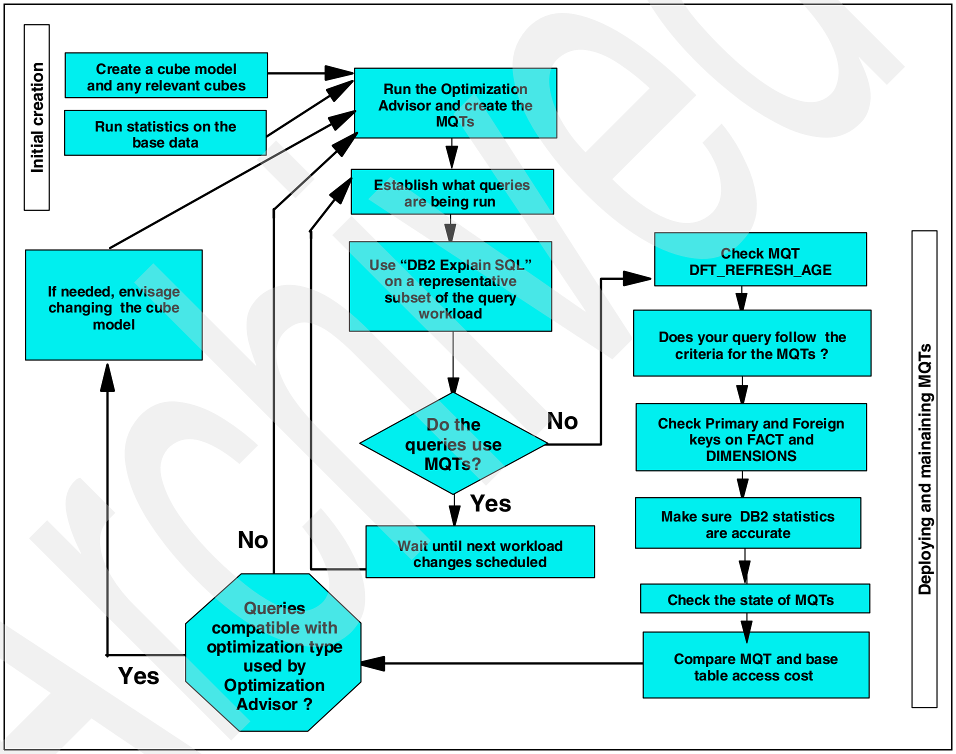


Figure 4-20 MQTs implementation steps process

The process depicted in Figure 4-20 basically shows what general steps to take when reviewing the MQT implementation, including the initial creation at the MQTs and the iterative process of maintaining MQTs.

The process as depicted is simplified and thus only includes steps that directly pertain to the creation and maintenance of the MQTs. This means that normal table maintenance such as reorganization and collecting statistics is not included, other than the initial statistics creation, as can be seen in the upper left-hand corner of the figure.

Prior to going into full production and during the initial creation of the MQTs, we suggest, if possible, that you start with a subset of the data for the base tables and run an iteration or two across the reduced data. This assumes that a meaningful query workload can be obtained at this stage. The advantage of this approach is that you can create the MQTs fairly quickly and determine whether they perform as expected or whether changes are required.

If we look back at Figure 4-20 on page 169, we can transform this into a little more detailed checklist:

1. Create a subset of the data for your base tables. This will initially reduce loading and refresh times of the base tables and MQTs as well as reducing any query times when creating the query workload.
2. Create a cube model and any relevant cubes. Obviously this is needed, as the Optimization Advisor depends on the metadata to suggest MQTs.
3. Run statistics on the base tables. This is especially important prior to running the Optimization Advisor and creating the MQTs for the first time, since without the statistics this process can be prolonged significantly as well as produce suboptimal MQTs.
4. Use the Optimization Advisor to create the MQTs.
5. Use DB2 Explain SQL on a representative subset of the query workload. The purpose of this is to see whether the query workload has changed since the MQTs were created or the assumptions about the workload were incorrect.
6. Validate if the query is using MQTs:
   a. Make sure that `DFT_REFRESH_AGE` is configured for the type of MQT being used (**0** for Immediate and **Any** for Deferred/Immediate)
   b. Make sure that the query matches the requirements to use the MQT. (More details are given in *DB2 UDB's High Function Business Intelligence in e-business*, SG24-6546, in section 2.7, “Materialized view matching considerations”.)
   c. Check that primary keys and foreign keys are defined on dimensions and fact tables.
   d. Check that statistics are updated on MQT and base tables and run them if not.
   e. Make sure that the MQTs are not in *check pending state.*

   f. Make sure that the cost of accessing the base table is not cheaper than accessing the MQT. For example, queries with predicates that use an index on the base table might be cheaper to perform compared to a tablespace scan on the MQT.
7. Check that most of the queries in the query workload are of the same type and match with the optimization type used when the Optimization Advisor was run:
   a. If not, analyze the different queries in the query workload and rerun Optimization Advisor
   b. If yes, eventually envisage changing the cube model and cube definition to better reflect the query workload. This should be done judiciously and always with the business and whole query workload in mind. For example, overly extending the hierarchies in the cubes may result in MQTs aggregated at a too low level possibly leading the DB2 optimizer to use the base tables because it is less expensive than accessing the excessively large MQTs.
8. In full production when query workload changes, revisit points (5) to (7) regularly.

Going through the list above, some tools might come in handy, as shown in Table 4-5.

Table 4-5 Implementation tools

Question	Tool to use
What queries are currently being run?	DB2 Snapshot monitor
Is the query using an MQT?	DB2 Control Center: Right-click the database and select Explain SQL or make sure the Explain tables are created and use the EXPLAIN statement in a DB2 Command Window.
How deep down in the hierarchies do the MQTs go?	Save your output from the Optimization Advisor or use the DB2 Control Center: right-click the MQT table and select Generate DDL
Are the DB2 parameters correctly set?	DB2 Command Window and DB2 Control Center

In the following sections we will elaborate and expand on the toolbox provided by DB2 to help monitoring MQTs.

4.6.1 What SQL statements are being run?

After the Optimization Advisor has been run and the MQTs have been created, we should check to make sure that DB2 is actually using them. The first step in this process is to determine what SQL is being run. Most of the front-end Business Intelligence and reporting tools have facilities for displaying and/or saving the SQL submitted to DB2. Retrieving the SQL from these tools may not be practical, however — especially in the cases where they are only available from the user workstations.

A way for the DBA of making certain that the SQL is stored for analysis is by capturing dynamic SQL using DB2's Snapshot Monitor. The DB2 Snapshot Monitor captures both the statement text and the statistics pertaining to the statements, including number of executions, number of rows read and updated, and the execution times. This will provide enough information to the DBA for further analysis into the types and frequency of the queries as well as the statements themselves for access path analysis. Note that the statements captured by the DB2 Snapshot Monitor are those originally submitted to DB2. They do not reflect any potential rewrite by the optimizer.

To get the statements from DB2, you attach to the instance and query the DB2 Snapshot Monitor:

```
ATTACH TO <instance> USER <userid> USING <userid>
GET SNAPSHOT FOR DYNAMIC SQL ON <database name>
```

This will provide point-in-time information from the SQL statement cache for the database.

You can also use the new TABLE function to specify the elements of interest from the snapshot. For example:

```
SELECT SNAPSHOT_TIMESTAMP, ROWS_READ, NUM_EXECUTIONS,
       PREP_TIME_WORST, TOTAL_EXEC_TIME, STMT_TEXT
FROM TABLE (SNAPSHOT_DYN_SQL ('database name', -1))
AS SNAPSHOT_DYN_SQL
```

Usually the queries run against a star schema are run as dynamic SQL. However, in the event that there are static SQL statements issued against the database, these statements can be retrieved from the SYSCAT.STATEMENTS catalog view of the database. For example:

```
SELECT TEXT FROM SYSCAT.STATEMENTS
```

However, try to minimize the use of static SQL in relation to star schemas, since these types of queries often are created and compiled at the inception of the program or application and may never be rebound since — even though the database may have changed radically. It also does not make sense to save on compile time by using static SQL when a typical query against a star schema runs for seconds, minutes, or even hours. And as a final point:

Important note: MQTs are never considered when static embedded SQL queries are executed.

4.6.2 Are the statements using the MQTs?

You can use DB2's EXPLAIN facility to gather information about the access path chosen by the optimizer and the rewritten (if appropriate) SQL.

Before using Explain, the Explain tables must be created. The Explain tables capture access plans when the Explain facility is activated. You can create them using table definitions documented in the SQL Reference, Volume 1, or you can create them by invoking the sample command line processor (CLP) script provided in the EXPLAIN.DDL file located in the 'misc' subdirectory of the sqllib directory. To invoke the script, connect to the database where the Explain tables are required, then issue the command:

```
db2 -tf EXPLAIN.DDL
```

The EXPLAIN statement is described in the SQL Reference, Volume 2. The **db2exfmt** tool, used for formatting the information in the Explain tables, is described in the Command Reference, and there is a sample available with DB2 Cube Views, MDSampleExplain.sql. It is located in the samples\olap\mdsample directory of the sqllib directory.

Explaining the chosen optimizer path can also be done through the Control Center. A feature of this tool is that if the Explain tables have not been created, they will be created for you. To access the tool, you can right-click the database you want the SQL explained for and you will see it, as shown in Figure 4-21.

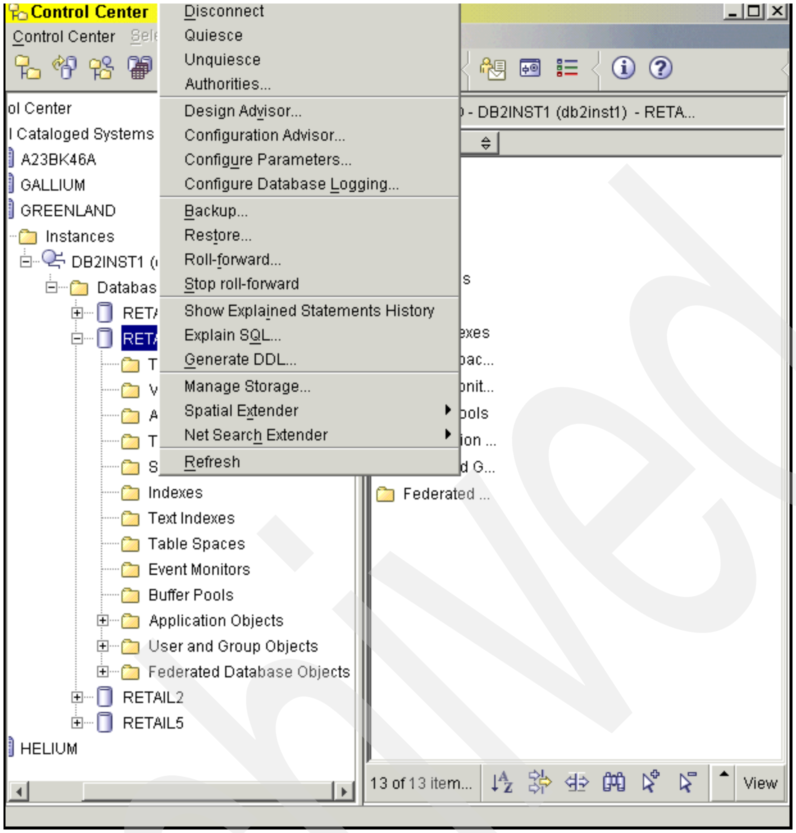


Figure 4-21 Explain SQL

Selecting Explain SQL will produce a new window as shown in Figure 4-22 (possible preceded by a message stating that the Explain tables have been created, if they were missing) where the SQL statement to be explained can be entered.

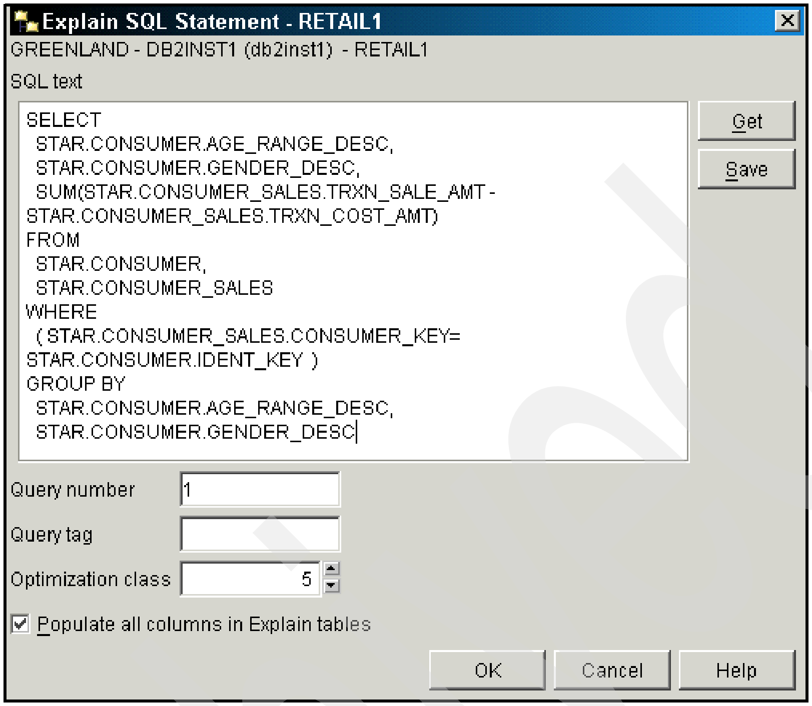


Figure 4-22 Explain SQL statement dialog

The advantage of using this approach to explaining SQL statements is that the access path chosen by the optimizer will be displayed graphically, which makes it easier to quickly determine whether for example query rewrites are performed. In the following example a query is explained graphically. The result set is displayed at the top of the graphic, and the tables from which the information is retrieved are at the bottom of the graphic.

Between the top and the bottom there are a number of different boxes, each describing the actions DB2 performs to get from the base data to the result set. In the case depicted in Figure 4-23 we see a simple scenario, where DB2 accesses the DB2INFO.MQT0000000001T02 Materialized Query Table to get the needed information using a table scan (TBSCAN). You can double-click each box to get a detailed explanation of each step, but often the Access Plan Graph provides enough information to determine where the problem is located.

Notice in Figure 4-23 that the cost timeron tally is displayed in every box in the graph. Often, if queries take too long, for example, the problem can be located to the place in the graph where the tally increases dramatically compared to the rest of the graph.

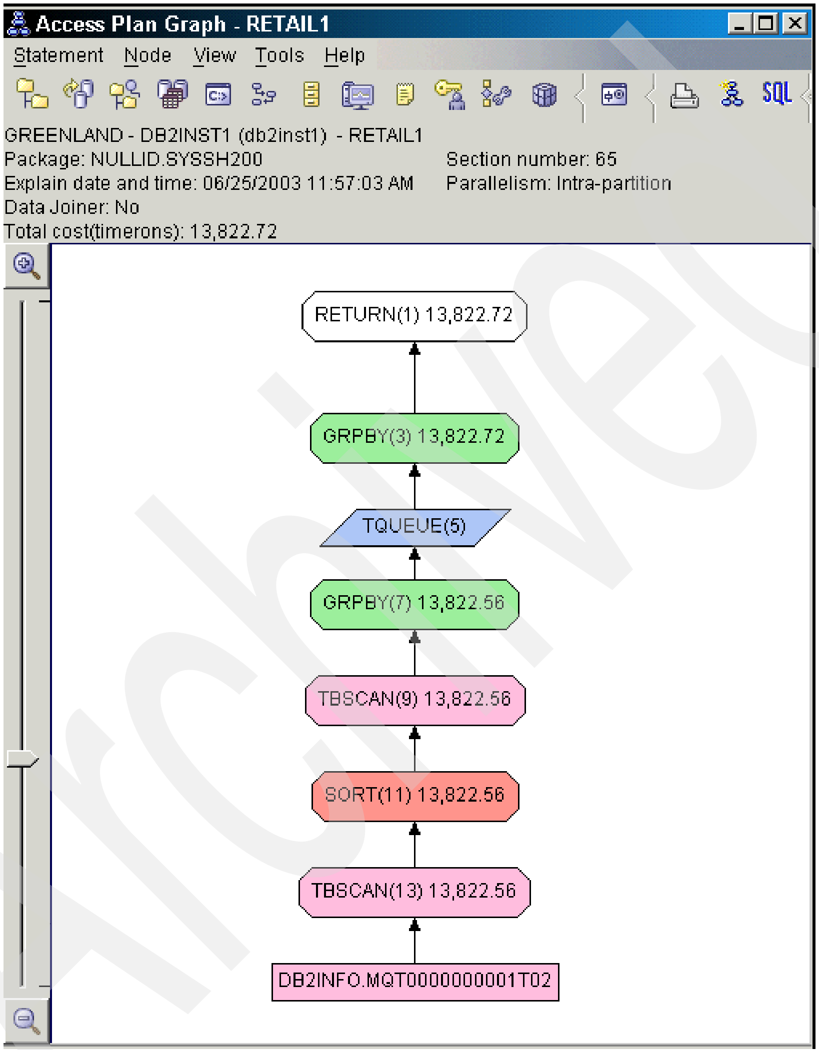


Figure 4-23 Explain SQL statement result

Another way of looking at the graph is to see if the expected tables are used. The query we ran was expected to use an MQT. As we are relieved to discover, the Access Plan Graph clearly shows us that this is the case. If the MQT was not used, we would see an Access Plan Graph as depicted in Figure 4-24.

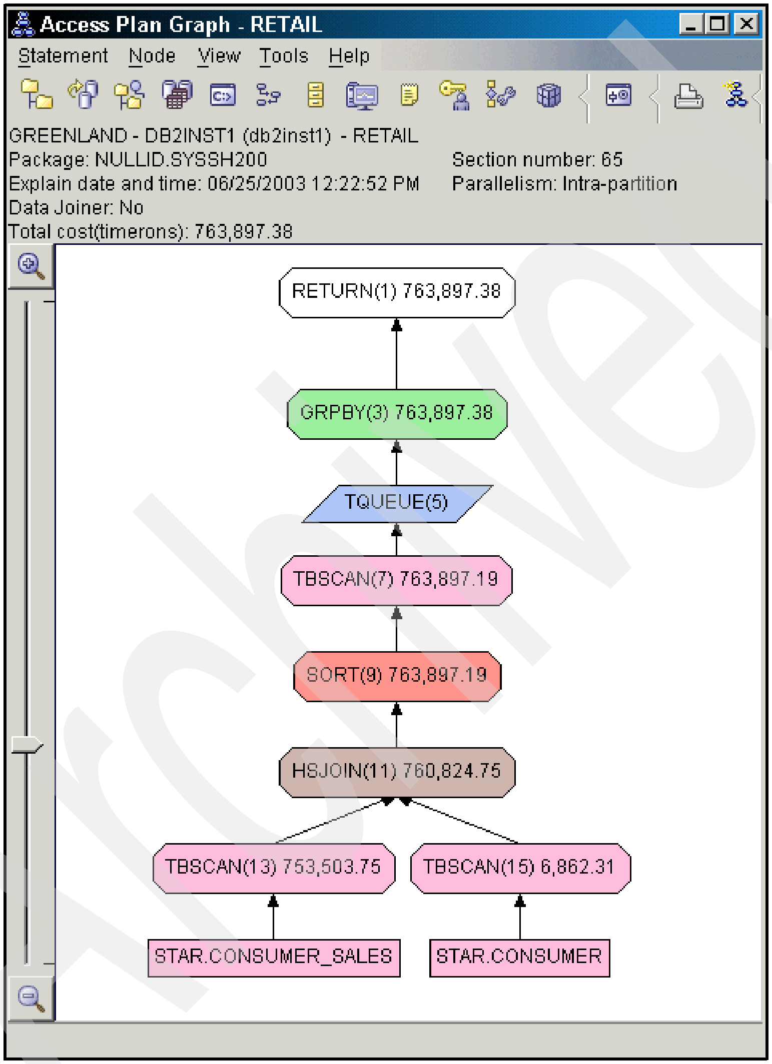


Figure 4-24 Explain SQL statement result without MQT

As can quickly be seen from the timeron tally, the table scan of the *STAR.CONSUMER_SALES* is, not surprisingly, very expensive. It is also easy to see that there are no MQTs being used.

We will not attempt to go into more detail about DB2 Visual Explain. Instead, we refer to the *Visual Explain Help Guide*, which provides detailed explanations of each of the elements displayed in the graph as well as providing an insight into how the graphs should be interpreted. A quick way of doing this is to double-click a box in the Access Plan Graph and click the **Help** button. From here, there is a specific description, as well as a general one, of what you see.

4.6.3 How deep in the hierarchies do the MQTs go?

If you study the output from the Optimization Advisor or use the DB2 Command Center to generate the DDL for the MQT in question, you will be able to see that there are one or more **GROUPING SETS** defined, as shown in Example 4-7.

Example 4-7 An abbreviated MQT

```
GROUP BY GROUPING SETS (
(
T2."CAMPAIGN_TYPE_DESC",
T2."CAMPAIGN_TYPE_CODE",
T2."CAMPAIGN_DESC",
T2."CAMPAIGN_ID",
T2."STAGE_DESC",
T2."STAGE_ID",
T2."CELL_DESC",
T2."CELL_ID",
T2."PACKAGE_DESC",
T2."PACKAGE_ID",
T2."COMPONENT_DESC",
T2."COMPONENT_ID",
T3."GENDER_DESC",
T3."GENDER_FLAG",
T3."AGE_RANGE_DESC",
T3."AGE_RANGE_CODE",
T4."CAL_YEAR_DESC",
T4."CAL_YEAR_ID",
T5."DEPARTMENT_DESC",
T5."DEPARTMENT_ID",
T5."SUB_DEPT_DESC",
T5."SUB_DEPT_ID",
T6."ENTERPRISE_DESC",
T6."ENTERPRISE_KEY",
T6."CHAIN_DESC",
T6."CHAIN_KEY",
T6."REGION_DESC",
```

```
T6."REGION_ID"
),
(
T2."CAMPAIGN_TYPE_DESC",
T2."CAMPAIGN_TYPE_CODE",
T2."CAMPAIGN_DESC",
T2."CAMPAIGN_ID",
T2."STAGE_DESC",
T2."STAGE_ID",
T2."CELL_DESC",
T2."CELL_ID",
T2."PACKAGE_DESC",
T2."PACKAGE_ID",
T2."COMPONENT_DESC",
T2."COMPONENT_ID",
T3."GENDER_DESC",
T3."GENDER_FLAG",
T5."DEPARTMENT_DESC",
T5."DEPARTMENT_ID",
T6."ENTERPRISE_DESC",
T6."ENTERPRISE_KEY",
T6."CHAIN_DESC",
T6."CHAIN_KEY",
T6."REGION_DESC",
T6."REGION_ID",
T6."DISTRICT_DESC",
T6."DISTRICT_ID"
```

Basically, what the grouping sets describe is how the MQT is to be aggregated across the dimension hierarchies. If you compare this to the cube model in DB2 Cube Views, you will be able to map where the MQT aggregates to, in each of the dimensions.

Once you have the grouping sets and the cube model, you can determine the hierarchies, by looking again in DB2 Cube Views, as shown in Figure 4-25.

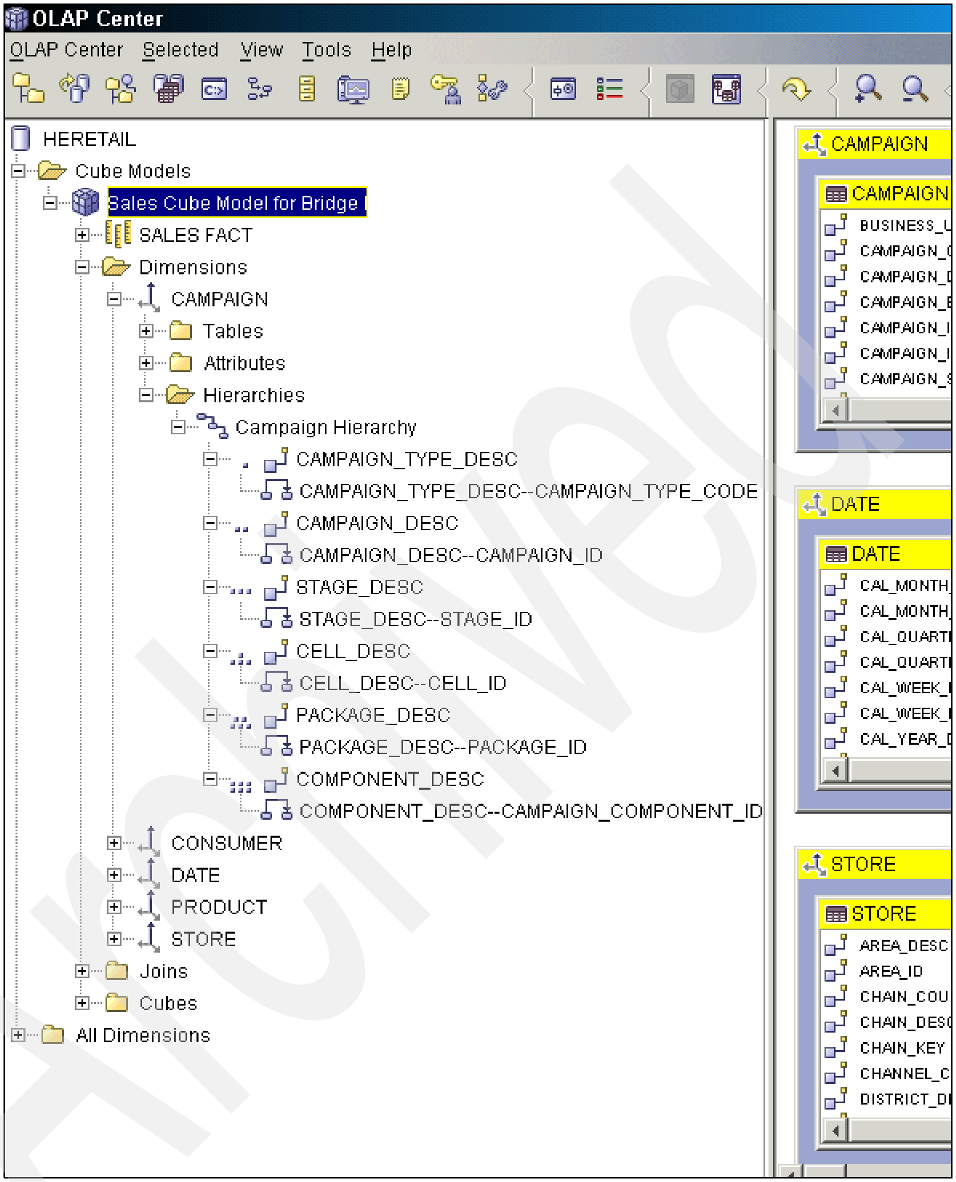


Figure 4-25 A cube model hierarchy

By looking at the first grouping set of Example 4-7 on page 178 you will see that there are four hierarchical levels of the campaign dimension present in the grouping set. The lowest level is the *CELL_DESC - CELL_ID* level, and we can thus determine the depth of the MQT within that grouping set on the campaign dimension. If you continue this analysis for the rest of the entries in the grouping set, you will determine where the cube is sliced by the MQT in that particular grouping set.

Continue this analysis for any other grouping sets and MQTs, and you will have determined all the slices made by the MQTs in your cube.

For an example of how you can visualize such a slice, please refer to Figure 4-6 on page 144, where two very shallow slices are depicted across the cube dimensions.

Now that we know how deep into the cube the MQTs go, we have a good foundation for determining how well any given query workload matches the MQTs built.

The analysis of such a query workload lies outside the bounds of this book, but a way to get started would be to find the queries that do not reroute to the MQTs (use explain for this) and see if you can group them into families depending on which dimensions they make use of and how deep down the dimensions they go.

Now order the families by size and by how many hierarchies they are from the closest slice of any MQT that covers the needed dimensions. Take the largest family and the family which are only one or two dimension hierarchies from being able to use an MQT, and explore the cost of changing your MQTs (for example, by using Cube Views) to match the query families. Continue until a large enough percentage of queries route to the MQTs or other constraints such as disk space are met.

4.6.4 Check the DB2 parameters

In the following sections we go into detail about which DB2 parameters have an influence on MQTs, either from a performance point of view or simply as an enabling measure.

Are the indexes being used?

You should also evaluate the use of the indexes you've created on the MQTs as well as on the base tables. As a starting point, you should have indexes on any high cardinality foreign key in the fact table, plus obviously, the primary keys of the dimension tables.

The **EXPLAIN** output will indicate if the indexes are being used. If they are not, use for example the Index Advisor (**db2advis**) to get recommendations on indexes to benefit your workload.

The syntax is:

```
db2advis -d <db name> [-t <time>] [-l <disk_space>]
          s "sql stmt" | -i <infile> | -w <workload name>
          [ -a <username> [/<password>] ]
          [ -o <output script> ] [ -p]
```

Note that only one of the following three options can be used: [s,i,w]

The options are shown in Example 4-8.

Example 4-8   db2advis options

```
-d database name.
    -p keep plans in explain tables.
    -t maximum duration for db2advis to run, in minutes.
       default is 1 minute, a value of 0 means unlimited duration.
    -l maximum disk space in megabytes.  default is
       unlimited.
    -s recommend database objects for this SQL statement.
    -i get SQL from this input file.
    -o place the database objects creation script in a file.
    -w get SQL from rows in the ADVISE_WORKLOAD table,
       specified by matching WORKLOAD_NAME.
    -a username to connect with. (optional password)
```

Create an input file called db2advis.in with the 5 lines provided in Example 4-9.

Example 4-9   db2advis input file

```
--#SET FREQUENCY 100
 SELECT COUNT(*) FROM EMPLOYEE;
 SELECT * FROM EMPLOYEE WHERE LASTNAME='HAAS';
 --#SET FREQUENCY 1
 SELECT AVG(BONUS), AVG(SALARY) FROM EMPLOYEE
          GROUP BY WORKDEPT ORDER BY WORKDEPT;
```

Run the following command and let it finish:

```
db2advis -d sample -i db2advis.in -t 5
```

For bigger workloads, the program will take longer.

Tip: The `db2advis` tool needs the Explain tables and the Advise tables to exist. You can create the them using the EXPLAIN.DDL script in the misc subdirectory of the sqllib directory.

Since the Index Advisor can use a file with the queries as input, along with frequency indications, it is ideally suited for our needs, since we are already working with a query workload for which we want to optimize the database.

See db2advis in the Command Reference for more information on the Index Advisor.

Are RUNSTATS current?

Make sure you've run RUNSTATS on the base tables as well as on the MQTs. Collect statistics on indexes, too. For example:

```
RUNSTATS ON TABLE MQT1 ON ALL COLUMNS WITH DISTRIBUTION AND DETAILED
INDEXES ALL
```

Important: Always make sure that `RUNSTATS` have been run on the base tables prior to building an MQT. DB2 depends on the statistics for accessing the base tables, and the REFRESH time of the MQTs may be extended considerably if the statistics are not present or current.

Are the DB2 special registers set?

The DB2 special registers are basically variables that tell DB2 how to treat the database in various situations. There are several ways of displaying and setting these values. The two ways most frequently used are either through the DB2 Command Center or via DB2 Command Window (possibly via a telnet connection if using an AIX® or Linux based system). Here we will focus mainly on which values to set, and will refer to the DB2 Administration Guide for further information as to how the DB2 Command Center works, for example.

Make sure the DB2 special registers are set to allow the DB2 optimizer to consider the types of MQTs you're interested in.

To determine the current setting, issue the following command in a DB2 Command Window:

```
VALUES (<special register>)
```

or specifically:

```
VALUES (CURRENT REFRESH AGE)
```

To set the relevant special registers:

```
SET CURRENT REFRESH AGE = ANY
```

(The default is 0)

This controls whether REFRESH IMMDIATE (0) or also REFRESH DEFERRED (ANY or 99999999999999.000000) MQTs can be used for optimization.

```
SET CURRENT QUERY OPTIMIZATION = 2 or >= 5
```

This instructs the optimizer to use query rewrite.

```
SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION = ALL
```

This controls what types of MQTs can be used for optimization. The default is SYSTEM, while NONE will disable query rewrites.

Note that the `CURRENT REFRESH AGE` special register must be set to a value other than zero for the specified table types to be considered when optimizing the processing of dynamic SQL queries.

In cases where a `SET` statement has not yet been executed, the special registers are determined by the value of the database configuration parameters. The database configurations parameters can, for example, be viewed and changed from the DB2 Control Center. Right-click the database and select **Configure Parameters**.

The special register values described above can be mapped to the database configuration keywords as shown in Table 4-6.

Table 4-6 Special register names vs. database configuration keywords.

Special register name	keyword ID
CURRENT REFRESH AGE	DFT_REFRESH_AGE
CURRENT QUERY OPTIMIZATION	DFT_QUERY_OPT

Is the MQT enabled for query optimization?

The ENABLE FOR QUERY OPTIMIZATION option is a constraint attribute on the table which tells the DB2 optimizer whether it can use the MQT for optimization. Make sure the MQT was created with ENABLE FOR QUERY OPTIMIZATION. That is the default.

Should you want the MQT not be included in the DB2 optimizers efforts to reroute queries, you can issue an ALTER TABLE statement with the option DISABLE QUERY OPTIMIZATION. The materialized query table will then not be used for query optimization. The table can still be queried directly, though.

Is the table really an MQT?

You can ALTER an MQT and make it a real table with the following command:

```
ALTER TABLE mqt SET MATERIALIZED QUERY AS DEFINITION ONLY;
```

You can check the status of a table with:

```
SELECT TABNAME, TYPE FROM SYSCAT.TABLES WHERE TABNAME = ‘mqt_name’;
```

Is the MQT accessible?

An MQT may be in a **CHECK PENDING NO ACCESS** state. This occurs:

1. After initial creation prior to population
2. When a staging table is created
3. On a staging table after a **SET INTEGRITY IMMEDIATE CHECKED** is run on a base table following a **LOAD INSERT**
4. On a REFRESH IMMEDIATE MQT after a **LOAD INSERT** and **SET INTEGRITY** on the base table

You can determine its status with:

```
SELECT TABNAME, STATUS
FROM SYSCAT.TABLES
WHERE TABNAME = ‘mqt_name’;
```

In all cases a **REFRESH TABLE <MQT tablename>** statement should clear the **CHECK PENDING** state.

4.6.5 Is the query optimization level correct?

Query rewrites will only be considered with the optimization level (DFT_QUERYOPT) set to 2, 5, 7 or 9. If the query optimization level is 0 or 1 query rewrites will not be considered. Using optimization level 3, most query rewrite rules are applied, including subquery-to-join transformations but routing of queries to materialized query tables will not be performed.

4.7 Optimization Advisor and cube model interactions

Having gone through the process of creating one or more MQTs through the help of the Optimization Advisor, you may wonder what kind of performance benefits you can expect from DB2 when query rewrites happen.

In general, we get the largest performance benefits in the top of the cube, where the measures are aggregated highly and the MQTs have few rows, compared to lower down, where the aggregations are less and the MQTs have more rows. We are, however, also helped by the fact that MQTs denormalize the base data (often further) and thus at the expense of disk space, eliminate many joins which often also are very costly. What this means is that we see substantial benefits from MQTs even when the aggregation is fairly low. In the tests we made we saw substantial performance benefits even where the MQTs built had slices that went through the lower half of the cube model dimensions (guided by our cube definitions).

It is, however, difficult to make firm recommendations as to how low you can go in the cube before the performance benefit becomes small. One of the main reasons for this that it is very difficult to say anything about data sparsity for a given set of base data. In addition, the data sparsity varies greatly between various sets of base data. The “point of diminishing returns” must, therefore, be determined iteratively, given there are no other constraints such as time or space. It is our experience that, after some initial experimentation, the basic recommendations of the Optimization Advisor with a few iterations performed quite well, given that we had no initial query workload with which to qualify our estimates. It was, however, fairly clear that the better idea you have about the query workload, the better matched the MQTs will be.

In the following sections a few examples are presented based on a database where we asked the Optimization Advisor to optimize for drill down. The reason for this was that we wanted to create MQTs that favored the top of the cube where we would get the greatest performance benefit.

This optimization will obviously not always be the right approach. In order to get an idea about what effect the Optimization Advisor query type specification has, Table 4-7 is presented.

Table 4-7 The Optimization Advisor query type behavior

Query Specification	General behavior
Drill down	The focus of the optimization is at the upper parts of the hierarchies and rollups may be done
Report	Like drill down, but without the top level rollups
Extract	The focus of the optimization is at the lower parts of the hierarchies
Drill through	Like extract, but with additional slices near the bottom

The big question is how to apply this in practice. For this we have tried to build a table for a selection of OLAP tools. Note that the table is only a suggestion for an initial optimization and may not completely fit with any given tool.

Table 4-8 Suggestions for Optimization Advisor query specifications

Product	OLAP type?	Queries	Optimization Advisor query specification
Spreadsheet add-in	Bridgeless ROLAP	Drill down	Extract or Drill down if data stays in DB2
Reporting	ROLAP	Report	Report
Hybrid Reporting tools	HOLAP	Report	Drill through
OLAP Analysis	ROLAP	Drill down	Drill down
Hybrid Analysis	HOLAP	Drill through to relational	Drill through
Multidimensional reporting and/or analysis	MOLAP	Only to the MOLAP cube	Extract

Since most cases need cubes specified under the cube model in the OLAP Center, it can be tempting to build a lot of them for any case that comes to mind. This is not recommended. Generally it is advisable to create the cubes predominantly for business reasons and to limit the number of cubes to a small number, preferably less than a handful for big cases.

4.7.1 Optimization Advisor recommendations

Here are a number of other practical recommendations:

- Many MQTs will extend the query compile time because the DB2 optimizer will have to consider all MQTs every time. Again — limit the number of cubes in your cube model. Regarding the actual performance penalty it is difficult to determine what it is on for example a per MQT per query basis since this varies widely depending on system hardware and database configuration. It would be a matter of benchmarking actual configurations to determine the impact of MQTs on SQL compile times, which lies outside the scope of this book.
- Using the refresh IMMEDIATE option on the MQTs is especially computationally intensive for MQTs spanning more than one dimension. In that case, prefer using the refresh DEFERRED option if possible, and plan your refresh of the MQTs carefully.

- Avoid creating multiple cube models on the same base tables. You will have difficulty maintaining and synchronizing the metadata between the cube models and you will most likely not have one complete set of metadata describing the entire data either.
- Resist the urge to build MQTs that go to the bottom of all dimensions (by building a cube reflecting the entire cube model). Even though any query could, in theory, be routed to the MQT, DB2 will most likely discard it because the cost of going to the MQT will be higher than going straight to the base fact table. The end result will be a lot of wasted disk space.
- If you are not getting the MQTs you want and changing the existing cubes does not work, try adding a cube to the cube model to provide additional hints to the Optimization Advisor.

The examples in the following sections are taken from query workloads generated by various OLAP tools, but for simplicity's sake we have selected a number of them that all using the same MQT, built for drill down queries. The script to create and refresh the MQT is provided in "MQT" on page 694.

Note that the MQT is specified with `REFRESH DEFERRED`. This was a deliberate selection by us when running the Optimization Advisor and was done in our test setup for flexibility reasons. We thus avoid placing the MQT in `CHECK PENDING` state if changes are made to the base table. This is, however, a selection the DBA must be very careful of using in a production environment.

Having inconsistencies between the base tables and the MQTs can result, from the users point of view, in getting different results depending on whether the DB2 optimizer chooses to use MQTs or not. Since the optimizer behavior is transparent to the end user, confidence in the data mart or data warehouse can quickly be lost even when inconsistencies are quite acceptable from a theoretical point of view. Database inconsistencies are, in practice, unnerving to the end users — even more so when using MQTs, since the end user have no way of knowing when an MQT is being used. The problem persists even in the cases where the end users are SQL literate because the DB2 optimizer's ability to rewrite the SQL hides the use of MQTs.

Important: Where possible, avoid using `REFRESH DEFERRED` when building MQTs unless all end users accept and understand that by allowing inconsistencies between MQTs and base tables, apparently similar queries can produce different results, in no easily predictable fashion.

If inconsistencies cannot be avoided, make sure that they are confined to a service window and that all users are aware of this.

Suppose that there are inconsistencies between the base tables and the MQTs. An example could be that you have an MQT on the fact table and the time dimension going down to the quarter level. Summing profits over the quarter will produce a certain result, having the DB2 optimizer using the MQT. Now suppose you do the same query but on the month level. This time the MQT will not be used and if the user sums three months of a quarter, the user will see a difference.

Now suppose the user does a query on the quarter level again (all the time knowing that the sums on quarters are off by a fraction) but this time adds region to the query, which by chance is not in the MQT. If the user now sums the numbers for the regions he will find that they match the sum for the three months he did earlier but not the sum for quarter. Apparently the sums for quarters are now correct? Yes, but only because the DB2 optimizer chooses to go to the base tables because region has not been aggregated in the MQT.

This behavior is not very easy to predict - especially if there are multiple MQTs and the queries span many dimensions. Often it is better to have the MQTs invalidated when the base tables are updated and experience a performance impact until the MQTs have been refreshed, than have a period where the users can not depend 100% on their results.

With the MQT listed above in mind we now will take a look at the query examples.

4.7.2 Query to the top of the cube

The query in Example 4-10 summarizes the profit from each transaction and groups it by age and gender to get the top five most profitable consumer groups. The query stays at the topmost part of the hierarchy and performs a large aggregation in terms of summing across the entire fact table:

Example 4-10 Query to the top of the cube

```
SELECT
  STAR.CONSUMER.AGE_RANGE_DESC,
  STAR.CONSUMER.GENDER_DESC,
  SUM(STAR.CONSUMER_SALES.TRXN_SALE_AMT - STAR.CONSUMER_SALES.TRXN_COST_AMT)
FROM
  STAR.CONSUMER,
  STAR.CONSUMER_SALES
WHERE
  ( STAR.CONSUMER_SALES.CONSUMER_KEY= STAR.CONSUMER.IDENT_KEY  )
GROUP BY
  STAR.CONSUMER.AGE_RANGE_DESC,
  STAR.CONSUMER.GENDER_DESC
```

Doing an explain without the MQTs yields the situation shown in Figure 4-26.

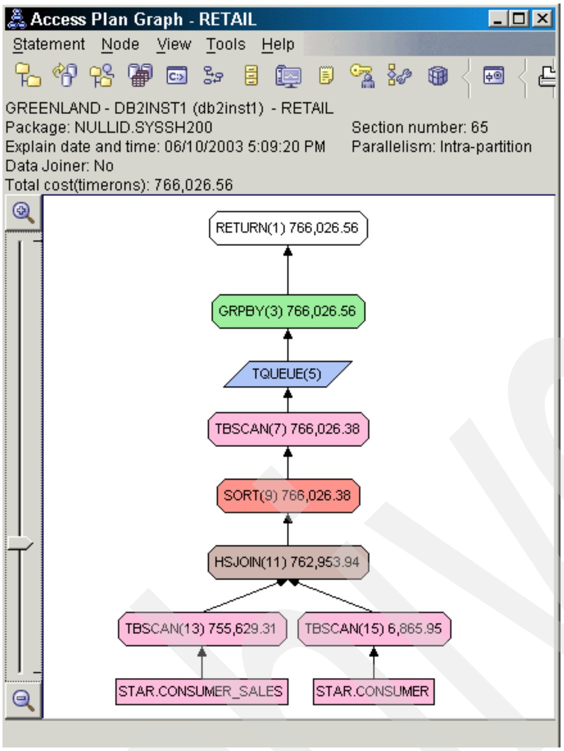


Figure 4-26 The top five most profitable consumer groups without MQTs

As can be seen in Figure 4-26, the cost of running the query without the query rewrite is 766,026.56 timerons.

Now, in Figure 4-27, we try the same query on the same data, but with the MQTs in place.

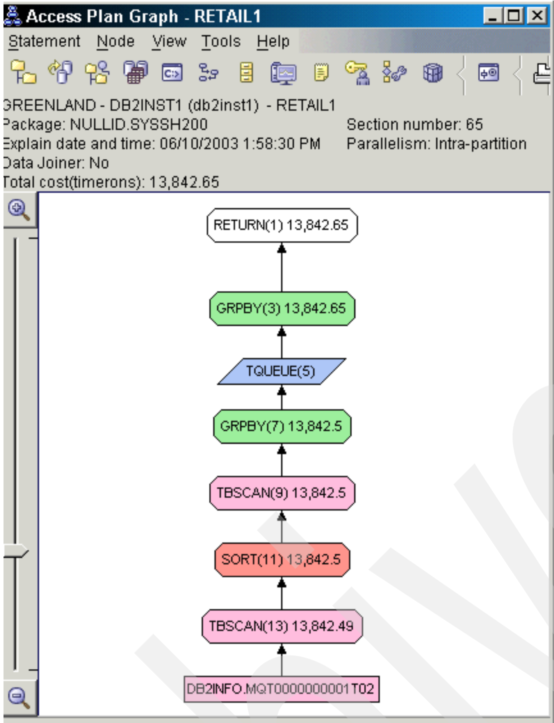


Figure 4-27 The top five most profitable consumer groups with MQTs

Figure 4-27 tells us that the cost of the query is now 1,3842.65 timerons — a performance improvement of much more than an order of magnitude.

Now, one may argue that there are no relevant indexes on the fact table and that this would change the picture we would be seeing, but placing indexes on the fact table is expensive in terms of space, especially since you would have to index the foreign key columns and you will still not have the preaggregations which we so richly benefit from here. Moreover it should be noted that the MQTs generated by the Optimization Advisor take up approximately 10% of the space occupied by the fact table — much less than would be occupied by a set of indexes covering the measures of the fact table.

4.7.3 Querying a bit further down the cube

The query in Example 4-11 is also fairly simple since it only joins one dimension with the fact table to get the sales amount and quantity by region area. Nevertheless, it is interesting to look at, since it is an example of a fairly often performed initial query of a drill down exercise which requests a complete aggregation of one or more measures of the fact table.

Example 4-11 Querying down the cube

```
Select T1."REGION_DESC" "c1" , T1."AREA_DESC" "c2" , sum(T2."TRXN_SALE_QTY")
"c3" , sum(T2."TRXN_SALE_AMT") "c4"
 from "STAR"."CONSUMER_SALES" T2, "STAR"."STORE" T1
 where T2."STORE_ID" = T1."IDENT_KEY"
 group by T1."REGION_DESC", T1."AREA_DESC"
 order by 1 asc , 2 asc
```

Running the query without the MQTs provides the access graph in Figure 4-28.

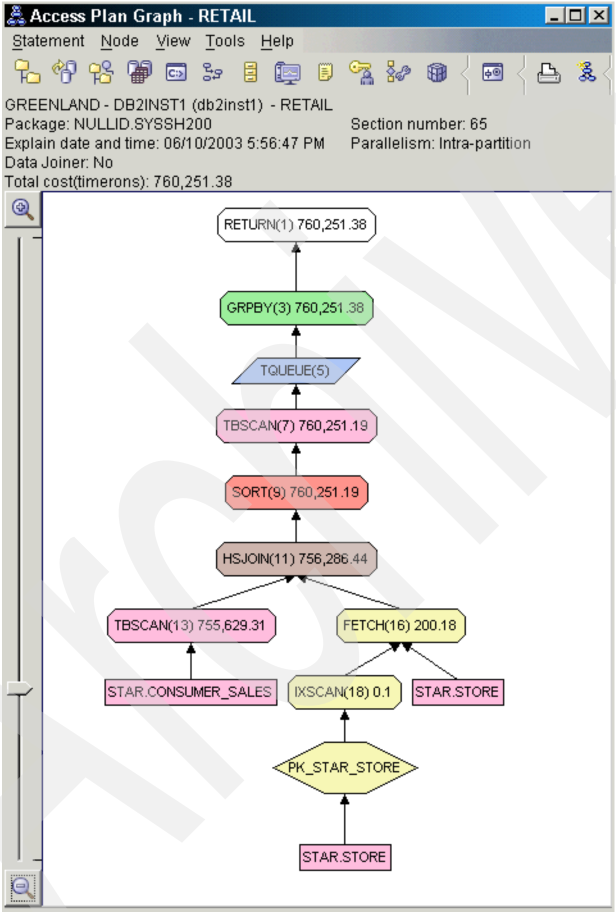


Figure 4-28 Sales amount and quantity by region area without MQTs

Figure 4-28 shows that the DB2 optimizer performs a tablespace scan on the fact table which basically is what makes the query so expensive. The reason for this is that all the rows of the fact table are needed to do the aggregation and, therefore, the optimizer will not even benefit from using the indexes on the fact table.

Now, in Figure 4-29, see what happens if we allow the optimizer to make use of the MQTs.

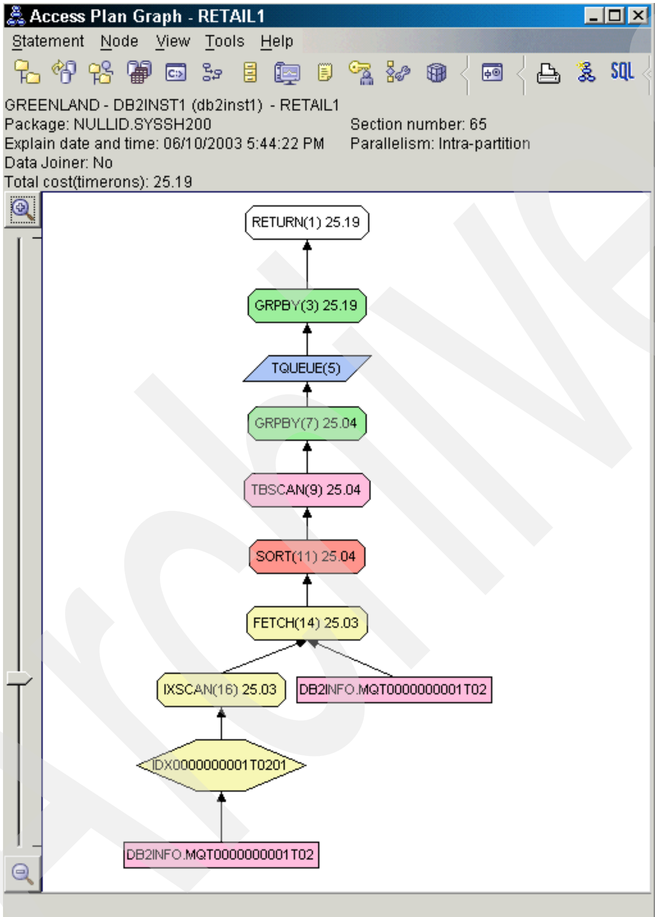


Figure 4-29 Sales amount and quantity by region area ***with*** *MQTs*

Figure 4-29 demonstrates how dramatic an improvement MQTs can be to an OLAP cube. The cost went down from 760,.251.38 timerons to 25.19 timerons. This time because the DB2 optimizer can take advantage of the index on the MQT as well as the preaggregations.

4.7.4 Moving towards the middle of the cube

The query in Example 4-12 goes down to the lower levels of the campaign dimension. Here we start to qualify the query as we are looking for the Coupon component of the campaigns.

Example 4-12 Query moving towards the middle of the cube

```
select T1."GENDER_DESC" "c1" ,
       T1."AGE_RANGE_DESC" "c2" ,
       T2."REGION_DESC" "c3" ,
       T3."CAMPAIGN_TYPE_DESC" "c4" ,
       T3."PACKAGE_DESC" AS "PACKAGE_DESC",
       T3."CELL_DESC" "c5" ,
       sum(T4."TRXN_SALE_QTY") "c6" ,
       sum(T4."TRXN_SALE_AMT") "c7"
  from "STAR"."CONSUMER_SALES" T4,
       "STAR"."CONSUMER" T1,
       "STAR"."STORE" T2,
       "STAR"."CAMPAIGN" T3
 where T4."CONSUMER_KEY" = T1."IDENT_KEY"
   and T4."STORE_ID" = T2."IDENT_KEY"
   and T4."COMPONENT_ID" = T3."IDENT_KEY"
   and T3."COMPONENT_DESC" = 'Coupon'
 group by T1."GENDER_DESC",
          T1."AGE_RANGE_DESC",
          T2."REGION_DESC",
          T3."CAMPAIGN_TYPE_DESC",
          T3."PACKAGE_DESC",
          T3."CELL_DESC"
 order by 1 asc , 2 asc , 3 asc , 4 asc , 5 asc, 6 asc
```

The aggregations are not as large as before since we can reduce the relevant number of rows needed from the fact table, as the work that needs to be done is less, but take a look at the timeron cost for doing the query against the base tables in Figure 4-30.

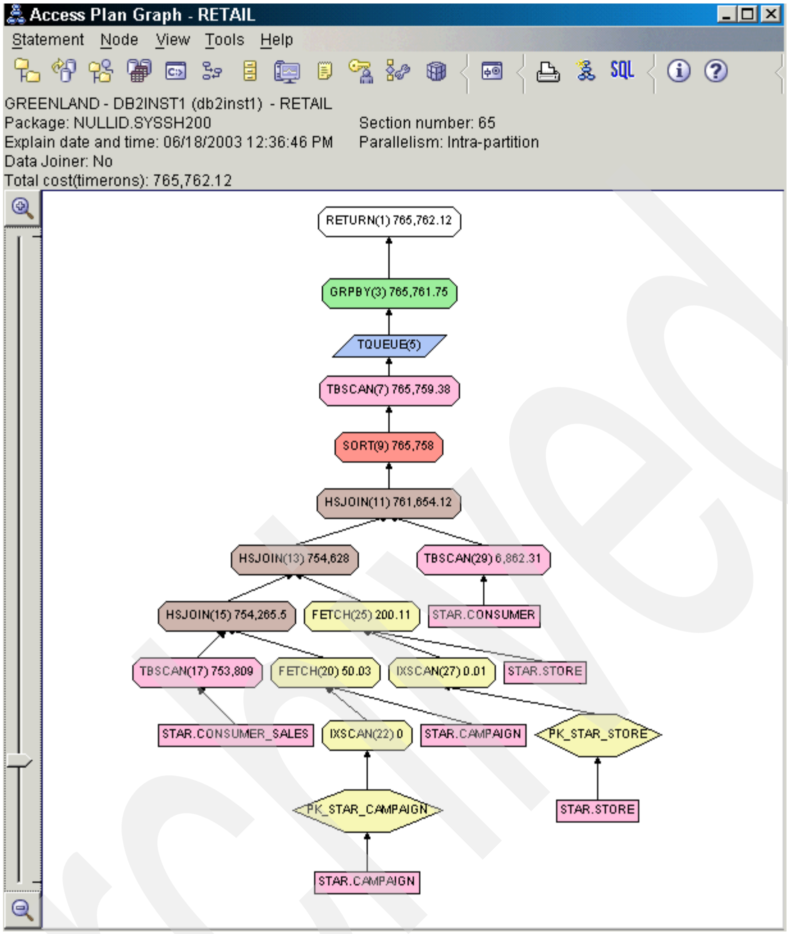


Figure 4-30 Sales through coupons campaign access graph

Even though we are only interested in the coupon campaigns, DB2 has no index to take advantage of (the COMPONENT_ID) and thus again performs a tablespace scan on the fact table. If there are many queries that reference this column on the fact table we might consider building an index, but let's take a look at what our MQT can do for this query.

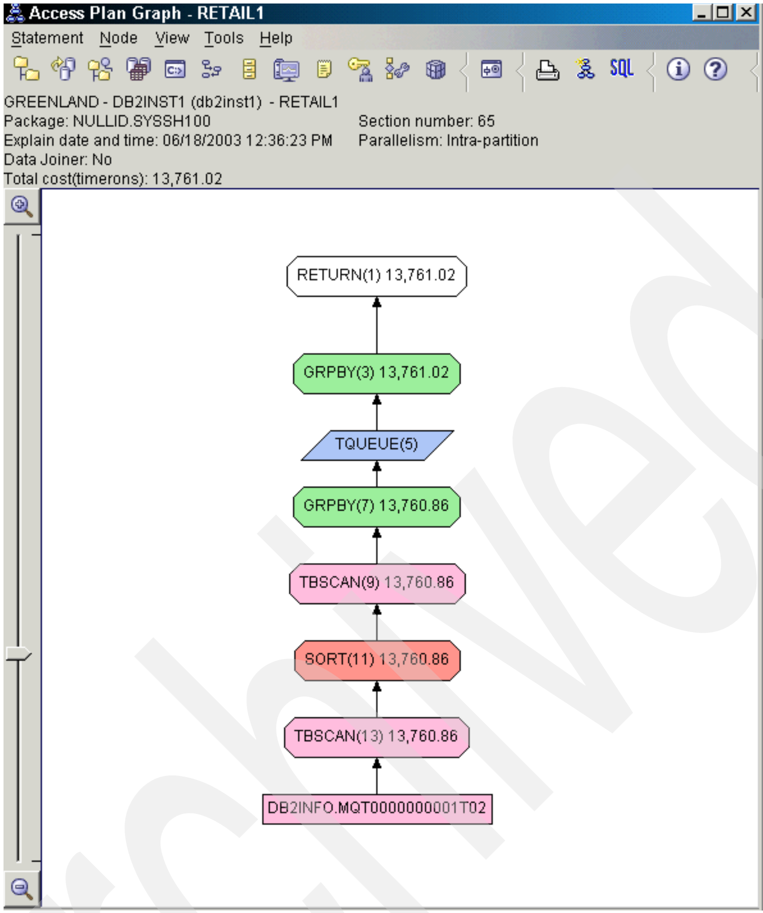


*Figure 4-31 Sales through Coupon campaigns **with** MQTs*

We have again reaped enormous benefits (from 765,762.12 to 13,701.02 timerons) from having the MQT with the aggregations in place.

We could continue our exploration into the realm of MQTs, but we think that these examples, even though they are quite simple, fairly represent the general performance benefits reaped from using MQTs.

4.7.5 Visiting the bottom of the cube

It should be noted that the deeper the exploration of the cube, the less benefits there are to be had from MQTs. This depends mainly on the size (in rows) of the MQTs compared to the base fact table. The closer the two numbers come, the less likely the DB2 optimizer is in considering the MQT and the less benefit there is in general from the MQT as individual aggregations become smaller and smaller. The sparsity of the fact table is also an important factor in determining how low in the cube we may go before we lose the benefits of MQTs.

All these factors are considered by the Optimization Advisor but it is nevertheless important to understand that these factors play an important role in determining how efficient the use of MQTs are. By building cubes that more or less encompasses the entire cube model and providing a large space allowance it is possible to make the Optimization Advisor build large MQTs that go very deep into the cube. However, space issues become severe as does the creation and refresh times of the MQTs as they come near the number of rows of the fact table so that the DBA responsible for their creation should survey the suggested MQTs carefully before deploying them in a production environment.

Prior to the actual deployment of the MQTs, we suggest doing the following:

- Determine the actual space requirements.
- Determine the necessary MQT refresh window as the MQT will be as large as or even larger than the base fact table.
- Perform an Explain on the resulting database with a representative query workload to determine that the MQTs are used and provide a substantial performance benefit compared to their possible large cost of creation and maintenance.

Generally it should be noted that if queries often are performed at the transaction level of the cube with few or no aggregations, MQTs will not be of much help. In this case we suggest exploring the use of indexes to boost performance in the cases where certain fact table columns often are chosen above others or, if that is not the case, in general rely on other means of performance optimization.

4.8 Performance considerations

MQTs provide significant performance improvements for your query environment, however, there are relevant and important aspects that you need to consider for your production query environment. There are two different situations where you can find opportunities to optimize, during the refresh (INCREMENTAL or FULL refresh) of MQTs and during the query execution time.

During the refresh of an MQT (either INCREMENTAL and FULL refresh), there is a time associated to join the dimensions and fact tables involved on the MQT. Depending on how well tuned your physical data model is, it provides significant results during population of MQTs. Also, there are different techniques that you can take to populate the MQTs like load instead of refresh, or avoid logging data.

During query execution time, the DB2 optimizer considers the MQT like regular tables when it comes to access plan strategies. Good indexes on MQTs also are important for query optimization as well update statistics on base tables and MQTs are required for the DB2 optimizer to be able to choose an MQT instead of accessing the base tables.

Another important aspect that you also need to consider for your query environment is related to the different approaches that you can apply to refresh the MQTs. You can for example, perform *INCREMENTAL* or *FULL* refresh on the MQTs, and you also perform *IMMEDIATE* or *DEFERRED* updates on existing MQTS. These different approaches can affect the availability of MQTs and cause impact in your query environment. In the following sections we discuss more of the details and techniques, and how you can implement them.

4.9 Further steps in MQT maintenance

The maintenance of the MQT is a very important process. You need to carefully evaluate the benefits and the pros and cons of different approaches, because it can affect the performance for queries as well the integrity for the information accessed by end users.

The MQT population process is totally dependent on the population process of the base tables (regular tables). If you have a batch window of time that you generally use to populate your data mart or data warehouse, you probably don't need to worry too much about the data latency between MQTs and base tables. However, if you need perform online load on base tables while end users are executing queries against them, you need to be aware that during that time, the end users might get different results on their reports. The good news is that DB2 provides you capabilities that allow you to control this situation; in other words, you can tell the DB2 Optimizer not to use the MQT during such times.

> **Note:** In this chapter the term Regular Tables or Base Tables is used to reference the underlying tables that are used to feed MQTs.

DB2 supports two different types of MQT: User Maintained and System Maintained.

Since DB2 Cube views only generate System Maintained MQTs, we do not cover User Maintained MQTs on this book. For information on User Maintained MQTs, please refer to the DB2 documentation, you also can obtain information from the redbook, *DB2 UDB's High Function Business Intelligence in e-business*, SG24-6546.

For the System Maintained MQTs, you can specify the frequency of maintenance as either:

- Refresh DEFERRED (point-in-time)
- Refresh IMMEDIATE (current time)

DB2 Cube Views Advisor allows you to select either refresh DEFERRED or refresh IMMEDIATE options. However, if for any reason, the SQL query for the MQT is not supported by an MQT as refresh IMMEDIATE, DB2 Cube Views automatically generates the MQT as refresh DEFERRED.

Detailed steps required to maintain the MQTs are provided in "Implementation guidelines" on page 203.

4.9.1 Refresh DEFERRED option

Refresh DEFERRED means that there is an acceptable latency between the time that you have incrementally updated your base tables and the time that you populate the MQTs. During a certain point-in-time, by explicitly executing a **`refresh`** command on the MQT, the delta information from the base tables is applied to the MQT. More details on the refresh of MQTs are provided in "Implementation guidelines" on page 203.

During this interval, by default, the MQT is available for queries, unless you explicitly execute the command **`Set Refresh Age 0`** (only MQTs defined as refresh IMMEDIATE are available for query rewrite).

For the refresh DEFERRED option, there are two different scenarios, which depend on the type of maintenance performed on the base tables:

1. Updates, Inserts, and Deletes:
   a. These automatically reflect the changes on the STAGING tables within the same unit of work.
   b. There is a latency between updates on the STAGING tables and the MQTs.
   c. The MQTs are still available to be used during *query rewrite* by the DB2 optimizer.
   d. Additional action is required to synchronize the MQTs. Transfer the changes from the STAGING Tables to the MQT.

2. Load and Import with Insert option:
   a. Data is inserted only in the Base Tables.
   b. Depending on the Load options on the base tables, the MQTs are placed in **Check Pending No Access** state and the DB2 optimizer does not route any query to these MQTs until it is refreshed again.
   c. There is a latency between updates on the STAGING tables and the MQTs.
   d. Additional action is required to compute the delta into the STAGING tables and from the STAGING tables to the MQTs.

4.9.2 Refresh IMMEDIATE option

For the refresh IMMEDIATE option, there are two different scenarios, which depend on the type of maintenance performed on the base tables:

1. Updates, inserts, and deletes:
   a. These automatically reflect the changes on the MQTs within the same unit of work.
   b. There is no latency between updates on the base tables and the MQTs.
   c. The MQTs are instantly available to be used during *query rewrite* by the DB2 optimizer.
   d. No additional action is required to synchronize the MQTs with the base tables.
2. Load and import with INSERT option:
   a. Data is inserted only on the base tables.
   b. Depending on the LOAD options on the base tables, the MQTs are placed in **Check Pending No Access** state and the DB2 optimizer does not route any query to these MQTs until it is refreshed again.
   c. Additional action is required to synchronize the MQTs with the base tables.

> **Note:** For information on *query rewrite*, please refer to the DB2 documentation.

4.9.3 Refresh DEFERRED versus refresh IMMEDIATE

The decision to use refresh DEFERRED or refresh IMMEDIATE needs to be careful evaluated by the DBAs. They must understand the application requirements as well as having the knowledge to exploit the technology.

Here is a list of considerations that you might use as a reference to make this decision:

- Refresh IMMEDIATE MQTs, like refresh INCREMENTAL MQTs can only have **`COUNT, SUM, COUNT_BIG and GROUPING`** aggregation functions.
- Latency of the data. The tolerance for latency depends on the application.
  - Some applications can accept a latency of the data for query, such as end-of-day, end-of-week, end-of-month. For example, data warehouses and strategic decision-making could accept a certain latency for the data. In fact, for some situations, it is a requirement for the application that the data is only refreshed during certain periods. In such cases, the MQT does not need to be kept in synchronization with the base tables, and the refresh DEFERRED option should be used.
  - For OLAP applications and tactical decisions, probably any MQT latency may be unacceptable and the IMMEDIATE option can be used.
- Refresh IMMEDIATE on MQTs with a high volume of insert, update, and delete activity could cause significantly performance overhead on the base tables.
- Refresh IMMEDIATE requires:
  - Extra Column with COUNT(*) for maintenance
  - Extra Column with COUNT(nullable_colum_name) on the select list for each nullable column that is referenced in the select list with a SUM.
- Refresh DEFERRED requires a *staging* table for INCREMENTAL refresh.
- The INCREMENTAL refresh might be faster on an MQT defined as refresh IMMEDIATE compared to an MQT defined as refresh DEFERRED because there is no need to use staging tables.
- Refresh DEFERRED MQTs can be kept out of synchronization.
- Load insert activity on base tables:
  - The MQTs defined as refresh IMMEDIATE option are unavailable while the **`Load Insert`** operation is being performed on the base tables, unless you specify **`ALLOW READ ACCESS`** on the load statement.
  - The MQTs defined as refresh DEFERRED are **available** while the **`Load Insert`** operation is being performed on the base tables.

4.9.4 INCREMENTAL refresh versus FULL refresh

When deciding between use INCREMENTAL refresh or FULL refresh on MQT, you need to consider the following:

- Refresh INCREMENTAL MQTs, like refresh IMMEDIATE MQTs can only have **`COUNT`**, **`SUM`**, **`COUNT_BIG`**, **`and GROUPING`** aggregation functions
- INCREMENTAL refresh increases the availability of the MQTs. The refresh operation can be faster than a FULL refresh.
- INCREMENTAL refresh requires an index on the group by columns, otherwise the performance can be slower than FULL refresh.
- INCREMENTAL refresh requires logging, unless alter table is specified on the MQT to turn off to Not Logging.
- The **`import replace`** or **`load replace`** option cannot be used on the underlying tables of an MQT that needs to be incrementally maintained. FULL refresh is required when used those options.
- INCREMENTAL refresh can generate updates and deletes of existing rows on the MQT.
- The frequency of INCREMENTAL refreshes can cause a logging overhead against the MQT:
  - More frequent refreshes have the potential to involve more updates against the MQT.
  - Less frequent refreshes may result in fewer updates because data consolidation may occur either on the staging table (for refresh DEFERRED MQT) or underlying table (for refresh IMMEDIATE MQT).
  - Less frequent refreshes could result in a large volume of data in the staging table (for refresh deferred MQT) that needs to be pruned and logged.

Note: If neither INCREMENTAL nor NOT INCREMENTAL is specified, the system determines whether incremental processing is possible; if not, FULL refresh is performed. If a staging table is present for the MQT that is to be refreshed, and incremental processing is not possible because the staging table is in a pending state, an error is returned (SQLSTATE 428A8).

FULL refresh is performed if the staging table or MQT is in an inconsistent state; otherwise, the contents of the staging table are used for incremental processing.

4.9.5 Implementation guidelines

MQTs can be used for many other purposes such as for example materialize SQL transformations on ETL process; on this documentation we are limiting our discussion on building MQTs to support business intelligence applications and more specifically for query reporting and DB2 Cube Views. For more details on how to implement MQTs, please refer to *DB2 UDB's High Function Business Intelligence in e-business*, SG24-6546.

The current version of DB2 Cube Views only generates scripts to perform FULL refresh on the MQT. If you plan to have MQTs incrementally maintained, you need to create the required scripts as well as updating the DB2 Cube Views scripts as required by the INCREMENTAL refresh process. The next session covers the scripts required for different scenarios to perform FULL and INCREMENTAL refresh on MQTs.

> **Note:** Since DB2 Cube Views generates the DDL to create the MQTs and these scripts are manually executed by the DBAs, they have the possibility to perform any change on it. We *do not* recommended to perform changes on the `Select Statement` for the MQT because such SQL is created based in some intelligence of sampling the source data as well the type of query/report that is performed against the MQT. If for any reason you need to change the SQL, make sure that your MQT still valid and being used by the end user query/reports.

FULL refresh on MQTs

A FULL refresh on MQTs can be performed either after the first initial load on the based tables as well after additional loads on the base tables. For both cases, the MQT can be defined as refresh IMMEDIATE or refresh DEFERRED.

> **Note:** Even if the MQT is defined as refresh IMMEDIATE, but when using the load utility (standard practice for data warehouse applications) to update the base tables, the data is not automatically propagated to the MQTs. An additional command is required to select, compute, and insert the data from base tables into the MQT.

Figure 4-32 shows the process flow and the major tasks required for the FULL refresh process on MQTs defined either as IMMEDIATE or DEFERRED.

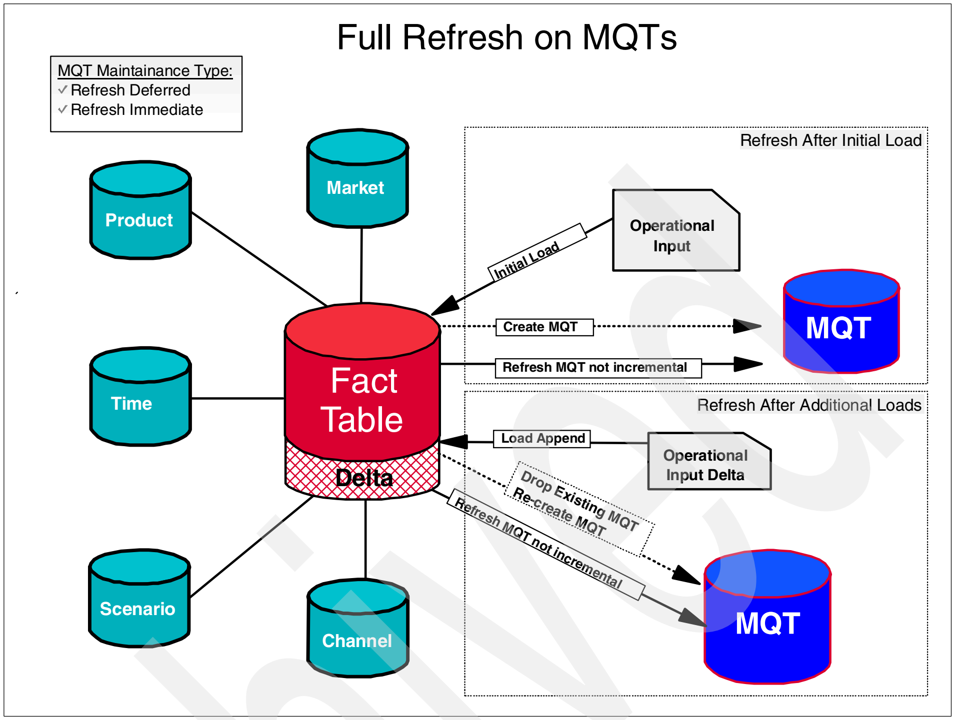


Figure 4-32 MQTs: FULL refresh for DEFERRED and IMMEDIATE

After you perform Initial load on the base tables and create the MQT, you can execute a command to perform a FULL refresh on the MQT. By issuing a **refresh** command against the MQT, DB2 selects and computes the data from the underlying tables (based on the select statement used to create the MQ)T and inserts it into the target MQT.

Assuming that you need to append large volumes of data into existing underlying tables and that you have decided to perform a FULL refresh on the MQTs after every load append, since this process does not automatically refresh the MQTs, you need perform execute a **refresh** command in order to synchronize the MQTs with the underlying tables. By issuing a **refresh** command against the MQT that is already populated, it first deletes the data from the MQT (unless you manually drop and recreate the MQT), selects and computes the entire data from the underlying tables (based on the select statement used to create the MQT), and inserts it into the target MQT.

> **Note:** When performing a FULL refresh on very large tables, it requires large temporary space for joins and to compute the data. Also, consider using NOT LOGGED INITIALLY on MQTs to avoid logging data during the refresh process. Please see additional information in “Estimating space required for MQTs” on page 216.

Table 4-9 and Table 4-10 provide a complete list of tasks required to implement FULL Refresh on REFRESH DEFERRED and REFRESH IMMEDIATE MQTs.

Table 4-9 Initial FULL refresh on refresh DEFERRED and IMMEDIATE MQTs

Step #	Step definition	Considerations
1	CREATE TABLE <affectable> (<prod_key> INTEGER NOT NULL, ... <sales> INTEGER NOT NULL, <misc> SMALLINT, …..) IN <fact_table_space>	Not Null constraint is required on surrogate key for referential integrity
2	CREATE TABLE <product> (<prod_key> INTEGER NOT NULL, <prod_name> VARCHAR (30) NOT NULL, <prod_group> VARCHAR (30)), …. IN <dimension_table_space>	Not Null constraint is required on the primary key for referential integrity
3	CREATE UNIQUE INDEX <uix_prod_key> ON PRODUCT (prod_key;	Recommended for performance purpose. See DB2 documentation for additional options on create index.
4	ALTER TABLE <product> ADD CONSTRAINT <pk_prod_key> PRIMARY KEY (prod_key);	Primary Key constraint is required for referential integrity
5	ALTER TABLE <fact_table> ADD CONSTRAINT fk_prod_key FOREIGN KEY (prod_key) REFERENCES <product> (prod_key) ON DELETE NO ACTION ON UPDATE NO ACTION NOT ENFORCED / ENFORCED ENABLE QUERY OPTIMIZATION	A FOREIGN KEY constraint is required for the DB2 optimizer to support MQTs on situation where the query does not match the number of tables defined on the MQT. It can be either ENFORCED or NOT ENFORCED.
6	DB2 Load from … insert/replace into <product> … DB2 Load from … insert/replace into <Fact_Table>	Initial Load on the Dimensions and Fact Table. **Note**: If an ENFORCED FOREIGN KEY constraint is defined, you first need to load the Dimension and at then load the fact table.

Step #	Step definition	Considerations
7a	DB2 CREATE TABLE <mqt> AS (SELECT SUM(t1.sales> AS <sales>, SUM(t1.misc> AS <misc>, COUNT(t1.misc) AS <count_misc>, COUNT(*) AS <count_of_rows>, <t2.prod_name> AS <prod_name>, <t2.prod_group> AS <prod_group>, …. FROM <fact_table t1, product t2, …> WHERE <t1.prod_key = t2.prod_key and …> GROUP BY <prod_name, …>) DATA INITIALLY DEFERRED **REFRESH DEFERRED** ENABLE QUERY OPTIMIZATION MAINTAINED BY SYSTEM IN <mqt_table_space> NOT LOGGED INITIALLY;	Create Table DDL for REFRESH DEFERRED MQTs. This example *does support* INCREMENTAL REFRESH . The COUNT(t1.misc) and COUNT(*) are required for INCREMENTAL REFRESH. **Note:** > The COUNT(t1.misc) and COUNT(*) are only required for Incremental Refresh. > For Refresh Deferred MQTs, DB2 Cube Views *does not* generate DDL for Incremental Refresh as well it does not generate the COUNT(*) and COUNT(nullable_measure_column) which are required for Incremental Refresh. >Columns that accept *null* and are listed on the GROUP BY clause can affect considerably the performance for the INCREMENTAL REFRESH; ENABLE QUERY OPTIMIZATION is required in order to the MQT be used by the DB2 Optimizer during query rewrite. The option NOT LOGGED INITIALLY is not required, however, it improves significantly the performance during FULL and INCREMENTAL refreshes on the MQTs.

Step #	Step definition	Considerations
7b	DB2 CREATE TABLE <mqt> AS (SELECT SUM(t1.sales> AS <sales>, SUM(t1.misc> AS <misc>, COUNT(t1.misc) AS <count_misc>, COUNT(*) AS <count_of_rows>, <t2.prod_name> AS <prod_name>, <t2.prod_group> AS <prod_group>, …. FROM <fact_table t1, product t2, …> WHERE <t1.prod_key = t2.prod_key and …> GROUP BY <prod_name, …>) DATA INITIALLY DEFERRED **REFRESH IMMEDIATE** ENABLE QUERY OPTIMIZATION MAINTAINED BY SYSTEM IN <mqt_table_space> NOT LOGGED INITIALLY;	Create Table DDL for REFRESH IMMEDIATE MQTs This example *does support* INCREMENTAL REFRESH . Db2 Cube Views automatically generates a COUNT(*) and a COUNT(nullable _column) for columns that accept nulls and are listed on the select clause using a SUM() function. This is a requirement for MQTs defined as REFRESH IMMEDIATE. The COUNT(t1.misc) and COUNT(*) are required for INCREMENTAL REFRESH and for REFRESH IMMEDIATE as well. **Note:** >DB2 Cube Views *does not* explicitly generate DDL with the INCREMENTAL option on the REFRESH command; However INCREMENTAL REFRESH might be automatically selected if INCREMENTAL REFRESH is possible. >Columns that accept null and are listed on the GROUP BY clause affect considerably the performance for the INCREMENTAL REFRESH The option NOT LOGGED INITIALLY is not required, however, it improves significantly the performance during FULL and INCREMENTAL refreshes on the MQTs.
8	REFRESH TABLE <mqt> NOT INCREMENTAL; Or SET INTEGRITY FOR <mqt> IMMEDIATE CHECKED	The Refresh table command performs a FULL refresh on the MQT. The Set Integrity also can be used and has the effect (FULL refresh on the MQT).
9	CREATE INDEX <index1> ON <mqt> (column list..)…	Optional you can create index on the MQT to improve query performance. Use the DB2 Index Advisor to identify required Index. Note: DB2 Cube Views generate index for the MQT.
10	REORG TABLE <mqt>;	Optional reorganize the MQT, especially if a cluster index was defined.
11	RUNSTATS ON TABLE <mqt> AND INDEXES ALL	Update the table and index statistics because they are used by the optimizer to determined the cost for query rewrite

Table 4-10 FULL refresh on refresh DEFERRED/ IMMEDIATE MQTs

Step #	Step definition	Considerations
1	DROP TABLE <mqt>; CREATE TABLE <mqt> AS (SELECT …..	To perform a full refresh on an existing MQT is recommended to drop and recreate the MQT table before perform the refresh command; Otherwise the refresh command needs to delete all rows from the MQT during the refresh process. Recreate the MQT as defined on Table-1.
2	Load from … insert/replace into <Fact_Table> …	Incremental Loads on the Fact Table.
3	REFRESH TABLE <mqt> NOT INCREMENTAL; Or SET INTEGRITY FOR <mqt> IMMEDIATE CHECKED	The refresh table command performs a FULL refresh on the MQT. The Set Integrity also can be used and has the effect (FULL refresh on the MQT).
4	CREATE INDEX <index1> ON <mqt> (column list..)…	Optional - you can create index on the MQT to improve query performance. Use the DB2 Index Advisor to identify required Index. Note: DB2 Cube Views generate index for the MQT.
5	REORG TABLE <mqt>;	Optional - reorganize the MQT, especially if a cluster index was defined.
6	RUNSTATS ON TABLE <mqt> AND INDEXES ALL	Update the table and index statistics because they are used by the optimizer to determined the cost for query rewrite

INCREMENTAL maintenance of refresh IMMEDIATE MQTs

MQTs defined as refreshed IMMEDIATE can be incrementally maintained via inserts/updates/delete on which changes (delta) are propagated immediately or it can also be incrementally maintained via the `Load Insert` utility on which changes (delta) are not propagated immediately.

Since `Load Insert` is a more common implementation for Business Intelligence applications, we are only covering the process to incrementally maintain MQTs using the Load Insert option.

Figure 4-33 shows the process flow and the major tasks required for the Incremental refresh process on MQTs defined as refresh IMMEDIATE:

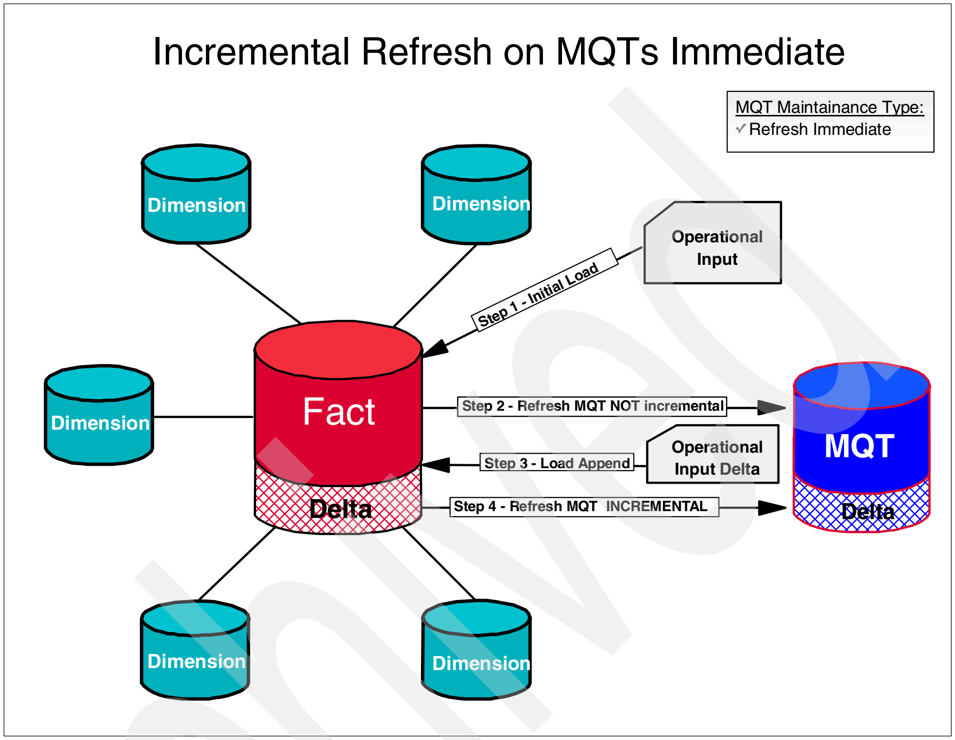


Figure 4-33 Incremental refresh on MQts IMMEDIATE

After you perform step-1 (Initial load on the base tables) and create the MQT, you need to execute a command to perform a FULL refresh not INCREMENTAL (Step-2) on the MQT. By issuing a refresh command (step-2) against the MQT, it computes the data from the underlying tables (based on the select Statement used to create the MQT) and inserts into the target MQT.

Assume that you need to append more data into existing base tables as well synchronize the MQTS. After you append the data into the underlying tables (step-3), you need to execute a command to incrementally refresh the MQT. By issuing a refresh command with the Incremental option (step-4) against the MQT, the delta information is selected and computed from the underlying tables (based on the select statement used to create the MQT) and inserted into the MQT table. This process can either insert new rows or update existing rows into the MQT.

Table 4-11 shows all required steps that you need to perform in order to incrementally refresh an MQT defined as refresh IMMEDIATE.

Table 4-11 Steps for INCREMENTAL refresh on refresh IMMEDIATE MQTs

Step #	Step definition	Considerations
1	ALTER TABLE <mqt> ACTIVATE NOT LOGGED INITIALLY;	This is not a required step; however, for large MQTs or when you are incrementally adding large volumes of data, by disabling the log for the MQT you can improve significantly the INCREMENTAL refresh process. If for any reason the refresh process fails, the MQT might become invalid and you need to Drop, Re-create and perform a FULL refresh on the MQT again.
2	CREATE INDEX <index_for_incremental> ON <mqt> (<mqt_group_by_columns>)…	To have good performance on the INCREMENTAL process, you need to define a non-unique index that guarantees uniqueness on the columns that are listed on the group by clause of the MQT. You also need to minimize the size of this index in order to avoid a big overhead during the refresh process.
3	RUNSTATS ON TABLE <mqt> AND INDEXES ALL;	Update the table and index statistics because they are used by the optimizer to determined the cost for query rewrite
4	Load from … insert into <Fact_Table> …	Incremental Load data on the Fact Table. Note: For INCREMENTAL refresh only the Insert option is supported.
5	SET INTEGRITY FOR <Fact_Table> IMMEDIATE CHECKED;	This command remove the “Check Pending” status from Fact Table;
6	REFRESH TABLE <mqt> INCREMENTAL; Or SET INTEGRITY FOR <mqt> IMMEDIATE CHECKED INCREMENTAL	The refresh table command with the INCREMENTAL option consider only the appended data from the underlying tables (fact table and dimensions) . This option can produce either insert of new rows on the MQT as well can update existing rows.
7	DROP INDEX <index_for_incremental>; CREATE INDEX <index1> ON <mqt> (column list..)…	Optional, You can remove the index created for the INCREMENTAL refresh process if you think it is not used by any other query; Optional you can create additional indexes on the MQT to improve query performance. Use the DB2 Index Advisor to identify required Index. Note: DB2 Cube Views generate index for the MQT.
8	REORG TABLE <mqt>;	Optional reorganize the MQT, especially if a cluster index was defined.
9	RUNSTATS ON TABLE <mqt> AND INDEXES ALL	Update the table and index statistics because they are used by the optimizer to determined the cost for query rewrite

INCREMENTAL maintenance of refresh DEFERRED MQTs

For a refresh DEFERRED MQT to be incrementally maintained, it *must* have a staging table associated with it. This temporary table is used to store the incremental data until an explicitly refresh command is executed against the MQT. The staging table associated with an MQT is created with the CREATE TABLE SQL statement.

MQTs defined as refreshed DEFERRED can be incrementally maintained via inserts/updates/delete on which changes (delta) are propagated immediately to the staging table, or it can also be incrementally maintained via the **`Load Insert`** utility on which changes (delta) are not propagated immediately to staging tables and to the MQTs.

Figure 4-34 shows the process flow and the major tasks required for the incremental refresh process on MQTs defined as refresh DEFERRED.

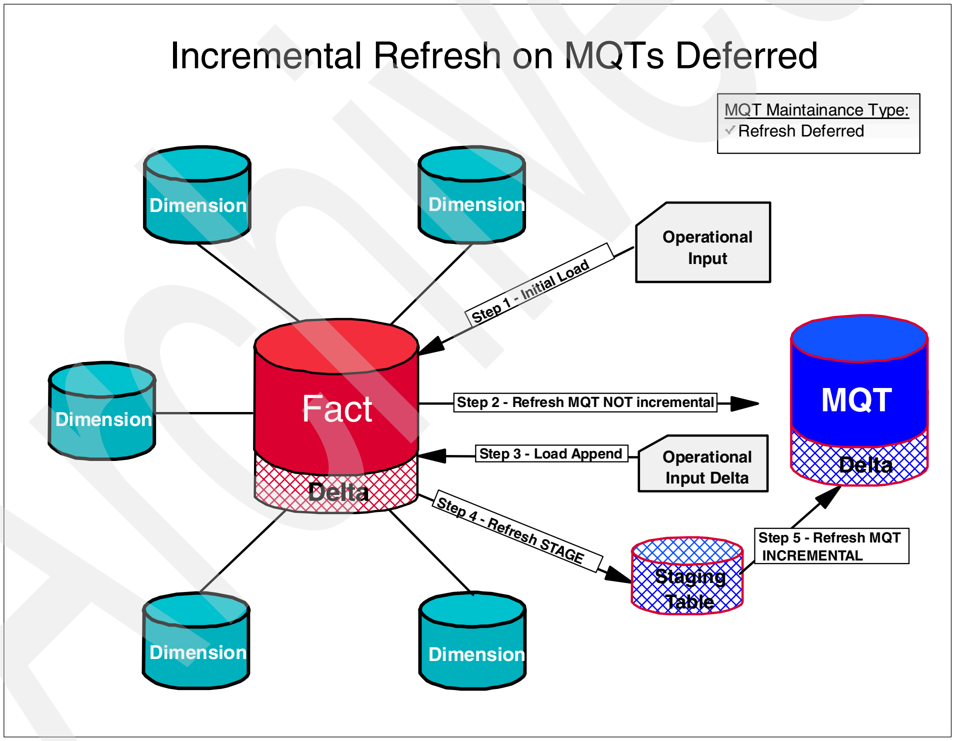


Figure 4-34 Incremental refresh on MQTs DEFERRED

After you perform step-1 (initial load on the base tables) and create the MQT, you need to execute a command to perform a FULL refresh (Step-2) on the MQT. By issuing a **`refresh`** command (step-2) against the MQT, it computes and aggregates the initial information from the underlying tables (based on the select statement used to create the MQT) and inserts the data into the MQT table.

Assume you need to append more data into existing base tables as well synchronize the MQTS. After you append the data into the underlying tables (step-3), you need to execute a command to incrementally refresh the MQT. By issuing a **`Set Integrity`** command with the Incremental option (step-4) against the staging table, it selects and computes the delta information from the underlying tables (based on the select statement used to create the MQT) and inserts it into the staging table.

On a separate step, you need to execute an additional **`refresh`** command against the MQT using the Incremental option. It extracts the data from the staging table and populates the MQT. This process can either insert new rows or update existing rows into the MQT.

Table 4-12 shows all required steps that you need to perform in order to incrementally refresh an MQT defined as refresh DEFERRED.

Table 4-12 INCREMENTAL refresh on refresh DEFERRED MQT

Step #	**Step definition**	**Considerations**
1	CREATE TABLE <mqt_staging> FOR <mqt> PROPAGATE IMMEDIATE;	Create the staging table to temporary store the appended data on the underlying tables. Since we are using Load Insert into the underlying tables, propagate IMMEDIATE option does effect the INCREMENTAL refreshes process.
2	ALTER TABLE <mqt> ACTIVATE NOT LOGGED INITIALLY;	This is *not* a required step; however, for large MQTs or when you are incrementally adding large volumes of data, by disabling the log for the MQT can improve significantly the INCREMENTAL refresh process. If for any reason the refresh process fail, the MQT might become invalid and you need to Drop, Re-create and perform a FULL refresh on the MQT again.

Step #	Step definition	Considerations
3	CREATE INDEX <index_for_incremental> ON <mqt> (<mqt_group_by_columns>)…	To have good performance on the INCREMENTAL process, you need to define a *non-unique* index that guarantees uniqueness on the columns that are listed on the group by clause of the MQT. You also need to minimize the size of this index in order to avoid a big overhead during the refresh process.
4	Load from … insert into <Fact_Table> …	Incrementally Load data on the Fact Table. Note: For INCREMENTAL refresh only the Insert option is supported.
5	SET INTEGRITY FOR <Fact_Table> IMMEDIATE CHECKED;	This command removes the “Check Pending” status from Fact table; It is required when the load was performed without the ALLOW READ ACCESS option.
6	SET INTEGRITY FOR <mqt_staging> IMMEDIATE CHECKED INCREMENTAL;	This step is to transfer the appended information from base table to the staging table. It select and compute the appended data from the underlying tables and insert into the staging table;
7	REFRESH TABLE <mqt> INCREMENTAL; Or SET INTEGRITY FOR <mqt> IMMEDIATE CHECKED INCREMENTAL	This step is to select the data from the staging table and populate the MQT table; It can produce either insert of new rows on the MQT as well can update existing rows.
8	DROP INDEX <index_for_incremental>; …. CREATE INDEX <index1> ON <mqt> (column list..)…	Optional you can remove the index created for the INCREMENTAL refresh process. Unless you want to keep it to be used by any other query; Optional you can create additional indexes on the MQT to improve query performance. Use the DB2 Index Advisor to identify required Index. Note: DB2 Cube Views generate index for the MQT.

Step #	Step definition	Considerations
9	REORG TABLE <mqt>;	Optional reorganize the MQT, especially if a cluster index was defined.
10	RUNSTATS ON TABLE <mqt> AND INDEXES ALL	Update the table and index statistics because they are used by the optimizer to determined the cost for query rewrite

4.9.6 Limitations for INCREMENTAL refresh

Limitations are specific to materialized query tables that are defined as refresh IMMEDIATE. The same limitations apply to queries used to create refresh DEFERRED tables associated with staging tables that we want to refresh incrementally.

- The query should also not contain correlated subqueries that require being part of the update.
- The query must involve at least one base table. A query using only table functions or common table expressions cannot be used.
- The query must not contain recursion.
- The query must not involve DB2 catalog tables. This is because updates to the catalog tables bypass the normal INSERT, UPDATE, or DELETE mechanisms for regular user tables.
- The query should not contain Outer Join.
- The query should not contain any functions that have side-effects or are non-deterministic.
- The query should not contain subqueries that require to be evaluated for each updated row.
- The query should not contain SELECT DISTINCT or DISTINCT in aggregate functions.
- The query should not contain complex SELECT list items, HAVING predicates or GROUP BY expressions that cannot be derived from lower-level aggregate data.
- The query should not contain STDDEV, VAR, MAX, MIN, AVG, COVARIANCE,CORRELATION.
- The query must contain COUNT(*) or COUNT_BIG(*).
- The query's GROUP BY items cannot be derived from aggregate functions in a lower query.
- All GROUP BY columns in the query must be in the SELECT list.

- In a partitioned environment the query must have the partition key as a subset of the GROUP BY items.
- The query should not contain any special registers like CURRENT TIMESTAMP.

4.10 MQT tuning

Since an MQT is just another table, any normal tuning technique applies, such as having RUNSTATS currently updated and having appropriated indexes defined.

The following are some recommendations that you need to carefully evaluate to implement in your production environment:

- Create index on MQT columns that are referenced on the `where` clause for most used queries (use the DB2 Index Advisory to help you identify appropriate indexes).
- Evaluate the possibility to use unique index with include columns on dimension tables. It can speed up retrieval time from these tables during the INCREMENTAL and FULL refresh of MQTs.
- Create a non-unique index on the MQT columns that guarantee uniqueness of rows in an MQT. In case of a partitioned MQT, the partition key should be a subset of the columns described above.
- Do not create an index on the staging table, since such indexes degrade the performance of appends to the staging table.
- For partition tables, make sure that you partition the staging table according to the partitioning of the MQT to promote collocated joins.
- Refresh of MQTs consumes CPU, I/O, and buffer pool resources, which ultimately impacts other users contending for the same resources. Refresh resource consumption can be reduced by combining multiple MQTs in a single refresh statement, since DB2 uses “multiple-query optimization” to share joins and aggregations required of each MQT in order to reduce the resource consumption.
- Reorganize tables (regular and MQTs) after incremental load, insert, and delete of large amounts of data.
- Collect statistics for underlying tables and on the MQTs:
  - After performing INCREMENTAL refresh on MQTs
  - After performing FULL refresh on MQTs
  - Perform any changes on existing MQTs (such as create, alter, or remove index, alter table)

4.11 Configuration considerations

When using MQTs, two of the main questions will be:

- How to estimate the memory required for MQTs
- How to estimate the storage required for MQTs

4.11.1 Estimating memory required for MQTs

In general, MQTs can help to reduce the amount of memory required in a query environment because they can help reduce the need for sorting. By avoiding sort data, you are avoiding use of a large amount of memory for sort purposes. However, there is a certain time where large amounts of memory can be required. It happens during the refresh of the MQT. Depending on the size of the tables involved on the MQT, the large amount of memory is required for SORTHEAP size.

The following recommendation apply only to non-clustered (or single node DB2) configurations:

- SORTHEAP:
  - SORTHEAP is usually very important for MQT REFRESH.
  - The size of the sortheap allocation depends on the complexity of the MQT.
  - Look at the Explain plan of the full select used by the MQT to estimate sortheap requirements and size accordingly....
  - If you have the memory to consume (32 bit versus 64 bit), this is a good candidate for over allocation.
  - You need to ensure that the dbm cfg sort heap threshold parameter is sized appropriately to support the sortheap specified.
  - Note that the sortheap allocation will usually be significantly less in the runtime environment.
- STATEMENT HEAP:
  - An MQT refresh may require a lot of statement heap. Otherwise, you may get an error like “statement too complex”.

4.11.2 Estimating space required for MQTs

DB2 Optimization Advisor provides you with a good estimate on the size for the MQT based on sampling of the data as already discussed in “Specify disk space and time limitations” on page 161. If you need to change the DDL (we don’t recommend it) for the MQT, you need to recalculate the space required for your MQT.

An MQT is a table, which contains pre-computed results whose definition is based on the result of a query execute against the underlying tables. Based on this, the size for an MQT is estimated using the same calculation as for a base table. For information on how to estimate the size of the MQT, please refer to Chapter 2 in the redbook, *Up and Running with DB2 UDB ESE: Partitioning for Performance in an e-Business Intelligence World,* SG24-6917.

Besides the space required for the MQTs, you also need additional *temporary* space for joins and aggregations.

The temporary tablespace is an important consideration during full REFRESH of MQTs. If the MQT has a lot of measures, rollups, and/or group by elements, more tempspace is generally required.

If the MQT's size estimate provided by DB2 Cube Views is very large, it is probably an indication that you may need more tempspace.

The following formula helps you to estimate the TEMP space required for refresh on an MQT:

```
TEMPSPACE Required = (# of pages required) * (pagesize) where
   # of pages required) = (Total # of rows in the MQT) * (10+10) / 2560)

      (10 + 10) = Is the size stored in the TEMP per MQT row. It is defined
      twice in situation where it needs to the DELETE of old data and
      INSERT of new data in the MQT.

      2560 is a number based on the fact that we store 256 rows per page
      (assuming 256 slots in a page of 10 bytes each).
```

Notes:

1. The Delete is referring to deleting the old data in the MQT. If it was an initial population, you do not need to account for this.
2. Note that the result of the formula gives the number of pages and the number 2560 is independent of the page size. Depending on the page size, we need to compute disk space accordingly. An 8K-page size would require double amount of disk compared to a 4K-page size.
3. Particular care needs to be taken at the catalog node because the refresh process of MQTs also uses TEMP space on this node. If you have a separate the catalog node from the data nodes with very little TEMP tablespace, you can have problems to perform full REFRESH on MQTs. Make sure you add additional TEMP space on this node to avoid any problems during the refresh process.

4.12 Conclusion

Building efficient MQTs is a difficult and time consuming task if done by hand. As an alternative, we show that using the OLAP Center's Optimization Advisor is a tool that not only eliminates the laborious MQT construction job for the DBA, but also does a very good job of constructing them based on the knowledge provided in the OLAP Center cube model and accompanying cubes.

We provide an insight into the working of the Optimization Advisor as well as MQTs in general and show what design considerations are beneficial to their construction.

By briefly analyzing some examples from our sample data mart, we show how MQTs may be deployed, ensuring that they are used, and ultimately how they may highly benefit queries.

Part 3

Access dimensional data in DB2

In this part of the book we describe how to access multidimensional data and metadata in DB2 through front-end tools such as OFFICE Connect and QMF for Windows, and through business partners tools such as Business Objects, Cognos, MetaStage, MetaIntegration metadata bridges, and MicroStrategy.

We also explain how to start building a Web service with OLAP capabilities.

5

Metadata bridges overview

This chapter introduces and summarizes some of the partner tools that provide a metadata bridge to DB2 Cube Views.

5.1 A quick summary

The way in which tools and applications can interface with the DB2 Cube Views metadata is via an API. The API is implemented as a DB2 stored procedure named db2info.md_message(). This API passes XML documents both in and out for all of its arguments.

The purpose of this section is to provide an introduction to those tools that can access the DB2 Cube Views metadata directly via the API, and to document the metadata bridges that make use of the API. For more information, please refer to the bridge article on the Developer Domain:

http://www7b.software.ibm.com/dmdd/library/techarticle/0305poelman/0305poelman.html

The bridges that are documented are the ones that were available for testing at the time of writing this book.

The advantage of having implemented the API as a DB2 stored procedure is that it becomes language neutral. Any programming language that can talk to DB2 can invoke this stored procedure.

To use the API, the calling program must construct XML documents to pass into the stored procedure. The program will also need to parse the XML that is returned by the stored procedure.

One of the considerations that a developer will need to decide upon when developing a bridge is whether the bridge will call the API to read the metadata, or read in exported metadata XML files — in the case of a pull from DB2 Cube Views. In the case of a push to DB2 Cube Views, the consideration is whether to call the API to create the metadata, or write the metadata to an XML file.

The advantage of using XML files is:

- The bridge can run independently of DB2 Cube Views, such as on another client or server box.

The disadvantages of using XML files are:

- When reading DB2 Cube Views XML, you cannot be sure that the metadata is valid and in sync with the relational schema in DB2.
- When producing DB2 Cube Views XML, you cannot be sure that the metadata can be successfully imported later on.

The advantages of using the API are:

- The bridge can use the VALIDATE operation to make sure any metadata it reads from DB2 is valid.

- The bridge can read additional information about referenced tables and columns by querying the DB2 system catalog tables.
- The bridge can see all metadata in DB2.

The disadvantage of using the API is:

- It may take longer to implement the bridge because you need to add the code to invoke the DB2 Cube View API. The program will need to produce operation XML and to parse the response XML.

As can be seen in Table 5-1, all partners to date have chosen to develop their bridges by reading from and writing to XML files.

Table 5-1 IBM business partner bridge implementations

Business Partner	**Product**	**Supported in product version/ release**	**One-way or two-way bridge**	**XML or API**	**Map to/from cube model/ cube**	**Support for incremental changes**
Ascential	MetaStage	V7.0	Two-way	XML	Cube and Cube Model Note 1	Yes
Business Objects	BO Universal Metadata Bridge	SP1® of BO Enterprise 6	Pull from DB2 Cube Views	XML	Cube and Cube Model	No
Cognos	DB2 Dimensional Metadata Wizard	MR1 of Cognos Series 7 Version 2	Pull from DB2 Cube Views	XML or CLI	Cube Model	No
IBM/Hyperion	Integration Server Bridge	DB2 OLAP Server V8.1	Two-way	XML	Cube and Cube Model	No
Meta Integration	MI Model Bridge	MIMB V3.1	Two-way	XML	Cube Model	No
MicroStrategy	MicroStrategy	MicroStrategy 7i Release 4	Pull from DB2 Cube Views	XML	Cube Model	No

Notes:

Note 1: Dependent upon source and target products.

These bridges and their functionalities will be constantly changing over time, and you will have to check directly with the tool partners for availability of these functions, specifically for two-way and incremental changes to functions.

6

Accessing DB2 dimensional data using Office Connect

Spreadsheets provide an intuitive and powerful front end to represent and manipulate business information. The main problem with most spreadsheets is their inability to seamlessly transfer information between the spreadsheet and a relational database like DB2. Often the users end up writing complex macros to do this. This process is buggy, expensive, difficult to maintain, and frequently beyond the skill set of the regular user.

IBM DB2 Office Connect helps users overcome current limitations by providing a simple GUI-based patented process that enables information in a spreadsheet to be transferred seamlessly to multiple databases.

This chapter describes how to access multidimensional data in DB2 using IBM DB2 Office Connect Analytics Edition.

6.1 Product overview

The Office Connect Analytics Edition is an add-in tool that you use with Microsoft® Excel that enables you to connect to a DB2 data source and retrieve Online Analytical Processing (OLAP) metadata to use in your Office Connect/Excel applications.

For example, you can obtain a multidimensional view of aggregate data for applications such as budgeting, cost allocation, financial performance analysis, and financial modeling. If you are working with sales data, you can analyze the data and then display revenue for:

- All years, or for specified periods of time, such as one year, one quarter, or one month
- All products or specified products
- All sales outlets or specified outlets
- All countries or specified countries or regions within a country

6.2 Architecture and components

IBM DB2 Office Connect Analytics Edition V4.0 is:

- Designed only for use with DB2 Cube Views
- Only available for Excel
- Implemented as a spreadsheet add-in in Excel. That is, you can add this tool to the Excel spreadsheet via the Menu option **Tools -> Add-ins**.
- Using ODBC for connectivity to the DB2 database. This means that you have to configure the database in DB2 that you connect as an ODBC system data source (on the Windows client) using either the ODBC Administrator or DB2 Configuration Assistant
- Using pivot table services to access data.

In order to use Office Connect to exploit the new OLAP aware DB2, you should have the following products installed:

- IBM DB2 Office Connect Analytics Edition V4.0
- Excel - Office2000 or XP and above
- DB2 UDB V8.1 FP2+ and above
- DB2 Cube Views V8.1 FP2+

The main components of the IBM DB2 Office Connect are shown in Figure 6-1.

- **Project Manager:** This is the Office Connect main console. It allows you to manage the various elements of your Office Connect project and bind cubes to your worksheets.
- **Connection Manager and Cube Import wizard:** This allows you to connect to a DB2 data source and select the cubes you want to import into Office Connect.

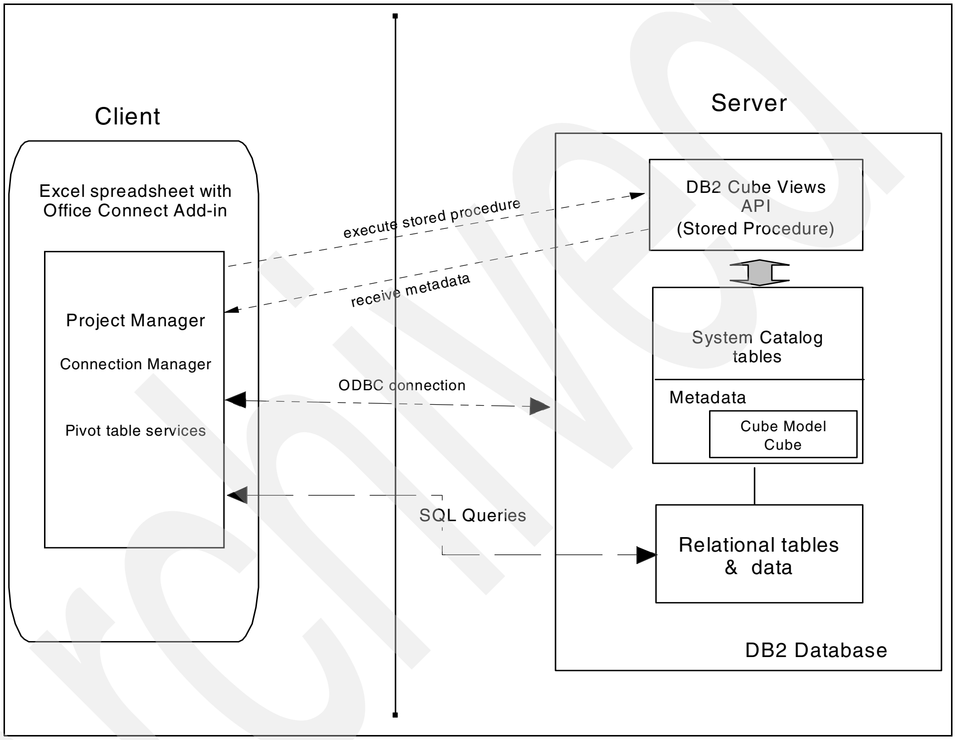


Figure 6-1 Architecture of Office Connect environment

> **Attention:** If you have both of Essbase and Office Connect add-ins active then Essbase owns the double click and the right click mouse actions.

IBM DB2 Office Connect accesses OLAP metadata in DB2 through the DB2 Cube Views API (implemented as a stored procedure in DB2).

The actual data retrieval (with the help of pivot table services) is through SQL queries.

6.3 Accessing OLAP metadata and data in DB2

Using Office Connect to access multidimensional metadata and data in DB2 is illustrated in the following sections.

The process flow is represented in Figure 6-2.

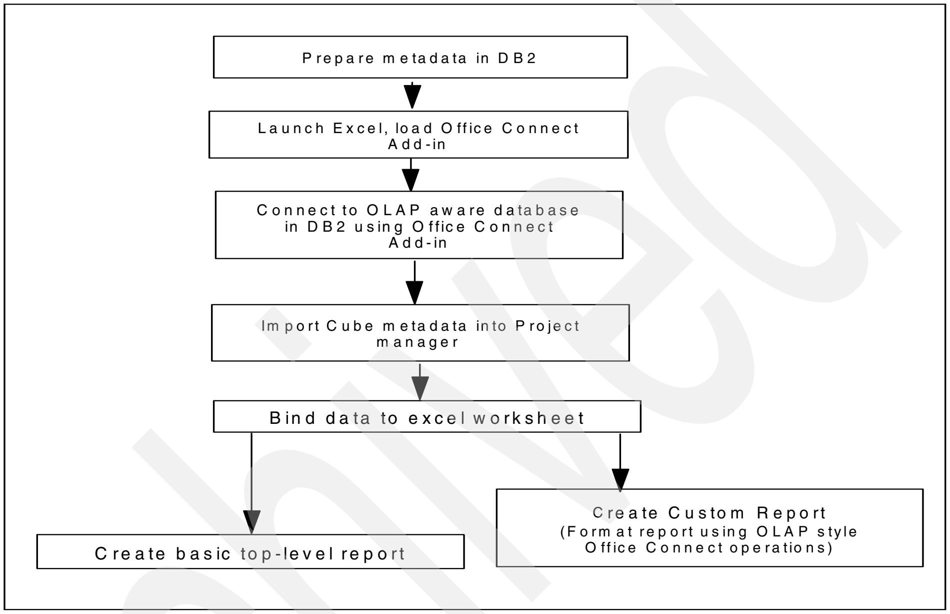


Figure 6-2 Process flow chart for Office Connect

6.3.1 Prepare metadata

First, the relational database in DB2 that is being used as the data source must be prepared for DB2 Cube Views (see 3.3.2, "Preparing the DB2 relational database for DB2 Cube Views" on page 86)

Second, a cube model needs to be created in DB2 Cube views. Office Connect is designed to work with only DB2 Cube Views cubes. Therefore, cubes (a subset of the cube model) also needs to be defined.

6.3.2 Launch Excel and load Office Connect Add-in

Once installed, Office Connect is available from the Excel menu bar (see Figure 6-3).

Note: We only used Excel XP through the examples and figures provided in this chapter. You may figure out some slight changes when using Excel 2000.

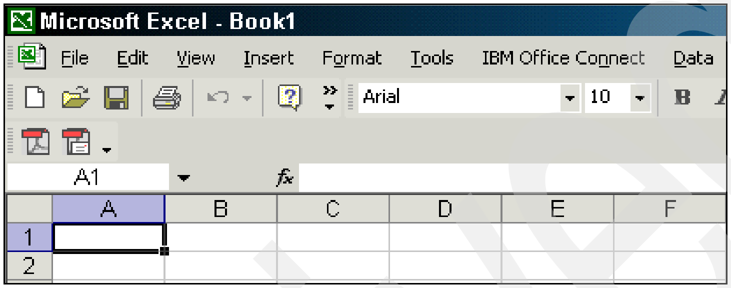


Figure 6-3 Office Connect

The Add-in can also be enabled/disabled from **Tools -> Add-ins** in Excel (see Figure 6-4).

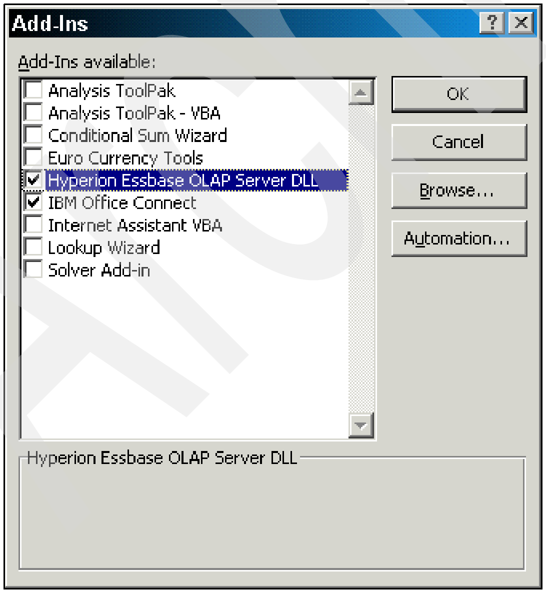


Figure 6-4 Office Connect Add-in

6.3.3 Connect to OLAP-aware database (data source) in DB2

From the Excel menu, select **IBM Office Connect -> Project Manager**. Provide the name of the ODBC data source (or select from the drop-down list), username, and password in the Connection Manager window in Figure 6-5.

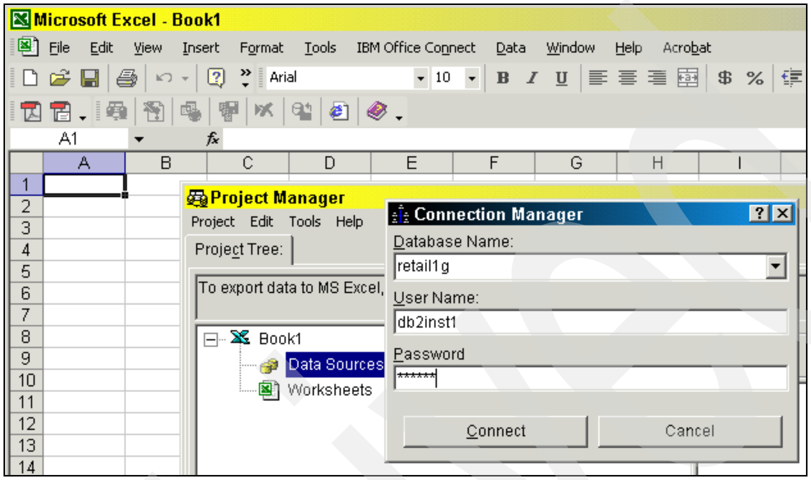


Figure 6-5 Provide connection information

6.3.4 Import cube metadata

1. Successful connection to the data source launches a Cube Import wizard (see Figure 6-6).

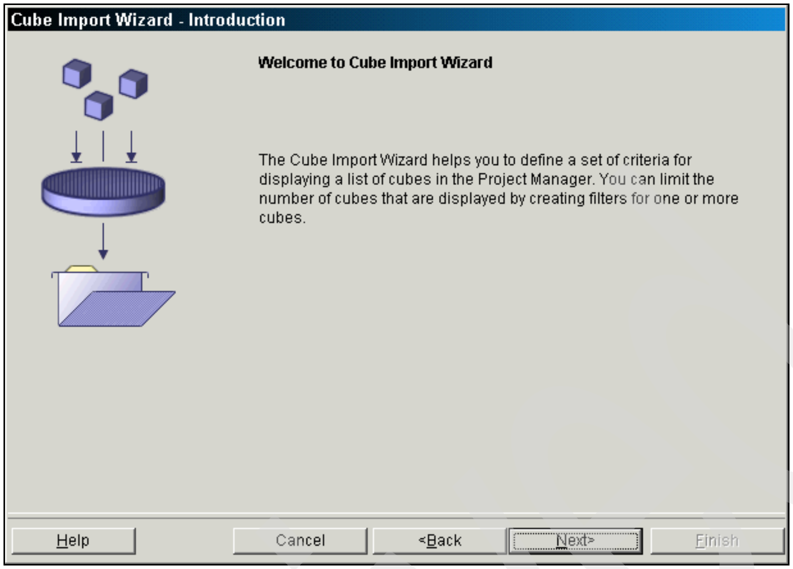


Figure 6-6 Cube Import Wizard

Select **Next** to select the cube metadata that you want to import (see Figure 6-7).

2. You have the option to select more than one cube at this time. If you select the cube model, then all the cubes defined within that cube model are also selected. The cube model 's name is retrieved to put the cube in context. The cube model has no other semantics within an Office Connect retrieval.

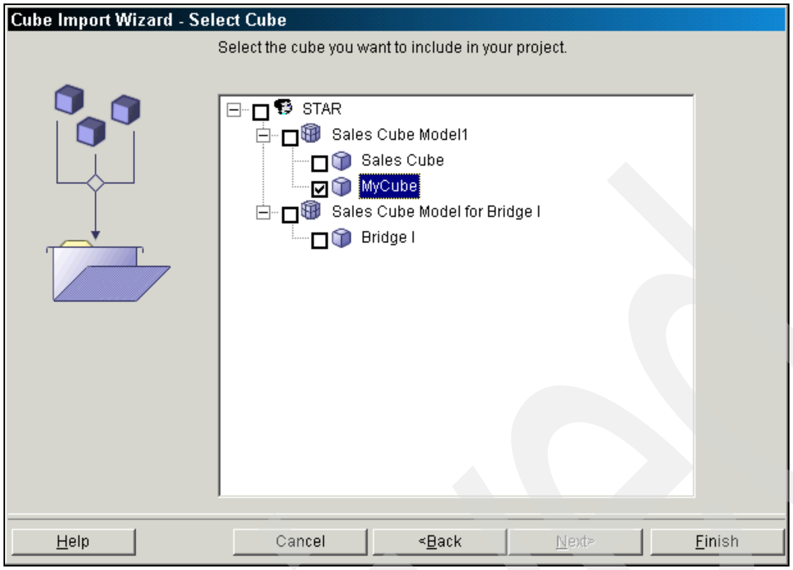


Figure 6-7 Select cube(s) for import

3. Select **Finish** to finish importing the metadata in to the Project Manager (see Figure 6-8).

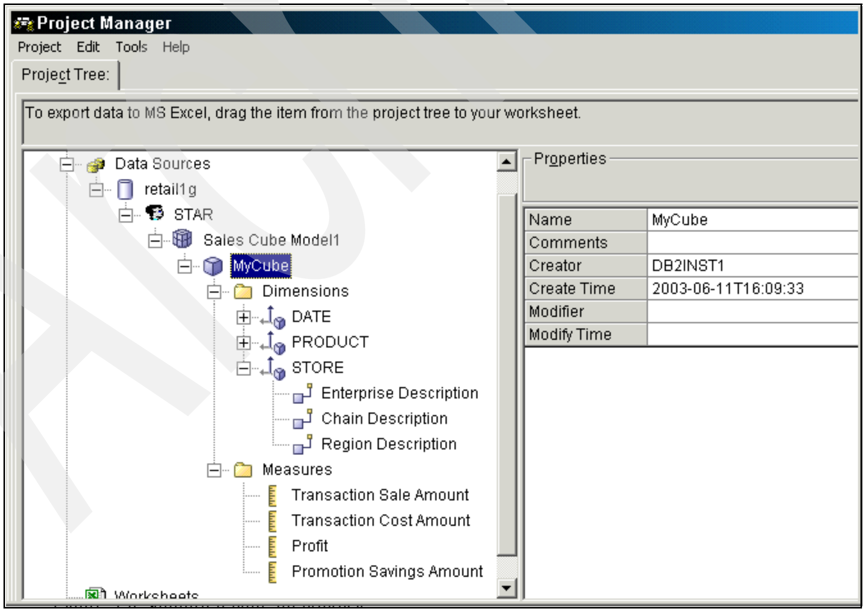


Figure 6-8 Imported cube metadata

You will see the selected dimensions and the members selected from its hierarchy and the selected measures.

6.3.5 Bind data to Excel worksheet

Binding data to an Excel worksheet means that you are retrieving data related to the cube into the spreadsheet.

To bind data to the Excel worksheet, you can right-click a selected cube and select export data to Microsoft Excel (see Figure 6-9) or left-click a selected cube and drag and drop it on to the Excel spreadsheet.

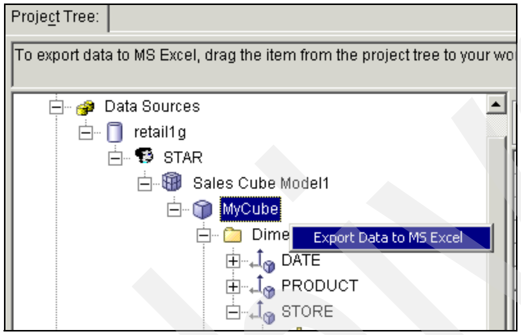


Figure 6-9 Export data to Microsoft Excel

You have the option to select the sheet that you want to export to (see Figure 6-10).

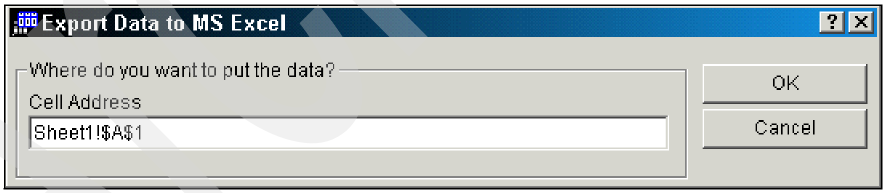


Figure 6-10 Select Excel sheet

Click **OK** to export the data to the Excel spreadsheet (see Figure 6-11). The spreadsheet report will now show data at the topmost level for all dimensions.

Tip: Selecting a cube and exporting data to Excel spreadsheet will produce a basic top-level report. That is, data is presented at the highest level of aggregation. Alternatively, you can perform a right-click operation on dimensions that you choose and measures that you are interested in and export data for that dimension or measure to the worksheet to produce a more customized report. This will present data at a lower level of aggregation and the user can further drill down to the level that he is allowed to.

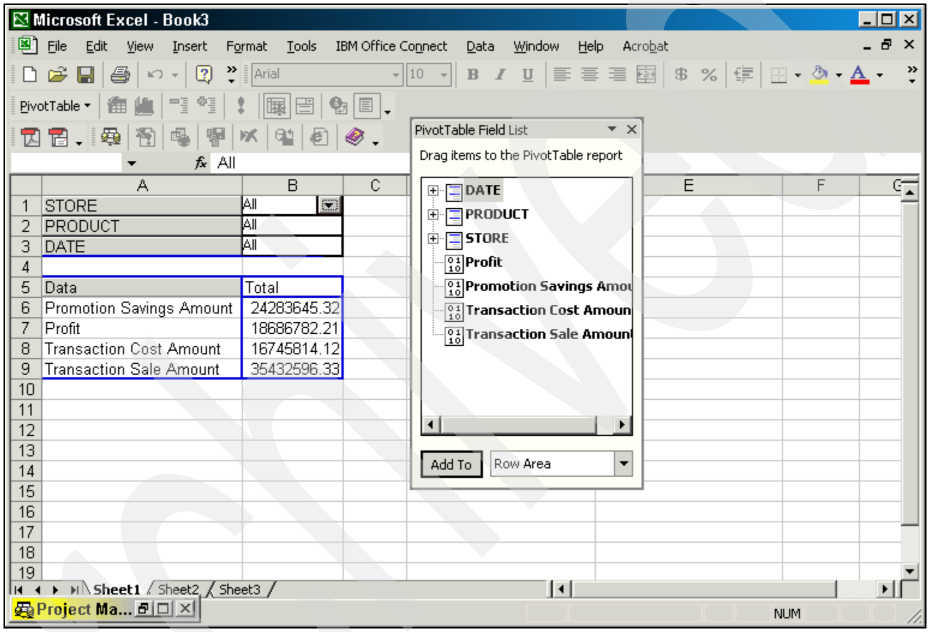


Figure 6-11 View data in Excel spreadsheet

In Figure 6-11, *STORE*, *PRODUCT* and *DATE* are the dimensions. The fields that are below (that is, **Data**) are the measures. This default report gives the Sales data measures for all stores, all product and for all years. This is the top most level of the aggregation. The level to which you can drill down to depends on the cube definition — in terms of how many measures you have subsetted for this cube and hierarchy levels for the dimensions.

You will also have a pop-up window called the Pivot Table Field List. This is discussed in 6.4, “OLAP style operations in Office Connect”

6.4 OLAP style operations in Office Connect

The Pivot Table Field List is essentially the area in the spreadsheet enclosed by the dimensions. So, in Figure 6-11, the table defined by rows 1 -9 and columns A, B is the pivot table. This area obviously changes as and when you drill down on members or remove columns or rows from a report.

You can add dimensions/members to the report from the Pivot Table Field List.

> **Note:** Pivot Table Field List option (window) is only available under Microsoft Office Excel XP. In Office 2000 there is no pivot table list and once a cube or component's of a cube dropped to create pivot table, Excel's pivot table wizard and Office Connect Project Manager are the only windows available to the user to add/remove the cube component.

The Pivot Table Field List can also be invoked from the Office Connect tool bar (see Figure 6-12)

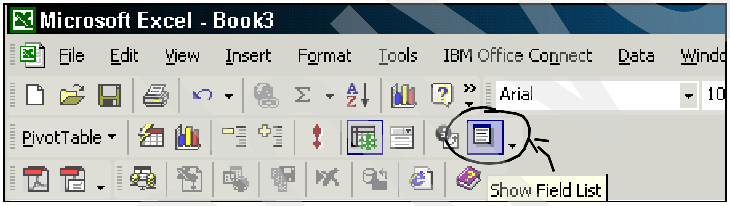


Figure 6-12 Show Pivot table field list

Office Connect uses these Excel pivot tables to view the data and also to pivot it. A pivoting action moves a dimension from a row to a column and vice versa.

Here are some of the common actions that you can perform in the Office Connect workspace.

Drag and drop

This type of action performed from the pivot table field list or from the pivot table itself.

To drag a dimension from either from the field list or the pivot table, simply left-click the name and release the mouse button to the location that you desire.

> **Note:** You can only drag and drop the dimensions not the member names.

For example, from the default report that we got (Figure 6-11), left-click *STORE* and release it to the left of **Data** to get a report as shown in Figure 6-13.

	A	B	C
1			
2	PRODUCT	All	
3	DATE	All	
4			
5	STORE->Enterprise Description	Data	Total
6	Enterprise	Promotion Savings Amount	24283645.32
7		Profit	18686782.21
8		Transaction Cost Amount	16745814.12
9		Transaction Sale Amount	35432596.33
10			

Figure 6-13 Drag and drop

Swap rows and columns

Again, this is a mouse action, selecting the dimension from a row, dragging and dropping it to a column, or conversely, dragging a dimension from a column and dropping it to a row.

> **Tip:** When using the mouse to move a dimension, a little spreadsheet icon appears and moves with the mouse pointer. It has a rectangle to represent the data, a long rectangle to the left to represent the row headings and a wide rectangle at the top to represent the column headings.
>
> There is a blue rectangle to represent where the dimension will be dropped:
>
> - If this blue rectangle is on the left, then the dimension will be dropped as a row heading.
> - If the blue rectangle is above the column heading rectangle, then the dimension will be dropped as a page heading.
> - If the blue rectangle is below the column heading rectangle, in the data area, then the dimension will be dropped as a column heading.

Double-click a cell to drill down

Simple double-click mouse operation on a cell (containing the dimension or member) in the pivot table to drill down on the hierarchy for that dimension.

You can also drill down using the **Show Detail** button on the tool bar.

Use member selection window to navigate or filtering

This a way to filter or hide certain members from a report. Single-click the dimension which will make a down-facing arrow available (see Figure 6-14).

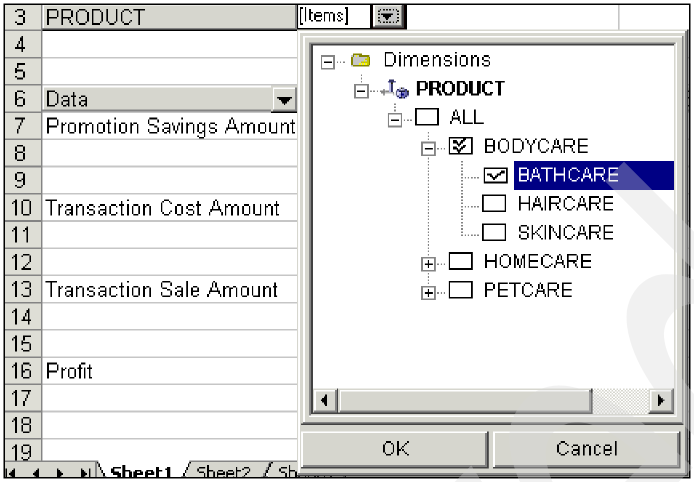


Figure 6-14 Member selection or filtering

From this member selection window, you can navigate and filter or deselect the members that you would like to remove from your report. You can use the help text for additional information.

Drilling up

Use the same member selection window to just select the upper level member that you require while deselecting the lower level members.

If you drag the dimension from the row or column back to the top, then again this will drill up for that dimension but to the top-most level.

You can also use the **Hide Detail** button on the tool bar.

Objects and pivot table report manipulation

Right-click a dimension in a row or a column and select *Order*. You can select *Move Left, Move Right, Move to beginning, Move to end,* as applicable (and valid). This changes the dimension's order and position (row or column) within a report.

Delete objects from report such as dimension or measure

Right-click the dimension and select *Hide Dimension* to remove information about that dimension from the report. To hide a measure, select the measure, right-click and choose *Hide Level*.

Different layout

The layout of the report can be changed using the pivot table layout wizard. Actions like drag and drop, swapping rows and columns, removing dimensions or uninteresting measures from the report can be performed and data for the modified report will be refreshed in a single step. This avoids making changes one by one which causes a refresh (via a SQL query) to occur after every change.

To launch the PivotTable layout wizard, right-click the pivot table and select **Wizard...** (see Figure 6-15)

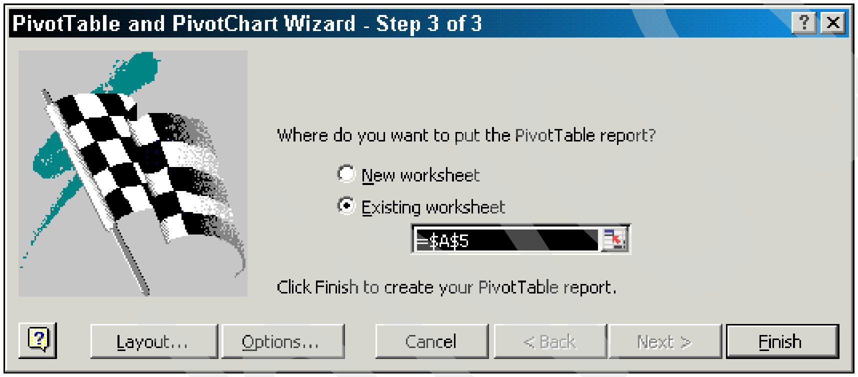


Figure 6-15 PivotTable wizard

Select **Layout** to launch the layout wizard (see Figure 6-16). Here you can drag out the uninteresting measures, swap dimensions between rows and columns, or remove dimensions from the report.

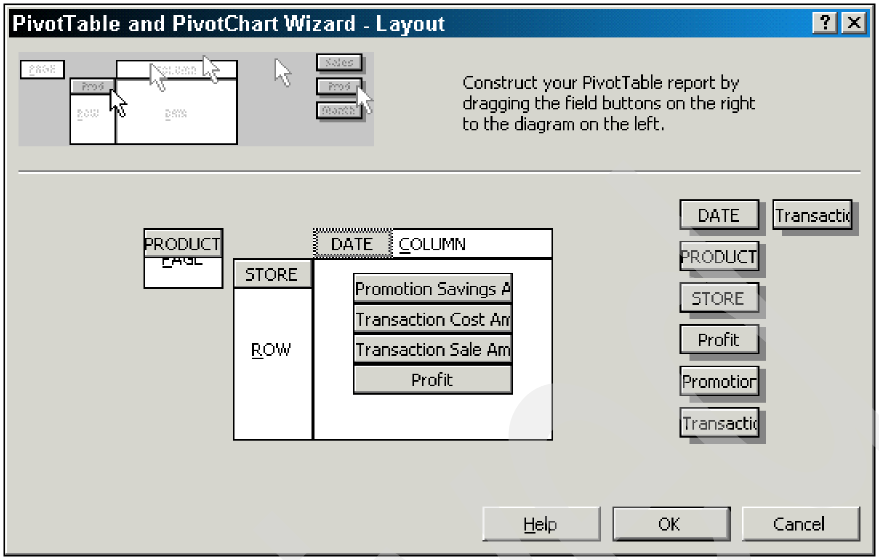


Figure 6-16 Layout wizard

Chart

Right-click anywhere on the pivot table report and select *Pivot Chart* to display the report as a chart (see Figure 6-17). You can use the *Chart Wizard* to select the type of chart.

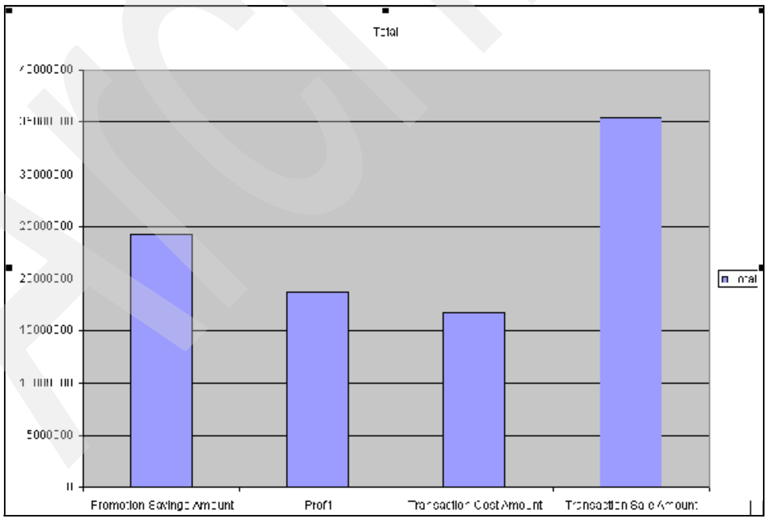


Figure 6-17 PivotChart

Format report

Right-click the pivot table report and select **Format Report** and choose one of the types available (there are totally 12). This will display the report in the style selected.

6.5 Saving and deleting reports

Use normal **File** -> **Save** operation from Excel to save the report.

You also the option to save the data source connection information from the project manager (**Project** -> **Save data source to file**)

With Office Connect, simply opening the saved report in Excel (without having saved data source information) does not need supplying the data source connection information again for that report or worksheet.

To delete a Office Connect report, right-click the report in Project Manager and select **Delete**.

6.6 Refreshing data

To refresh data for a static Office Connect workbook or worksheet (that is, a worksheet that does not reflect current data), select from the Excel menu bar, **IBM Office Connect** -> **Data** -> **Refresh Workbook** or **Refresh Worksheet.** This causes a (ODBC) connection to the data source and data is retrieved (based of SQL query it uses).

6.7 Optimizing for better performance

In Office Connect, the action that is being performed most frequently is a drill down operation. That is, the user is always first presented with the data at the top most level. The user then drills down to the sections of the data that is most interesting.

Office Connect requires to have cubes defined to mimic the SQL queries that you expect when using Office Connect.

Office Connect drill down queries may benefit better performance when using MQTs recommended by the DB2 Cube Views Optimization Advisor. If you selected the drill down query type, the Optimization Advisor has optimized for the top levels of the dimensions, as the user is typically starting at the top and drilling down. Without MQTs all of the aggregation will be done at query time for each query at the higher levels of the dimensions and therefore take longer.

Question: How do you check if the report SQL query from Office Connect is exploiting the MQT once it is built?

Answer: Extract the SQL query from Office Connect (by enabling SQLDebug Trace) and use it in DB2 Explain. This will show whether the query is being routed to the MQT or not.

The subsections 6.7.1, "Enable SQLDebug trace in Office Connect" and 6.7.2, "Use DB2 Explain to check if SQL is routed to the MQT" explain how to do this.

6.7.1 Enable SQLDebug trace in Office Connect

1. From Excel File menu, choose **Properties**
2. In the Properties window, choose the **Custom** tab (see Figure 6-18)
   a. Type **SQLDebug** in the **Name** field.
   b. Type **True** in the **Value** field.
   c. Click the **Add** button.
   d. Click **OK**.

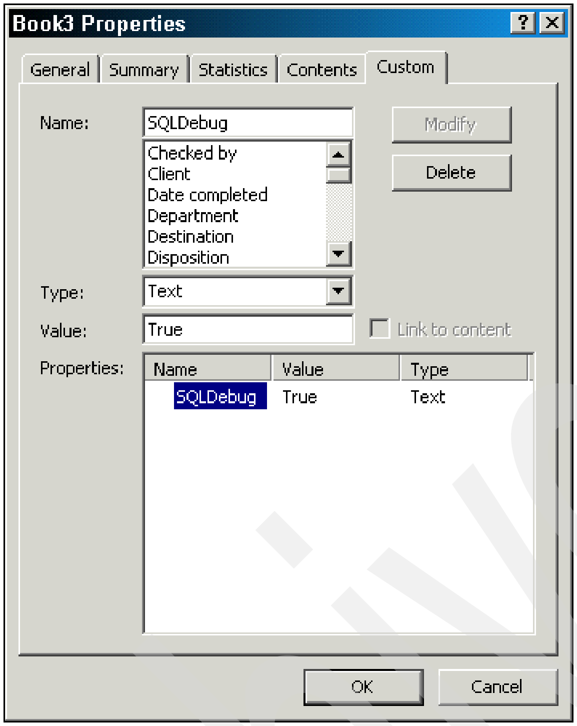


Figure 6-18 SQLDebug

This enables SQL trace in Office Connect.

Now, any type of drill action should give the SQL that the query is using. After performing a query in Office Connect, an SQLDebug window will appear that displays the SQL that has just been submitted.

Save the SQL using copy/paste to perform DB2 Explain. Example 6-1 shows a SQL query that was used for retrieving the top most level of a cube.

Example 6-1 SQL for a retrieval in Office Connect

```
Select ' ',' ',' ',' ',' ',' ',' ',SUM("STAR"."CONSUMER_SALES"."TRXN_SALE_AMT")
as "TRXN_SALE_AMT",SUM("STAR"."CONSUMER_SALES"."TRXN_COST_AMT") as
"TRXN_COST_AMT",SUM("STAR"."CONSUMER_SALES"."TRXN_SALE_AMT" -
"STAR"."CONSUMER_SALES"."TRXN_COST_AMT") as
"Profit",SUM("STAR"."CONSUMER_SALES"."PROMO_SAVINGS_AMT") as
"PROMO_SAVINGS_AMT"  From "STAR"."CONSUMER_SALES"
```

6.7.2 Use DB2 Explain to check if SQL is routed to the MQT

From the Control Center for DB2 UDB on Windows (**Start** -> **Program** -> **IBM DB2** -> **General Administration Tools** -> **Control Center**), connect to the database and from the right-click option, choose **Explain SQL**.

Use the SQL that you saved from the SQLDebug trace to obtain the access plan graph that DB2 uses. This graph will show whether DB2 will choose the MQT for the data retrieved by this query (see Figure 6-19).

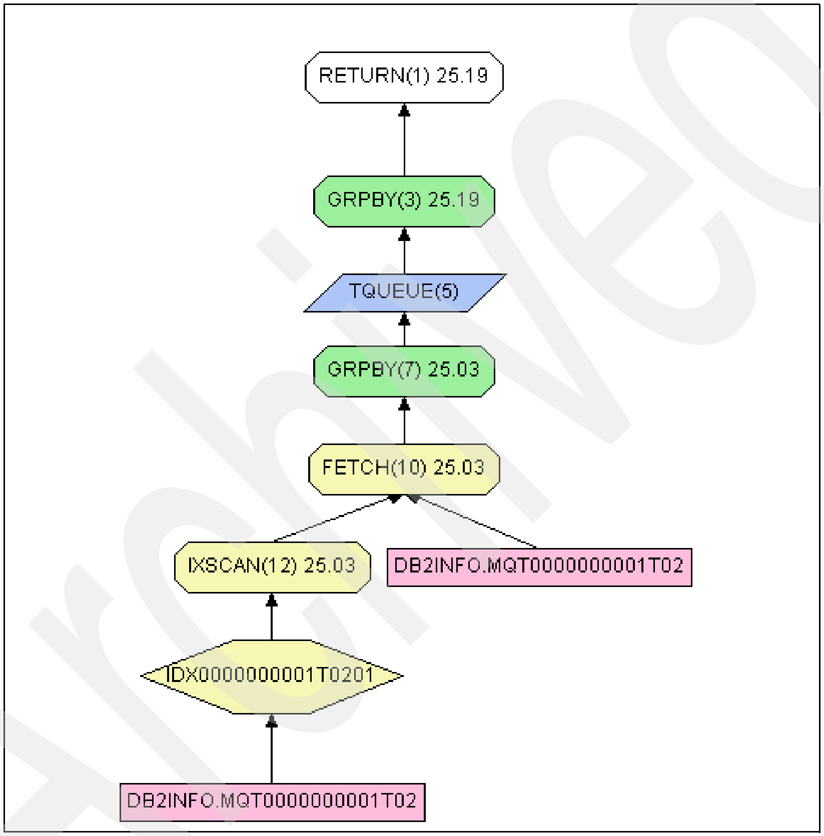


Figure 6-19 Access plan graph

The following scenario illustrates the benefit of using MQTs (in other words, optimizing the cube model in DB2 Cube Views) for Office Connect.

6.7.3 Scenario demonstrating benefit of optimization

Suppose we are producing a report that shows sales data for *Skincare* type of products for the *West* region, year-to-date (see Figure 6-20).

	A	B
1	STORE	West
2	PRODUCT	SKINCAR
3	DATE	All
4		
5	Data	Total
6	Promotion Savings Amount	434134.64
7	Profit	344716.61
8	Transaction Cost Amount	377202.79
9	Transaction Sale Amount	721919.4

Figure 6-20 Customized report

From a basic top-level report (refer to Figure 6-11) that we start with, we first drill down to show data only for the *West*' region. Example 6-2 shows the SQL used to retrieve the data for this drill down action.

Example 6-2 Drill down to West region SQL

```
   SUM("STAR"."CONSUMER_SALES"."TRXN_SALE_AMT") as
   "TRXN_SALE_AMT",SUM("STAR"."CONSUMER_SALES"."TRXN_COST_AMT") as
   "TRXN_COST_AMT",SUM("STAR"."CONSUMER_SALES"."TRXN_SALE_AMT" -
   "STAR"."CONSUMER_SALES"."TRXN_COST_AMT") as
   "Profit",SUM("STAR"."CONSUMER_SALES"."PROMO_SAVINGS_AMT") as "PROMO_SAVINGS_AMT"
   From  "STAR"."CONSUMER_SALES" inner join "STAR"."STORE" ON
   "STAR"."CONSUMER_SALES"."STORE_ID"="STAR"."STORE"."IDENT_KEY" Where
   (("STAR"."STORE"."ENTERPRISE_DESC"='Enterprise                          ' And
   "STAR"."STORE"."CHAIN_DESC"='Chain Retail Market              ' And
   "STAR"."STORE"."REGION_DESC"='West') ) Group by
   "STAR"."STORE"."ENTERPRISE_DESC","STAR"."STORE"."CHAIN_DESC","STAR"."STORE"."REGION_D
   ESC"
```

Without having implemented any MQT, the time (in timerons) for this query was 362,752.31.

We then drill down on products to include data only for *SKINCARE* (see Example 6-3).

Example 6-3 Drill down to SKINCARE products

```
Select ' ','
','STAR"."PRODUCT"."DEPARTMENT_DESC","STAR"."PRODUCT"."SUB_DEPT_DESC","STAR"."STORE"."EN
TERPRISE_DESC","STAR"."STORE"."CHAIN_DESC","STAR"."STORE"."REGION_DESC",SUM("STAR"."CONS
UMER_SALES"."TRXN_SALE_AMT") as
"TRXN_SALE_AMT",SUM("STAR"."CONSUMER_SALES"."TRXN_COST_AMT") as
```

```
"TRXN_COST_AMT",SUM("STAR"."CONSUMER_SALES"."TRXN_SALE_AMT" -
"STAR"."CONSUMER_SALES"."TRXN_COST_AMT") as
"Profit",SUM("STAR"."CONSUMER_SALES"."PROMO_SAVINGS_AMT") as "PROMO_SAVINGS_AMT"  From
"STAR"."CONSUMER_SALES" inner join "STAR"."PRODUCT" ON
"STAR"."CONSUMER_SALES"."ITEM_KEY"="STAR"."PRODUCT"."IDENT_KEY"  inner join
"STAR"."STORE" ON "STAR"."CONSUMER_SALES"."STORE_ID"="STAR"."STORE"."IDENT_KEY" Where
(("STAR"."STORE"."ENTERPRISE_DESC"='Enterprise                         ' And
"STAR"."STORE"."CHAIN_DESC"='Chain Retail Market                ' And
"STAR"."STORE"."REGION_DESC"='West') ) AND
(("STAR"."PRODUCT"."DEPARTMENT_DESC"='BODYCARE                            ' And
"STAR"."PRODUCT"."SUB_DEPT_DESC"='SKINCARE                            ') ) Group by
"STAR"."PRODUCT"."DEPARTMENT_DESC","STAR"."PRODUCT"."SUB_DEPT_DESC","STAR"."STORE"."ENTE
RPRISE_DESC","STAR"."STORE"."CHAIN_DESC","STAR"."STORE"."REGION_DESC"
```

Again, without having any MQTs implemented, the explain SQL in DB2 shows that the time (in timerons) for this query was 49.184.94

See Figure 6-21 and Figure 6-22 for the access plan graphs for these two queries.

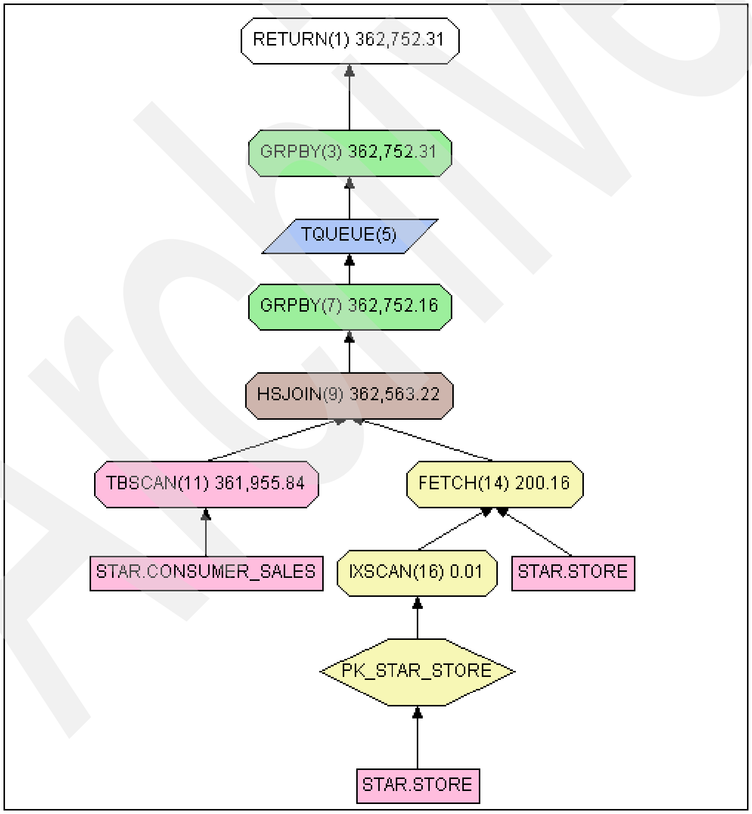


Figure 6-21   Access plan graph - STORE

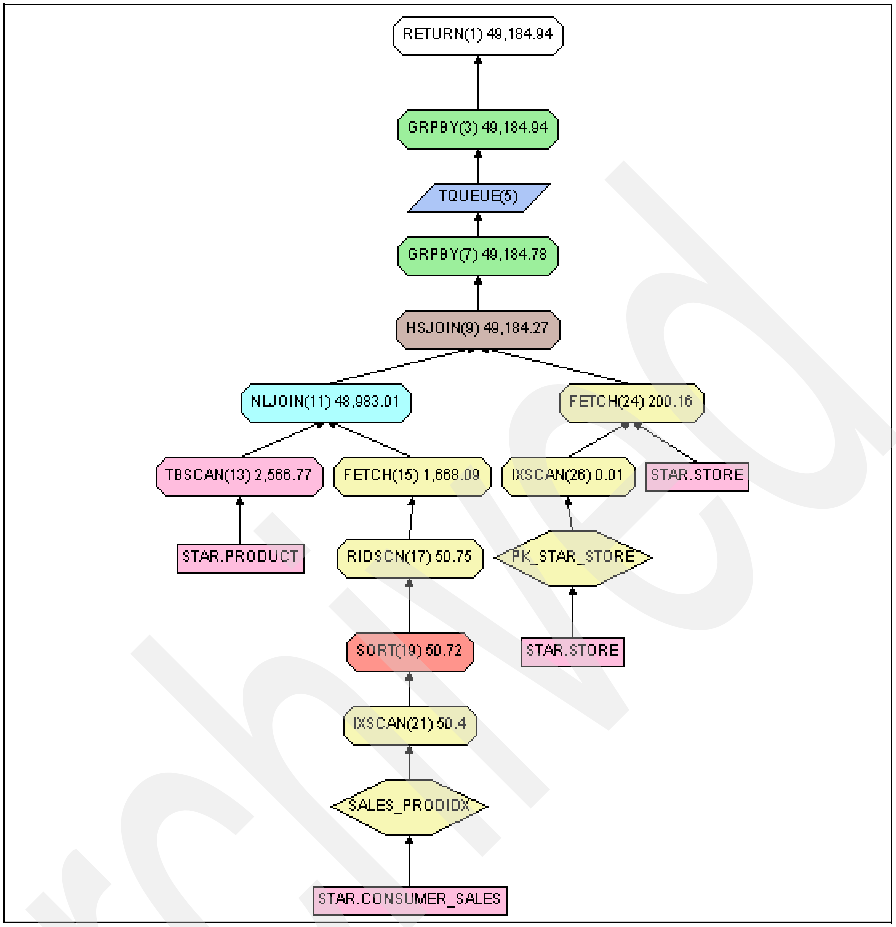


Figure 6-22 Access plan graph - PRODUCT

After using the Optimization Advisor to create MQTs, the corresponding times (to drill down on *STORE* and *PRODUCT* are 25.19 and 25.19

As obvious from this scenario, there is a significant performance benefit in optimizing the cube model for drill down queries, as summarized in Table 6-1.

Table 6-1 Drill down times

Drill down time for a sample query (in timerons)	**Without MQT**	**With MQT**
Query1	362,752.31	25.19
Query2	49,184.94	25.19

7

Accessing dimensional data in DB2 using QMF for Windows

This chapter presents certain deployment scenarios for the IBM QMF for Windows product when accessing DB2 Cube Views, and describes some new OLAP capabilities in QMF for Windows.

7.1 QMF product overview

IBM DB2 Query Management Facility (QMF) has a rich product history spanning two decades. Its foundation is built upon strong query and reporting facilities that provide the end-user with seamless access to data that is stored in any database in the IBM DB2 family of databases. There are several DB2 QMF family product offerings. These products include:

- DB2 QMF for TSO/CICS®: Runs in the TSO and CICS environments
- DB2 QMF High Performance Option (HPO): Provides enhanced performance management and administrative capabilities for TSO/CICS environments.
- DB2 QMF for Windows: Provides an easy-to-use graphical interface on the windows platforms
- DB2 QMF for WebSphere®: Provides a three-tier QMF architecture requiring only a thin Web browser client.

The scope and breadth of the QMF Product Family portfolio has provided for the continuance of integration of the latest technologies. This chapter will discuss the new integration offerings of QMF for Windows with multidimensional data analysis through the use of OLAP technology.

7.2 Evolution of QMF to DB2 Cube Views support

Customer surveys have revealed that some customers use QMF to perform analysis similar to informal application of OLAP. Users would run a series of queries and generate data reports. They would examine these reports and based upon their findings, they would decide on the next set of queries to run and so forth. The process would continue iteratively until the objective of the analysis was complete. At first glance, it may seem odd that a user would choose QMF to do OLAP-like processing when several high-end OLAP tools are available on the market. This oddity can be addressed with two explanations:

1. The QMF user may not even be familiar with the concept of OLAP processing. The user is simply trying to obtain the answers to their business questions in the most straightforward and descriptive manner through QMF queries and reports.
2. The database administrator (DBA) or user may be familiar with the concepts behind OLAP but may not possess the necessary skills, resources, or business initiative to invest in OLAP products.

In either case, the main point is that QMF was currently fulfilling the need of some customers to do primitive OLAP-like functions in addition to providing for their query and reporting needs.

7.3 Components involved

QMF for Windows is a front-end tool accessing directly DB2 Cube Views through the stored procedure implementing the DB2 Cube Views API (see Figure 7-1). In order to exploit the new OLAP functionality, installation of the following software products is required:

- QMF for Windows v7.2f or above
- DB2 Cube Views v8.1
- DB2 Universal Database Version 8.1 FixPack 2
- Supported server systems:
  - On Microsoft Windows:
    - Windows NT® 4 or Windows 2000 32-bit
  - On AIX:
    - AIX Version 4.3.3 32-bit, AIX 5L™ 32-bit, or AIX 5L 64-bit
  - On Linux:
    - Linux Red Hat 8 (kernel 2.4.18/ glibc 2.2.93-5) 32-bit, or Linux SuSE 8.0 (kernel 2.4.18/ glibc 2.2.5) 32–bit

      For the latest information on distribution and kernel levels supported by DB2, go to:

      `http://www.ibm.com/db2/linux/validate`
  - On Sun Solaris Operating System:
    - Solaris 8 32-bit, or Solaris 9 32-bit
- Supported client component:
    - Windows NT 4, Windows 2000, or Windows XP 32–bit

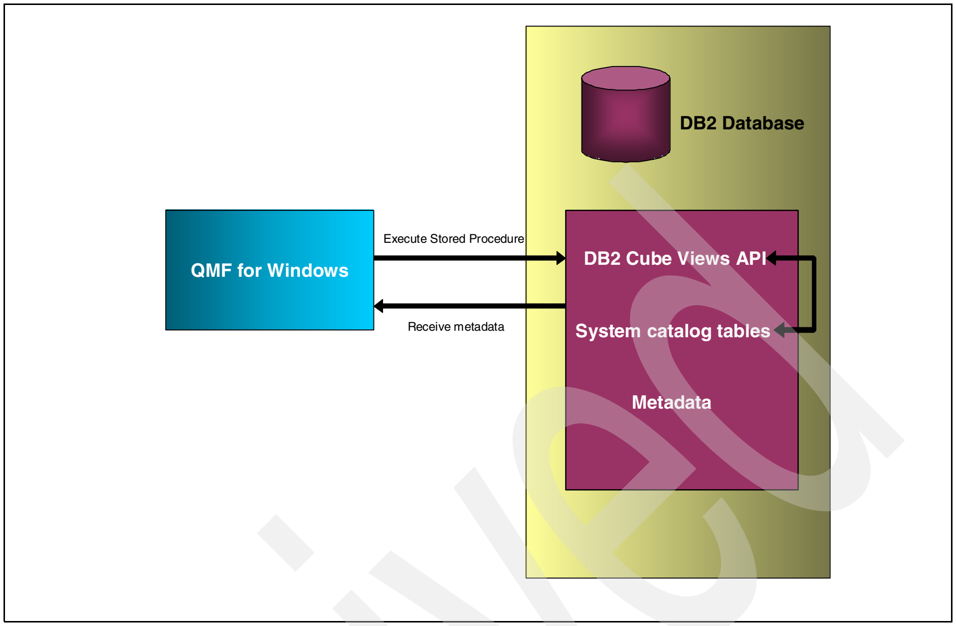


Figure 7-1 Components required for QMF for Windows with DB2 Cube Views

All communications between QMF for Windows and DB2 Cube Views occur via XML.

7.4 Using DB2 Cube Views in QMF for Windows

QMF for Windows allows for the creation of several types of QMF objects:

- Query
- Form
- Procedure
- List
- Job
- Map

Prior to the release of QMF for Windows v7.2f, the types of queries supported were SQL, Prompted and Natural Language. The introduction of a new OLAP query object type was the necessary feature that brought the OLAP construct of a cube into the QMF data space (see Figure 7-2). To create a new OLAP query, select File->New... to display the new object window.

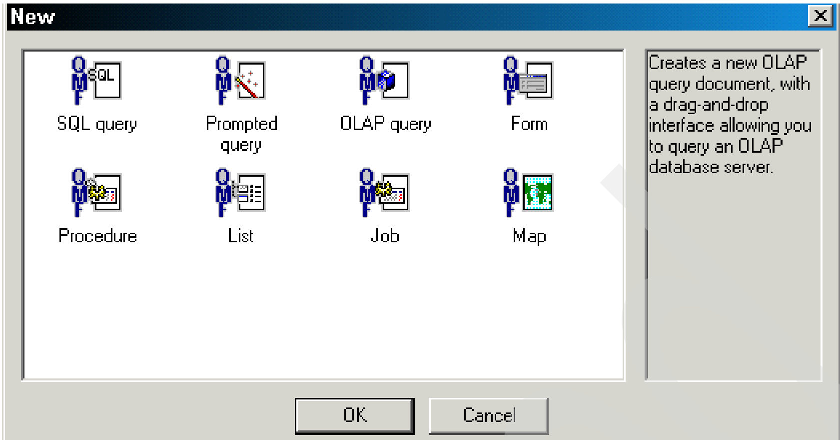
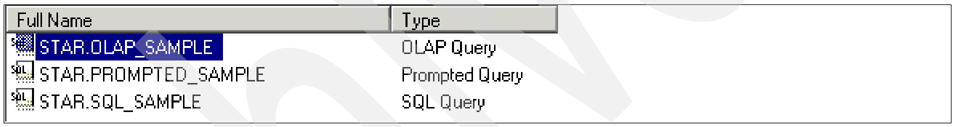


Figure 7-2 New object window for QMF for Windows

The new OLAP query can be saved at the server level in the QMF control tables as type *OLAP Query.*



Full Name	Type
STAR.OLAP_SAMPLE	OLAP Query
STAR.PROMPTED_SAMPLE	Prompted Query
STAR.SQL_SAMPLE	SQL Query

Figure 7-3 List of queries saved at the server

This new OLAP query object provides a drag-and-drop interface enabling the user to build an OLAP query. The building of the OLAP query begins with the use of the OLAP query wizard after OLAP query is selected for the **New** window.

7.4.1 QMF for Windows OLAP Query wizard

The OLAP Query wizard proceeds step-by-step through the OLAP query definition process:

1. Select a server. The servers listed are databases defined in QMF for Windows Administrator (see Figure 7-4).

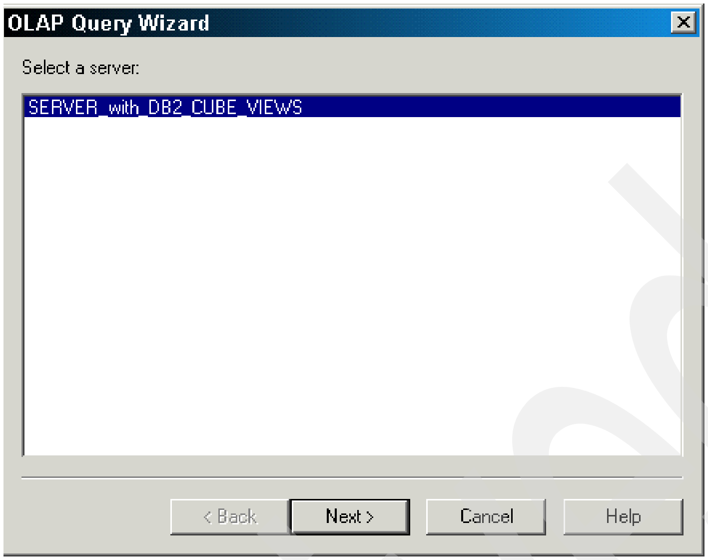


Figure 7-4 OLAP Query wizard server

If DB2 Cube Views has not been installed or properly configured on the server selected, an error message will occur as shown in Example 7-1.

Example 7-1 Unsupported cube error message

```
QMF for Windows cannot communicate with the specified database in order to
retrieve OLAP metadata. This might be because the database does not support IBM
DB2 Cube Views. For more information, press F1.
```

2. Choose how to sort the cube list: schema or model (see Figure 7-5). Upon completion of this step, QMF for Windows retrieves and sorts the list of cubes by invoking the stored procedure of DB2 Cube Views to obtain the existing cube definitions from the DB2 Cube Views catalog tables. If no cubes are found on the server, an error message will occur.

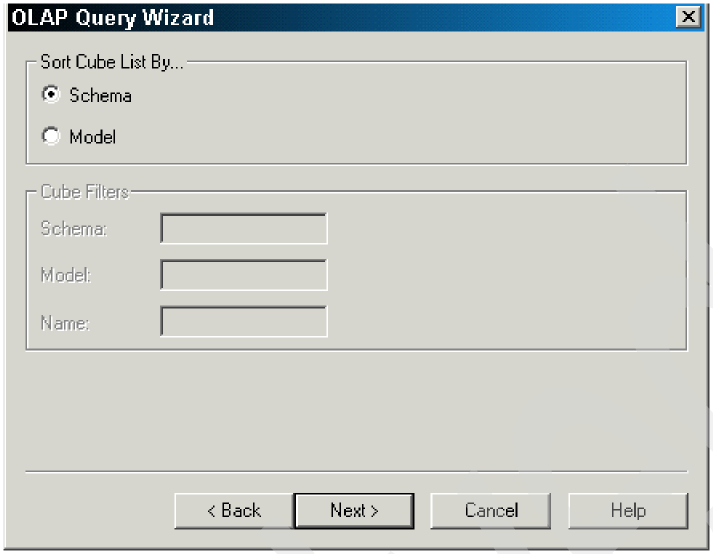


Figure 7-5 OLAP Query wizard sort

a. The cube list sorted by schema begins with the server name, followed by each schema name that contains one or more cubes and concludes with all cubes owned by the schema name (see Figure 7-6).

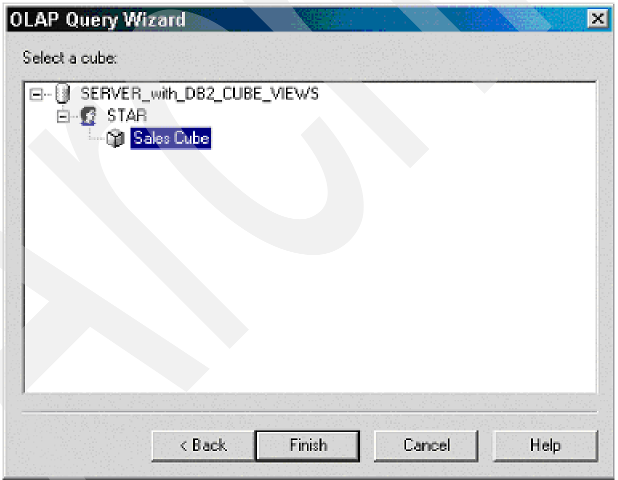


Figure 7-6 OLAP Query wizard cube schema

b. The cube list sorted by model begins with the server name, followed by each cube model that contains one or more cubes and concludes with all cubes derived from the cube model (see Figure 7-7).

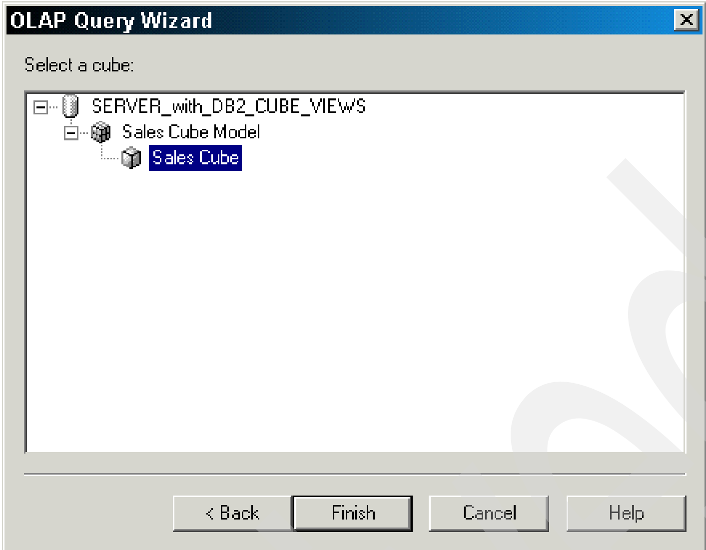


Figure 7-7 OLAP Query wizard cube

3. Select the cube to be associated with the QMF OLAP Query.

   Upon selection of the cube, the complete description of all associated metadata is retrieved.

7.4.2 Multidimensional data modeling

QMF for Windows v7.2f contains an advanced graphical user interface that provides the viewing and manipulating of multidimensional data. These are the three major enhancements to the graphical user interface representation:

- Object Explorer
- Layout Designer
- Query Results View

These enhancements provide a powerful environment for business intelligence analysis.

7.4.3 Object Explorer

The Object Explorer is a tool bar that can be floating or docked on the right or left vertical panel of the QMF for Windows interface. The Object Explorer uses a tree control structure approach to display Dimension and Measure metadata objects. Business names for metadata objects as defined in DB2 Cube Views are used in the Object Explorer for easy recognition by the user. By DB2 Cube Views metadata definition, cubes do not use multiple hierarchies because cube dimensions allow only one cube hierarchy per cube dimension. Therefore, there is one hierarchy per dimension in the cube and the hierarchy levels are displayed in the Object Explorer as shown in Figure 7-8.

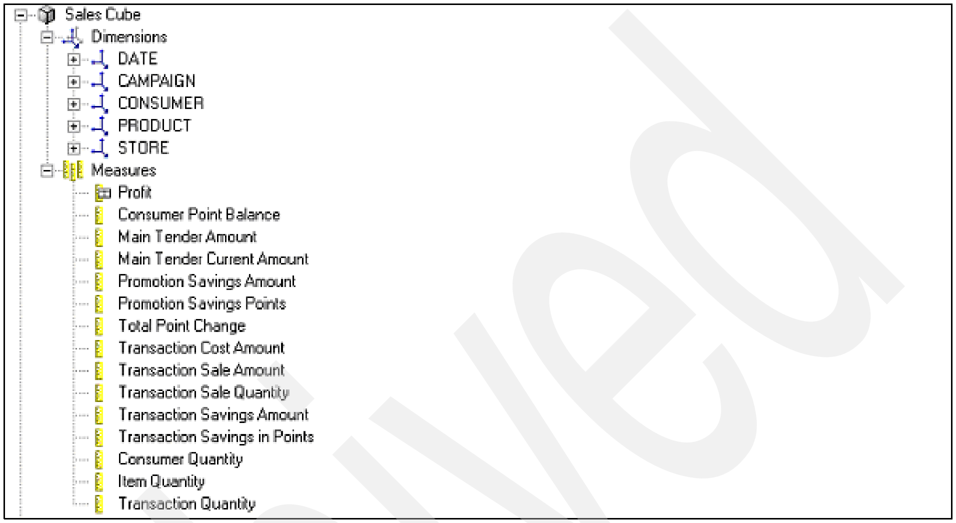


Figure 7-8 View of the cube in Object Explorer

Hierarchy levels are listed in order of precedence from highest to lowest as shown in Figure 7-9.

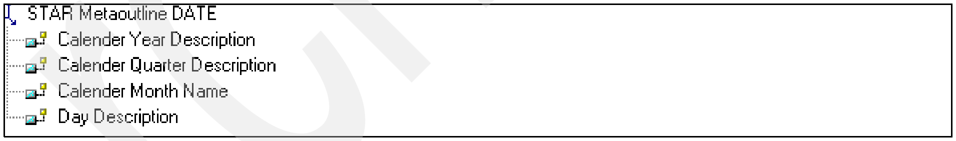


Figure 7-9 Hierarchy levels in Object Explorer

A tool tip can be displayed by placing the mouse over a metadata object in the Object Explorer. The tool tip consists of the actual metadata object name, its business name, its data type and the aggregation, if applicable.

7.4.4 Layout Designer

The Layout Designer is a design tool that can be floating or docked by default across the bottom panel of the QMF for Windows interface. To add the Layout Designer, select View and place a check mark next to Layout Designer.

There are three groups in the Layout Designer:

- Top Dimensions
- Side Dimensions
- Measures

The Layout Designer in Figure 7-10 enables the user to drag and drop attributes into the various groupings to create an interactive view of the multidimensional data. The top and side groups will contain dimensions. The measure group contains measures.

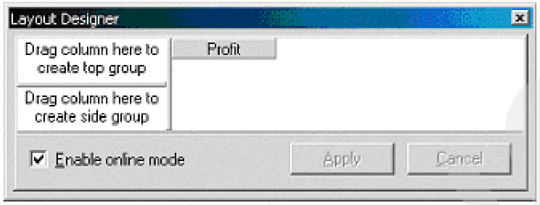


Figure 7-10 Default Layout Designer toolbar

An option on the Layout Designer is to enable online mode. When this option is selected, changes made in the Layout Designer will automatically result in updates to the Query Results View.

When the enable online mode is not checked as in Figure 7-11, the Query Results View appears greyed out, and updates made to the Layout Designer will not take effect until the user selects Apply.

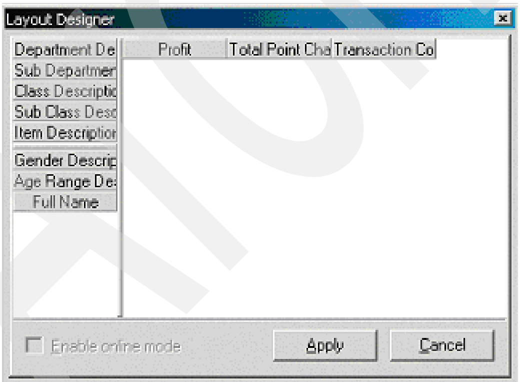


Figure 7-11 Layout Designer without enable online mode option

The query layout can also be created by using drag and drop within the lower portion of the tree control in the Object Explorer entitled Layout. The Layout Designer and the Layout tree control contain the identical query information.

Initially, dimensions are displayed in their most rolled-up form. Rolled-up means that lower level(s) of the dimension are not displayed. Drill-down is the opposite of rollup. Drill-down exposes lower levels of the dimension's hierarchy. To drill-down in a dimension, click the plus sign. To roll back up, click the minus sign.

7.4.5 Query Results View

By default, the initial result set of an OLAP Query will contain a single cell of the first measure listed, rolled up to the highest level of aggregate with no top or side dimensions. From this point, the user can build upon the result set adding or removing additional dimensions and measures. Unlike a typical SQL query that is written and subsequently executed, the OLAP Query object will automatically run the generated SQL when changes are made to the Query Results View. This implementation enables a user to avoid the challenges of writing complex SQL containing OLAP functions. As QMF Prompted Query enables the user to select the data and data conditions of interest without knowledge of SQL or table structures, the OLAP query allows the user to interact with the DB2 Cube Views catalog without having knowledge of the underlying metadata objects.

The Query Results View appears in the middle panel by default. The actions of dragging and dropping dimensions and measures into the Layout Designer are reflected in changing to the Query Results View. The Query Results View is constantly refreshed on each change made in the Layout Designer. This task is accomplished via under the covers with SQL generated by QMF for Windows. The SQL execution and status is indicated by the message line in the lower left hand corner of the application. As with a regular SQL query, the user can cancel the operation of the SQL generated OLAP query by selecting the Cancel Query button or menu option.

When a cube model is selected for an OLAP query, the default result set will contain the first measure, aggregated up to the highest level.

Filter option

The OLAP Query Filter command brings to the front a window that allows for the user to select what values to include in the results. This filter panel in Figure 7-12 allows the user to determine precisely which values are available. A checked box indicates that the value is included and an unchecked box indicates that the value is not included. This filter also serves to re-add values that have previously been excluded from the results. Changing the filter values requires the OLAP query to execute SQL behind the scenes to generate the new results set.

On the right-click menu of a Measure or Dimension in the Object Explorer, Layout Designer or Query Results View, the Remove from Layout and Filter Out options have the same effect as de-selecting an item in the OLAP Query Filter window. The default for the filter option is that all attributes are selected and included in the Query Results View.

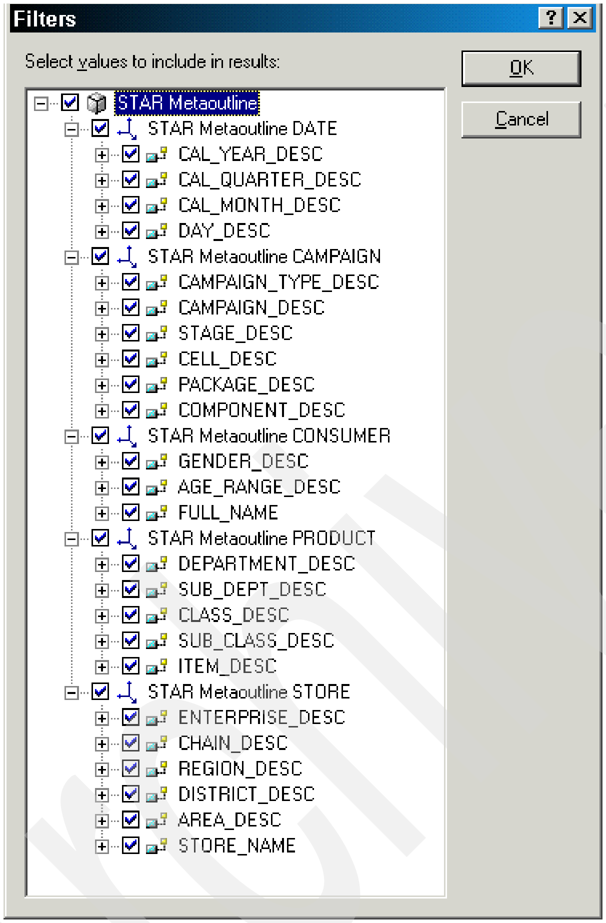


Figure 7-12 Default filter window

Working with filters

The following filter selections would calculate a result set containing the *New Product Introduction* Campaign Type in the *Central* region for the years 2000 and 2001. The exclusion of different values via a filter can affect the results of the measures. Filter choices can produce an empty result set. At least one value for each level must be selected when specifying a filter.

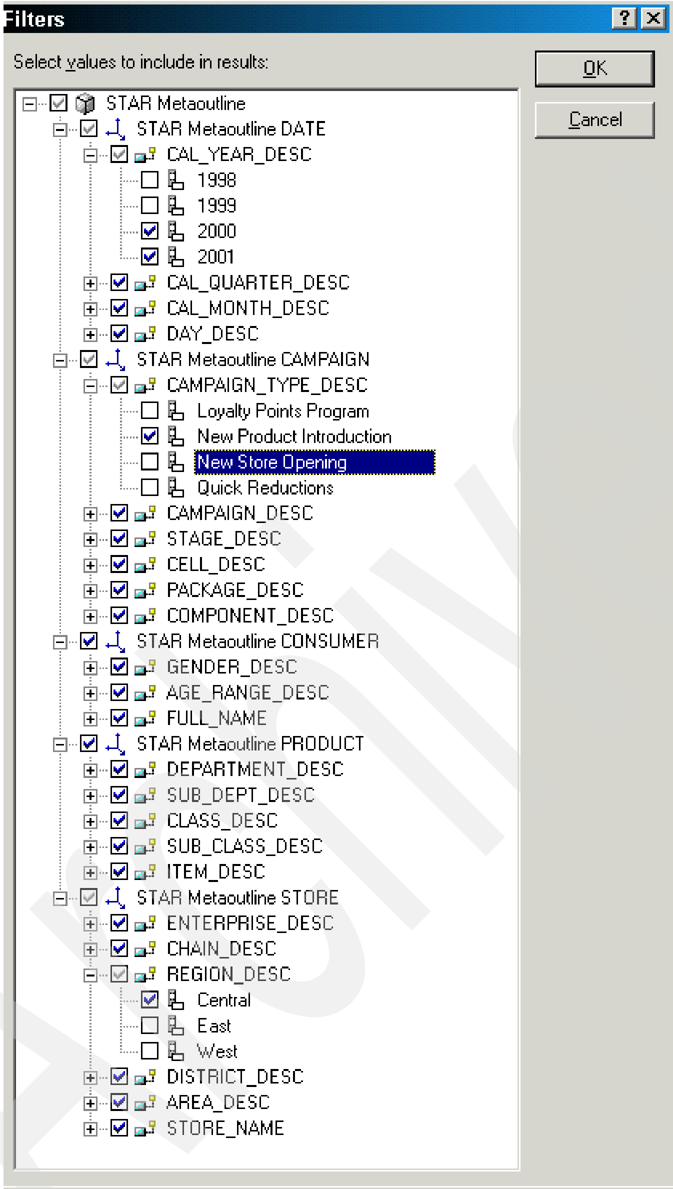


Figure 7-13 Filter window with options

It can be determined from the Object Explorer window whether any filters are in place: a filter symbol is located in the upper left-hand corner of the existing metadata object icon.

Formatting options

Formatting options shown in Figure 7-14 can also be applied to columns in the Query Results View. To add formatting, select the desired column and either use the right-click option or the formatting tool bar to change the formatting parameters. You can specify column heading names, data text colors, background colors, and data format.

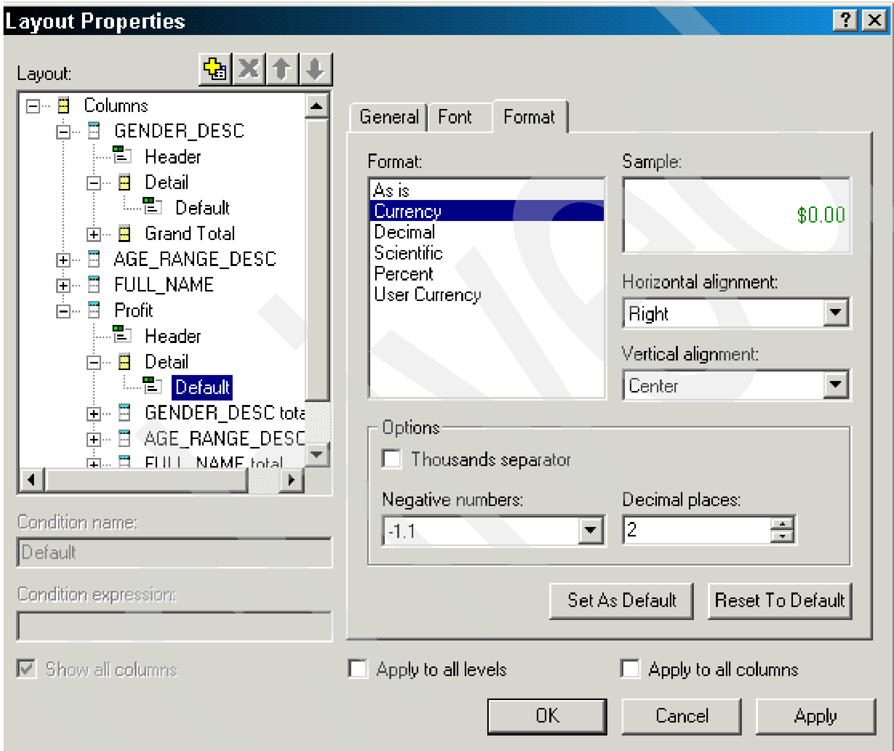


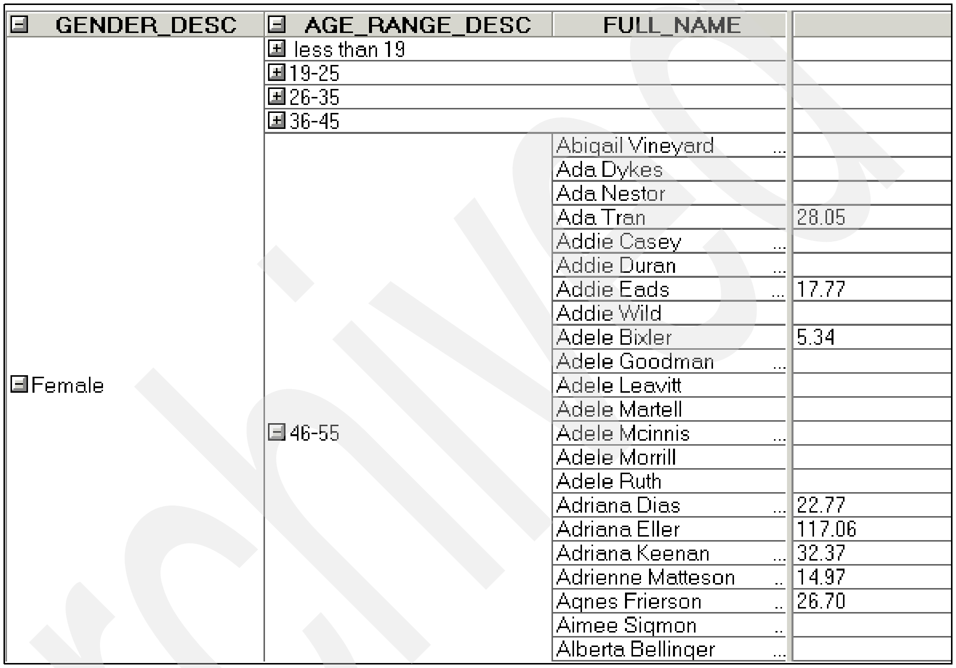
Figure 7-14 Formatting options

OLAP functionality

QMF for Windows provides the mechanisms by which the user can employ OLAP techniques while performing multidimensional data analysis. These techniques include drill down, drill up, rollup, pivot, slice and dice, and drill through.

Drill down

Drill down refers to a specific analytical OLAP technique when the user traverses among levels of data ranging from the highest, most summarized level to the lowest, most detailed level. The drill down path is defined by the hierarchy within the cube dimension. To increase the granularity of the result set, the drill down feature can be employed in the QMF for Windows Query Results View. Simply click the plus (+) sign preceding the data value to expand the level. Drill down can also be accomplished through right-clicking a column header within the Query Results View and selecting drill down. In Figure 7-15, the level Female is drill down twice to display the full names of those women between ages 46-55. The corresponding profit that each individual person produced is displayed in the Profit column.



GENDER_DESC	AGE_RANGE_DESC	FULL_NAME	
Female	less than 19		
	19-25		
	26-35		
	36-45		
	46-55	Abigail Vineyard ...	
		Ada Dykes	
		Ada Nestor	
		Ada Tran	28.05
		Addie Casey ...	
		Addie Duran ...	
		Addie Eads ...	17.77
		Addie Wild	
		Adele Bixler	5.34
		Adele Goodman ...	
		Adele Leavitt	
		Adele Martell	
		Adele Mcinnis ...	
		Adele Morrill	
		Adele Ruth	
		Adriana Dias ...	22.77
		Adriana Eller	117.06
		Adriana Keenan ...	32.37
		Adrienne Matteson ..	14.97
		Agnes Frierson ..	26.70
		Aimee Sigmon ..	
		Alberta Bellinger ...	

Figure 7-15 Drill down operation

Drill up

Drill up refers to a specific analytical OLAP technique when the user traverses among levels of data ranging from the lowest, most detailed level to the highest, most summarized level. The drill up path is defined by the hierarchy within the cube dimension and is the same as the drill down path. To decrease the granularity of the result set, the drill up feature can be employed in the QMF for Windows Query Results View. Simply click the plus (-) sign preceding the data value to expand the level. Drill up can also be accomplished through right-clicking a column header within the Query Results View and selecting drill up.

By default, dimensions are displayed drilled up to the highest level of the hierarchy (see Figure 7-16).

8	⊞ GENDER_DESC	Profit
9	⊞Female	102326.70
10	⊞Male	35556.68
11	⊞Unknown	89690.16
12	All values	227573.54

Figure 7-16 Drill up operations

Roll up

Roll up refers to a specific analytical OLAP technique involving the computation of the data relationships between all levels of a hierarchy in a dimension. These data relationships are often summations though any type of computational relationship or formula that might be defined.

The *All values* row represents the value of all of the collective hierarchy levels rolled up to the highest level of aggregation.

	1	2
1	⊞ GENDER_DESC	Profit
2	⊞Female	387264.24
3	⊞Male	96032.45
4	⊞Unknown	190027.75
5	All values	673324.44

Figure 7-17 Roll up operations

Pivot

Pivot refers to a specific analytical OLAP technique of changing the dimensional orientation of the result set. Pivot can be accomplished in QMF for Windows by changing one of the top dimensions into a side dimension and vice versa or swapping dimensions.

Slice and dice

A slice is an OLAP term which describes a two-dimensional page of a cube (see Figure 7-18). One or more dimensions are fixed to a single value, resulting in the variation of values in remaining two dimensions. Slice and dice refers to a user-driven process of navigating by interactively specifying the slices via pivots and drill down/up. QMF for Windows users can accomplish slice and dice by employing the techniques discussed earlier to perform pivots and drill down/up on the Query Results View.

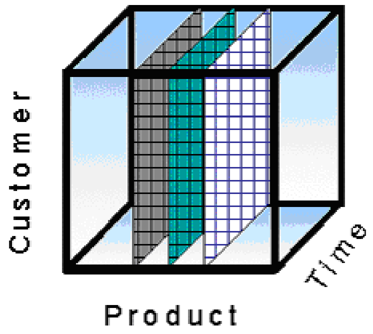


Figure 7-18 Slices of the product dimension

Drill through

Drill through refers to a specific analytical OLAP technique of switch from a cube (multidimensional data model) to the primary relational data. Since QMF for Windows is a complete relational query and reporting tool, the underlying relational tables that develop the cube can be accessed and viewed as in Figure 7-19.

	1	2	3	4	5
1	IDENT_KEY	CONSUMER_KEY	FULL_NAME	FIRST_NAME	LAST_NAME
2	1	10021231	Mandy Jacobson	Mandy	Jacobson
3	2	10434231	Sonya Gunderson	Sonya	Gunderson
4	3	11904231	Ethel Goodin	Ethel	Goodin
5	4	12793231	Elena Fletcher	Elena	Fletcher
6	5	15012231	Candace Lash	Candace	Lash
7	19	32176231	Stephanie Shepherd	Stephanie	Shepherd
8	20	32701231	Cassandra Stinson	Cassandra	Stinson

Figure 7-19 Portion of CONSUMER table from a relational view

7.5 OLAP report examples and benefits

In order to support multidimensional data via DB2 Cube Views, QMF for Windows v7.2f provides users with the abilities to *describe, visualize* and *manipulate* multidimensional data:

- *Describe* is accomplished through the QMF OLAP Query object.
- *Visualize* is achieved via the Object Explorer and Query Results View.
- *Manipulate* is fulfilled by use of the Layout Designer.

7.5.1 Who can use OLAP functionality?

Because of its easy-to-use interface, QMF for Windows v7.2f can be tailored to the OLAP requirements of virtually any educated worker from a senior-level executive to a skilled business analyst, or even the average manager, sales person, or novice user. Different members of an organization can access shared OLAP queries, make data or formatting modifications, and save these modified queries, thereby building a base of OLAP queries that suit the needs of each individual user. The results from the analysis of the OLAP query can also be printed.

7.5.2 Before starting

In order to enable QMF for Windows to generate OLAP queries against our retail store's data, the DBA would use DB2 Cube Views ***OLAP Center*** to create meaningful metadata objects for the user.

To begin OLAP analysis with QMF for Windows, one cube object derived from a cube model has to be defined with OLAP Center since QMF for Windows builds its OLAP query based upon a cube. Since metadata objects are saved at the server level, different users of QMF for Windows could access the any existing cube objects and would not need to initially use the OLAP Center before creating OLAP queries in QMF for Windows.

Figure 7-20 represents our scenario cube named Sales Cube. Sales Cube is defined by a star schema with one center fact table, CONSUMER_SALES and five dimension tables: CONSUMER, DATE, STORE, CAMPAIGN and PRODUCT.

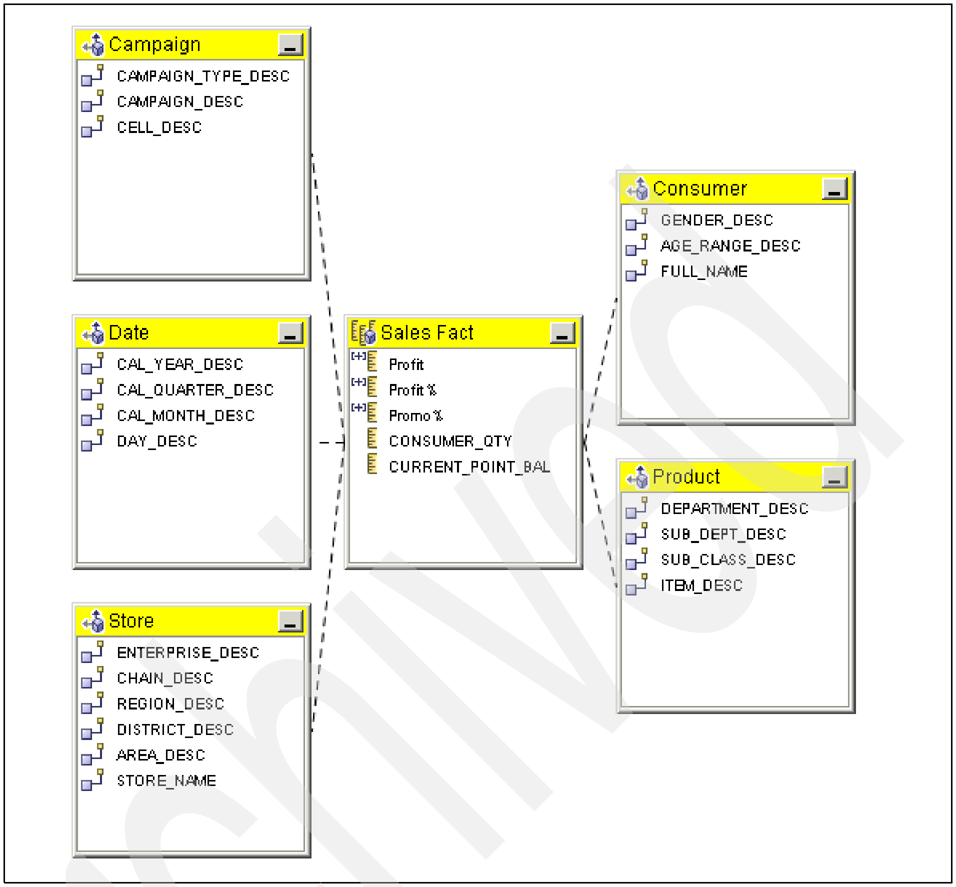


Figure 7-20 Sales cube example in DB2 OLAP Center

7.5.3 Sales analysis scenario

Let us suppose we have a national sales manager for a retail store chain who uses QMF for Windows to access and analyze sales data collected by the various stores throughout the country. On a regular basis, the analysis requires the manager to incorporate data attributes from several tables that contain customer names and descriptions, store names and locations, marketing campaigns, product information and sales figures over time.

OLAP query example 1

The manager wants to determine which are the most profitable gender/age categories in the western region of the United States.

1. Begin by creating a new OLAP Query object. Select File->New and choose the OLAP Query icon.
2. Follow the OLAP Query wizard to select the appropriate server and cube from the given cube list.
3. After the initial result set is retrieved, drag and drop the Consumer dimension into the Side Dimension group indicated in the Layout Designer.
4. Drag and drop the Profit measure into the Measures group indicated in the Layout Designer. Anytime dimensions and measures are added or removed from the result set, the SQL is generated and sent by QMF for Windows to DB2 to process the request.
5. Select the **Filter** option. Under dimension Store, expand Region Description and deselect Central and East attributes. This will result in the inclusion of only values from the west region.

In Figure 7-21, It can be seen that the most profitable groups are Unknown_less than 19, Female_26-35, Female_36-45 and Female_19-25.

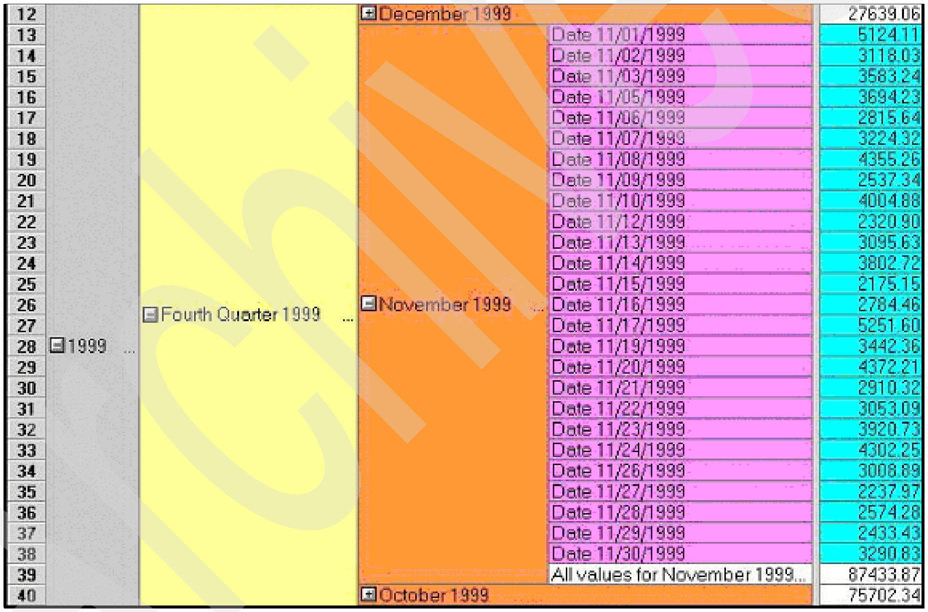
	1	2	3
1	GENDER_DESC	AGE_RANGE_DESC	Profit
2	Female	less than 19	25644.93
3		19-25	43978.11
4		26-35	86541.15
5		36-45	74882.54
6		46-55	61197.53
7		56-65	42953.50
8		66-75	22831.12
9		>75	6391.85
10		Unknown	22843.51
11		All values for Female ...	387264.24
12	Male	less than 19	5518.37
13		19-25	15798.27
14		26-35	26425.13
15		36-45	13509.62
16		46-55	15383.52
17		56-65	10338.11
18		66-75	5750.61
19		>75	1510.82
20		Unknown	1798.00
21		All values for Male	96032.45
22	Unknown	less than 19	188095.41
23		19-25	441.88
24		26-35	1233.22
25		36-45	18.74
26		46-55	4.98
27		56-65	17.88
28		>75	215.64
29		All values for Unknown ...	190027.75
30	All values		673324.44

Figure 7-21 OLAP report 1: most profitable consumer groups in the West region

OLAP query example 2

The manager is considering running a promotional sale during the month of November. The manager wants to know what the most profitable sales day in November 1999 was in order to determine the best date for a promotional day in November of the upcoming year.

1. Begin by creating a new OLAP query or modify the previous OLAP query by removing the Consumer from the Query Results View by right-clicking **Consumer** in the Layout Designer and selecting Remove from Layout. Also remove the filter option.
2. Place the Date dimension in the Side Dimension group.
3. Place Profit in the Measures Group.
4. Drill down into the fourth quarter of the year 1999 and we see in Figure 7-22 that November 17, 1999 was the most profitable day of sales.



12			December 1999		27639.06
13				Date 11/01/1999	5124.11
14				Date 11/02/1999	3118.03
15				Date 11/03/1999	3583.24
16				Date 11/05/1999	3694.23
17				Date 11/06/1999	2815.64
18				Date 11/07/1999	3224.32
19				Date 11/08/1999	4355.26
20				Date 11/09/1999	2537.34
21				Date 11/10/1999	4004.88
22				Date 11/12/1999	2320.90
23				Date 11/13/1999	3095.63
24				Date 11/14/1999	3802.72
25				Date 11/15/1999	2175.15
26			November 1999	Date 11/16/1999	2784.46
27		Fourth Quarter 1999		Date 11/17/1999	5251.60
28	1999			Date 11/19/1999	3442.36
29				Date 11/20/1999	4372.21
30				Date 11/21/1999	2910.32
31				Date 11/22/1999	3053.09
32				Date 11/23/1999	3920.73
33				Date 11/24/1999	4302.25
34				Date 11/26/1999	3008.89
35				Date 11/27/1999	2237.97
36				Date 11/28/1999	2574.28
37				Date 11/29/1999	2433.43
38				Date 11/30/1999	3290.83
39				All values for November 1999...	87433.87
40			October 1999		75702.34

Figure 7-22 OLAP report 2: most profitable sales

OLAP query example 3

Suppose now that the manager is interested in analysis of the historical consumer buying trends. Specifically, over the period of one year from 1998 to 1999, has the sales profit from Females ages 56-65 increased?

1. We can modify the previous OLAP query from example 2.

2. Pivot on the Time dimension by moving the Time dimension from the Side Dimension group to the Top Dimension group.
3. Add the Consumer dimension to the Side Dimension Group.
4. Add Profit to the Measure Group.
5. Drill down into the Female level and ascertain in Figure 7-23 that Females 56-65 have increased the profit margin by close to 5% from 1998 to 1999.

	1	2	3	4	5
1	CAL_YEAR_DESC		1998	1999	All values
2	GENDER_DESC	AGE_RANGE_DESC	Profit	Profit	Profit
3	Female	less than 19	4275.72	21369.21	25644.93
4		19-25	8024.23	35953.88	43978.11
5		26-35	15099.75	71441.40	86541.15
6		36-45	14883.71	59998.83	74882.54
7		46-55	11416.26	49781.27	61197.53
8		56-65	7856.43	35097.07	42953.50
9		66-75	3975.56	18855.56	22831.12
10		>75	988.70	5403.15	6391.85
11		Unknown	4649.99	18193.52	22843.51
12		All values for Female	71170.35	316093.89	387264.24

Figure 7-23 OLAP report 3: consumer buying trends

7.6 Maintenance

When operating OLAP queries, the user should take care about:

- Invalidation of OLAP queries
- Performance issues

7.6.1 Invalidation of OLAP queries

If the metadata cataloged in DB2 Cube Views is modified or deleted, existing QMF OLAP query objects may not be functional depending on the changes made to the underlying metadata structures. An error message will be issued when opening a previously saved QMF OLAP query if any of the referenced metadata objects are no longer valid within the catalog tables of DB2 Cube Views. DBAs should take care in preserving DB2 Cube View's metadata objects that contain QMF OLAP query dependencies, that is, a saved OLAP query references the metadata.

7.6.2 Performance issues

Some OLAP queries may require significant time to complete and a significant number of rows (or bytes) to be fetched. Certain limits can be increased or eliminated to prevent the cancellation of demanding OLAP queries. From QMF for Windows Administrator, select the specified server, select the Resource Limits tab, and edit the corresponding resource group schedule.

Select the **Limits** tab. The following limits may need to be adjusted to successfully run high demanding OLAP queries:

- Maximum Rows to Fetch:
  - Warning Limit
  - Cancel Limit
- Maximum Bytes to Fetch:
  - Warning Limit
  - Cancel Limit

Note: A limit with a specification of zero implies that no limit exists.

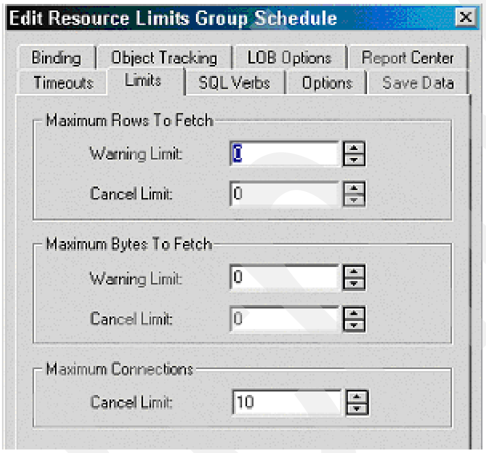


Figure 7-24 Resource Limits Group in QMF for Windows Administrator

7.7 Conclusion

QMF for Windows v7.2f provides support for multidimensional data analysis through the introduction of the OLAP query, enhancements to the graphical-user interface and support of DB2 Cube Views. For more information on QMF for Windows and the QMF Family, go to:

http://www.ibm.com/qmf

8

Using Ascential MetaStage and the DB2 Cube Views MetaBroker

This chapter describes an end-to-end deployment scenario, using Ascential MetaStage, Ascential DataStage, and the Ascential DB2 Cube Views MetaBroker®. It explains how to implement and to use MetaStage's metadata exchange and analysis functions, and discusses the benefits of its use. The objective is to provide more detailed information and a tutorial about the way metadata can be leveraged to manage an overall data warehouse implementation from initial data modeling design, subsequent ETL design and database population, and finally, through OLAP cube creation.

8.1 Ascential MetaStage product overview

MetaStage is the platform services component of the Ascential Enterprise Integration Suite shown in Figure 8-1, responsible for management of metadata. MetaStage is a persistent metadata Directory that uniquely synchronizes metadata across multiple separate silos, eliminating rekeying and the manual establishment of cross-tool relationships. Based on patented technology, it provides seamless cross-tool integration throughout the entire Business Intelligence and data integration lifecycle and toolsets. MetaStage provides full investment protection that reduces the guesswork associated with iterative updates, allowing you to assess the impact of change, and understand the full meaning (and potential) of your data without ambiguity.

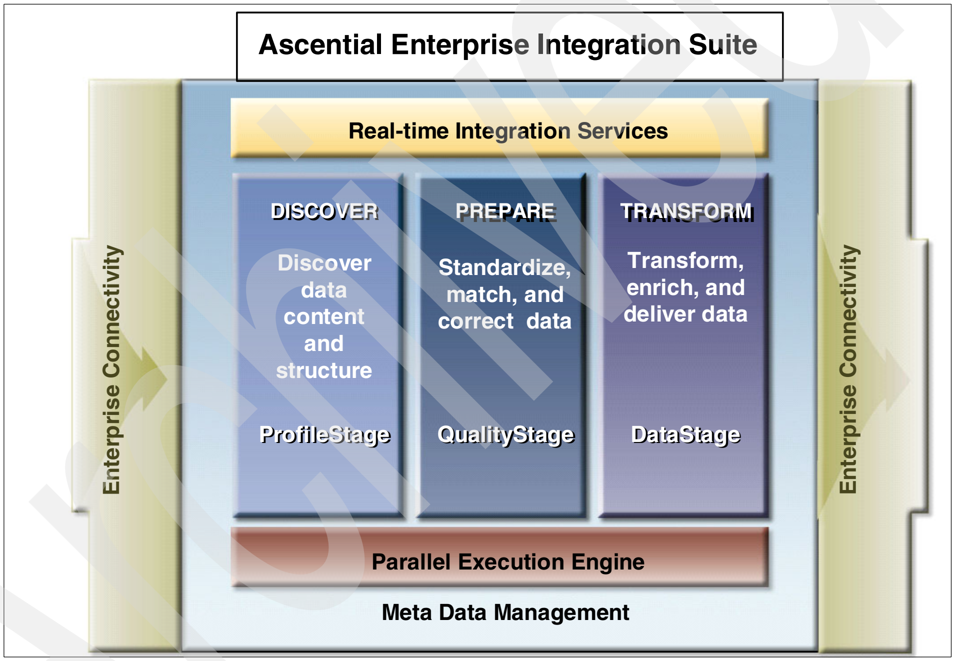


Figure 8-1 Ascential Enterprise Integration Suite

MetaStage provides a new approach to the management of metadata. Regardless of your architecture, MetaStage lets you develop a high-quality enterprise resource over which you have real control.

This innovative architecture consists of five major software components:

1. **MetaStage Directory:**

   The MetaStage Directory is a server-side database that is configured by the MetaStage administrator, and utilized by the rest of the MetaStage clients and external applications. Ascential's patented translation and identity technology — advanced Meta Model embedded within MetaStage and MetaBrokers — can decompose any tools' capabilities into the metadata equivalent of the Periodic Table of Elements. As MetaStage fundamentally understands what each tool requires at the atomic level, it can easily recompose metadata for any other tool that shares common atoms (or concepts) on demand. Ascential calls this atomic level representation and commonality technique *semantic overlap*. This also enables MetaStage to automatically recognize whether a metadata asset within the metamodel is, in fact, the same thing.

MetaStage Directory is capable of sourcing, sharing, storing, and reconciling a comprehensive spectrum of metadata:

a. Business metadata: Business rules, definitions, business domains, terms, glossaries, algorithms and lineage information using business language (for business users).

b. Technical metadata: Defines source and target systems, and their table and fields structures and attributes, derivations and dependencies (for specific users of Business Intelligence, OLAP, ETL, Data Profiling, and ER Modeling tools).

c. Operational metadata: Information about operational application execution (events) and their frequency, record counts, component-by-component analysis, and other related granular statistics (for operations, management, and business users).

d. Project metadata: Documents and audits development efforts, assigns stewards, and handles change management (for operations, stewards, tool users, and management).

Since all of the Directory's objects are versioned, a complete chronological audit trail of events is provided. An administrator can roll back a Directory to a known point in time, or consolidate it for optimal operational efficiency. As companies usually span multiple time zones and territories, the entire MetaStage deliverable is National Language Support (NLS) and Unicode enabled. Multi-tiered access with roles such as administrator, developer, subscriber and steward can be configured to ensure that the right metadata is delivered to the right person, at the right time — either singularly, or integrated with corporate directories such as LDAP.

2. **MetaBrokers and MetaArchitect:**

   Using MetaStage and its MetaBrokers, your company can leverage best-of-class modeling, data profiling, data quality, ETL, OLAP and Business Intelligence tools and be assured of protecting its investment.

   Ascential's high fidelity and round-trip integrity approach leverages each tool's true metadata format and meaning, enabling you to exploit the full value of your data quickly, without guesswork or labor intensive manual intervention and on-going maintenance. MetaBrokers come in five groups:

a. The first group deals with data model design, and includes tools such as CA ERwin, Oracle Designer, and the Unified Modeling Language (UML).

b. The second group deals with OLAP and Business Intelligence tools such as Cognos PowerPlay, Business Objects, and Hyperion.

c. The third group deals with ETL tools such as Ascential DataStage and Informatica PowerCenter.

d. The fourth group enables the sharing of operational metadata. This allows critical DataStage operational metadata to be perfectly reconciled with its associated design within the directory. Conceptually, this can be used with other tools' event metadata, provided that it conforms to the prescribed format and meaning.

e. The fifth group is a custom MetaBroker capability called MetaArchitect. MetaArchitect is a repeatable mechanism used to establish relationships and interchanges with a third party tool's metadata when no MetaBroker currently exists. It can also be used for special requirements such as Stewardship, DataStage, or Glossary information exchange. Using an existing metamodel, such as DataStage, MetaArchitect can alias the existing base model into a form that would not otherwise be possible. This enables rich bi-directional metadata exchange via Common Separated Values (CSV) or XML Metadata Interchange (XMI) file formats, with optional XSL-T style sheets for granular XML vocabulary formatting. MetaArchitect is the most expedient and consistent approach for the integration of home grown and commercial repositories such as CA Advantage and ASG Rochade.

3. **MetaStage Explorer:**

   MetaStage Explorer is a *power user* client interface for inspecting and interacting with the metadata in the MetaStage Directory. It delivers sophisticated metadata navigation and analysis functions. To minimize manual intervention, key Explorer functions have been script-enabled to permit users to focus on high-value analysis and management activities. The MetaStage Explorer delivers these key capabilities:

   – ***Impact analysis***: Using one of two model-oriented browser capabilities, you can traverse the underlying metadata objects using any tools' Meta Model representation to understand their relationships. More powerfully, you can immediately determine where an individual ERwin table is used, and what depends on that definition for its daily function, such as a Business Objects universe, Cognos Impromptu catalog, or DataStage design. With any change to an original ERwin design, you know exactly how data is flowing into a warehouse, or what BI tools' reports could be affected adversely — and managed accordingly — using other Explorer functions. This information can be propagated online, or in file format, to multiple audiences in business-based HMTL or technically-oriented XML form, using XSL-T-based customization and MetaStage's versatile model-driven query, reporting, and documentation tool.

– ***Data lineage*****:** Using the same underlying approach and reporting mechanism as Impact Analysis, Data Lineage integrates design and operational metadata in order to present developers, or end users, with a full perspective of not only the origin of the data in front of them, but the path it traveled — including all the applied derivations. While these can be numerous and potentially complex, it fundamentally answers the key question “where did this data come from?” Or put another way, for any given table, it can find its source or target. This information can be presented graphically showing the number of rows processed, accounting for every design component used. Or it can be presented as an aggregate number of rows processed across multiple DataStage servers and related tools against a given table. This could also include the derivation expression itself, making it ideal for audit, documentation, or immediate business analyst comprehension.

– ***Metadata synchronization and integration***: Cross-tool metadata can be automatically distributed to tools and users through MetaStage's advanced publish and subscribe capabilities. This provides systematic metadata synchronization and integration via all MetaBrokers, including MetaArchitect-based custom MetaBrokers. Subscriptions can either be a single metadata snapshot, or a recurring distribution event in nature, and subscribers are automatically notified via e-mail whenever relevant changes are made, ensuring good version control and change management best practices.

4. **MetaStage Browser** (and SQL access):

   MetaStage Browser enables you to immediately understand the context and meaning behind your data. That means you can make faster, more accurate decisions. An intuitive, Web-based thin-client is presented for navigating definitions, searching for key words, or performing custom metadata entry. As part of a corporate Metadata Dictionary project, the Browser highlights metadata relationships and artifacts that help you know whom to call about a specific “profit calculation.” It categorizes your metadata into appropriate hierarchical business terms, glossaries and business domains — such as cost centers, departments, and SIC codes and industries — for speedy navigation and comprehension.

Under the covers, it utilizes an administratively controlled, SQL-accessible portion of MetaStage Directory. Built using industry standard Java technology, the MetaStage Browser provides a reusable template for integrating your metadata into your own information delivery environment — such as an Enterprise Information Portal, Business Intelligence tools, and Microsoft Excel — or other application or Web technologies using industry-standard SQL.

8.1.1 Managing metadata with MetaStage

"Define data once and use it many times" is the basic principle of managing metadata. Figure 8-2 illustrates a recommended overall flow of metadata among tools as you build the data warehouse and DB2 Cube Views as part of a complete Data Warehousing project.

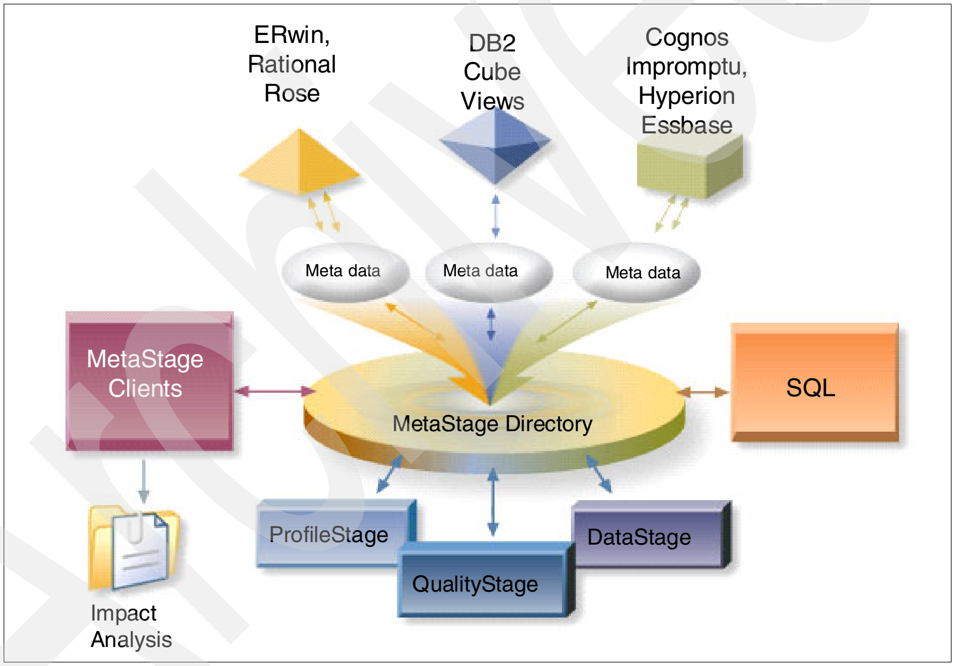


Figure 8-2 Metadata flow

Figure 8-3 shows this flow of metadata in a typical data warehouse lifecycle flowing from source system analysis to conceptual and physical models to Business Intelligence (BI) from left to right.

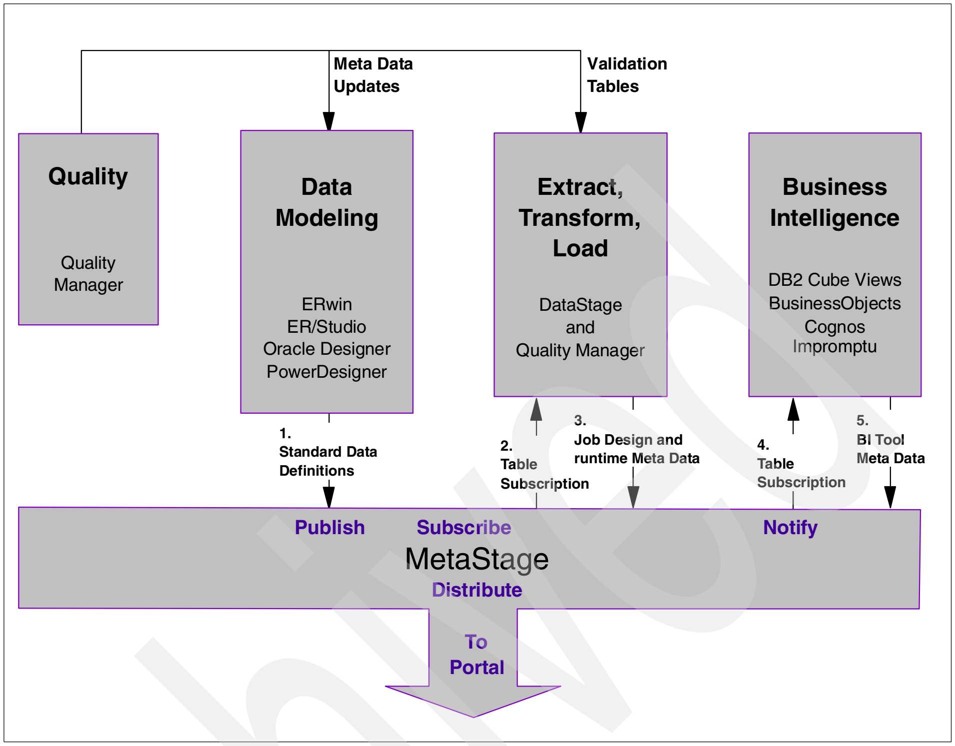


Figure 8-3 Recommended metadata flow

This flow of metadata from design to end user illustrates the implementation of a publish and subscribe paradigm that MetaStage uses to enable an organization to formalize metadata policies and procedures.

Exporting metadata with a MetaStage MetaBroker is a four-step process:

1. Creating a user-defined category and filling it with the objects you want to export
2. Publishing the contents of the user-defined category
3. Subscribing to the publication
4. Exporting the contents of the publication to which you have subscribed

This process lets you control who has the authority to make data public and to export it to other tools.

Examining Figure 8-3 a little more closely, keeping in mind the publish and subscribe paradigm, a warehouse project might typically start with a set of conceptual and physical data models, or source system analysis. To create data models, you may use any tool of your choosing. While it is recommended that you standardize on a data modeling tool in you organization, MetaStage does not force you to standardize on any one modeling or Business Intelligence tool. In fact, this is the beauty of MetaStage. If, for example, you find that it is more productive for one data modeling group to use UML class diagrams with Rational® Rose® and another group to use ERwin, MetaStage does not prevent you from doing this.

Note: MetaStage supports UML 1.1, 1.3 and 1.4 via XMI file format. Therefore, any modeling tool that supports XMI export (as for example Rational Rose) is automatically supported by MetaStage.

Step 1 in Figure 8-3 shows that when the warehouse data models are stable they can be made available to other uses by publishing the metadata objects in MetaStage. Once published, the data model definitions then become the standard metadata definitions that subsequent warehouse processes will use. In the flow depicted by Figure 8-3, the users of the data model definitions are the Extract, Transform, Load (ETL) operation and the BI process.

Simultaneously, the ETL and BI development groups can subscribe to the data model definitions provided by data modelers. By subscribing to the standard data model definitions the ETL and BI processes can no operate in parallel using the same metadata definitions achieving a maximum level of reuse and consistency. After the ETL and BI developers have completed their respective tasks, the specific metadata definitions for the ETL and BI processes can now be published as shown in Steps 3 and 5 in Figure 8-3. Now that MetaStage has the complete set of metadata definitions that span the enterprise data warehouse process, business and technical metadata can be selectively distributed to end users via either:

- Generated HTML documentation (customizable by XSL transformations)
- Directly, via SQL against an administratively controlled relational schema
- Customizable JSP-based standard Web interface

Since MetaStage stores all metadata definitions in its directory powerful analysis can be preformed on both design and runtime process metadata. MetaStage provides powerful query, impact analysis and data lineage capabilities to better understand the nature of the business and technical metadata in your environment.

Figure 8-4 shows the metadata flow in a little more detail and includes DB2 Cube Views as a subscriber and publisher of metadata definitions. It is worth while at this point to make reference to the way that MetaStage and MetaBrokers share metadata definitions. Figure 8-4 shows that each tool in the warehouse environment stores its own copy of metadata definitions in some kind of physical storage. Each MetaBroker for the respective tool will read metadata definitions from the respective physical storage.

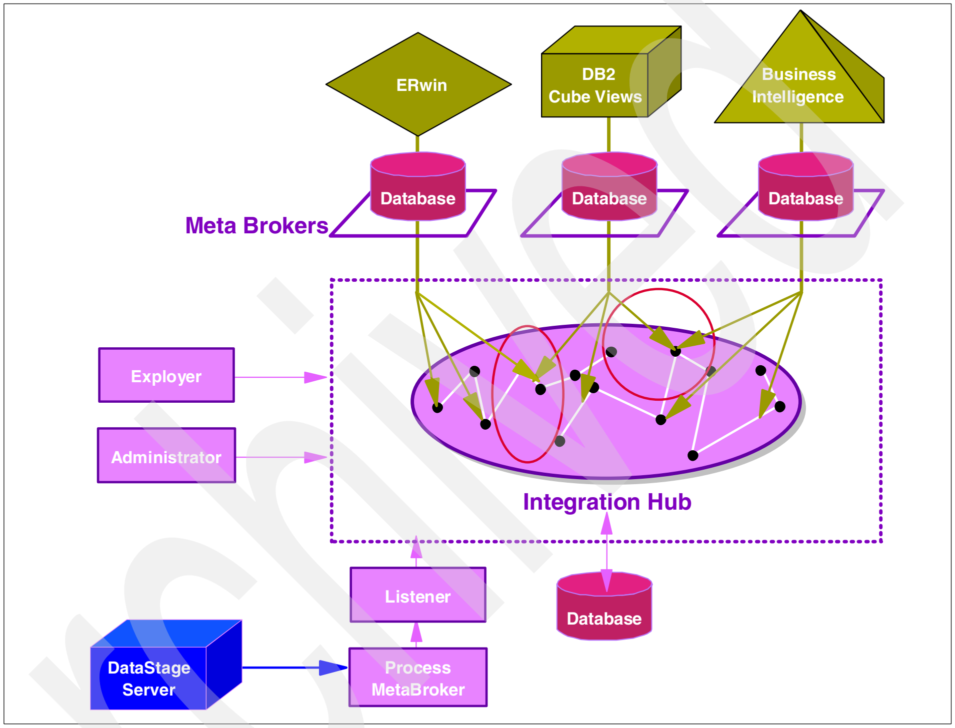


Figure 8-4 Metadata flow with DB2 Cube Views

The unique semantic translation that only Ascential MetaBrokers can do occurs after the MetaBroker has read each tool's specific metadata. Once read, the MetaBroker will perform semantic translation into atomic semantic units and store each semantic unit in the Directory. Once stored in the Integration Hub, the semantics of each unit is preserved for use by any other MetaBroker. The units highlighted by Figure 8-4 reflects the semantic equivalency of atomic units in the Directory are available for read by any other MetaBroker where there is metadata equivalency between tools. Ascential calls this *semantic overlap*. The picture conceptualizes the semantic overlap between different tool's metadata models. Furthermore, having understood the full extent of each tools metadata model, and the overlap, MetaStage can ready shared metadata between tools.

For example, a table stored by ERwin has the equivalent meaning in DB2 Cube Views and most BI tools. If the ERwin MetaBroker stores a table metadata definition object, the same table object is then available to be read by the DB2 Cube Views MetaBroker or any other MetaBroker that has a semantically equivalent metadata definition.

To drill down one step further into the flow of metadata definitions with MetaStage, we will use Figure 8-5 to illustrate the process. Figure 8-5 shows that metadata definitions always flow in and out of MetaStage via a MetaBroker.

The method of metadata definition access depends on the physical storage mechanism used by each specific tool to be integrated using a MetaBroker. The two methods of integration are via a file format or Application Programming Interface (API).

Note: API includes native tool APIs such as COM, C++, Java, SQL and others dependent on the data source access provided by the tool.

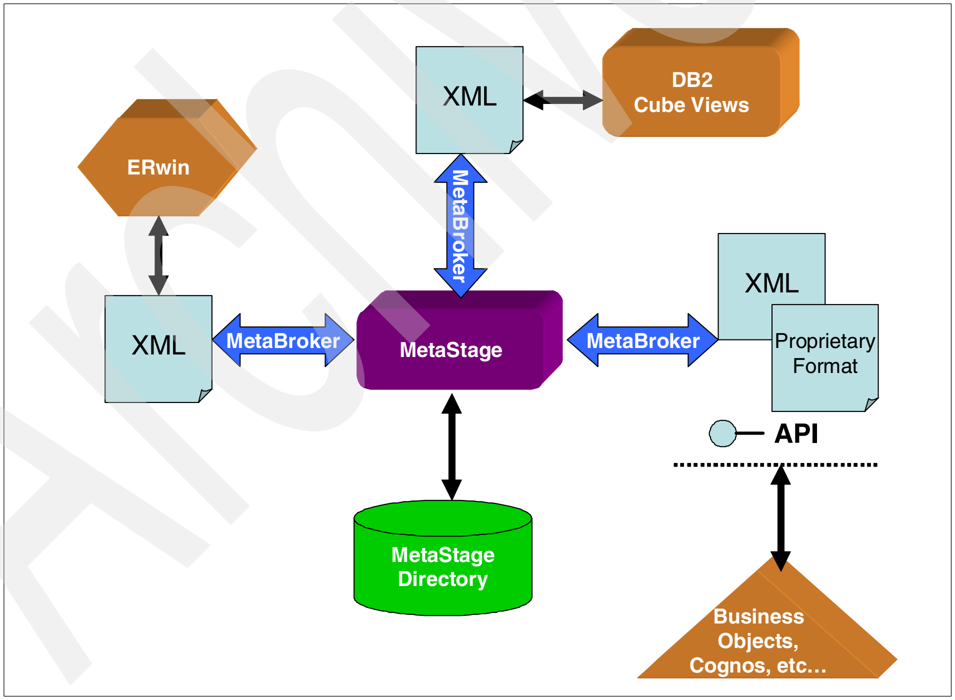


Figure 8-5 Detailed metadata flow

In the example shown in Figure 8-5, the ERwin MetaBroker will access an ERwin XML export file as its source metadata. Similarly, the DB2 Cube Views MetaBroker will use the XML file format defined by DB2 Cube Views as the interchange format. For other tools that will be integrated in the warehouse environment, such as BusinessObjects or others, the interchange format will vary depending on the access methods provided by each respective tool.

Unlike Figure 8-3 on page 277, Figure 8-5 implies no sequence of flow. Often this depends upon each company's best practice approach to metadata management, and the tools involved. Figure 8-3 on page 277 depicts a recommended flow of metadata for a specific warehouse project circumstance where we start with data models. However, MetaStage does not enforce this flow of metadata. In the following sections we will explore concrete examples of the flow of metadata in various scenarios to get a head start in developing DB2 Cube Views cube models and generally integrating DB2 Cube Views in your data warehouse environment.

8.2 Metadata flow scenarios with MetaStage

In this section, we will discuss the following metadata scenarios:

- Getting a fast start by importing ERwin dimensional metadata into DB2 Cube Views
- Leveraging existing data warehousing-oriented metadata with MetaStage
- Performing cross-tool impact analysis
- Performing data lineage analysis using DataStage design and operational metadata

8.2.1 Importing ERwin dimensional metadata into DB2 Cube Views

In this scenario we will see how to leverage an ERwin dimensional model to use as the base for developing a DB2 Cube Views cube model. For this example we will use the sample star schema shown in Figure 8-6.

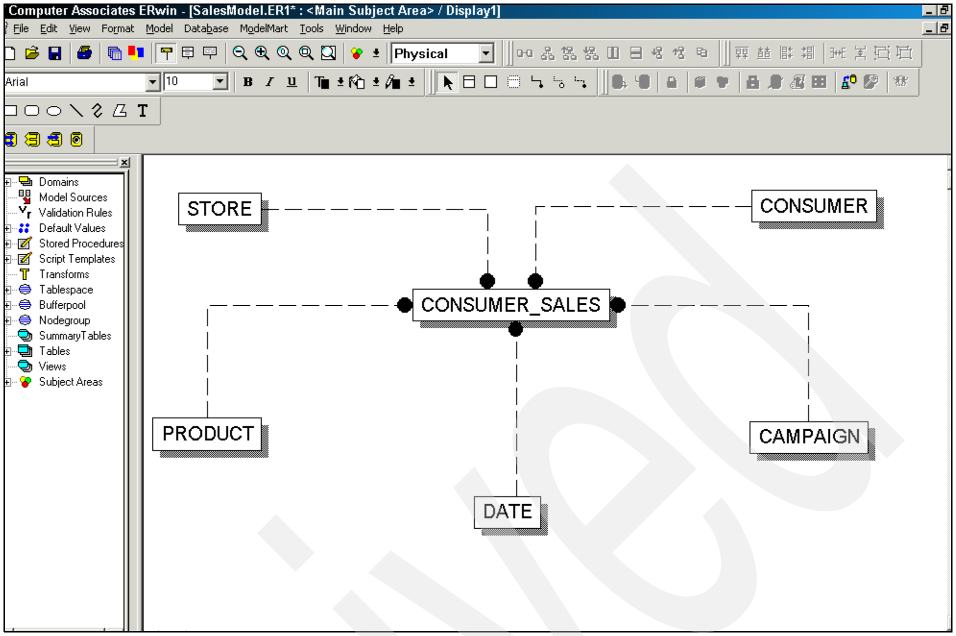


Figure 8-6 ERwin 4.1 sales star schema

ERwin 4.1 has the ability not only to design your star schema for use in DB2 Cube Views, it can tag each table in the star schema as playing an OLAP role so that other tools can use that information. In ERwin 4.1 you can tag a table as being a Fact or Dimension table.

To capture the dimensional model from ERwin 4.1 you must first tag each table in your star or snowflake schema as being a fact or a dimension. Applying this additional metadata is what makes the abstraction from your physical data structure to a dimensional structure. Without such dimensional metadata, the MetaBroker simply sees a relational data structure.

Figure 8-7 shows that the *CONSUMER_SALES* table has been defined as a Fact.

> **Note:** You must manually select each table as being a fact or dimension for the appropriate XML tag to be generated in the ERwin XML export. Choosing **Calculate Automatically** will not produce dimensional metadata in the XML export.

Similarly, the other tables in the model have been defined, but this time as dimensions. This is very useful information to OLAP tools, but the metadata is captured in ERwin 4.1 metadata definitions. MetaStage is able to read this information into the Directory and make it available to DB2 Cube Views so that the metadata describing which tables are facts and dimensions do not have to be redefined.

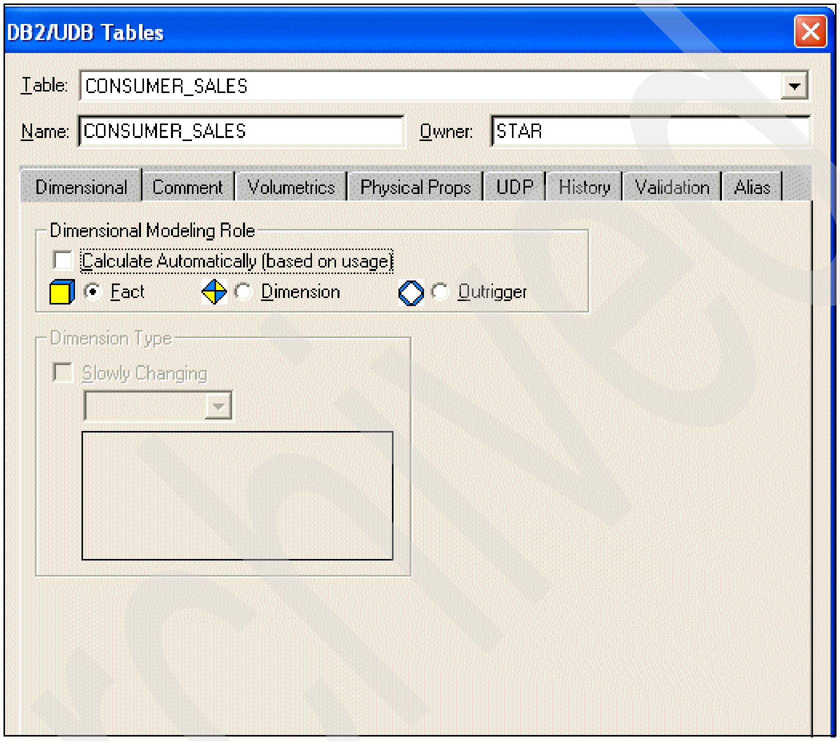


Figure 8-7   ERwin 4.1 dimensional dialog

A summary of the process involved to get ERwin 4.1 dimensional metadata into DB2 Cube Views is shown by Figure 8-8.

1. The ERwin 4.1 metadata must be exported to an XML file format.
2. MetaStage uses the ERwin 4.1 MetaBroker to import the ERwin 4.1 XML file format.
3. The relevant metadata objects are exported to the DB2 Cube Views XML file format.
4. The DB2 Cube Views XML file format is imported into OLAP Center.

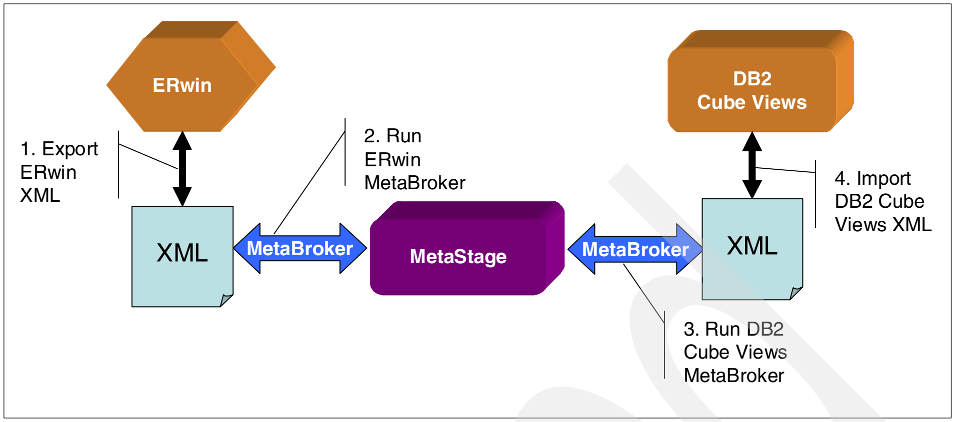


Figure 8-8 Summary of using ERwin 4.1 dimensional metadata

To provide the ERwin 4.1 dimensional metadata to DB2 Cube Views, you must first export your ERwin 4.1 model to an XML file format and import the ERwin 4.1 metadata into MetaStage. You do this by performing a MetaStage import using the ERwin 4.1 XML file format as the source. Figure 8-9 shows the MetaStage import dialog used to import the ERwin 4.1 metadata.

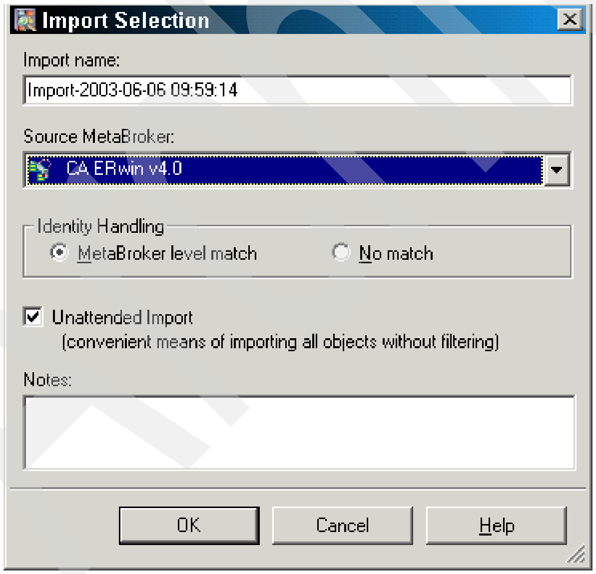


Figure 8-9 MetaStage ERwin import dialog

By selecting Computer Associates ERwin v4.0, the ERwin MetaBroker is invoked and the parameters screen shown in Figure 8-10 will be displayed. You must select the ERwin 4.1 XML file using the parameters dialog.

Note: The ERwin 4.0 MetaBroker is forward compatible with ERwin 4.1.

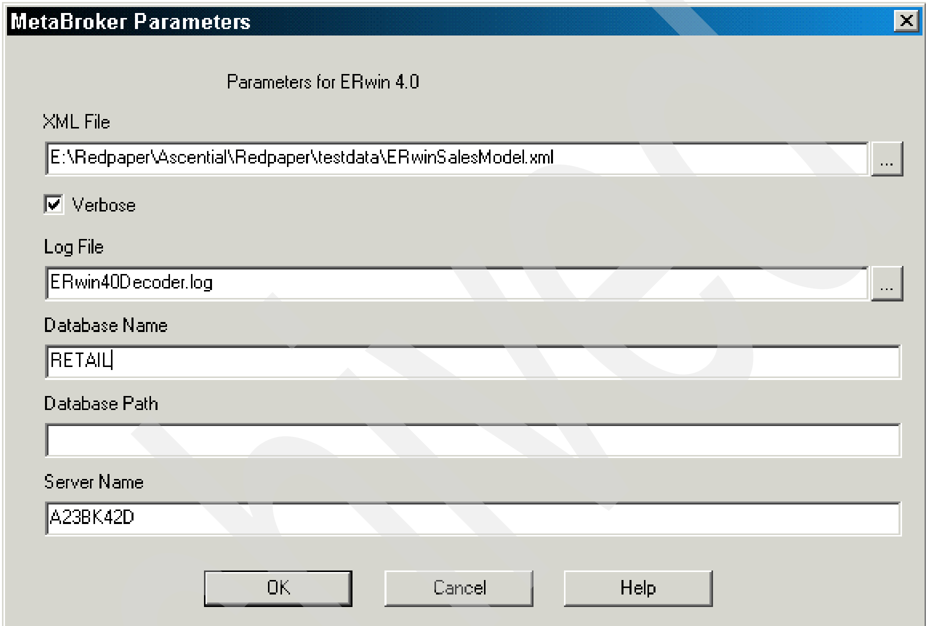


Figure 8-10 ERwin import parameters dialog

Figure 8-11 shows the ERwin 4.1 metadata in MetaStage after the import is complete.

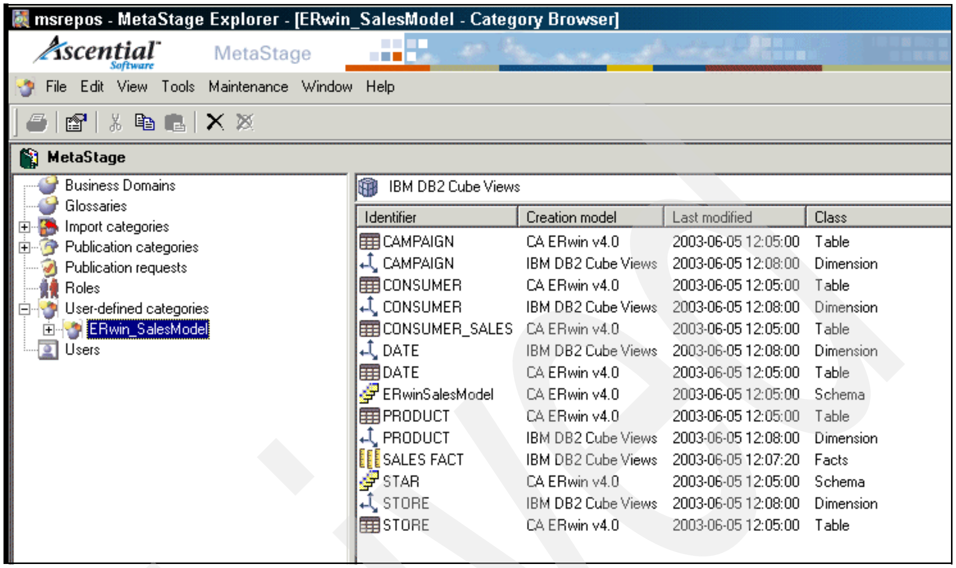


Figure 8-11 ERwin sales model imported into MetaStage

Figure 8-11 shows the major subset of metadata that can be shared with DB2 Cube Views. We can see that each ERwin table and its respective OLAP object is imported into MetaStage. These objects can now be exported to DB2 Cube Views so that further refinement of the cube model can occur in DB2 Cube Views.

Exporting metadata to DB2 Cube Views consists of:

1. Copying objects available to be published to a User Defined Category
2. Publishing metadata objects in the User Defined Category
3. Running the subscription wizard to create a subscription to the published objects

The objects shown in Figure 8-10 on page 285 were published, and Figure 8-12 shows the MetaStage New Subscription wizard dialog.

As shown in Figure 8-8 on page 284, the DB2 Cube Views MetaBroker creates an XML file containing the metadata definitions that must be imported into DB2 Cube Views subsequent to the completion of the export of metadata from MetaStage.

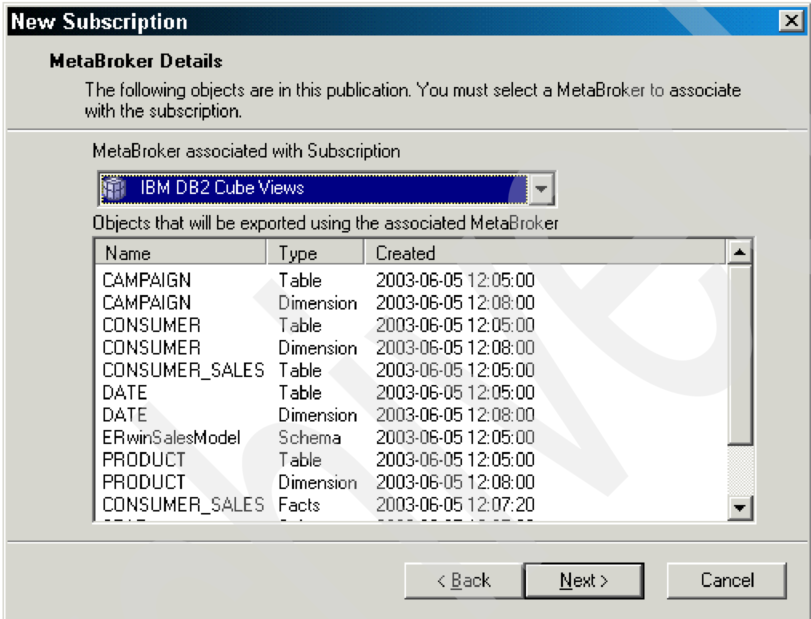


Figure 8-12 MetaStage new subscription dialog

The DB2 Cube Views MetaBroker requires the name and location of the XML file to produce, as shown by Figure 8-13. This XML file contains the source metadata definitions that will be imported to DB2 Cube Views using the XML import feature.

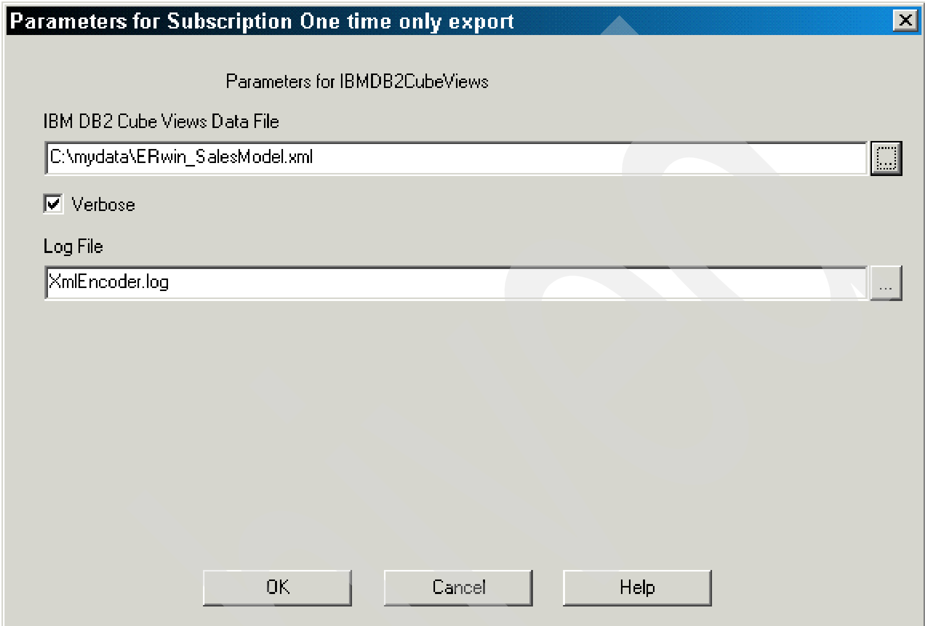


Figure 8-13 DB2 Cube Views export parameters

Finally, we see that Figure 8-14 shows the base level cube model in DB2 Cube Views OLAP Center. From this point the DB2 Cube Views user can continue to define OLAP metadata such as hierarchies and attribute relationships.

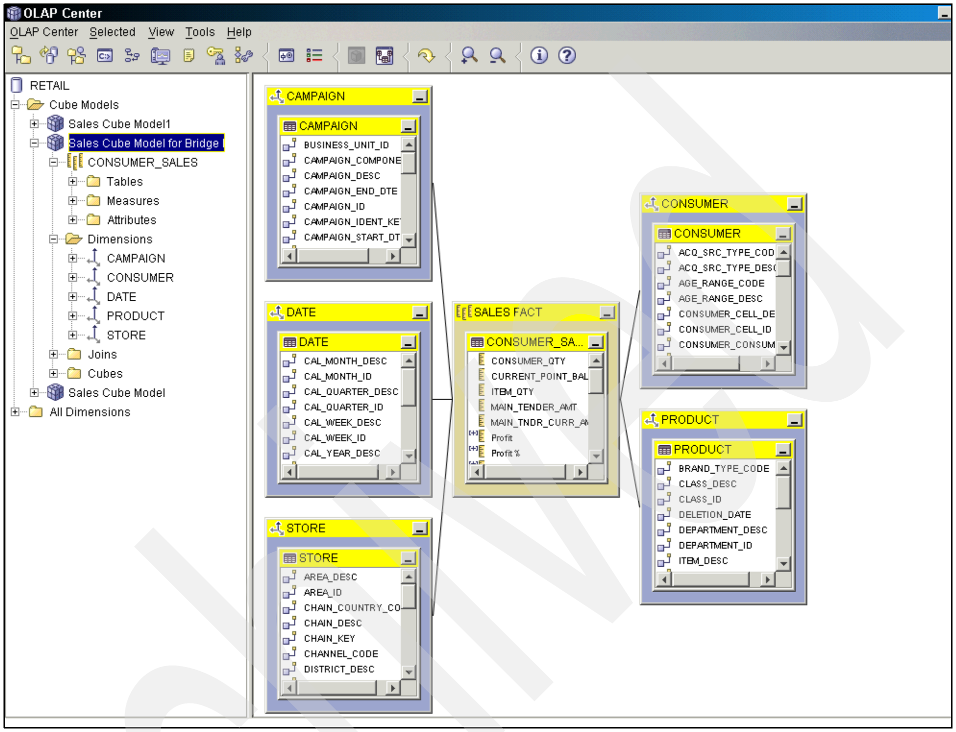


Figure 8-14 DB2 Cube Views Sales model from ERwin

8.2.2 Leveraging existing enterprise metadata with MetaStage

In this scenario we will see how to reuse existing dimensional metadata in DB2 Cube Views. Specifically, we will integrate a Hyperion Essbase Integration Server cube model with DB2 Cube Views. This scenario will apply as well to IBM DB2 OLAP Server™ and OLAP Integration Server metadata.

In Figure 8-15 we can see the Hyperion cube model. Since there has already been an investment in developing a cube model, it makes sense to reuse the same cube model in other parts of the organization. In this case we want to make the cube model available to DB2 cube views. To do this with MetaStage, we must first import the Hyperion MOLAP database model into MetaStage and then export it to DB2 Cube Views.

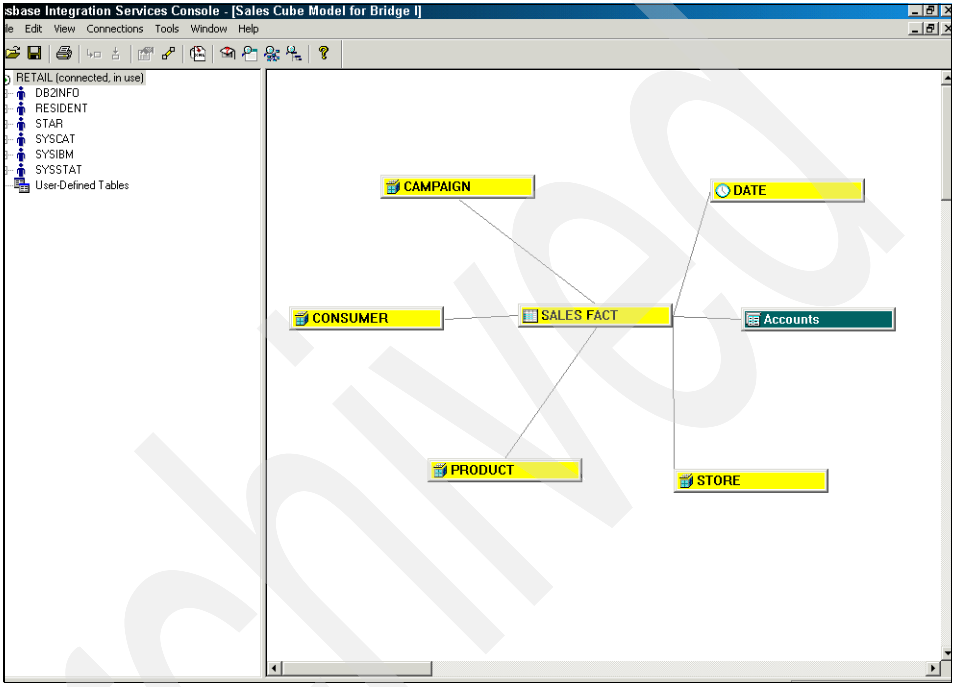


Figure 8-15 Hyperion Essbase Integration Server cube model

Similar to the scenario in 8.2.1, “Importing ERwin dimensional metadata into DB2 Cube Views” on page 281, Figure 8-16 shows the flow of metadata from Hyperion to DB2 Cube Views. This process involves four basic steps:

1. Export the Hyperion Essbase cube model in an XML file format.
2. Run the Hyperion Essbase MetaBroker to import the cube model metadata into MetaStage.
3. Export the Hyperion Essbase cube model metadata to a DB2 Cube Views XML file format by running the DB2 Cube Views MetaBroker.
4. Import the DB2 Cube Views XML using DB2 OLAP Center.

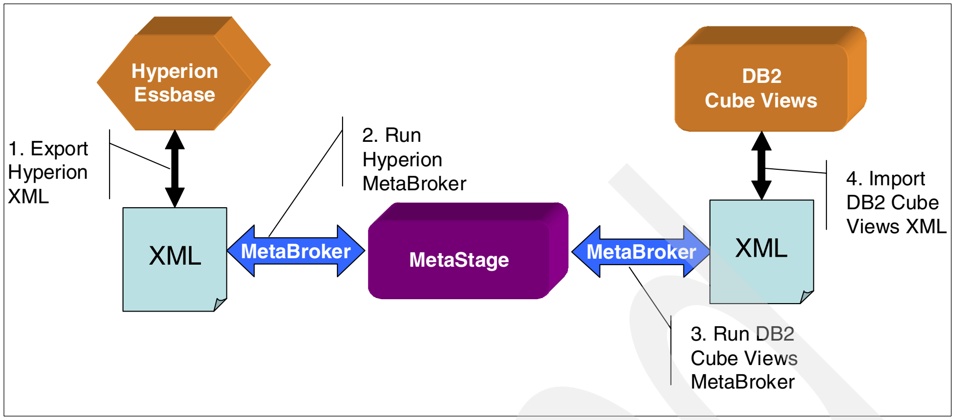


Figure 8-16 Summary metadata flow from Hyperion Essbase to DB2 Cube Views

Once you have exported the Hyperion metadata model to an XML file format, MetaStage can be used to import the Hyperion metadata into MetaStage. Figure 8-17 shows the import selection dialog.

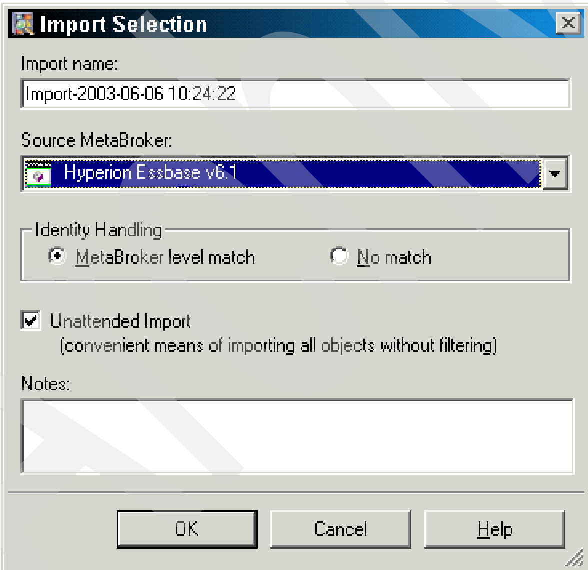


Figure 8-17 MetaStage Hyperion import dialog

After selecting the Hyperion MetaBroker to import the Hyperion metadata, we will see the metadata in MetaStage shown in Figure 8-18.

Note: The Ascential Hyperion 6.1 MetaBroker is forward compatible with release 6.5.

Here you must provide the Hyperion metadata XML file format for the MetaBroker to import.

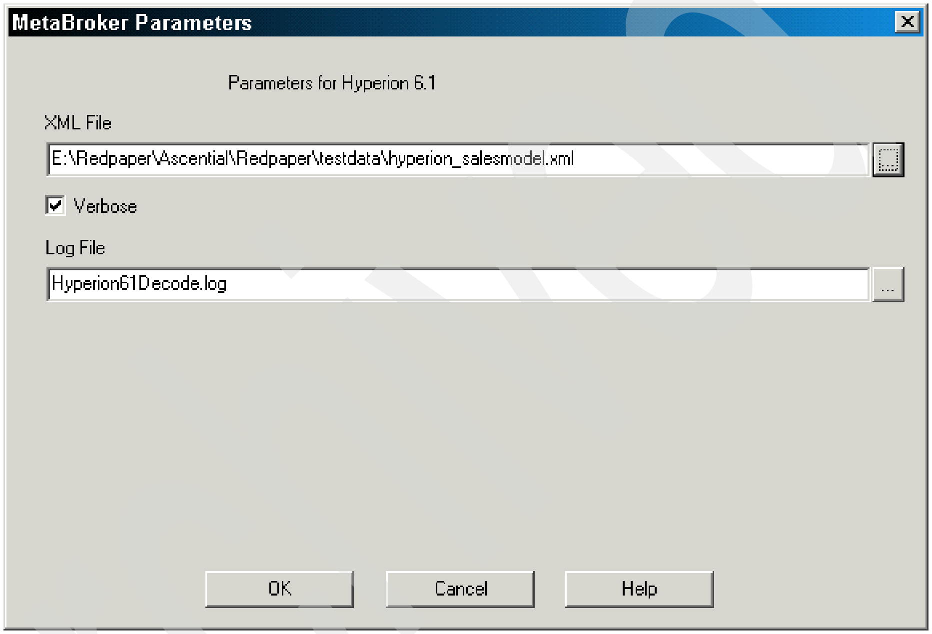


Figure 8-18 Hyperion import parameters dialog

After running the Hyperion MetaBroker to import the Hyperion metadata, we will see the metadata in MetaStage shown in Figure 8-19.

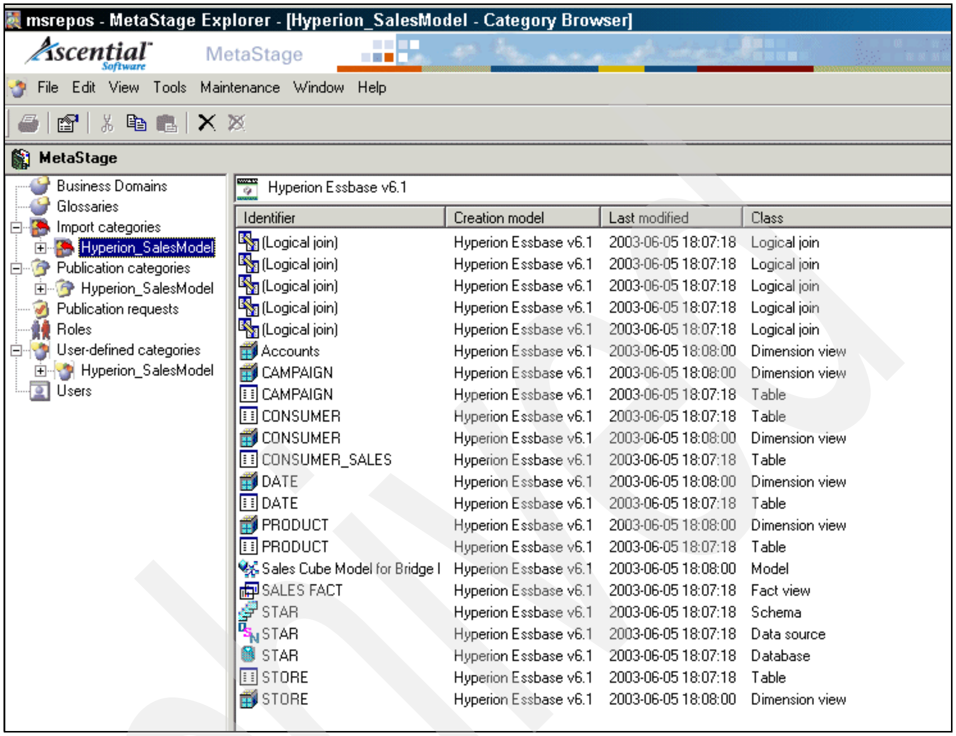


Figure 8-19 Hyperion metadata in MetaStage

Once the Hyperion metadata is in MetaStage, we export this metadata to DB2 Cube Views by subscribing to this metadata.

Figure 8-20 shows the new subscription dialog in MetaStage.

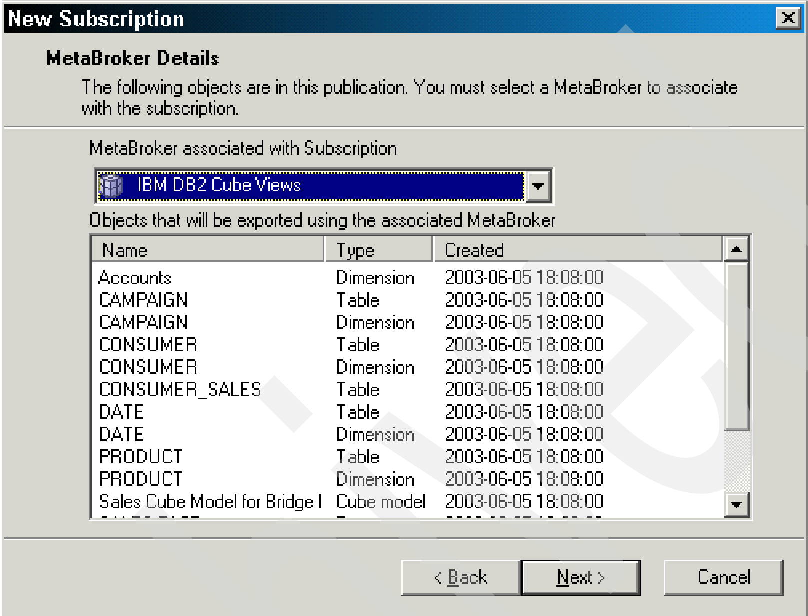


Figure 8-20 DB2 Cube Views subscription to Hyperion metadata

Selecting DB2 Cube Views and following the subsequent screens in the wizard will run the DB2 Cube Views MetaBroker shown in Figure 8-21.

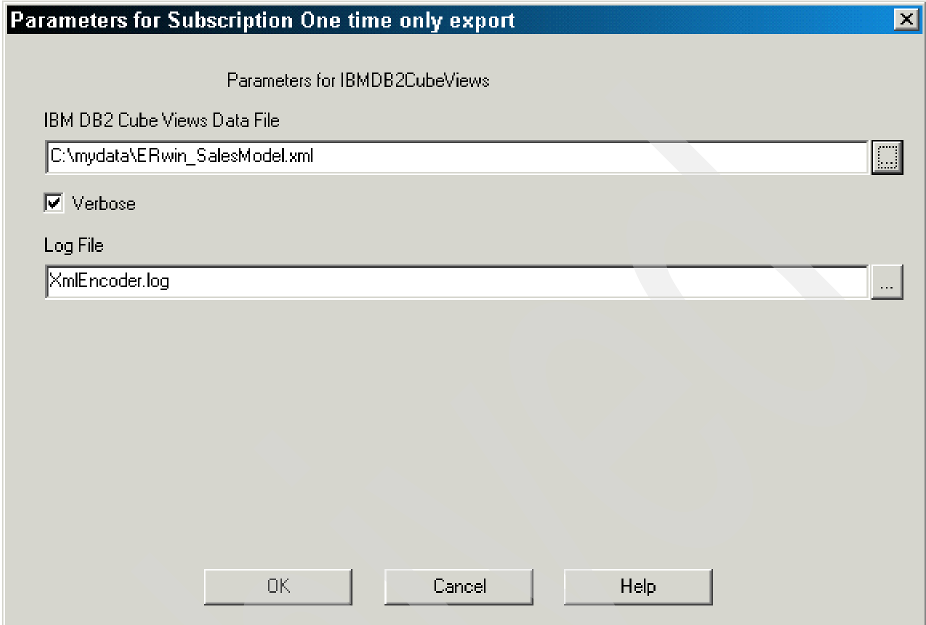


Figure 8-21 DB2 Cube Views MetaBroker parameters

You must specify the location of the DB2 Cube Views XML file for the MetaBroker to produce. After running the DB2 Cube Views MetaBroker, an XML file will be produced. Import this XML file into DB2 Cube Views using OLAP Center. The Hyperion cube model is now stored in DB2 Cube Views and ready for enhancement and use. The resultant cube model in DB2 Cube Views is shown in Figure 8-14 on page 289.

8.2.3 Performing cross-tool impact analysis

In this section we will examine the power of performing cross tool impact analysis with MetaStage. Within MetaStage, developers can perform in-depth dependency analysis to rapidly assess the impact of a change from a data warehouse, back to the operational data store, back to the staging database, all the way to the original data sources and the data modeling tool that might have been used to specify them. This delivers the functionality necessary to ensure that changes to data structures do not corrupt critical downstream reports.

To show cross tool impact analysis we will look at the column *TRXN_SALE_AMT* defined in the ERwin data model shown in Figure 8-22.

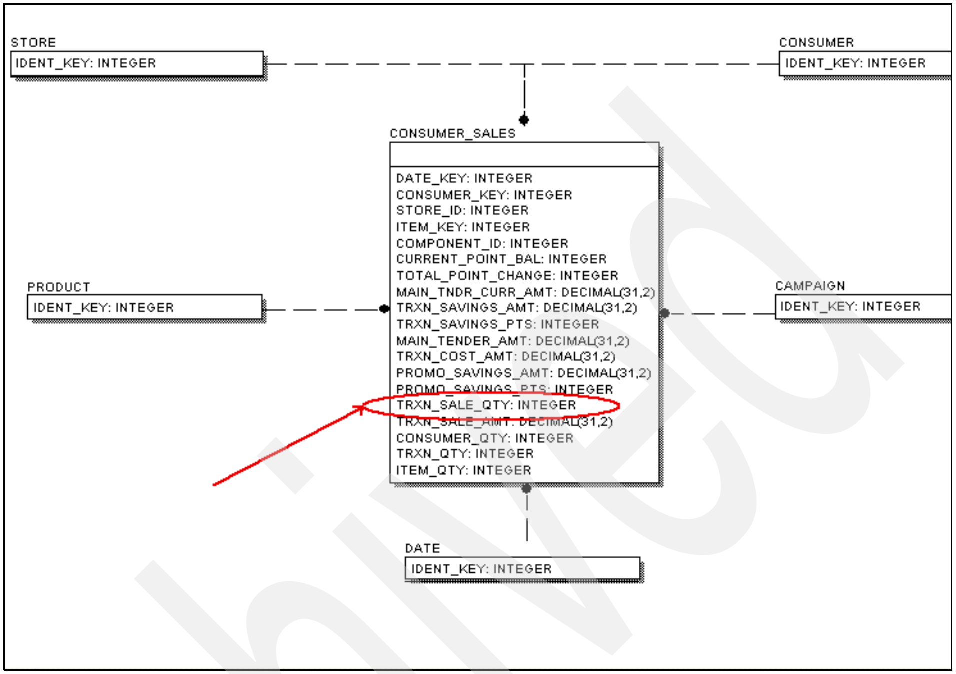


Figure 8-22 ERwin Sales data model

To perform cross tool impact analysis, MetaStage must have stored in its Directory metadata from all the tools you want to include in the analysis. For this example we will use metadata from ERwin and DB2 Cube Views. In 8.2.1, “Importing ERwin dimensional metadata into DB2 Cube Views” on page 281, we saw how to import the ERwin metadata into MetaStage and then export this metadata to DB2 Cube Views. We will assume that this step has been performed.

Assuming that we already have the ERwin metadata in the MetaStage Directory we now need to import the DB2 Cube Views metadata into MetaStage. To do this you must export the appropriate cube model from OLAP Center into an XML file format for the DB2 Cube Views MetaBroker to read.

From OLAP Center, choose the menu option **OLAP Center >Export** to see the export dialog shown in Figure 8-23. You must choose a cube model and location to export the metadata.

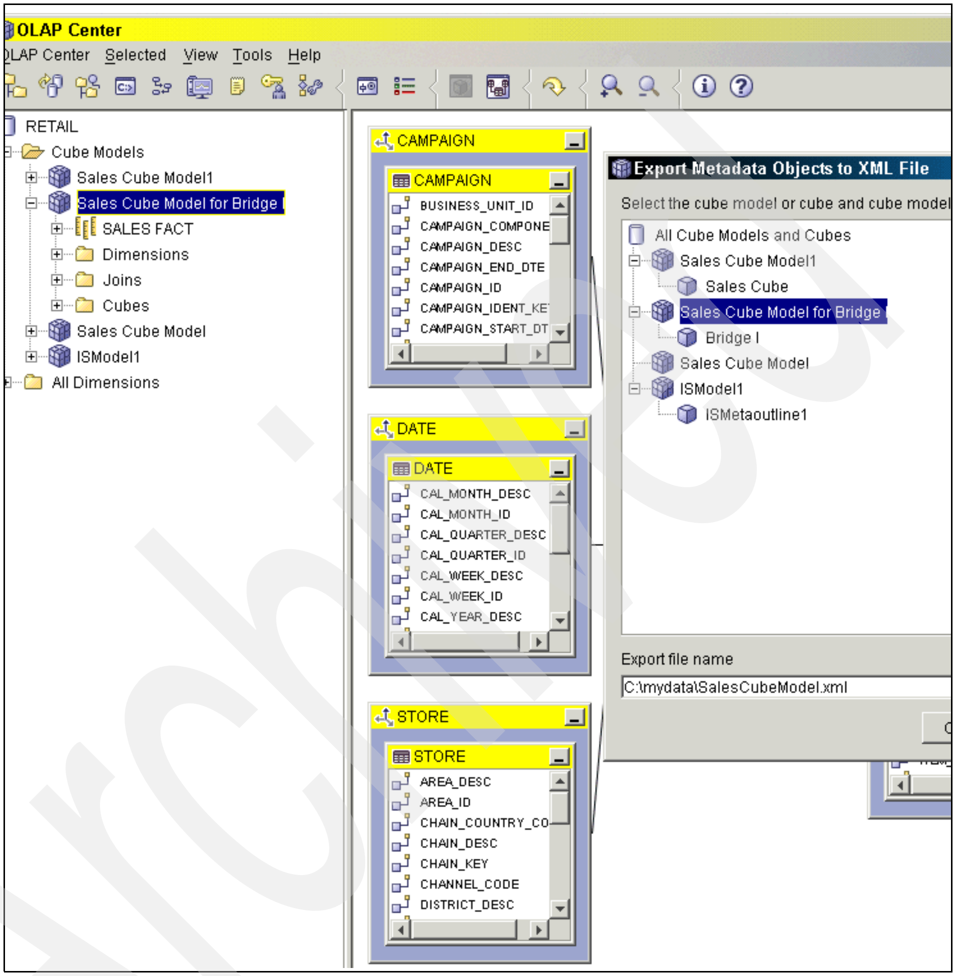


Figure 8-23 OLAP Center export dialog

When the OLAP Center XML export is complete, you will have an XML source file to use with the DB2 Cube Views MetaBroker. The DB2 Cube Views MetaBroker will read the metadata from this file to import into MetaStage.

You must import the DB2 Cube Views metadata by running the MetaBroker. Invoke a MetaStage import: at the **Import Selection** dialog, choose **IBM DB2 Cube Views** as shown in Figure 8-24 to run the import.

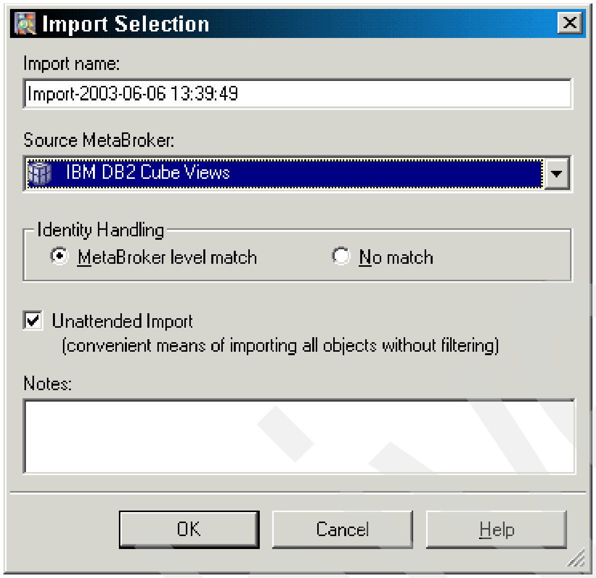


Figure 8-24 MetaStage import selection dialog

The DB2 Cube Views MetaBroker will require the location of the source XML file. Enter the location as shown by Figure 8-25 and then run the MetaBroker.

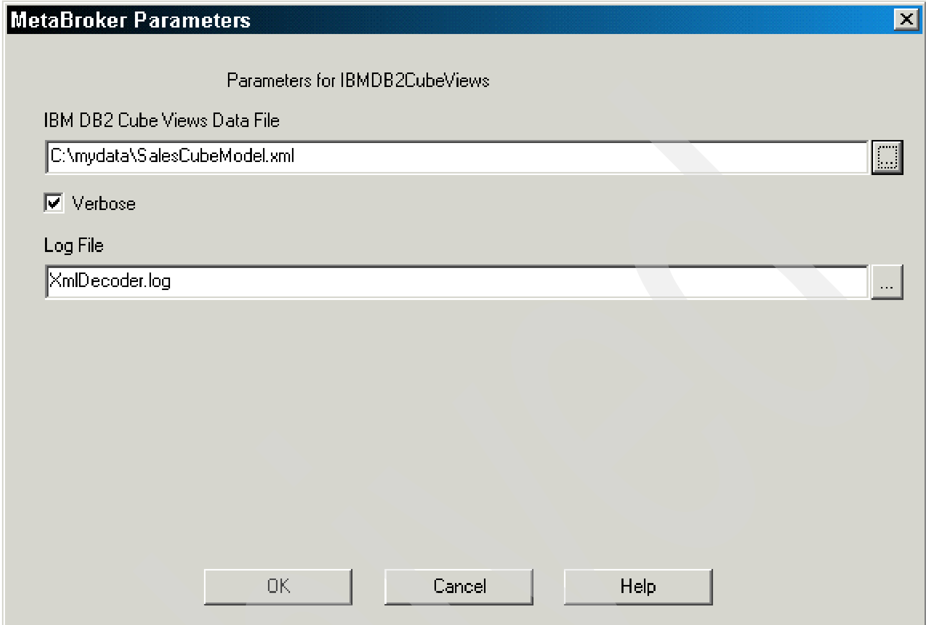


Figure 8-25 DB2 Cube Views MetaBroker parameters

After running the DB2 Cube Views MetaBroker you will have all of the metadata relating to your cube model in MetaStage shown in Figure 8-26. Notice that the ERwin metadata was already imported into MetaStage.

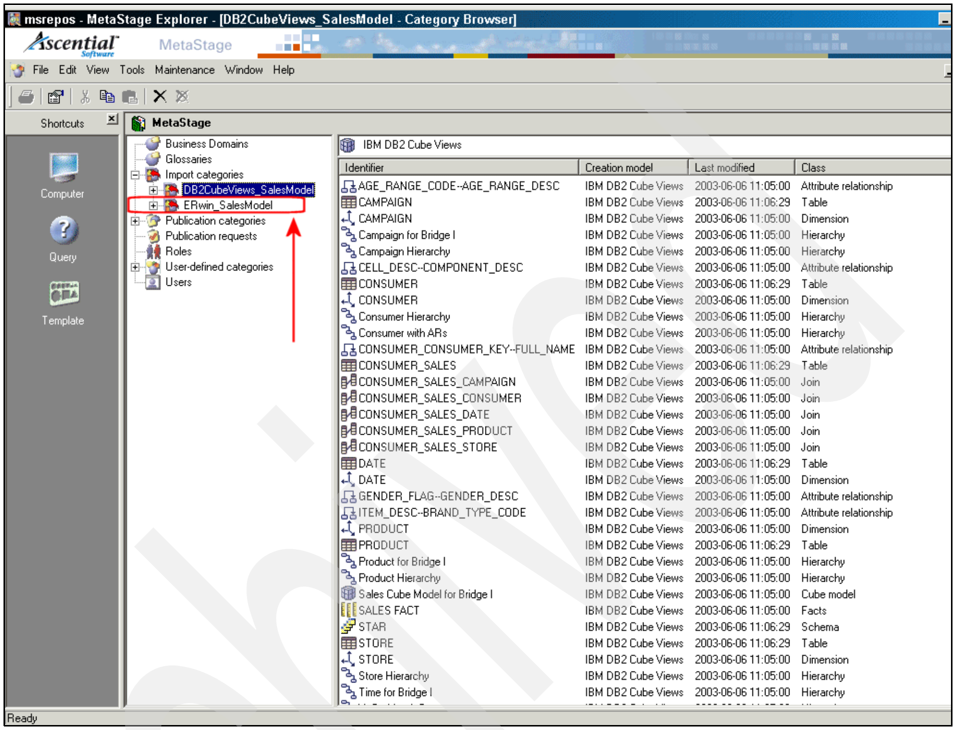


Figure 8-26 MetaStage after ERwin and DB2 Cube Views metadata import

Before we can run cross tool impact analysis queries we must run the **MetaStage Object Connector**. From MetaStage choose **Tools>Object Connector** to open the Object Connector dialog shown in Figure 8-27.

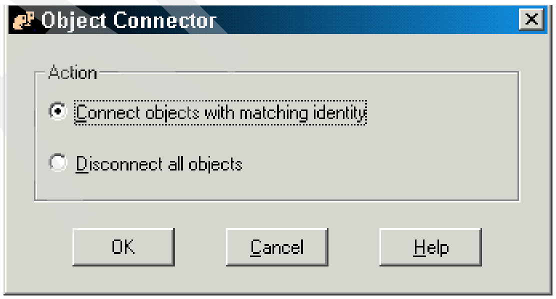


Figure 8-27 Object connector dialog

Running the Object Connector will establish (set) a special MetaStage relationship between objects called **Connected To**. You set the **Connected To** relationship when you want to designate objects as being semantically equivalent across different tools. For example, if you use the same object in two different tools, and import it from each into MetaStage as we have just done with ERwin and DB2 Cube Views, it appears in the MetaStage directory as two different objects. You can then set the **Connected To** relationship between the two instances and their contained objects (for example, columns) in order to keep track of the relationship between the objects and their child objects.

This enables you to run cross-tool impact analysis, to determine which connected and contained objects are affected if you make a particular change (see Figure 8-28). For example, you may ask, “If I change the design of my target DB2 table in ERwin, will my DataStage design and reports continue to work?”

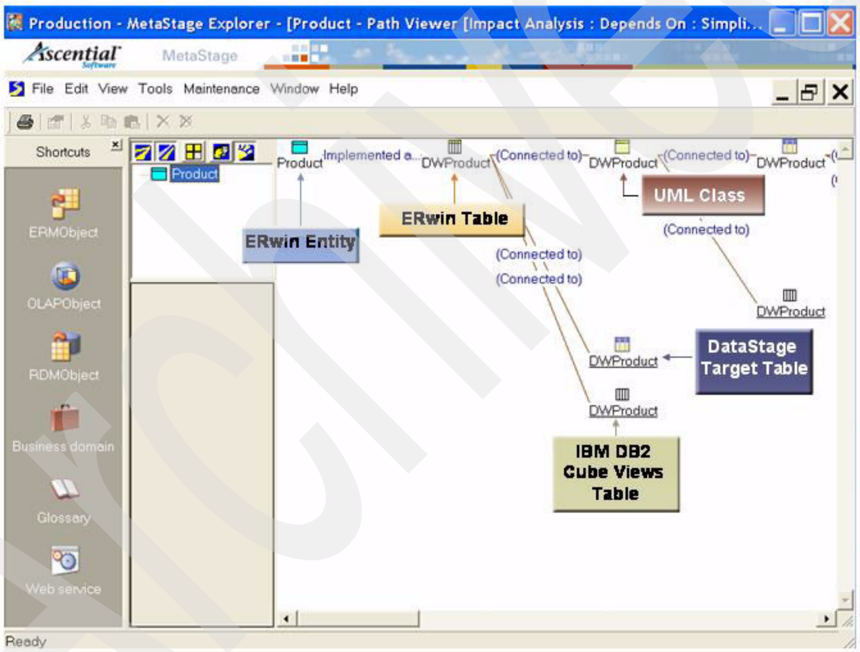


Figure 8-28 Impact analysis report showing Connected_To relationships

The Object Connector will automatically search the MetaStage Directory for objects that have equivalent identities and connect them and their respective child objects using the special **Connected To** relationship.

Note: Each object has an identity that usually includes its name.

Run the Object Connector to connect semantically equivalent objects in the MetaStage Directory. When all equivalent objects have been connected cross tool impact analysis can be performed. In our scenario we are interested in the column *TRXN_SALE_AMT* shown in Figure 21. In our example we want to show the impact of making a change to the *TRXN_SALE_AMT*. If we want to make a change to a column in our data model, we would typically make the change in the tool that stores the master copy of our data model. In this case, ERwin is storing the master copy of the data model metadata. Therefore, we should make the change in ERwin. Functionally in MetaStage this means that it will be more effective if we make ERwin the context from which we run our impact analysis query.

To make the ERwin copy of the *TRXN_SALE_AMT* to the root of our impact analysis query we must change the context in MetaStage so that we are browsing the ERwin metadata. When we run the impact analysis query however, we will see the impact of making a change to *TRXN_SALE_AMT* from ERwin across to DB2 Cube Views. Impact analysis queries always begin from some object. We will call this object the root object. Therefore, in our example *TRXN_SALE_AMT* will be the root of our impact analysis query.

Change the context in MetaStage to the ERwin **Import Category** shown in Figure 8-29.

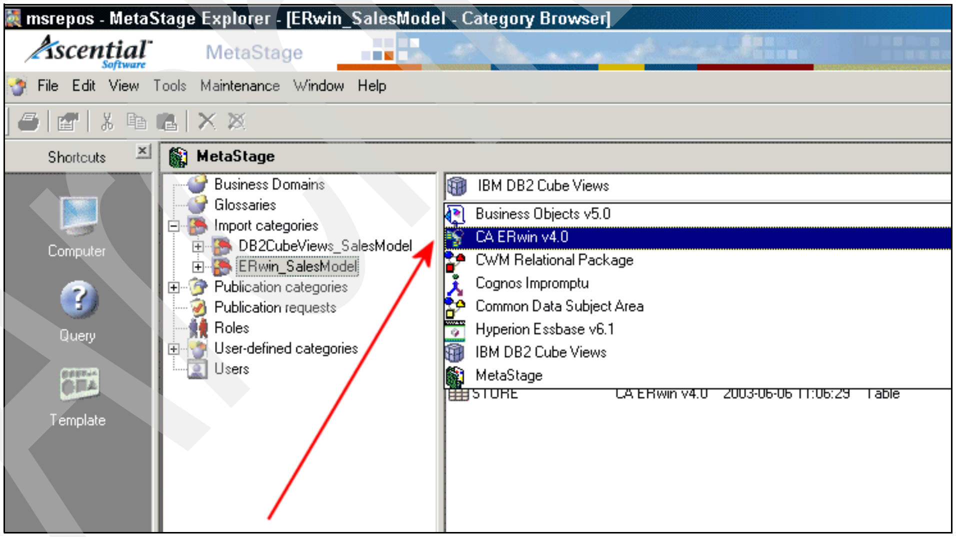


Figure 8-29 Switch from sourcing ERwin metadata view

Now use the **Browse Views** box to change to the ERwin view as shown in Figure 8-30.

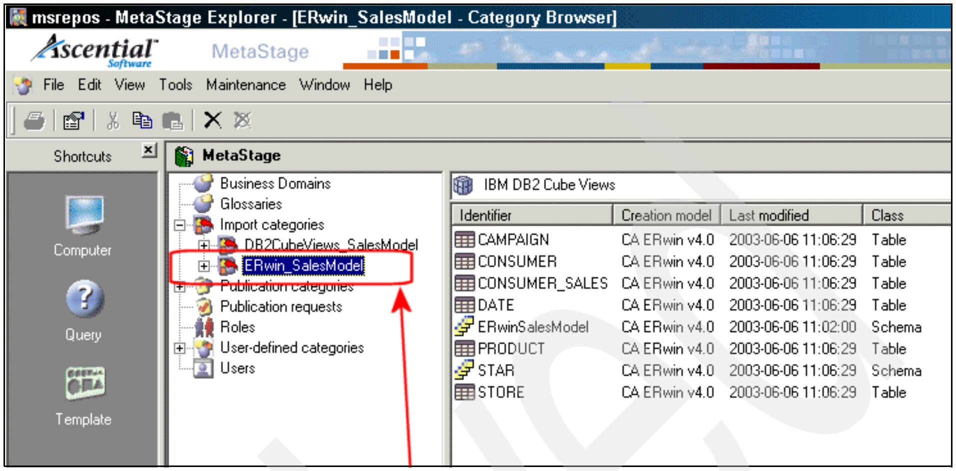


Figure 8-30 Select IBM DB2 Cube Views of ERwin metadata

You will now see the sales data model from the ERwin perspective shown in Figure 8-31.

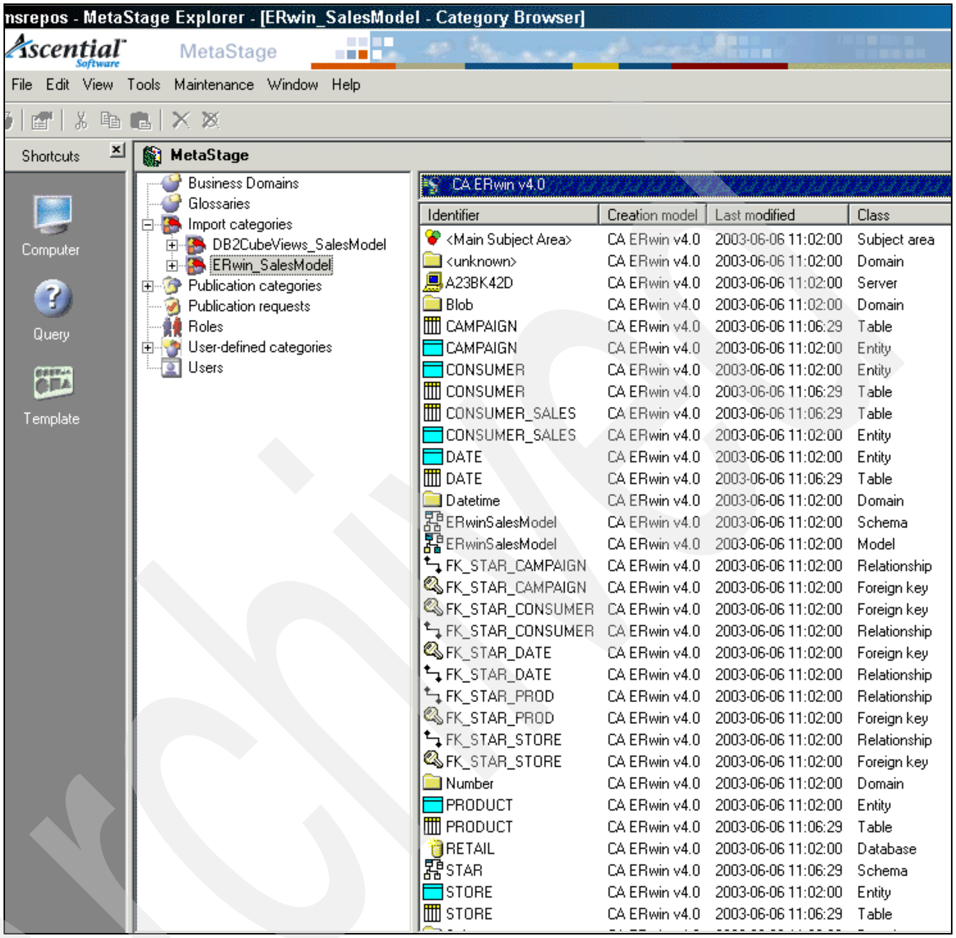


Figure 8-31 ERwin import category

From Figure 8-22 on page 296 we already know where the *TRXN_SALE_AMT* is: it is part of the CONSUMER_SALES table. To run the impact analysis query on the *TRXN_SALE_AMT* we need to navigate to the column object. To do this, right-click the CONSUMER_SALES table object and select **Browse from CONSUMER_SALES>CA ERwin 4.0** as shown in Figure 8-32.

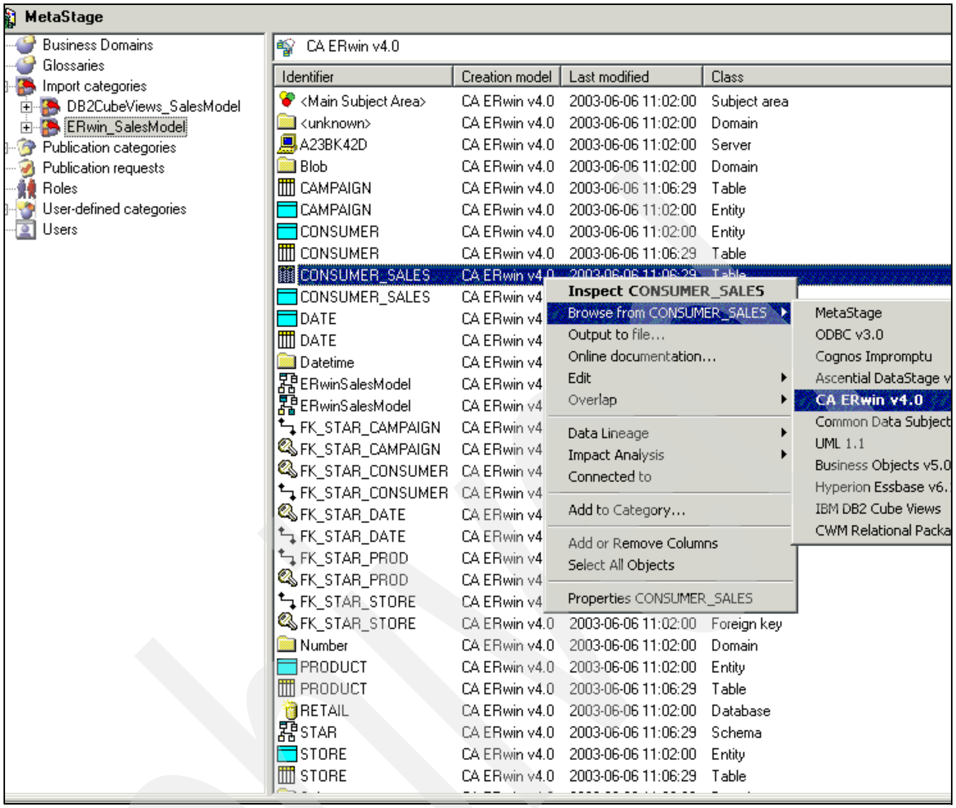


Figure 8-32 Browse from ERwin

You will be presented with the *CONSUMER_SALES* object. Then, navigate to the *TRXN_SALE_AMT* and right-click the object to select the menu option **Impact Analysis > Where Used** as shown in Figure 8-33.

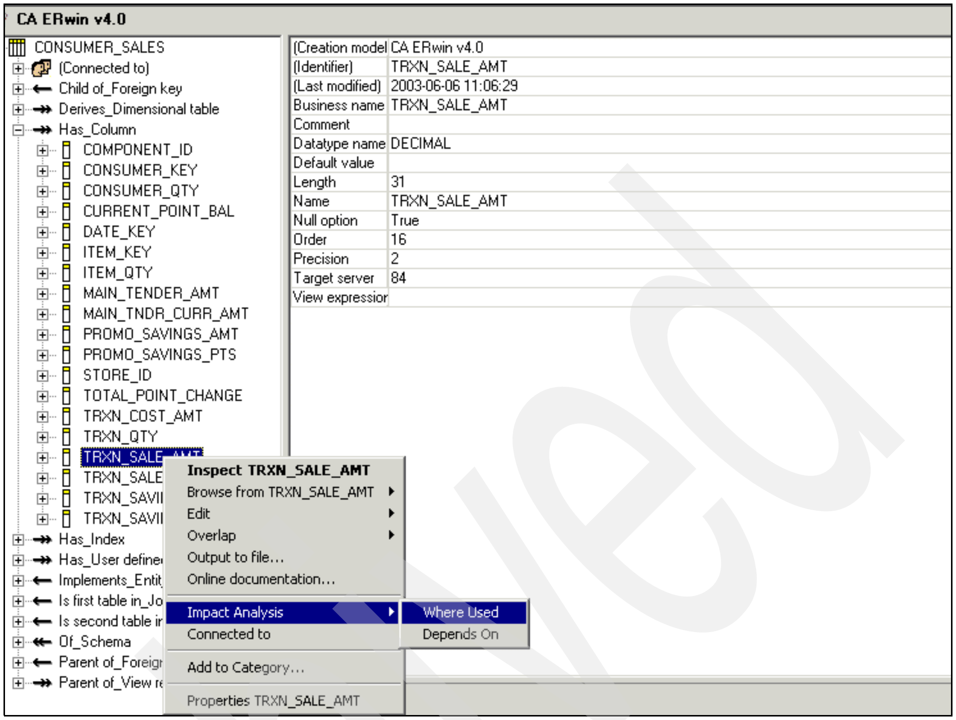


Figure 8-33 ERwin Where Used impact analysis menu

After running the impact analysis query, you will see the screen in Figure 8-34, showing the impact analysis across tools with the viewing context being that of ERwin only. To show both the ERwin and the DB2 Cube Views context on the same screen, click the button **Show Connected Objects via creation view**.

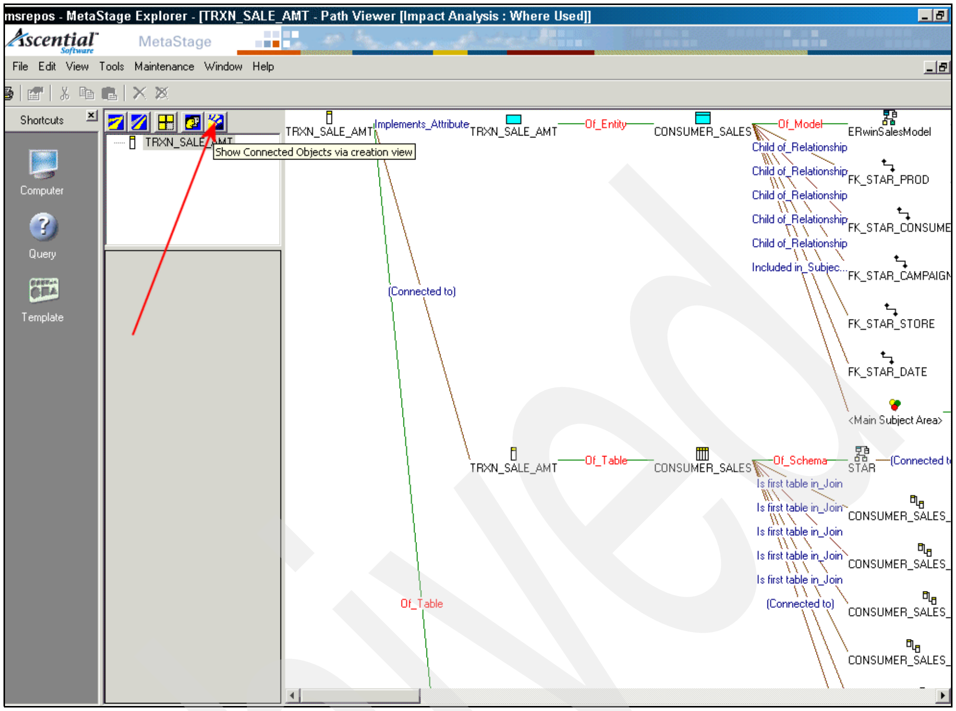


Figure 8-34 Impact analysis without connected objects via creation view

After clicking the **Show Connected Objects via creation view** button, you will be presented with a new impact analysis query path viewer. You will be able to navigate around the path viewer canvas by using the horizontal and vertical scroll bars.

If we scroll down to the bottom of the path viewer canvas, shown by Figure 8-35, we can see the impact of making a change to column *TRXN_SALE_AMT* to DB2 Cube Views. We can see that *TRXN_SALE_AMT* has a relationship *Of_OLAPMember* to the measure *Profit* that is subsequently used in other *Measures*.

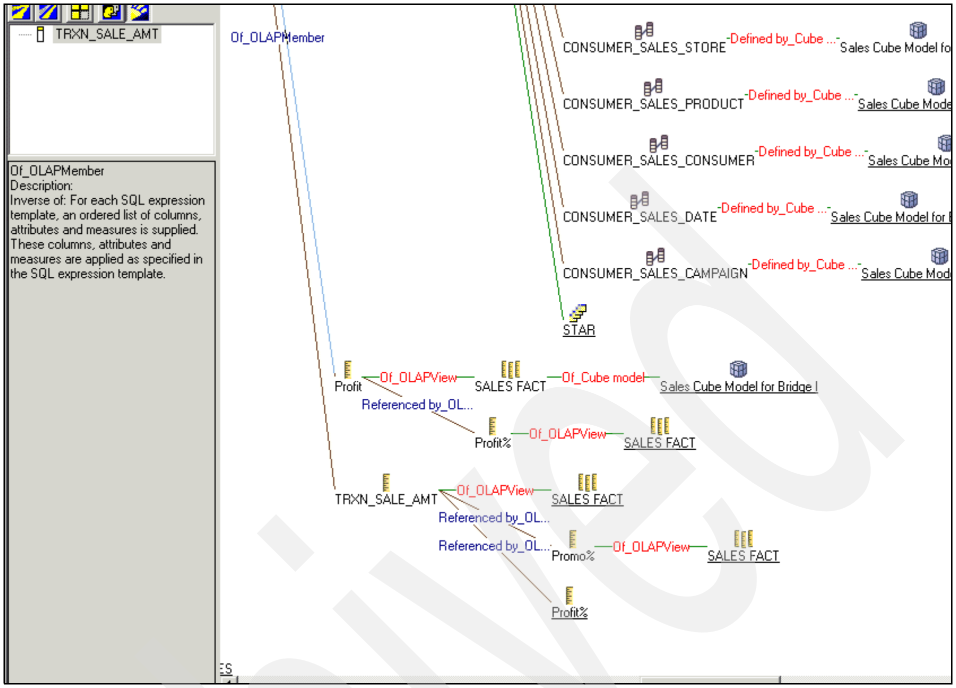


Figure 8-35 Impact analysis path viewer with creation view context

In addition to assessing the impact of making a change to *TRXN_SALE_AMT* in DB2 Cube Views, the impact analysis path viewer will show the impact to any other object to which the column is connected. In this case we will be able to browse the impact to ERwin itself by making the change.

Therefore, before making the change to *TRXN_SALE_AMT*, the data modeler must communicate with the OLAP developers to ensure that the change will not affect the *Profit* and other measures in DB2 Cube Views.

> **Note:** After creating the Impact Analysis, the user is able to do a right-mouse click and create HTML documentation.

8.2.4 Performing data lineage and process analysis in MetaStage

A major benefit of storing your metadata in the MetaStage directory is that you can investigate the history of your overall project and the potential impact of changing it. You can also examine the history of how your data warehouse was populated using Ascential DataStage. This section describes how you can use MetaStage to answer questions such as these:

- "When did this process last run and was it successful?" (*process analysis*)
- "Where did the last three writes to table A come from?" (*data lineage*)

Process analysis uses process metadata to tell you the history of process runs, including success or failure or warnings, parameters used, time and date of execution. It focuses on the path between job designs and the events they generate. This information is useful, for example, if you want to check whether past jobs ran successfully or run with errors.

Data lineage uses process metadata to tell you the history of an item of data, including its source, status, and when it was last modified. It focuses on the source table in a DataStage job and the derivations, transformations and lookups that connect it to a target table in the Operational Data Store or datamart. This information is useful, for example, if you are trying to resolve a data warehousing design problem, and need to collect information about the way the information was transformed for the business user from the source system.

Data lineage overview

A data lineage path (see the example in Figure 8-36) shows the source table, the target table, the links between them, and the events involved for a DataStage job whose events were captured by the MetaStage Process MetaBroker. After you run a job and capture process metadata, as described above, you can create a data lineage path from captured objects by selecting a DataStage object type that has semantic overlap with the following MetaStage classes:

- Data Schema (DataStage: DSN)
- Data Store (DataStage: DataStore)
- File (DataStage: File)
- Data Collection (DataStage: TableDefinition)

Figure 8-36 shows an example of a data lineage query showing a simple DataStage source to target mapping.

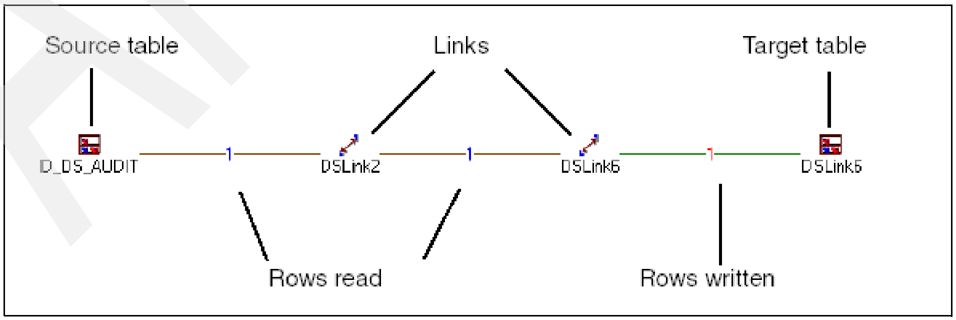


Figure 8-36 MetaStage data lineage example

When you right-click a captured object in one of the above classes, and choose **Data Lineage> Find Sources**, or **Data Lineage> Find Targets**, a data lineage path appears in the Path Viewer if a path is available for that object. The path includes the source data collection, the target data collection, the links that connect them, and either the number of rows read and written or the time and date of the event.

Data lineage queries allow you to answer the following types of questions:

- Which jobs updated table *Sales* in the last two days?
- What was the overall status of DataStage job *CashItems*, and did it report any unusual occurrences related to table *Sales*?
- What data sources did job *CashItems* use? How exactly did it transform them into *Sales*?

Process analysis overview

MetaStage captures process metadata to keep track of the execution history of warehouse activity processes, such as DataStage job runs. You can use Process Analysis on this process metadata to investigate how running various processes has affected your data warehouse. You can discover details about when jobs run, what parameters they use, whether they are successful, and if not, why not.

> **Note:** Data lineage queries can also report on the success or failure of processes, but only insofar as the processes affect specific tables that were written to or read from. Process analysis queries look at executable objects such as DataStage jobs and all the resources touched by the events and activities they generate.

Jobs that run successfully generate events associated with the data resources they access. Jobs that are aborted generate events identifying the point at which failure occurred.

When you capture process metadata with the Process MetaBroker, these events, along with the related activities and resources, are stored in the MetaStage Directory. MetaStage uses objects of the Event, Activity, and Software Executable classes to create Process Analysis paths. There are two types of process analysis paths:

- **Find Runs:** These paths start with a specified software executable and continue through sets of events to the resources touched. (In the MetaStage view, a DataStage job design is an instance of the Software Executable class.)

- **Find Executables:** These paths start from a specified event or activity (a run or component run) and continue to the software executable from which it originated.

Both paths follow the same data model containment and dependency relationships used in impact analysis, but process analysis paths provide a more direct display of the relationships between executables and runs.

Figure 8-37 is a Find Runs path example, taking *ActivityJob01UD*, as the source software executable.

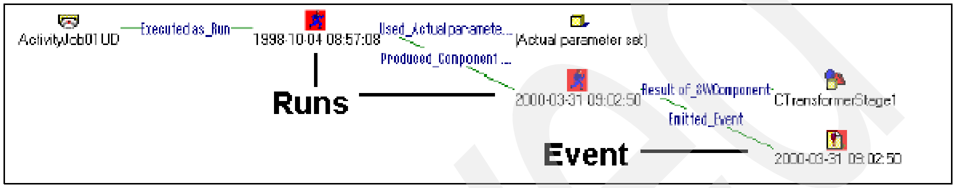


Figure 8-37 Process analysis example

The shading of the two run icons and the shaded area around the event icon (lower right) illustrates what you see if a run of the selected executable fails. This shading appears as red in the MetaStage Path Viewer and indicates failed runs. You can trace the failure to a specific event in the run, and see that, in this case, the failure is associated with *CTransformerStage1*. You can right-click the event and inspect it to see the error message DataStage wrote to the log, which is stored in the Event's Message attribute. By inspecting the *Actual parameter set* object, you can see the parameter values used for this run.

Note: The same process analysis can be done using the DataStage view.

Capturing operational metadata from DataStage

In this scenario we will use the Process MetaBroker provided with MetaStage to capture metadata about running DataStage jobs. For this example, we will run a DataStage job that has already been built to load our sales model star schema. In this example, the DataStage job assumes that a transactional or other system produces flat sequential files containing data to load into the sales model star schema.

The basic flow of data in the DataStage job shown in Figure 8-38 assumes that some other system produces flat sequential files. Although DataStage has the capability to access almost any kind of source system (including, but not exhaustive, Siebel, Oracle Applications, SAP R/3, PeopleSoft, J.D. Edwards, RDBMS, mainframe flat files and database sources, JMS, WebSphere MQ, Web services, and XML) to load directly into the DB2 star schema, this example does not show this configuration. The DataStage job will read the data from each file, transform the data, and load the results into the respective tables. As the job runs, certain operational process metadata will be produced and captured by the MetaStage Process MetaBroker.

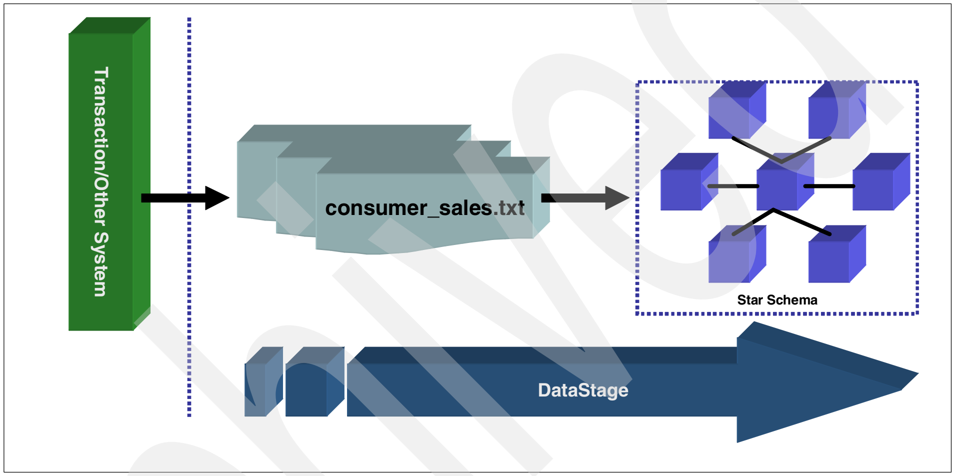


Figure 8-38 DataStage job flow

MetaStage uses the Process MetaBroker to capture metadata generated when you run data warehouse activities. This metadata provides MetaStage with information about the following things:

- Time when a data warehouse update occurred
- Completion status of the data warehouse update
- Tables that were updated when it ran

Configuring the operational metadata components

The complete set of operational metadata components involved in producing process metadata for MetaStage are shown in Figure 8-39.

> **Note:** For operational process metadata component installation instructions, refer to the MetaStage documentation “Installation Instructions.”

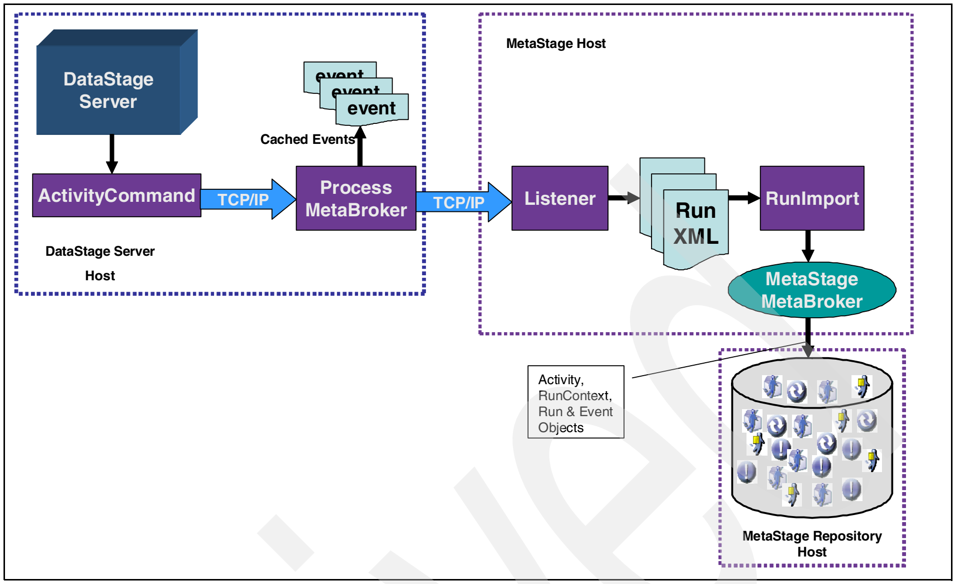


Figure 8-39 Operational metadata components

The Process MetaBroker is installed and configured on the server host running DataStage. When a DataStage Server job runs, the job uses the **ActivityCommand** interface installed by the Process MetaBroker to communicate process events via TCP/IP to the Process MetaBroker. As the DataStage Server job continues to run, events will be cached in the Process MetaBroker events directory specified in the Process MetaBroker configuration file.

When an end event is received by the Process MetaBroker signaling that the DataStage Server job has completed, the Process MetaBroker is ready to transmit the run to MetaStage. The Process MetaBroker will transform each individual event file for a particular run into a single XML file and send it to the Listener running on the MetaStage host. The Listener is a process that runs on the MetaStage host and listens on a particular port defined in a configuration file. The Listener's purpose is to wait for the Process MetaBroker to send completed runs to the MetaStage host.

When the Process MetaBroker packages up a run in an XML file, it will connect to the Listener on the MetaStage host and transmit the XML file. The Listener will store the XML file in a directory specified in a configuration file. Once the run XML files are on the MetaStage host, they can be imported into the MetaStage Directory using the **`RunImport`** utility. The **`RunImport`** utility will read the run XML file/s and import them into the MetaStage Directory. Performing a **`RunImport`** will result in Activity, RunContext, Run and Event objects being created in the

MetaStage Directory. These objects will have relationships to DataStage ETL, design metadata objects so that data lineage and process analysis can be performed.

Each main operational metadata component, Process MetaBroker, Listener and RunImport, has a configuration file that allows for many user options to provide automation and flexibility.

How to configure a DataStage project to produce operational metadata that will be used for both data lineage analysis of the DataStage design and its operational metadata is documented in detail in Appendix A, “DataStage: operational process metadata configuration and DataStage job example” on page 639.

Importing operational metadata

Two major steps are needed to capture the operational metadata:

1. Configure the operational metadata components
2. Create DataStage jobs that will produce process metadata.

When these have been completed, we will perform the steps required to import the process metadata so it is ready for data lineage and process analysis queries.

Process metadata is useful on its own. However, process metadata is most valuable when it can be used to trace back to events that happen to or in relation to design metadata or physical metadata. For MetaStage to provide the most valuable data lineage and process analysis results, the DataStage job design metadata must already be in MetaStage.

We must now import the DataStage job design metadata into MetaStage. To do this we will create a DataStage import category called *DataStage_p0* in MetaStage as shown in Figure 8-40.

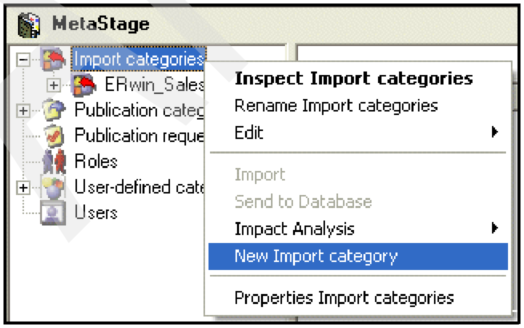


Figure 8-40 MetaStage: new import category

Now we will import the contents of DataStage project *p0* into the *DataStage_p0* Import category shown in Figure 8-41.

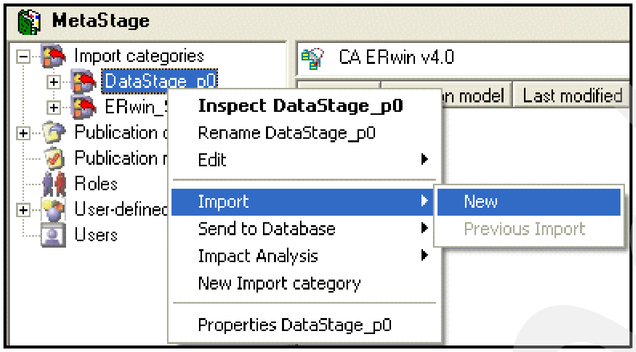


Figure 8-41 Importing multiple DataStage job designs from a DataStage project

After clicking **New** as shown in Figure 8-41, the **Import Selection** dialog will be presented. Here we select **Ascential DataStage v7** as the source MetaBroker shown in Figure 8-42.

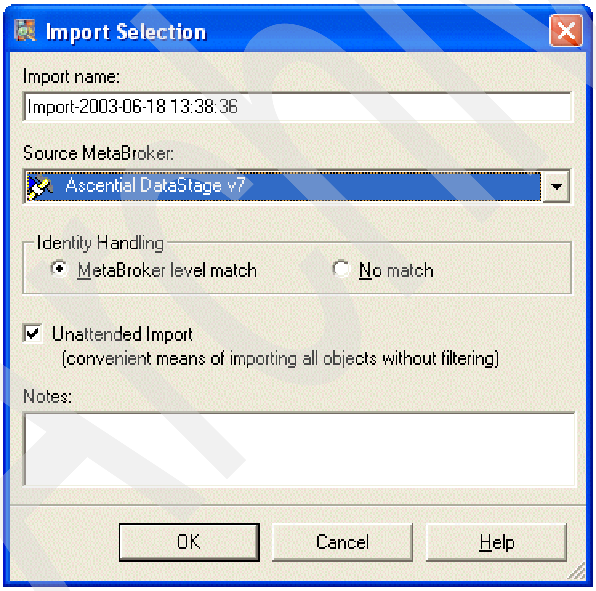


Figure 8-42 MetaStage: DataStage import

We will accept the defaults and click **OK**. The DataStage MetaBroker parameters dialog will be shown. Accept the defaults and click **OK**.

The DataStage login dialog will be shown. For our example, we are connecting to the host *wb-arjuna* and the project p0 as shown in Figure 8-43. We will click **OK** here and the DataStage MetaBroker will import the contents of the *p0* DataStage project into MetaStage.

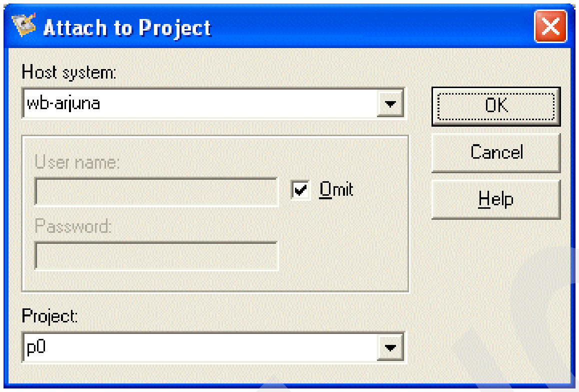


Figure 8-43 DataStage login

We now have all the data model and job design metadata in MetaStage. Before we can run the DataStage jobs, we must look at the *Locator* concept a little more.

Locators in MetaStage

Because of the inconsistencies in certain ODBC drivers, MetaStage cannot always match captured table definitions to the identical table definitions previously imported from DataStage. When MetaStage does match a captured table definition to a previously imported table definition, it does not create a new object in the directory, but instead connects the run-time process metadata information to the originally imported table definition. You can then use data lineage and process analysis to see which events touched this table definition when the job was run, and to see which column definitions the table definition contains. (If you are viewing the objects in the MetaStage view, table definitions are called *data collections*, and column definitions are called *data items*.)

When it cannot match a table definition, MetaStage creates a new table definition with the same name as the table definition in the job design, and adds it to the directory. However, this table definition will have no column definitions, because the Process MetaBroker does not capture column definitions during runtime. The value of its Creation Model attribute is *MetaStage* instead of *DataStage*.

To avoid any mismatch in process metadata and DataStage metadata, we will create a Locator table in the database where the DataStage job is running. To ensure that imported and captured table definitions always match, create locator tables in the source and target databases used by the DataStage job. The Locator table must include the name of the computer, the software product, and the data store, and must be created with the appropriate permissions so that it can be accessed by the necessary users.

When you import table definitions from these databases into DataStage, a fully qualified locator string is created for them based on the information in the Locator table. This locator information remains with the table definitions when they are imported into MetaStage or captured by the Process MetaBroker.

The Locator table will be used as a lookup while the DataStage jobs are running so that the DataStage engine can create event files with the correct Locator path for DataStage objects. The DDL to create the locator table is:

```
CREATE TABLE MetaStage_Loc_Info (
   Computer varchar(64),
   SoftwareProduct varchar(64),
   DataStore varchar(64));
```

To create the locator table we submit the SQL above to the DB2 connection we established in Figure A-13 on page 652 which was RETAIL. Next we must insert an entry in the Locator table for the DataStage Server engine to use. The SQL for our example is:

```
insert into db2admin.MetaStage_Loc_Info (Computer, SoftwareProduct,
DataStore) values ('wb-arjuna', 'DB2', 'RETAIL');
```

For our example the values for the SQL insert statement can be obtained by using the MetaStage Class Browser. We will open the MetaStage Explorer and click the **Computer** icon in the Shortcut bar on the left.

In our example, there will be two Computer objects, one created by the ERwin MetaBroker and one created by the DataStage MetaBroker. We will expand the *wb-arjuna* object imported by ERwin and further expand the *Hosts_Resource* and *Created_Resource* relationships. As shown in Figure 8-44, the values displayed for our example are the values we wish to insert into the Locator table.

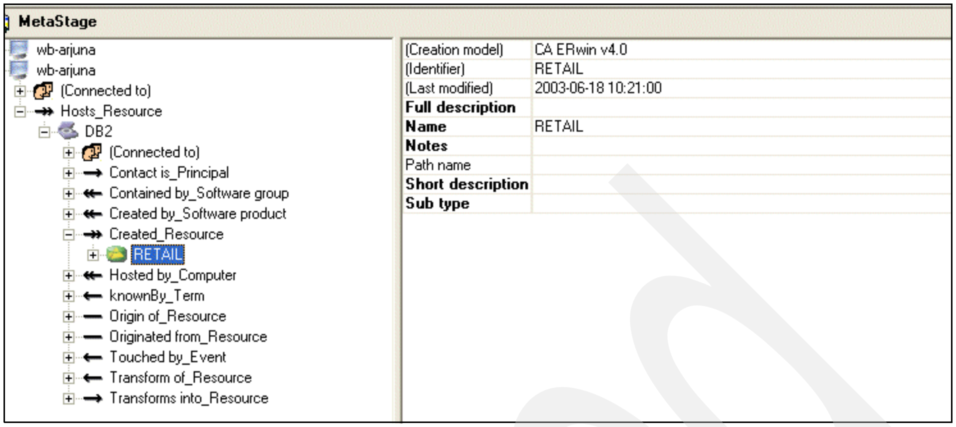


Figure 8-44 MetaStage: computer instances

This means that when TableDefinition objects are created in the process metadata, the Locator path used will be the path we specify in the locator table MetaStage_Loc_Info described above. After submitting the SQL to insert the Locator path entry, the table will have the following row:

```
SELECT * FROM MetaStage_Loc_Info;
Get Data All:
"COMPUTER", "SOFTWAREPRODUCT", "DATASTORE"
"wb-arjuna", "DB2", "RETAIL"
1 row fetched from 3 columns.
```

Now we have created and populated the Locator table we proceed to run the DataStage jobs to produce process metadata. Since we have already configured the Process MetaBroker and Listener, capturing process metadata for our jobs is a simple matter of running the DataStage jobs.

We will open the DataStage Director shown in Figure 8-45 to run our DataStage jobs. We have two jobs for this example, *LoadDimensions* and *LoadFacts*.

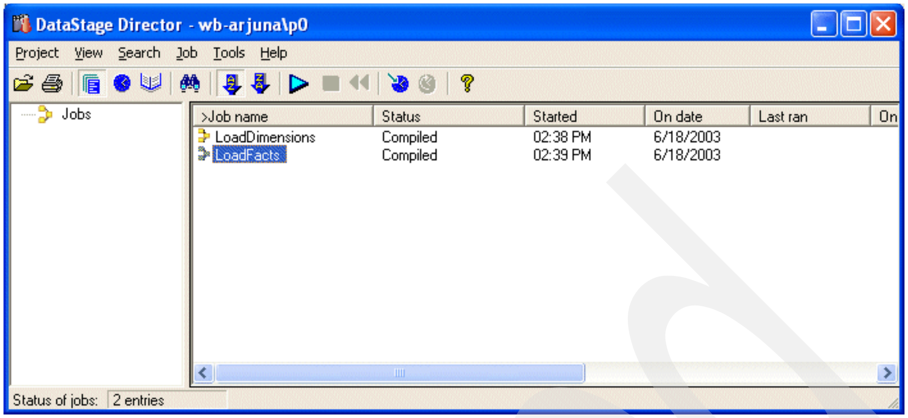


Figure 8-45 DataStage Director

1. First we will run the *LoadDimensions* job. To do this we will highlight the *LoadDimensions* job and click the **Run Now** button from the tool bar. Running this job will produce a run XML file on our Listener host. The location of the run XML file will be the value we entered into the Listener configuration file in Table A-3 on page 644. When the *LoadDimensions* job is complete we will run the *LoadFacts* job to produce another run XML file. The results of our DataStage job runs are shown in Figure 8-46.

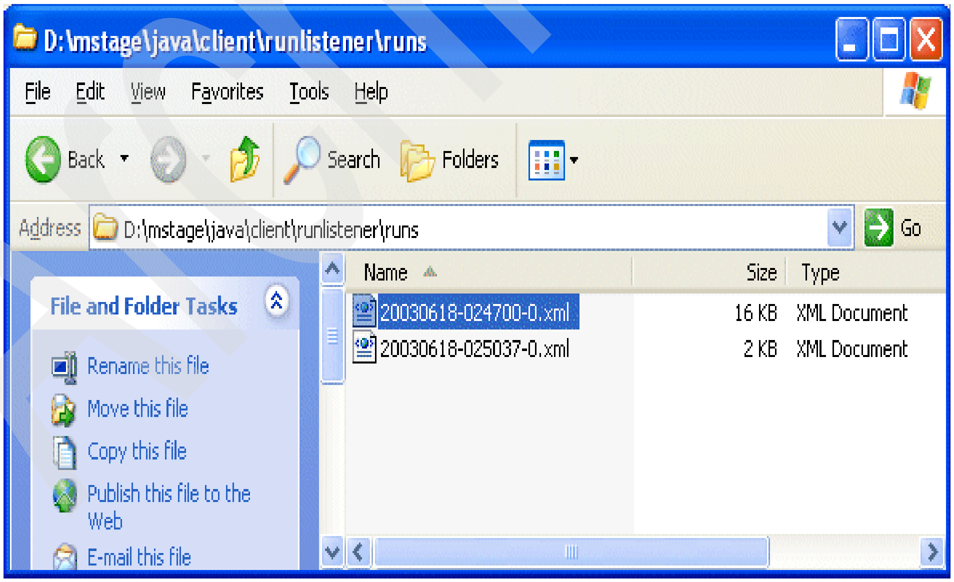


Figure 8-46 DataStage run results

For our example, we ran two jobs and we have two resultant XML files. All the events and activities associated with the job runs are contained in the XML files. A sample of the run XML is shown in Example 8-1.

Example 8-1 Sample run XML

```
<Event Type="Write" StartedAt="2003-06-18T18:44:52"
FinishedAt="2003-06-18T18:44:55" RowCount="17">
- <DataResourceLocator>
  <LocatorComponent Class="Computer" Name="wb-arjuna" />
  <LocatorComponent Class="SoftwareProduct" Name="DB2" />
  <LocatorComponent Class="DataStore" Name="RETAIL" />
  <LocatorComponent Class="DataSchema" Name="STAR" />
  <LocatorComponent Class="DataCollection" Name="CAMPAIGN" />
  </DataResourceLocator>
```

We can see that a **Write** event started and affected the LocatorComponent: **wb-arjuna->DB2->RETAIL->STAR->CAMPAIGN**. We can see that the DataStage Server used our Locator entry as part of the Locator path that was inserted into the run XML.

Now that we have the run XML files produced, we will import the runs into the MetaStage Directory using **RunImport** shown in Figure 8-39 on page 313. **RunImport** is designed to be scheduled to run on a regular basis after DataStage runs your warehouse activities. RunImport can be scheduled using any Windows command scheduler including the Windows @ scheduler. Therefore it is recommended that the MetaStage Explorer be shut down during the **RunImport** process.

For our example we will simply open a command window and run the default RunImportStart.bat file provided with the installation of **`RunImport`**. We will navigate to the RunImport installation directory and run the batch command:
`D:\mstage\java\client\runimport>RunImportStart.bat`

The output from the RunImport for our example is shown in Figure 8-47.

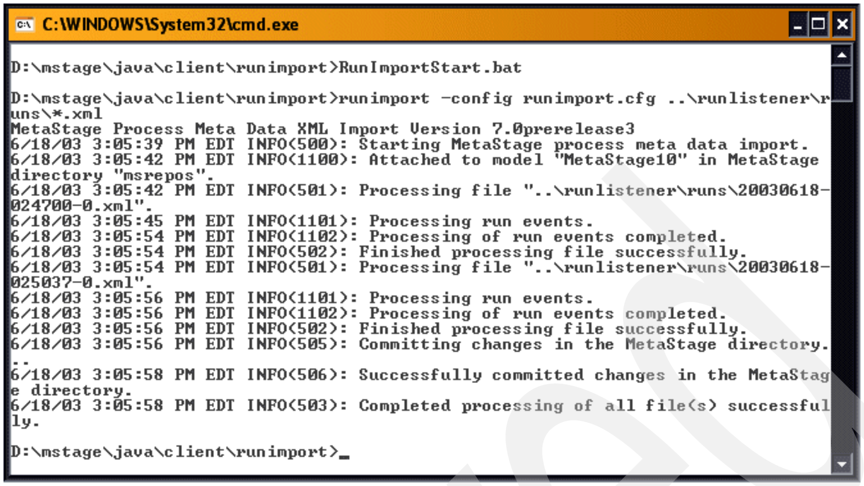


Figure 8-47 RunImport output

We can see that the two run XML files were successfully processed and committed to the MetaStage Directory. Any associated RunImport log information will be in the log file location specified in the RunImport configuration file as shown in Table A-4 on page 646.

Running a data lineage query

In our example we have now populated the MetaStage Directory with DataStage run process metadata. We are now in a position to run data lineage queries in MetaStage to examine what happened to our design metadata during a run.

For our data lineage query we will look at the *CONSUMER* dimension table and what happened to it during DataStage job runs. To do this we will open the MetaStage Explorer and examine the *CONSUMER* TableDefinition object.

In Figure 8-48, we have opened the MetaStage Explorer and clicked on the **DataStage_p0** Import category to show the DataStage objects. Highlighted is the *CONSUMER* TableDefinition object.

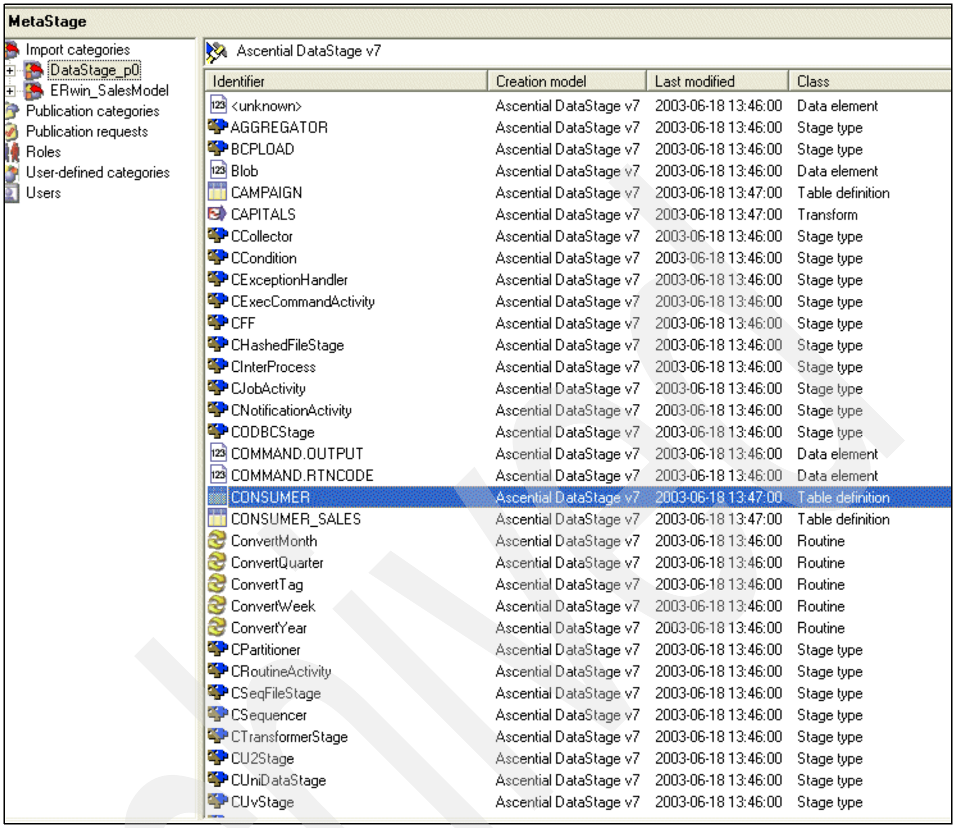


Figure 8-48 MetaStage category browser

We will now right-click the *CONSUMER* object to expose the context menu for the object. For our data lineage example we will find the find the sources of the *CONSUMER* object as shown in Figure 8-49.

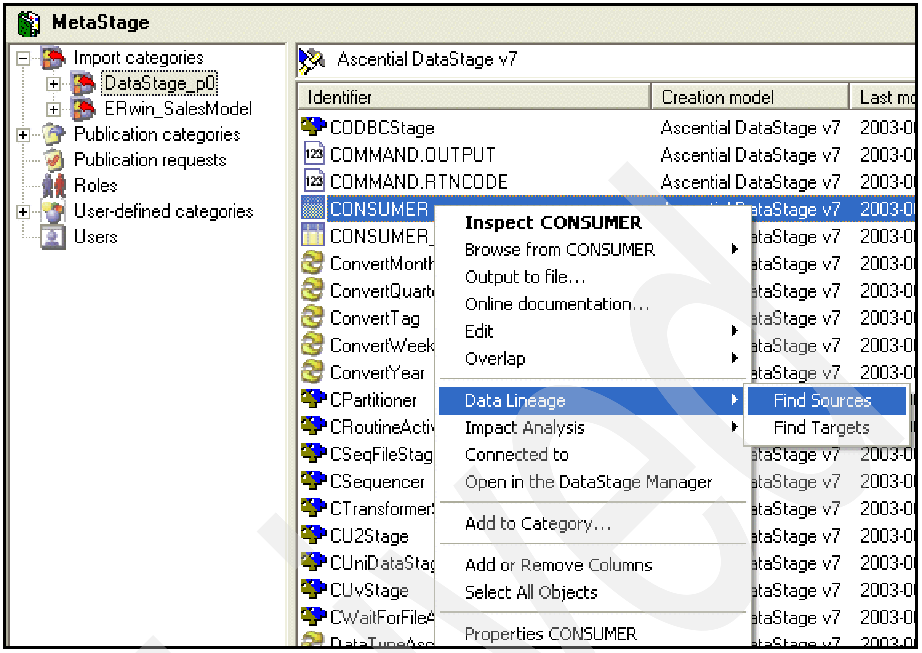


Figure 8-49 Data lineage menu

By clicking the **Find Sources** menu option, we see the data lineage path shown in Figure 8-50.

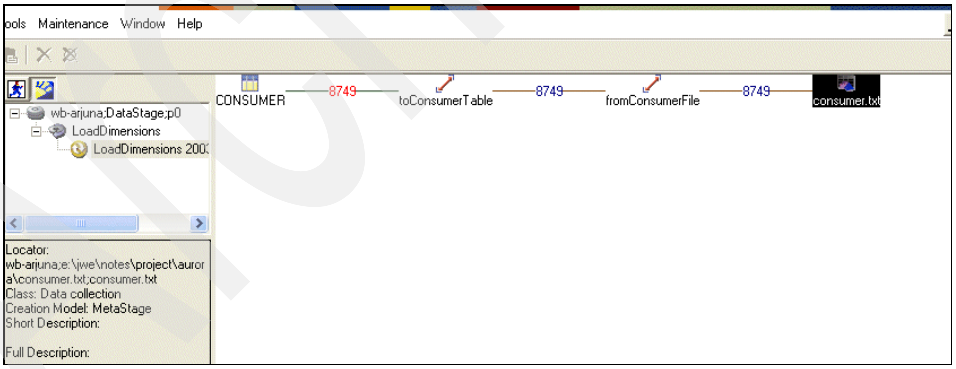


Figure 8-50 Data lineage path

We see from Figure 8-50 that the *CONSUMER* table was loaded from the consumer.txt file and that in this particular job 8749 rows were inserted into the table. The value in red indicates a write event and the values in blue indicate a read event. Each object on the data lineage can be inspected in detail to find out more information about the particular object.

For example, the *toConsumerTable* link could be opened to drill down into the transformations that occurred to each column on the Link object.

Running an operational analysis query

In this example we will perform an analysis looking at all of the DataStage job runs for the *LoadFacts* job design from DataStage. We will analyze the DataStage runs associated with this particular design.

To do this we will open the MetaStage Explorer and click the *DataStage_p0* import category and scroll down to the *LoadFacts* job design shown in Figure 8-51.

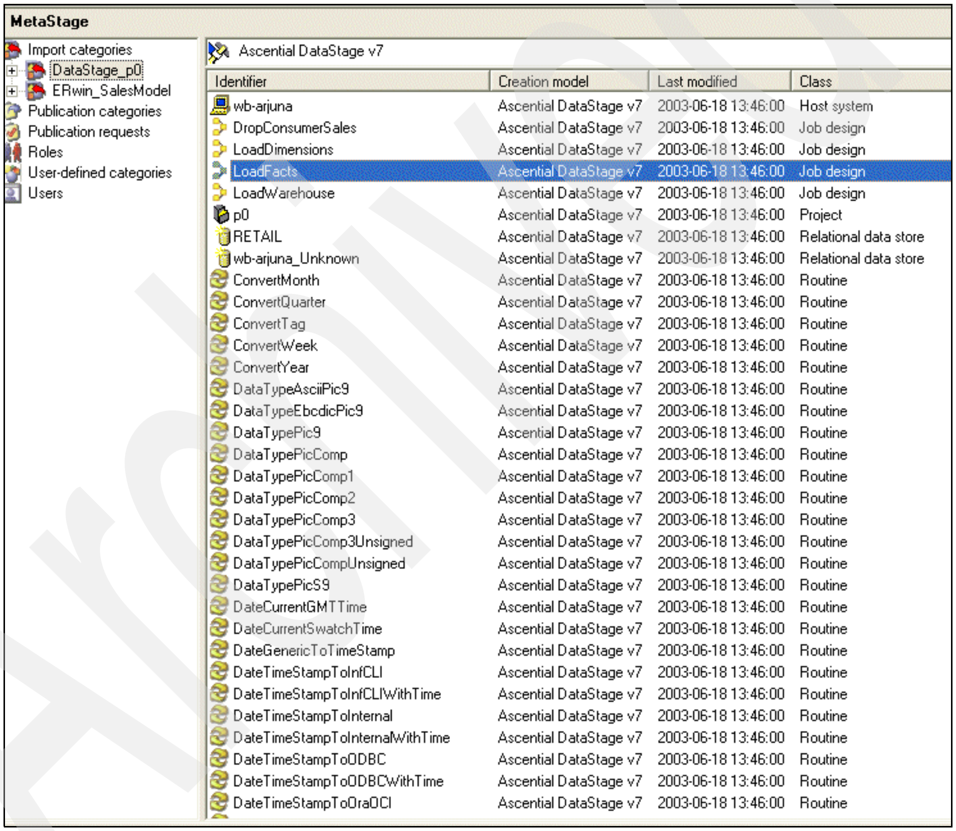


Figure 8-51 MetaStage category browser

We will right-click the *LoadFacts* job design object to expose the context menu shown in Figure 8-52.

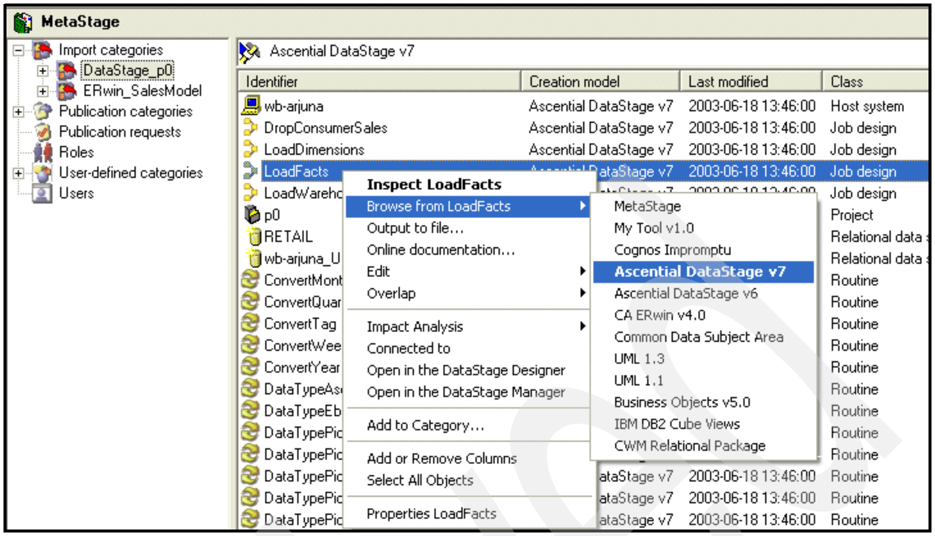


Figure 8-52 Browse from menu

We will choose the **Browse from LoadFacts -> Ascential DataStage v7** menu. This will give us a tree control from which we can browse the *LoadFacts* job design object in more detail. Now we have the ability to browse relationships from the *LoadFacts* job design object. From here we expand the *Compiles into_Compiled* job relationship. Since we started examining the job design, we need to find the actual compiled instance of that job design that ran on the DataStage Server. Figure 8-53 shows that we have right-clicked the *LoadFacts* compiled job to expose the context menu.

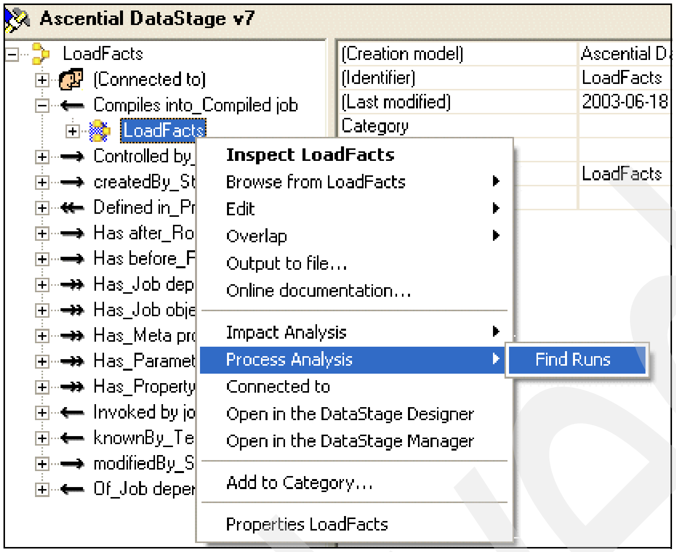


Figure 8-53 Process analysis menu

On the menu we can choose to run the **Process Analysis->Find Runs** query. Running the query results in the process analysis path shown in Figure 8-54.

Figure 8-54 Process analysis path

We can see that the compiled job ran on 2003-06-18 and that there was some of problem in its execution. We know that there was a problem because the ending event *toConsumerSalesTable* has a red icon. We can examine in more detail the reason for the problem running the job by inspecting the *toConsumerSalesTable* event object. If we double-click the event object we can see more detail about the event. Figure 8-55 shows the actual DataStage Server message.

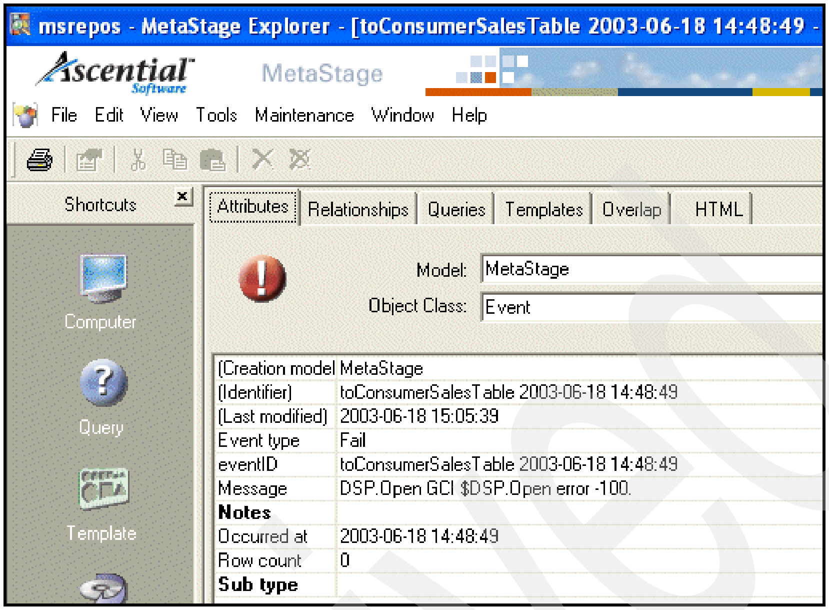


Figure 8-55 Inspect event

In our example there is a problem opening the *CONSUMER_SALES* fact table. We know that the *CONSUMER_SALES* table is the problem because Figure 8-54 shows us that the *toConsumerSalesTable* link emitted the event. The DataStage Server error code can be looked up and problem can be rectified.

8.3 Conclusion: benefits

The management of metadata (the definition of the data itself) in any data integration infrastructure is critical for maintaining consistent, clear and accurate analytic interpretations, and for lowering the maintenance costs associated with managing complex integration processes (such as a data warehouse) among multiple constituents, projects and tools. To be effective, metadata must be seamlessly shared across multiple tools — without the loss of information, fidelity or consistency. Furthermore, the analysis and management of business definitions and rules across the entire Business Intelligence, data integration and data management infrastructure should occur without custom coding, fragmentation, re-keying or manual intervention.

We have seen in the previous sections that MetaStage can provide tremendous value, not only as a simple integration path to exchange metadata with other tools and DB2 Cube Views, but as an enterprise metadata management tool.

In addition to the exchange of metadata with DB2 Cube Views, MetaStage provides tight integration with all your warehouse tools. This will provide metadata consistency and optimize design metadata sharing and reuse that will reduce costs due to time delays during development and production. In addition, MetaStage stores a persistent directory of all your metadata in a location that can be integrated into fail-safe disaster recovery systems to protect your metadata investment.

Finally, MetaStage's ability to synchronize all your enterprise metadata will enhance your business decision making by providing tightly integrated metadata that is consistent and timely.

9

Meta Integration of DB2 Cube Views within the enterprise toolset

This chapter describes certain deployment scenarios for using Meta Integration Technology products. It explains in each scenario how to implement and to use the metadata bridges.

9.1 Meta Integration Technology products overview

Meta Integration Technology, Inc. is a Silicon Valley, California based software vendor specialized in tools for the integration and management of metadata across tools from multiple vendors, and multiple purposes including data and object modeling tools, data Extraction, Transformation, and Load (ETL) tools, Business Intelligence (BI) tools, and so on. The need for data movement and data integration solutions is driven by the fact that data is everywhere underneath business applications. The same applies for metadata: metadata is also everywhere underneath the data and object modeling tools, as well as within the repositories of the ETL, Data Warehouse, Enterprise Application Integration, and Business Intelligence development tools. Meta Integration offers metadata movement solutions for the integration of popular development tools with IBM DB2 Cube Views, as illustrated in Figure 9-1.

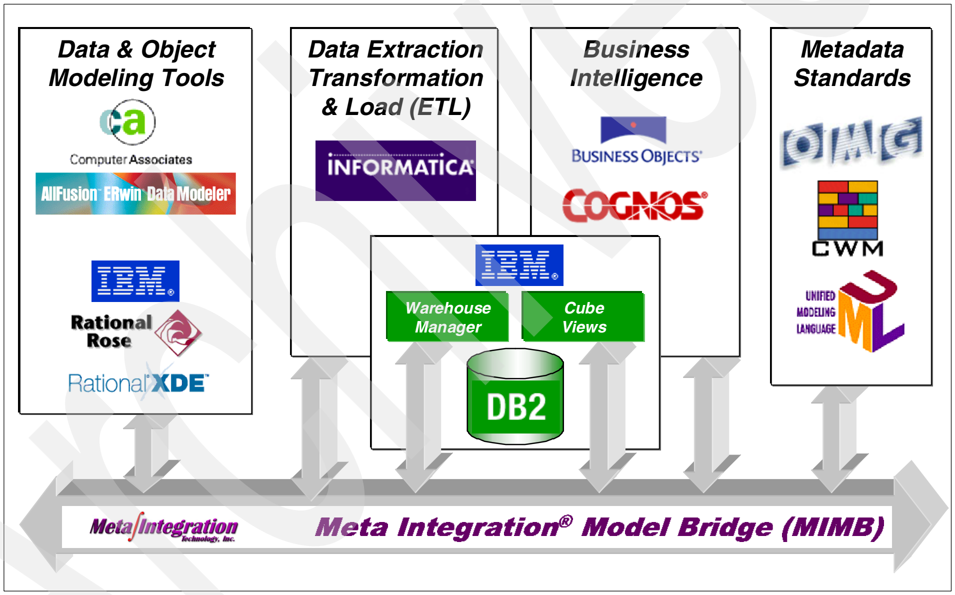


Figure 9-1 A sample of typical metadata movement solutions

9.1.1 Meta Integration Works (MIW)

MIW is a complete metadata management solution with sophisticated functionalities such as the Model Browser, the Model Bridges, the Model Comparator, the Model Integrator, and the Model Mapper all integrated around a powerful metadata version and configuration management as shown in Figure 9-2.

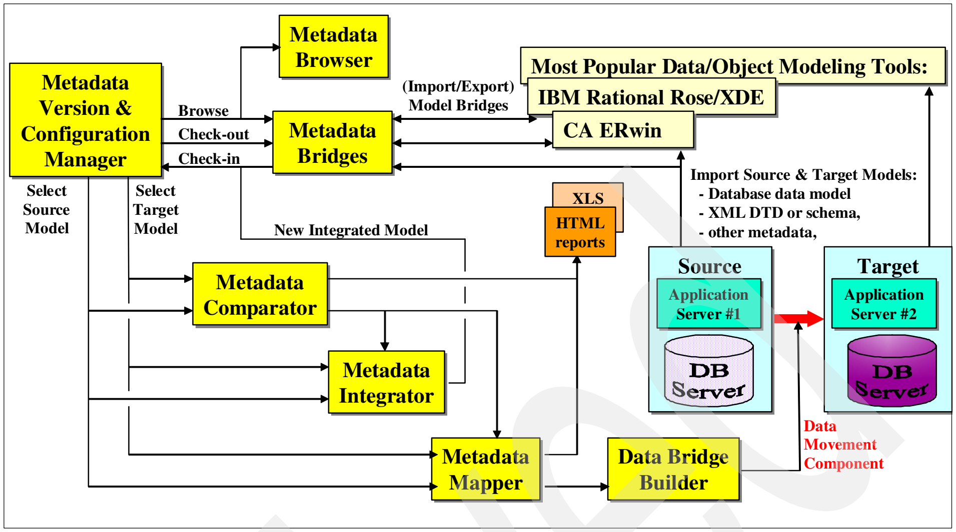


Figure 9-2 Meta Integration functionality

MIW is a powerful metadata management solution, and integrates well with today's best practices in software development, as it provides a unique component based approach to the ETL tool market. Indeed, the MIW development environment generates C++ based data movement components that can be easily integrated (plug and play) with any Windows or UNIX based business applications. Multiple data movement components can be produced for various purposes such as:

- Legacy Data Migration (LDM)
- Enterprise Application Integration (EAI)
- Data Warehousing (DW) and datamarts.

The code of the produced data movement components can be reviewed through any Quality Assurance (QA) processes, and does not depend on any middleware (free of any run-time cost at deployment time). The Model Mapper provides the mapping migrations required to support the perpetual changes in the source and destination data stores. Indeed, one of the key features of MIW is the built-in support for change management facilitating the maintenance and/or generation of new versions of the data movement components as needed. Data Connectors are available for most popular databases via ODBC (as DB2), as well as for XML data sources (as HL7 for Health Care) to service the expanding needs in the fields of EDI, e-business, and enterprise information portals.

MIW is entirely written in Java, and can be connected to a local or centralized metadata repository.

9.1.2 Meta Integration Repository (MIR)

MIR is based on a modern 3-tier architecture as shown in Figure 9-3 with support for multi-users, security, and concurrency control. The repository metamodel integrates standards like the OMG CWM and UML, and supports XMI compliant metadata interchange. MIR can manage massive amounts of metadata and make it persistent on most popular RDBMS like DB2, Oracle or SQL Server. The underlying repository database is fully open allowing users to build their own metadata Web portals, or use their existing data tools to perform metadata reporting, mining, and even intelligence.

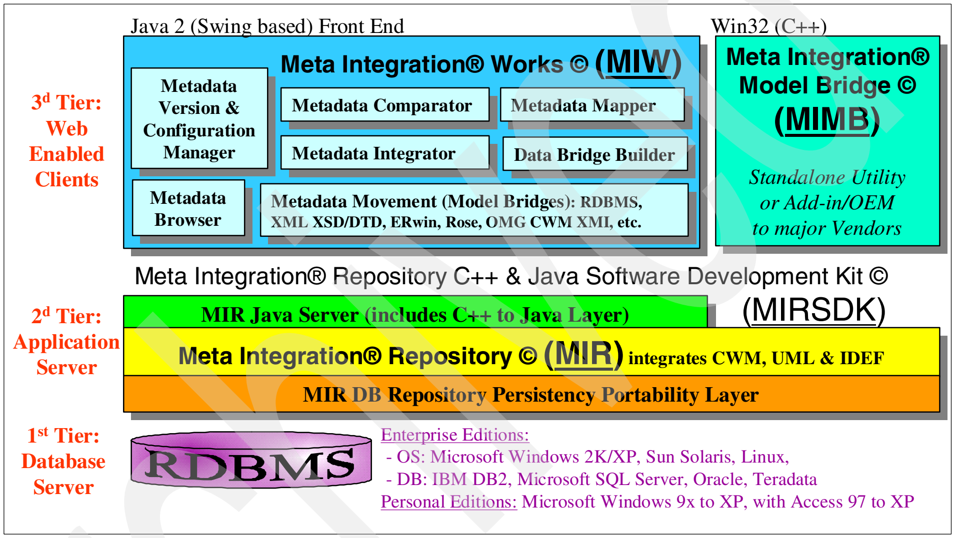


Figure 9-3 Meta Integration architecture

Open database access to the repository for:

- Web enabled end user Enterprise Metadata Portal
- Metadata Intelligence and Reporting

9.1.3 Meta Integration Model Bridge (MIMB)

MIMB is a utility for legacy model migration and metadata integration. MIMB also operates as an add-in integrated inside popular modeling, ETL, and BI tools. With over 40 bridges, MIMB is the most complete metadata movement solution on the market. MIMB supports most popular standards and the market leading tool vendors, as illustrated in Figure 9-4.

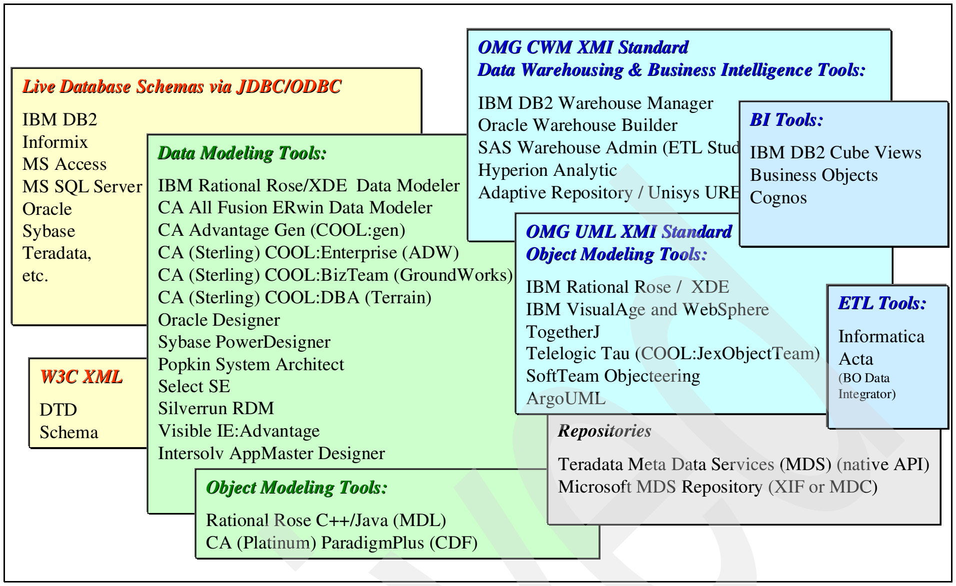


Figure 9-4 Meta Integration supported tools

More details on supported tools and versions are available at:

http://www.metaintegration.net/Products/MIMB/SupportedTools.html
http://www.metaintegration.net/Products/MIMB/AboutVendors.html
http://www.metaintegration.net/Products/MIMB/AboutStandards.html

9.2 Architecture and components involved

Meta Integration Model Bridge (MIMB) as a standalone utility (or add-in metadata movement component to popular ETL/BI tools) is based on the non-persistent version of the Meta Integration Repository (MIR) in memory. Each MIMB Import bridge creates metadata that can be reused by any MIMB export bridge. In other words, Meta Integration does not create point-to-point bridges, as illustrated in Figure 9-5.

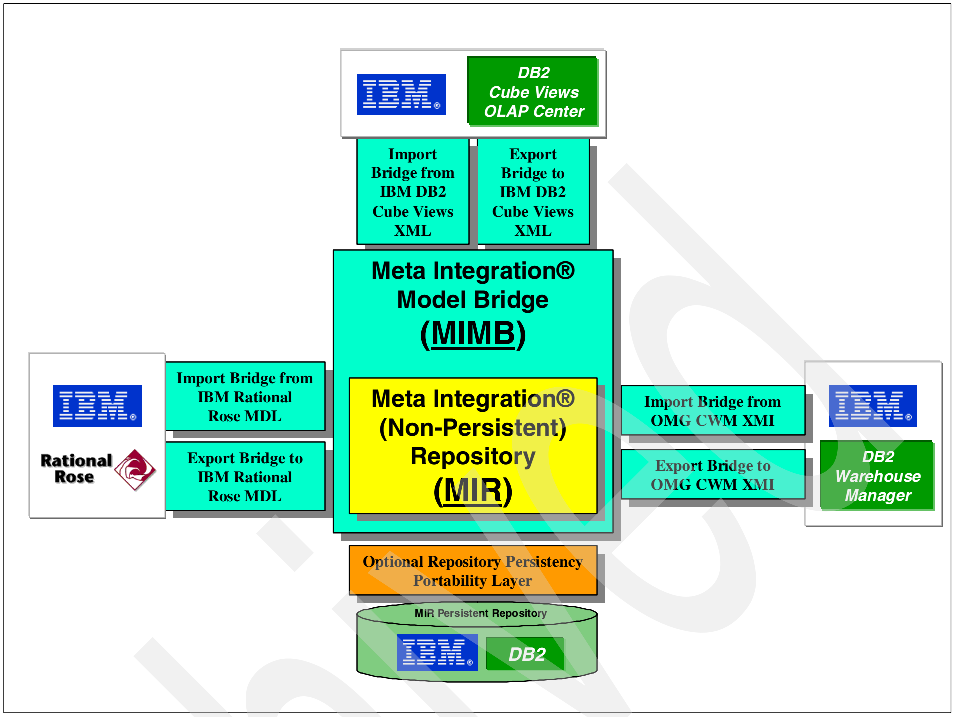


Figure 9-5 A metadata integration solution example

9.3 Metadata flow scenarios

MIMB provides bi-directional metadata movement solutions for the integration of IBM DB2 Cube Views with the development tools of the enterprise.

The exchange of metadata between various tools and DB2 Cube Views using metadata bridges is motivated by several business cases (tools integration in the enterprise, documentation...) and helps data warehouse specialists, database administrators, data modelers and application developers in the following ways:

- Forward engineering of a data model created in a design tool or an ETL tool to a DB2 Cube Views cube model. This metadata movement capability allows a data modeler to reuse metadata already designed and available in the enterprise to quickly create a cube model in DB2 Cube Views, therefore saving time when creating the OLAP metadata and leveraging the existing metadata, such as business names and descriptions that are not likely to be stored in the database.

- Reverse engineering of a DB2 Cube Views cube model into a design model. This metadata movement capability allows extracting and reusing the metadata of a cube model created in DB2 Cube Views to quickly create a model in a data modeling tool, an object modeling tool, or an ETL tool in order to document the model, develop a software application or other purposes.

The generic metadata flows can be summarized as in Figure 9-6.

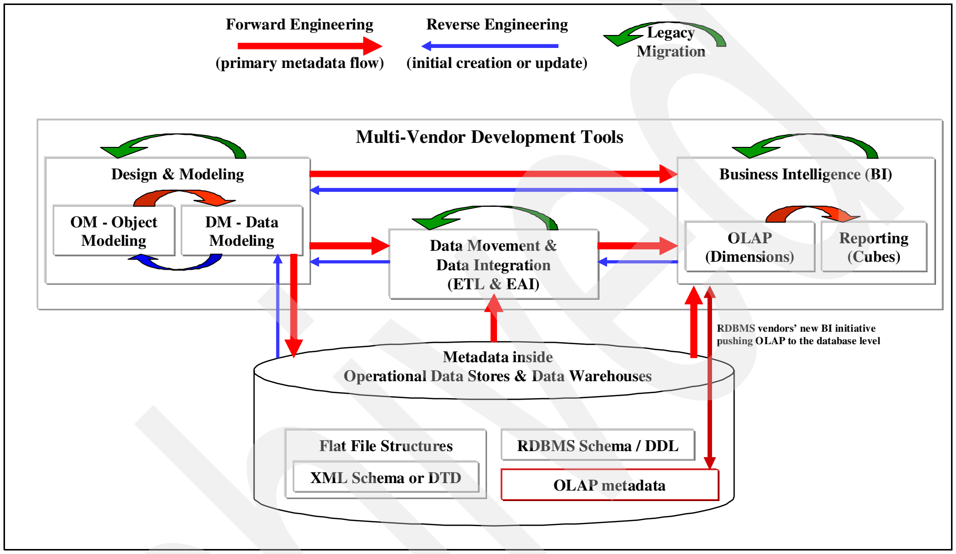


Figure 9-6 Business cases for metadata movement solutions

The tools vendors themselves can provide some of these metadata movements, for example, IBM Rational® Rose® provides bi-directional integration between UML object modeling and physical data modeling. Similarly, BI vendors provide the forward engineering from their OLAP dimension design tool to their OLAP based reporting tool. However, large corporations use best-of-breed tools from many vendors. In such case, MIMB can play a key role implementing all the metadata movement required for the integration of their development tools, as illustrated in Figure 9-7.

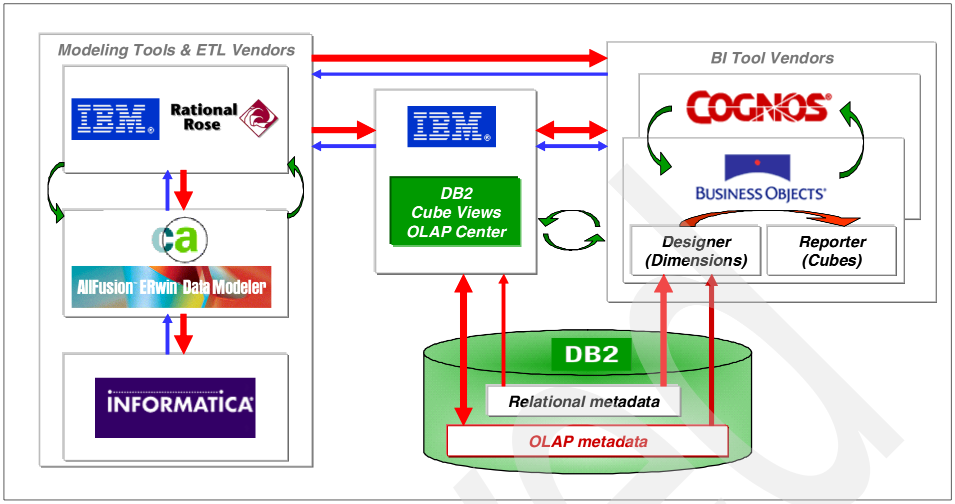


Figure 9-7 Possible metadata movement solutions for DB2 Cube Views

We will demonstrate the implementation of the 7 metadata movement scenarios shown in Figure 9-8, based on popular modeling tools, ETL, and the OMG CWM metadata standard.

As each tool has its own tricks and each MIMB bridge has its own set of import/export options, each scenario has been written as an independent piece and can be read separately based on your interests.

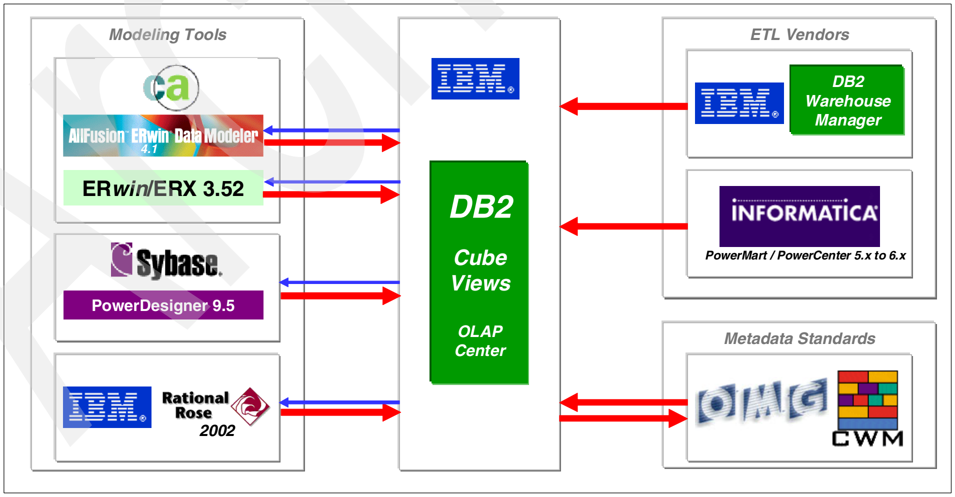


Figure 9-8 Metadata movement scenarios illustrated in this chapter

9.4 Metadata mapping and limitations considerations

These are the four primary scenarios:

1. Forward engineering from popular data modeling tools like IBM Rational Rose or IBM Rational XDE™, and Computer Associates ERwin Data Modeler
2. DB2 Cube Views Integration with ETL tools like Informatica and DB2 Warehouse Manager
3. DB2 Cube Views Integration with BI vendors like BO and Cognos
4. DB2 Cube Views support for metadata standards like OMG CWM XMI.

The current MIMB v3.1 provides IBM DB2 Cube Views import and export bridges for IBM DB2 OLAP Center, and is available for download at:

http://www.metaintegration.net/Products/Downloads/

This version 3.1 provides very complete support for the foregoing user cases (1) and (2) of forward engineering:

- An ERwin star schema sample model is provided with instructions to generate the DB2 Cube Views dimensions, facts, and cube model.
- However, MIMB v3.1 provides currently incomplete support for the foregoing user cases (3) and (4), due to current BI/OLAP limitations in the Meta Integration Repository (MIR) metamodel of v3.x.

> **Note:** To get the most up-to-date information on new versions and releases, concerning metamodel extensions and support for change management and impact analysis between all the integrated data modeling, ETL, and BI tools, check the following site regularly:
>
> http://www.metaintegration.net/Products/MIMB

9.4.1 Forward engineering from a relational model to a cube model

In a forward engineering scenario:

- The relational tables are used to specify where the tables are located in DB2
- The fact tables are used to create the cube model facts object
- The measure columns of the fact tables are transformed into measure objects
- The dimension and outrigger tables are transformed into dimension objects
- The dimension and outrigger columns are transformed into dimension attributes
- The foreign key relationships are used to build joins

- The business name, description and data type of relational objects are also converted

The produced cube model can then be edited DB2 OLAP Center to enrich it with additional OLAP metadata such as hierarchies, levels, cubes, calculated measures and more.

9.4.2 Reverse engineering of a cube model into a relational model

In a reverse engineering scenario:

- The relational tables referenced by the cube model are converted to the destination tool.
- The joins are analyzed in details to create relationships when possible.
- The OLAP dimensions, facts, attributes and measures business name, description are also converted to the destination tool.

The generated model can be edited in the destination tool to further document it, and add information that was not contained in the source cube model XML file. This missing information can be physical information (such as indexes or tablespaces) that can be retrieved automatically from the database using the destination tool's database synchronization features, or it can be logical information, such as generalizations (super type sub type entities) or UML methods.

For more mapping information, please read the MIMB software documentation, which includes the complete mapping specification of each bridge. This documentation can be consulted online at:

http://www.metaintegration.net/Products/MIMB/

9.5 Implementation steps scenario by scenario

This section describes the implementation steps for both forward and reverse engineering for the following metadata exchanges:

- Metadata integration of DB2 Cube Views with Computer Associates AllFusion ERwin Data Modeler versions 4.0 to 4.1
- Metadata integration of DB2 Cube Views with Computer Associates ERWin versions 3.0 to 3.5.2
- Metadata integration of DB2 Cube Views with Sybase PowerDesigner versions 7.5 to 9.5

- Metadata integration of DB2 Cube Views with IBM Rational Rose versions 2000e to 2002
- Metadata integration of DB2 Cube Views with the OMG CWM and UML XMI standards
- Metadata integration of DB2 Cube Views with DB2 Warehouse Manager via the OMG CWM XMI standard
- Metadata integration of DB2 Cube Views with Informatica PowerMart

The DB2 Cube Views cube model we used is shown in Figure 9-9.

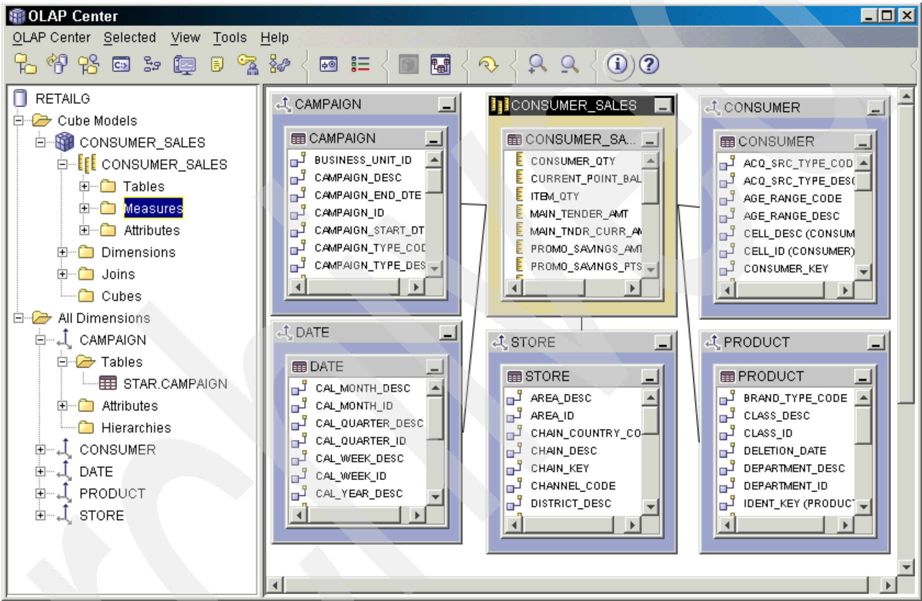


Figure 9-9 The cube model used

9.5.1 Metadata integration of DB2 Cube Views with ERwin v4.x

Computer Associates AllFusion ERwin v4.x Data Modeler is one of the leading database design tools. It supports DB2 as a target database system and allows designing star-schemas databases via its dimensional modeling notation.

Forward engineering from ERwin v4.x to DB2 Cube Views

The goal of this scenario is to demonstrate how an existing ERwin v4.x model can be converted to a DB2 cube model.

The overall process of this metadata conversion is as follows:

1. Using ERwin v4, create the star schema model
2. Using ERwin v4, generate the SQL DDL for this database
3. Using DB2, run this SQL script to create the tables and columns of this schema
4. Using ERwin v4, save the model as XML
5. Using MIMB, convert this ERwin v4 XML file into a DB2 Cube Views XML file
6. Using DB2 Cube Views, import this DB2 Cube Views XML file

Each step of this process is described in the following paragraphs.

1) Using ERwin v4.x, create the star schema model

The ERwin v4 model used in this scenario is shown in Figure 9-10. This database model is in the form of a star-schema and the bridges will use the dimensional information specified in the ERwin v4 model to create a cube model.

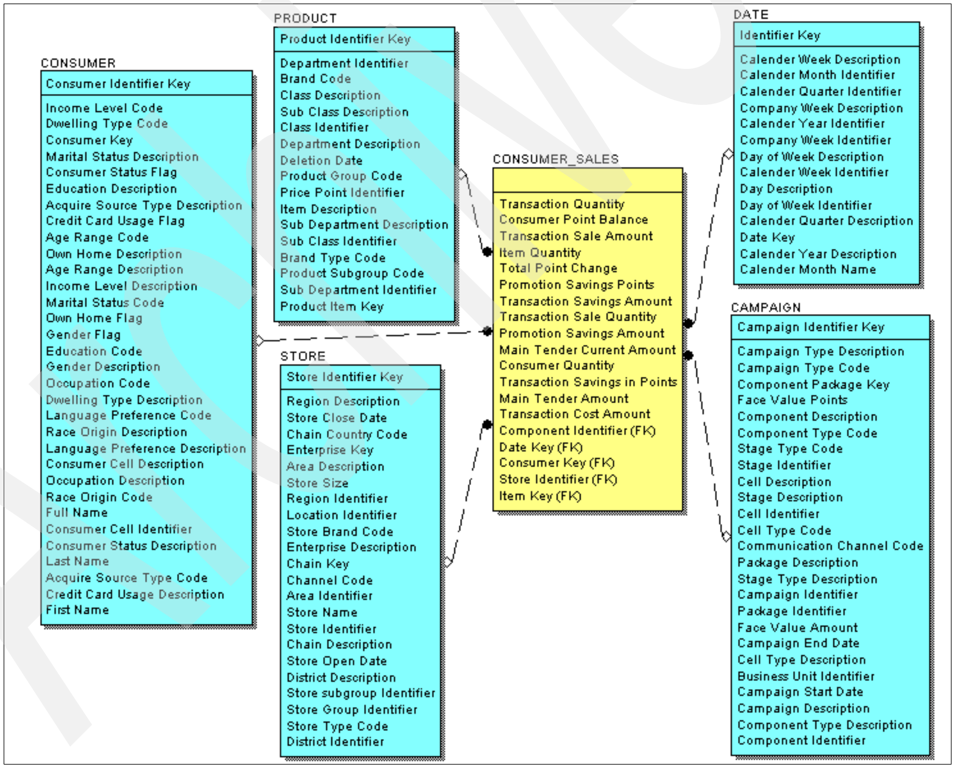


Figure 9-10 Logical view of the ERwinv4 model

During the implementation of this data model in ERwin v4, the dimensional modeling features were enabled, as shown in Figure 9-11. These features can be activated in the menu **Model -> Model Properties** and in the tab **General**.

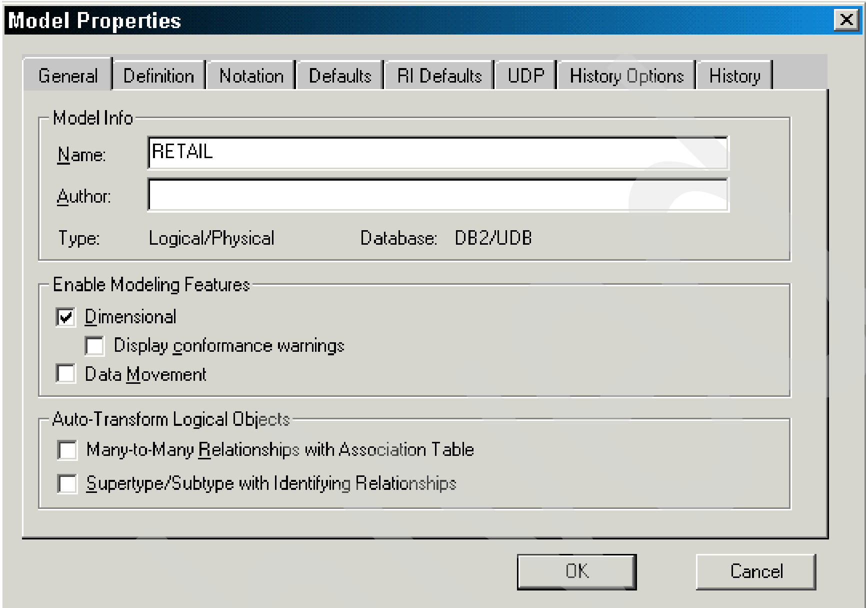


Figure 9-11 Enabling the ERwin v4 dimensional features

This option enables an additional **Dimensional** panel in the **Table** properties window shown in Figure 9-12, so that we can specify the role of each table (**Fact**, **Dimension** or **Outrigger**).

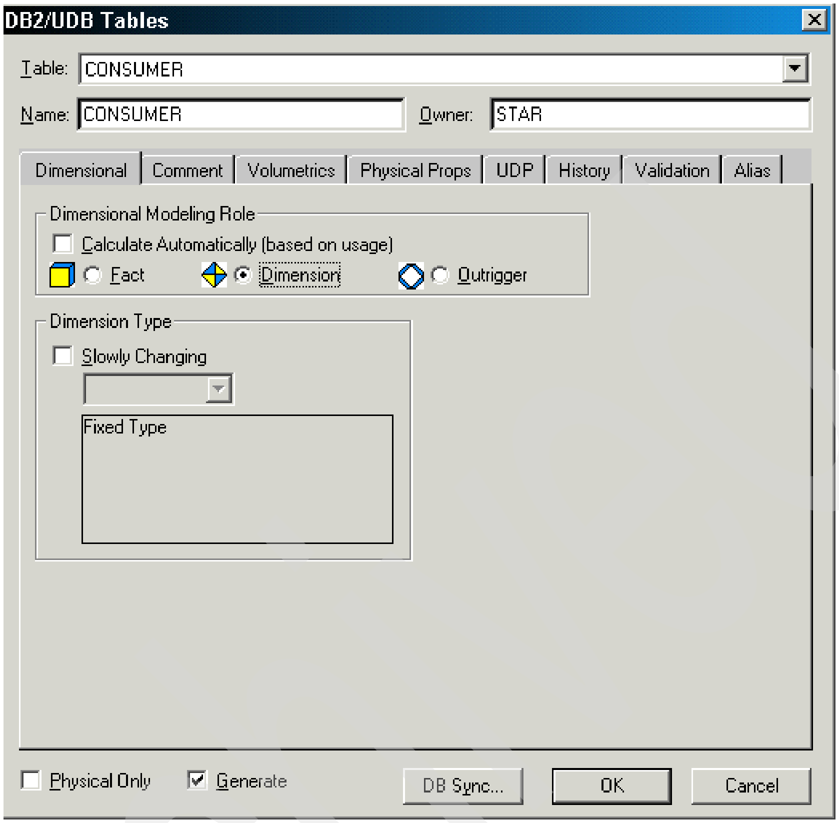


Figure 9-12 Specifying the table dimensional roles

> **Note:** The role of each table should be set explicitly, so that it is saved in the ERwin v4.x XML file format and the bridges can used it. Otherwise, if ERwin v4.x computes the dimensional role of the table automatically, it will not be saved in the ERwin v4.x XML file.

2) Using ERwin v4.x, generate the SQL DDL for this database

Once the model has been designed, the SQL DDL can be generated and the database created in DB2 UDB. In the ERwin v4.x model Physical View, choose the menu **Tools -> Forward Engineer/Schema Generation** to generate the SQL script as shown in Figure 9-13.

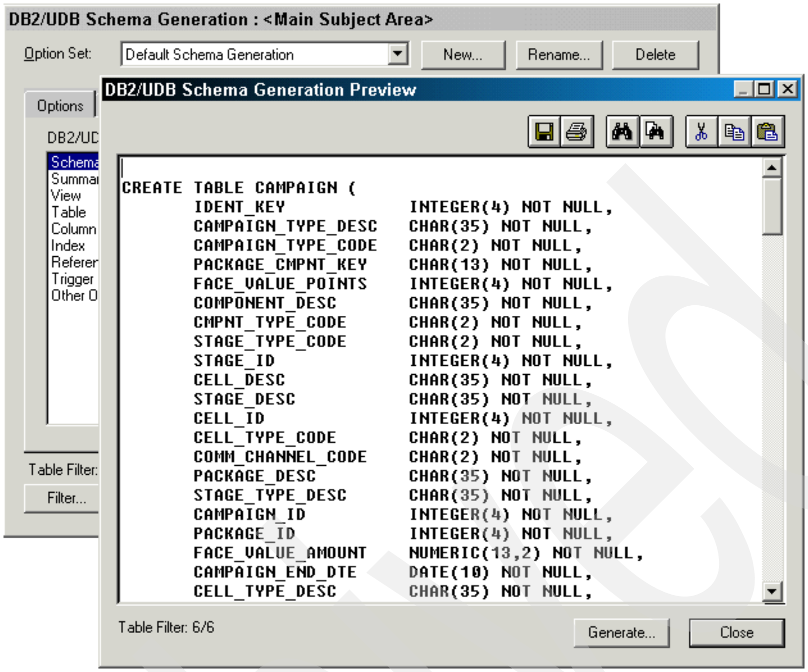


Figure 9-13 DB2 schema generation

3) Using DB2, create the tables and columns of this schema

The SQL script in Figure 9-13 can be executed to create the DB2 Tables. Here is how to execute it using the DB2 Command Window tool:

```
db2 connect to MDSAMPLE
db2 set current schema = STAR
db2 –tvf C:\Temp\star.sql
```

> **Note:** The database schema must be created in DB2 before the cube model is imported into DB2 Cube Views.

At this point, the database has been setup and is ready to receive the cube model metadata.

4) Using ERwin v4.x, save the model as XML

The next step of this process is to save the ERwin v4 model as an XML file. The bridge will use this file as input.

When the model is loaded in ERwin v4.x, choose **Save As** from the **File** menu, select the XML format type in the **Save as type list**, type the file name for the model you are saving in the **File name text** box, and click **Save**.

Note: If the ERwin v4 model is logical and physical (business names have been defined in the logical view), the **Save as XML** process described above will not properly save the physical names into the XML file, if ERwin v4.x automatically computed these physical names.

To work around this issue, you can use an alternate **Save as XML** feature of ERwin v4 located in menu **Tools -> Add-Ins' -> Advantage Repository Export**. It produces a slightly different XML file format where the physical names are expanded.

This issue does not occur if the ERwin v4.x model is physical only.

5) Using MIMB, convert ERwin XML into DB2 Cube Views XML file

Start the MIMB tool and select the import bridge labeled **CA ERwin 4.0 SP1 to 4.1**, and import your ERwin v4.x XML file, as shown in Figure 9-14.

The MIMB *validation* feature checks that the model is valid according to the rules of the MIR metamodel. If something is wrong (a key is empty or a column does not belong to any table or a foreign key does not reference a primary key), it will display a warning or error message.

The *subsetting* feature allows you to create a subset of the model so that the model exported to the destination tool only contains the few tables you chose.

Both features are described in the online documentation:

http://www.metaintegration.net/Products/MIMB/Documentation/

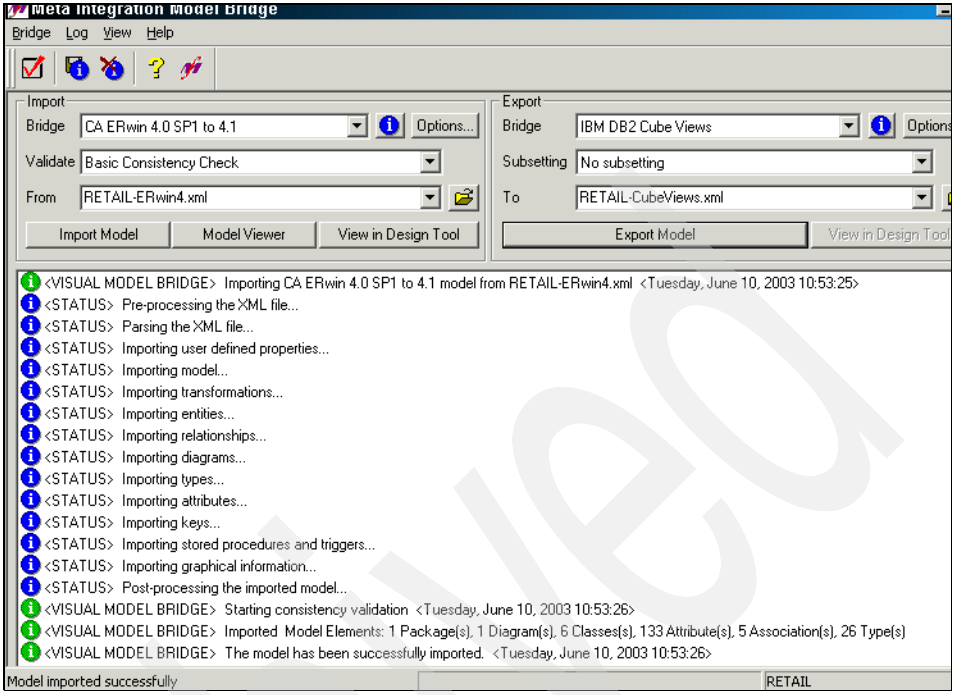


Figure 9-14 Importing the ERwin v4 model into MIMB

Select the export bridge labeled **IBM DB2 Cube Views** and click the **Options** button to specify the export parameters as shown in Figure 9-15.

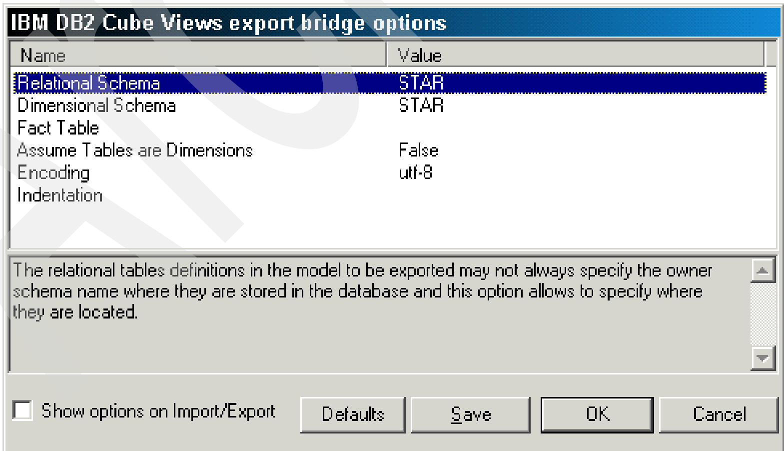


Figure 9-15 Specifying the export bridge parameters

The export parameters used in this scenario are as follows:

- The DB2 schema for the tables of the model is *STAR*, as the model may not always specify where each table is located.
- The cube model to be created will be located in the same *STAR* DB2 schema.
- We specify that the source encoding of the ERwin v4 model is utf-8.
- The other options are left with their default value.

Close this window, specify the name of the DB2 Cube Views XML file to be created, and click the **Export** button.

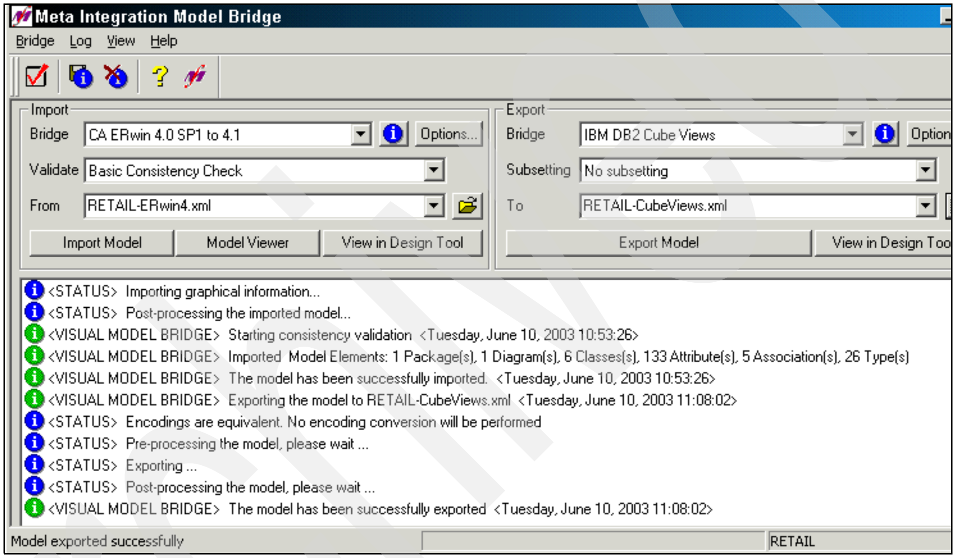


Figure 9-16 Exporting the model to DB2 Cube Views

6) Using DB2 Cube Views, import this DB2 Cube Views XML file

At this point, the cube model XML file has been created and is ready to be opened into the DB2 OLAP Center graphical tool. Just start OLAP Center, connect to your database, and choose **Import** in the OLAP Center menu as shown in Figure 9-17.

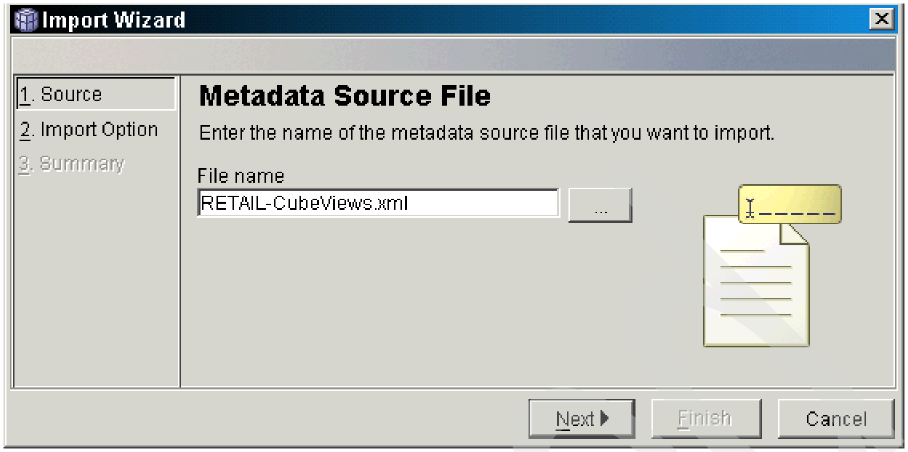


Figure 9-17 Specifying the XML file to import into OLAP Center

The content of the XML file is displayed in Figure 9-18, which allows controlling how this metadata should be imported, in case there is already some metadata in place and object name collision should occur:

- Either update the existing objects with the new imported version.
- Or keep the current version of the metadata.

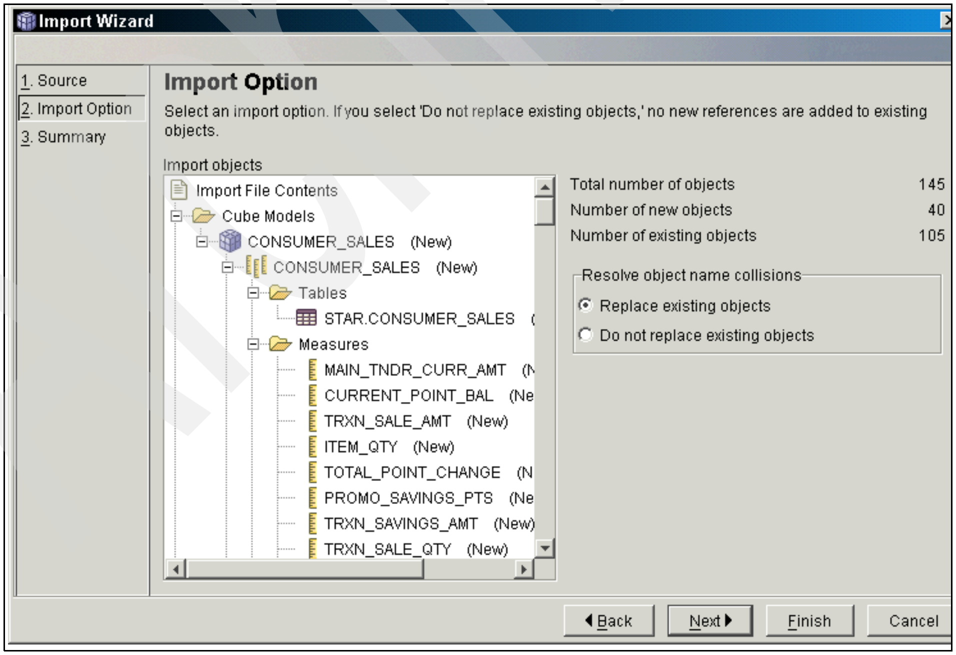


Figure 9-18 Controlling how the metadata is imported into OLAP Center

Finally, the ERwin star schema metadata converted and imported into OLAP Center will provide the DB2 cube model in Figure 9-9 on page 339.

The business names and descriptions defined in ERwin v4.x are also converted to the cube model, as shown in Figure 9-19.

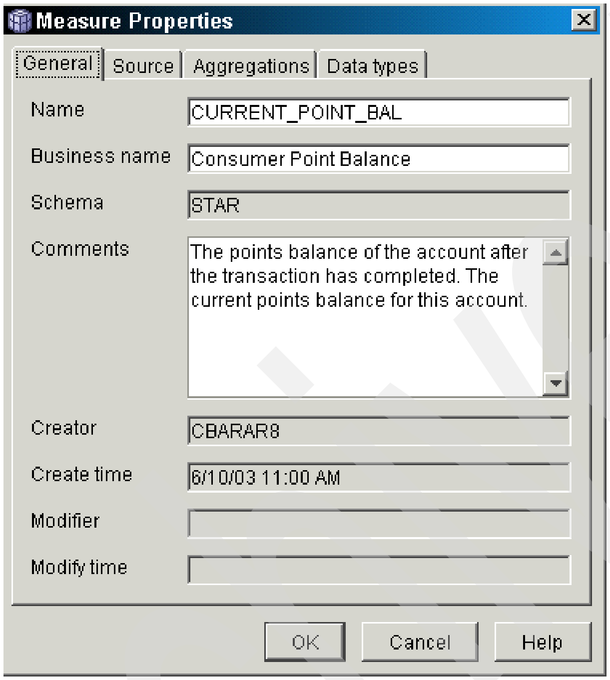


Figure 9-19 The ERwin v4 business names and description are also converted

Congratulations, the ERwin v4.x star schema model was converted to DB2 Cube Views!

Reverse engineering from DB2 Cube Views to ERwin v4.x

The goal of this scenario is to demonstrate how an existing DB2 Cube Views model can be converted into an ERwin v4.x Model.

The overall process of this metadata conversion is as follows:

1. Using DB2 Cube Views, export your cube model as an XML file.
2. Using MIMB, convert this DB2 Cube Views XML file into an ERwin v4.x XML file.
3. Using ERwin v4.x, import this XML file.

Each step of this process is described in the following paragraphs.

1) Using DB2 Cube Views, export your cube model as an XML file

The DB2 Cube Views model used in this scenario is the one shown in Figure 9-9 on page 339.

The first step of the conversion process is to save this cube model into an XML file. Use the OLAP Center menu **OLAP Center -> Export** and select the cube model to be exported as shown in Figure 9-20.

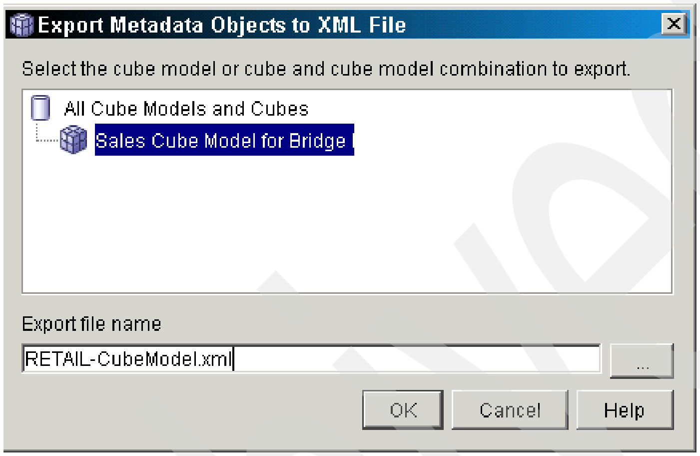


Figure 9-20 Exporting from the DB2 cube model as XML

2) Using MIMB, convert DB2 Cube Views XML into ERwin XML file

Start the MIMB software, select the import bridge labeled **IBM DB2 Cube Views** and import your model. Select the export bridge labeled **CA ERwin 4.0 SP1 to 4.1**, select the name of the export ERwin v4 XML file, and click the **Export Model** button as shown in Figure 9-21.

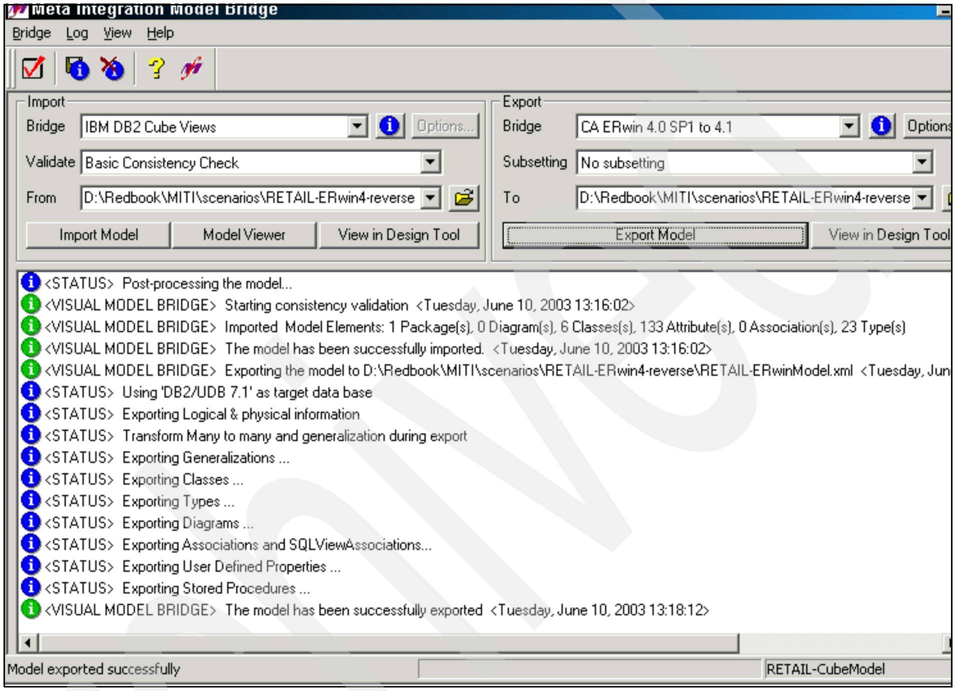


Figure 9-21   Converting the cube model XML file to an ERwin v4 XML file

3) Using ERwin v4.x, import this XML file

At this point, you can open the generated XML file into ERwin v4 using menu **File -> Open**. When the file choice window appears, select XML Files (*.xml) in the 'Files of type list box and select the XML file produced by MIMB.

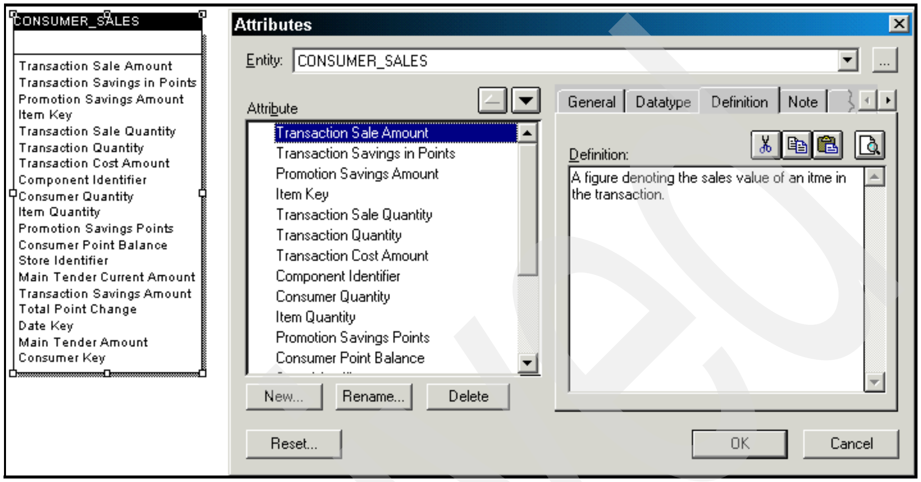


Figure 9-22 Cube model converted to ERwin v4 with business names

The cube model converted to ERwin v4.x contains the business names and descriptions, and the logical view is shown in Figure 9-23.

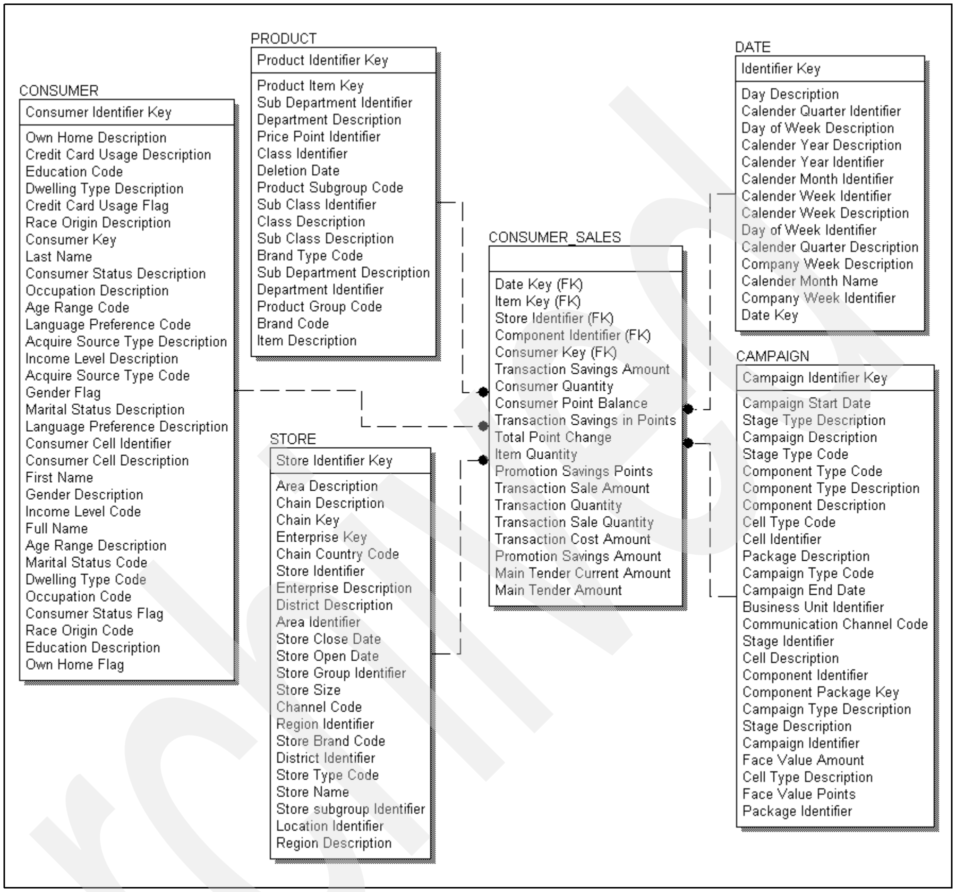


Figure 9-23 Logical view of the ERWin model

Congratulations, the cube model was converted to ERwin v4.x!

9.5.2 Metadata integration of DB2 Cube Views with ERwin v3.x

Computer Associates ERwin 3.x is still one of the leading database design tools. It supports DB2 UDB as target database system and allows designing star-schemas databases via its dimensional modeling notation.

Note: The ERwin v3.x ERX file format is very widely used as a de facto standard means of exchanging relational database metadata. Many design tools support it and therefore this scenario can also be used to interact and exchange metadata with them.

A non-exhaustive list of such tools would include Embarcadero ER/Studio, Microsoft Visio, Sybase PowerDesigner, Computer Associates Advantage Repository, Casewise, and Informatica PowerCenter.

Forward engineering from ERwin v3.x to DB2 Cube Views

The goal of this scenario is to demonstrate how an existing ERwin model can be converted to a DB2 cube model.

The overall process of this metadata conversion is as follows:

1. Using ERwin, create the star schema model.
2. Using ERwin, generate the SQL DDL for this database.
3. Using DB2, run this SQL script to create the tables and columns of this schema.
4. Using ERwin, save the model as ERX.
5. Using MIMB, convert this ERwin ERX file into a DB2 Cube Views XML file.
6. Using DB2 Cube Views, import this DB2 Cube Views XML file.

Each step of this process is described in the following paragraphs.

1) Using ERwin, create the star schema model

The ERwin model used in this scenario is the one shown in Figure 9-24. This database model is in the form of a star-schema and the bridges will use the dimensional information specified in the model to create a cube model.

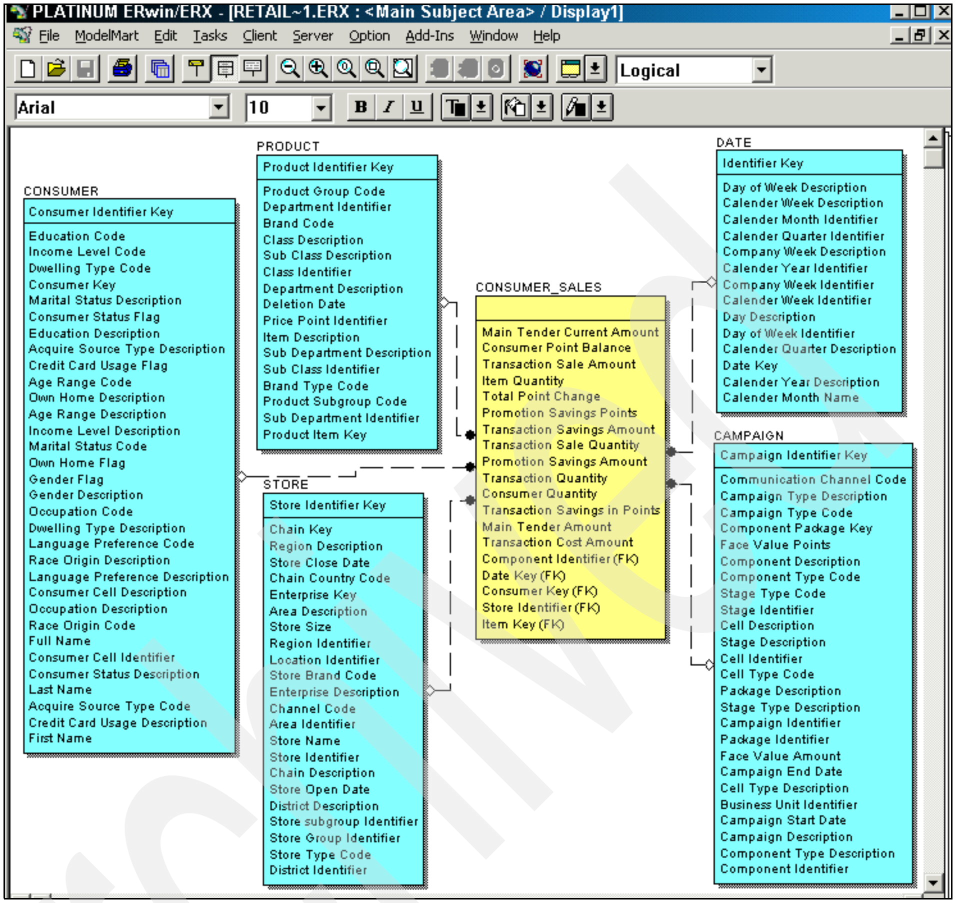


Figure 9-24 Logical view of the ERwin model

During the implementation of this data model in ERwin, the dimensional modeling features were enabled, as shown in Figure 9-25. These features can be activated in the menu **Options -> Preferences** and in the tab **Methodology**.

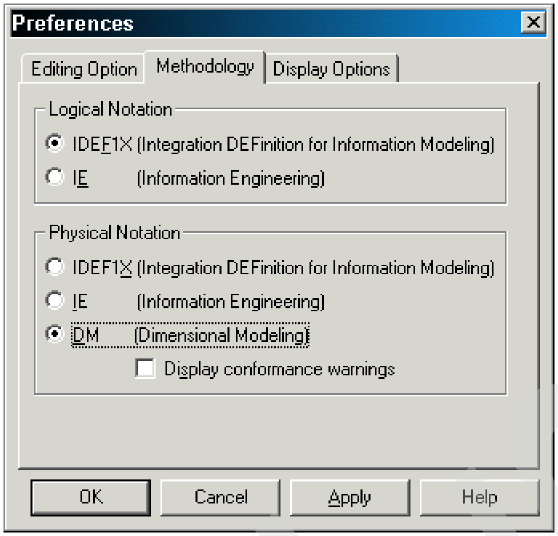


Figure 9-25 Enabling the ERwin dimensional features

This option enables an additional **Dimensional** panel in the tables properties, so that we can specify the role of each table (for example **Fact**, **Dimension** or **Outrigger**), as shown in Figure 9-26.

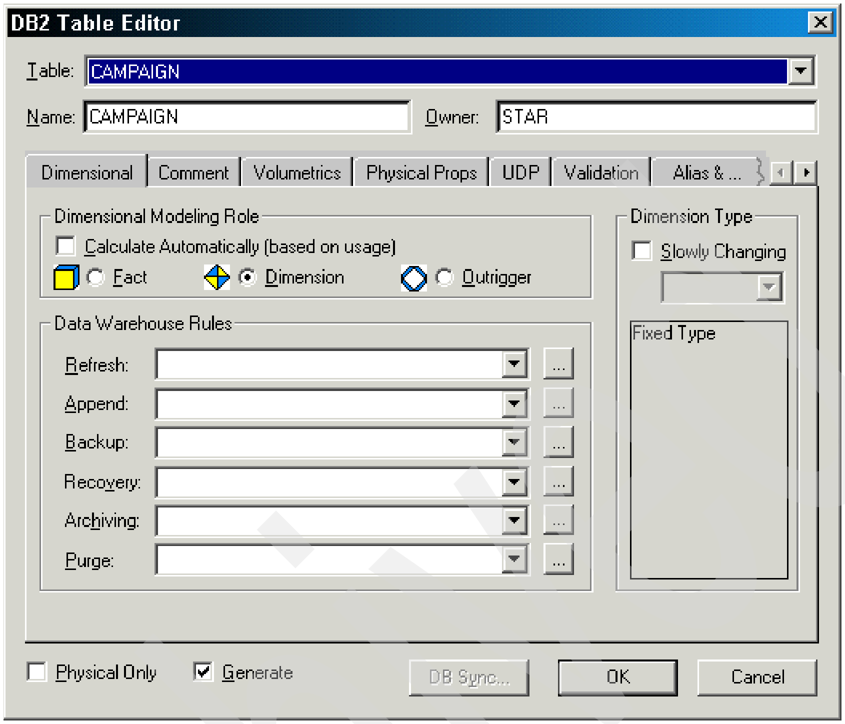


Figure 9-26 Specifying the tables dimensional roles

Note: The role of each table should be set explicitly, so that it is saved in the ERwin ERX file format and the bridges can used it. Otherwise, if ERwin computes the dimensional role of the table automatically, it is not saved in the ERwin ERX file

2) Using ERwin, generate the SQL DDL for this database

Once the model has been designed, the SQL DDL can be generated and the database created in DB2 UDB. In the ERwin model physical view, choose the menu **Tasks -> Forward Engineer/Schema Generation** to generate the SQL script as in Figure 9-13 on page 343.

3) Using DB2, create the tables and columns of this schema

The SQL script can be executed to create the DB2 Tables through the DB2 Command Window tool (or any other tool).

4) Using ERwin, save the model as ERX

The next step of this process is to save the ERwin model as an ERX file. The bridge will use this file as input.

When the model is loaded in ERwin, choose **Save As** from the File menu, select the ERX format type in the File format area, type the file name for the model you are saving in the **File name** text box and click **OK** as shown in Figure 9-27.

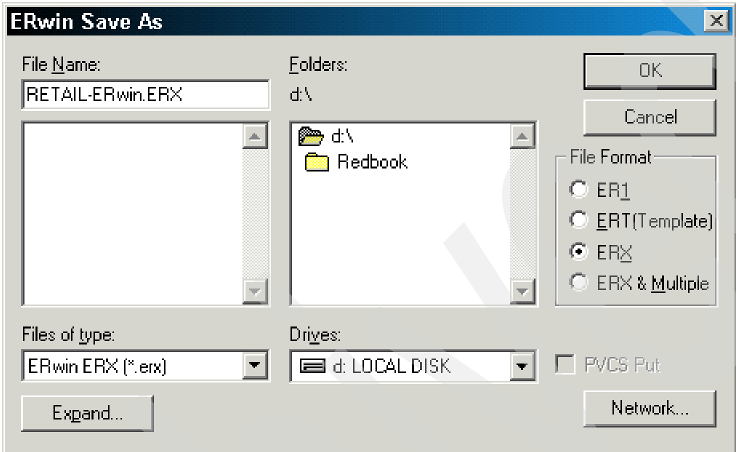


Figure 9-27 Saving the model as ERX

Note: When saving a logical and physical model, the physical names of tables, columns, and keys may not always be saved into the ERX file. Indeed, when ERwin is used to manage the automatic generation of physical names from logical names, only the generation rules are saved.

One solution is to make sure all physical names are explicitly set, therefore not relying on any generation rules from the logical names.

Alternatively, when saving a model as ERX, the dialog box offers a button called **Expand**, which opens another dialog box labeled **Expand Property Values**. Select the **DB2** tab of this window, and check the appropriate names to expand (column name) as shown in Figure 9-28.

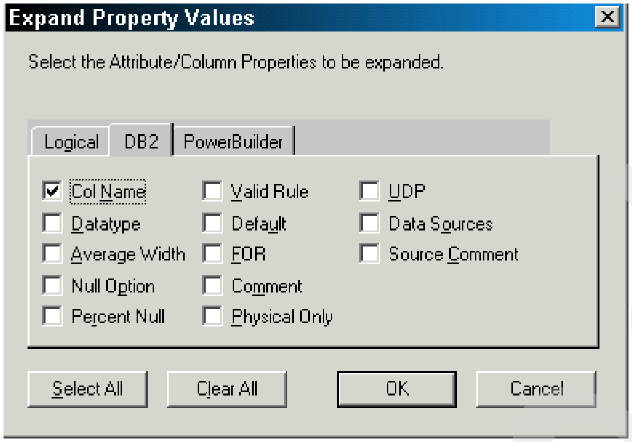


Figure 9-28 ERwin names expansion feature

5) Using MIMB, convert ERwin ERX file into DB2 Cube Views XML

Start the MIMB tool and select the import bridge labeled “CA ERwin 3.0 to 3.5.2”, and import your ERX file, as shown in Figure 9-29.

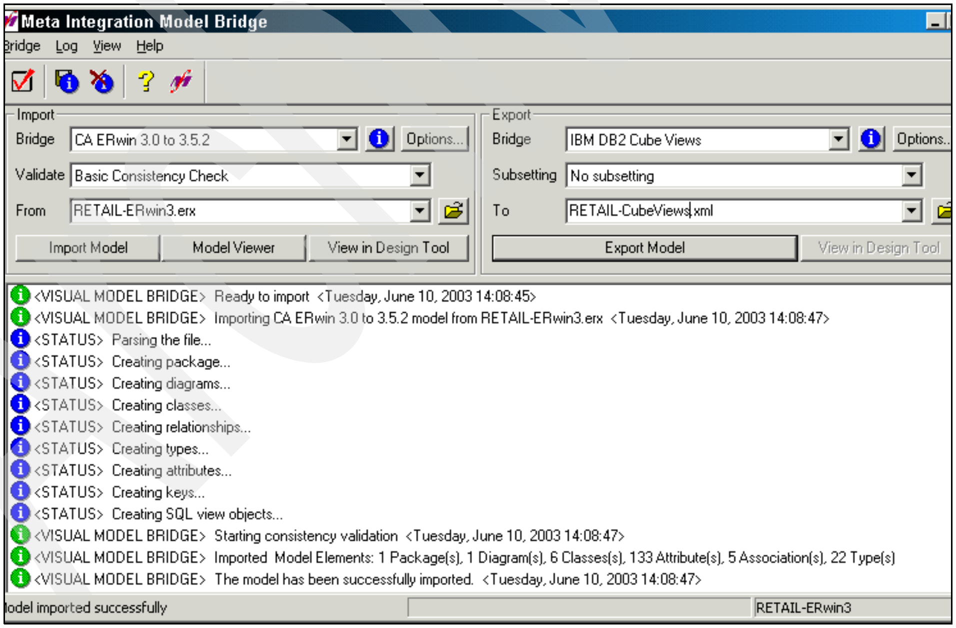


Figure 9-29 Importing the ERwin model into MIMB

Select the export bridge labeled **IBM DB2 Cube Views** and click the **Options** button to specify the export parameters as in Figure 9-30.

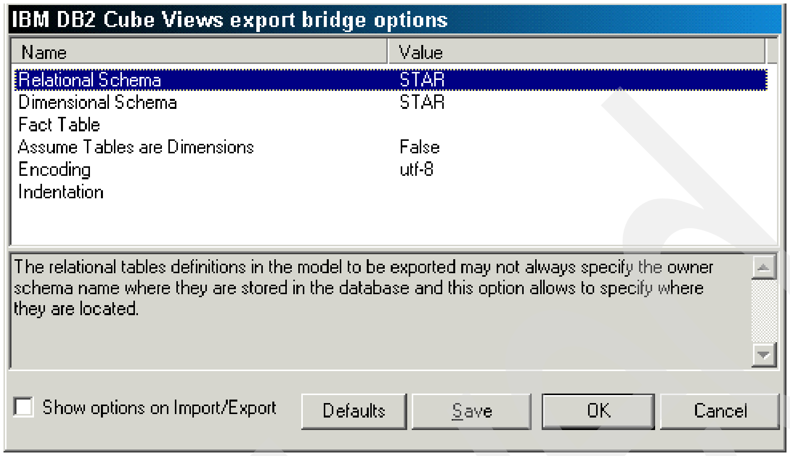


Figure 9-30 Specifying the export bridge parameters

The export parameters used in this scenario are as follows:

- The DB2 schema for the tables of the model is *STAR*, as the model may not always specify where each table is located.
- The cube model to be created will be located in the same *STAR* DB2 schema.
- The other options are left with their default value.

Close this window, specify the name of the DB2 Cube Views XML file to be created, and click the **Export** button (see Figure 9-31).

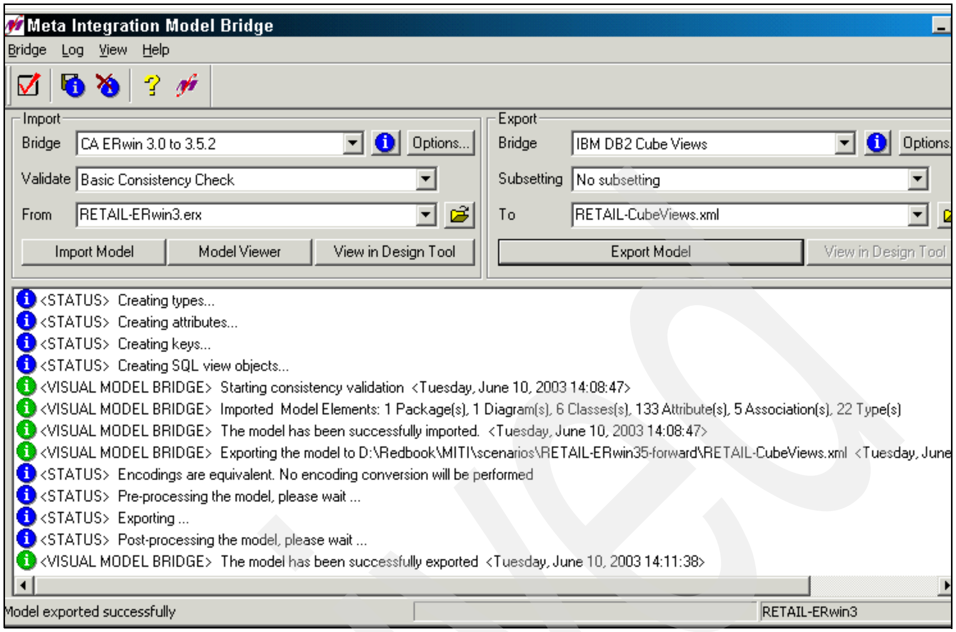


Figure 9-31 Exporting the model to DB2 Cube Views

6) Using DB2 Cube Views, import this DB2 Cube Views XML file

At this point, the cube model XML file has been created and is ready to be opened into the OLAP Center graphical tool. Just start OLAP Center, connect to your database and choose **Import** in the OLAP Center menu to get the display in Figure 9-32.

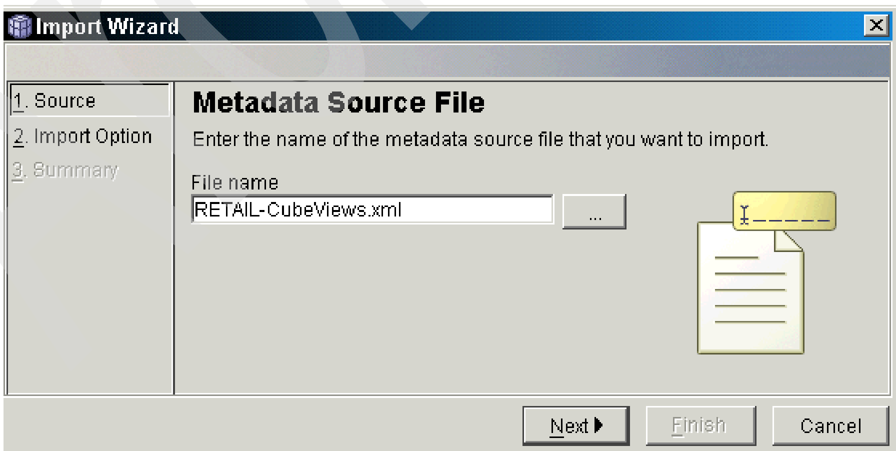


Figure 9-32 Specifying the XML file to import into OLAP Center

The content of the XML file is displayed in Figure 9-33, which allows controlling how this metadata should be imported, in case there is already some metadata in place and object name collision should occur:

- Either update the existing objects with the new imported version.
- Or keep the current version of the metadata.

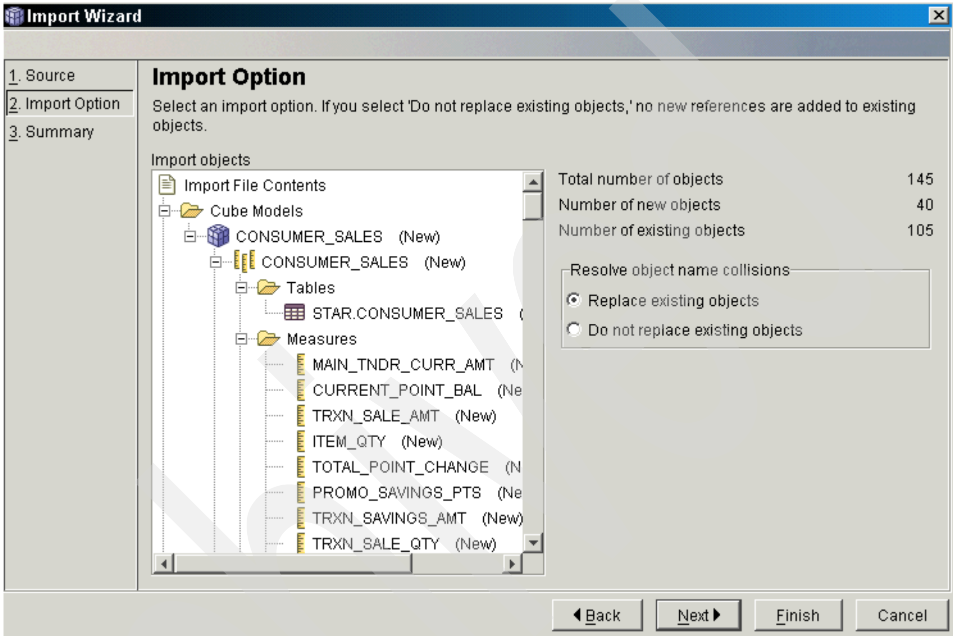


Figure 9-33 Controlling how the metadata is imported into OLAP CEnter

Finally, the ERwin star schema metadata converted and imported into OLAP Center will provide the DB2 cube model in Figure 2-9 on page 15.

The business names and descriptions defined in ERwin are also converted to the cube model, as shown in Figure 9-34.

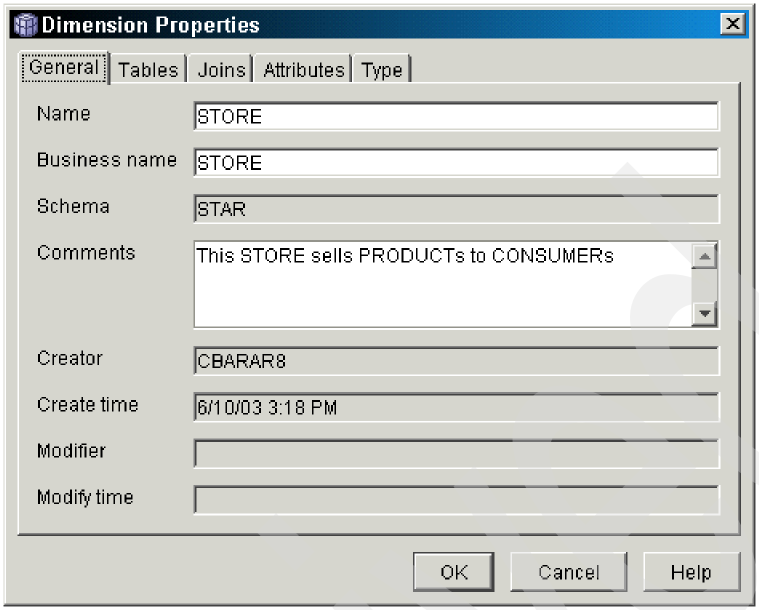


Figure 9-34 The ERwin business names and descriptions are also converted

The ERwin objects' business name and description are also converted.

Congratulations, the ERwin star schema model was converted to DB2 Cube Views!

You can now edit this model in DB2 OLAP Center to enrich it with additional OLAP metadata such as hierarchies, levels, cubes, calculated measures, and more.

Reverse engineering from DB2 Cube Views to ERwin v3.x

The goal of this scenario is to demonstrate how an existing DB2 Cube Views model can be converted into an ERwin model.

The overall process of this metadata conversion is as follows:

1. Using DB2 Cube Views, export your cube model as an XML file.
2. Using MIMB, convert this DB2 Cube Views XML file into an ERwin 3.x ERX file.
3. Using ERwin, import this ERX file.

Each step of this process is described in the following paragraphs.

1) Using DB2 Cube Views, export your cube model as an XML file

The DB2 Cube Views model used in this scenario is the one shown in Figure 9-9 on page 339.

This step has already been detailed in “1) Using DB2 Cube Views, export your cube model as an XML file” on page 349. The DB2 cube model is saved into an XML file using **OLAP Center > Export** in DB2 Cube Views.

2) Using MIMB, convert DB2 Cube Views XML file into ERX file

Start the MIMB software, select the import bridge labeled IBM DB2 Cube Views’ and import your model. Select the export bridge labeled “CA ERwin 3.0 to 3.5.2”, select the name of the export ERwin ERX file, and click the **Export Model** button.

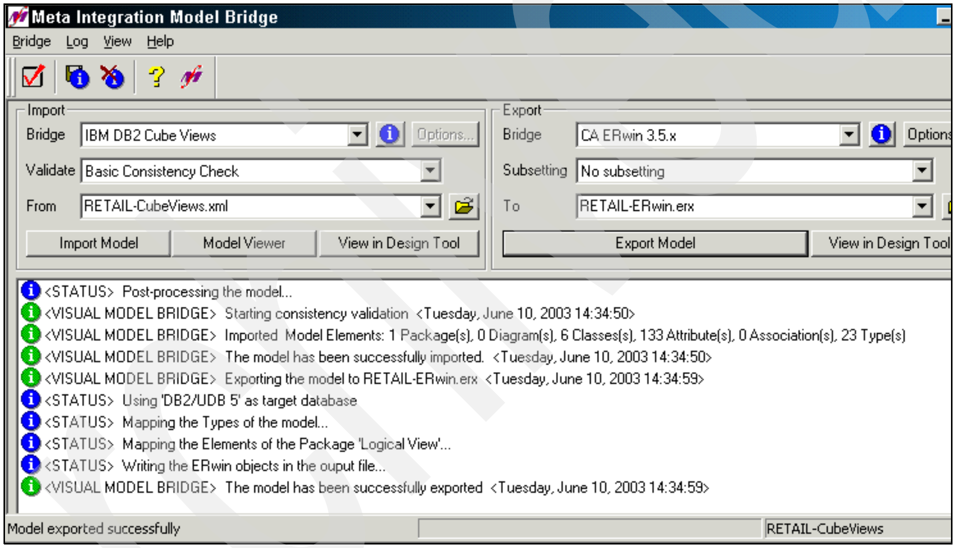


Figure 9-35 Converting the DB2 cube model XML to an ERwin ERX file

3) Using ERwin, import this ERX file

At this point, you can open the generated ERX file into ERwin using menu **File -> Open.** When the file choice window appears, select ERwin ERX (*.erx) in the **List files of type** list box and select the ERX file produced by MIMB as shown in Figure 9-36.

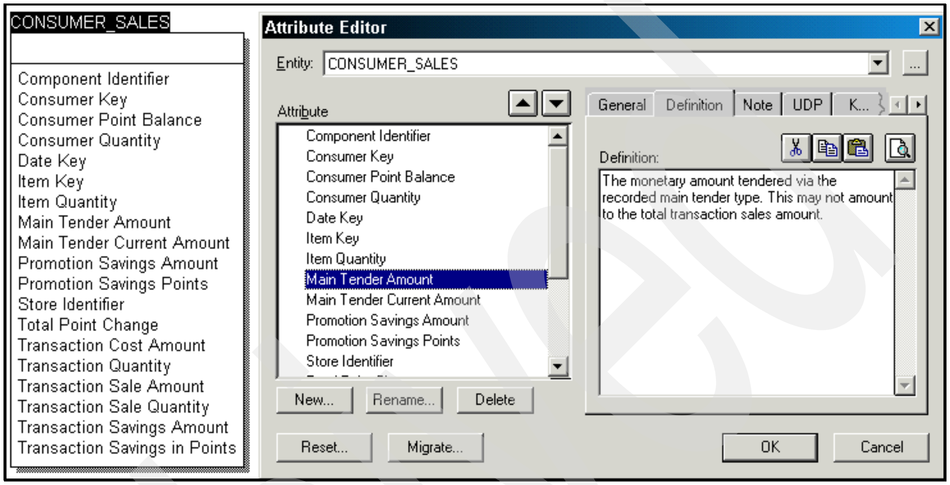


Figure 9-36 DB2 cube model converted to ERwin

The cube model converted to ERwin contains the business names and descriptions, and a logical view is displayed in Figure 9-37.

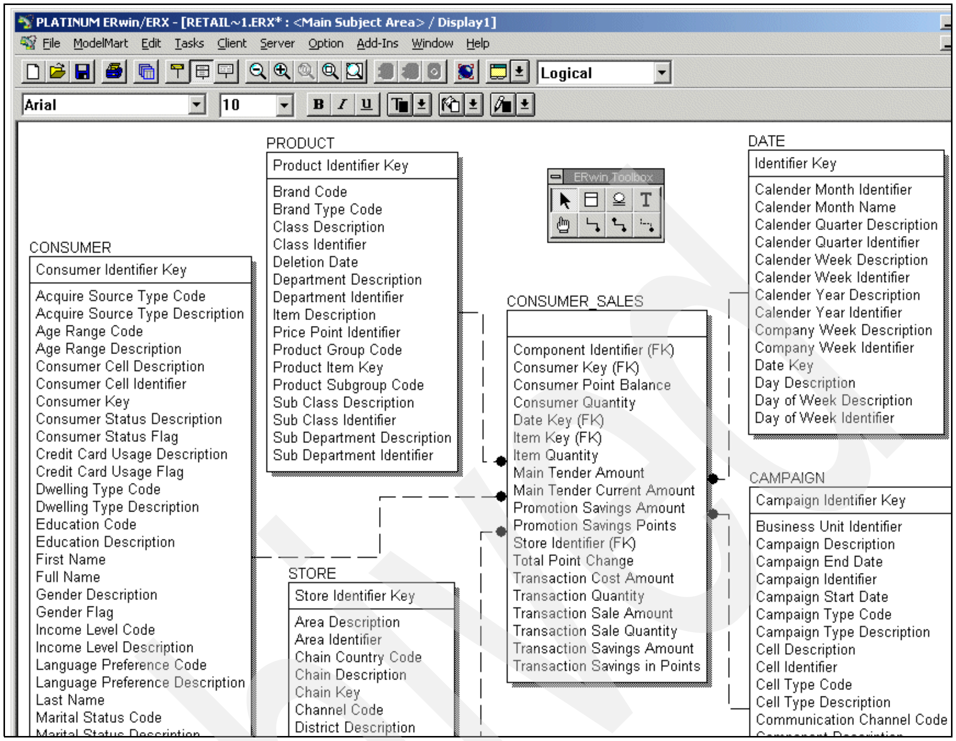


Figure 9-37 The cube model reversed engineered to ERwin 3.x

Congratulations, the cube model was converted to ERwin!

9.5.3 Metadata integration of DB2 Cube Views with PowerDesigner

Sybase PowerDesigner is one of the leading database design tools. PowerDesigner allows designing Conceptual Data Models (CDM) as well as Physical Data Models (PDM). It supports DB2 UDB as target database system of a physical data model, provides star schema database design features in *physical diagrams* and also provides multidimensional/OLAP modeling features in *multidimensional diagrams*. In this scenario, we will demonstrate the forward and reverse engineering of star schema in PowerDesigner PDM *physical diagrams*.

Forward engineering from PowerDesigner to DB2 Cube Views

The goal of this scenario is to demonstrate how an existing PowerDesigner PDM model can be converted to a DB2 cube model.

The overall process of this metadata conversion is as follows:

1. Using PowerDesigner, create the star schema PDM model.
2. Using PowerDesigner, generate the SQL DDL for this database.
3. Using DB2, run this SQL script to create the tables and columns of this schema.
4. Using PowerDesigner, save the model as PDM XML.
5. Using MIMB, convert this PowerDesigner XML file into a DB2 Cube Views XML file.
6. Using DB2 Cube Views, import this Cube Views XML file to import the metadata.

Each step of this process is described in the following paragraphs.

1) Using PowerDesigner, create the star schema PDM model

The PowerDesigner model used in this scenario is the one shown in Figure 9-38. This database model is in the form of a star-schema, and the bridge will use the dimensional information specified in the model to create a cube model.

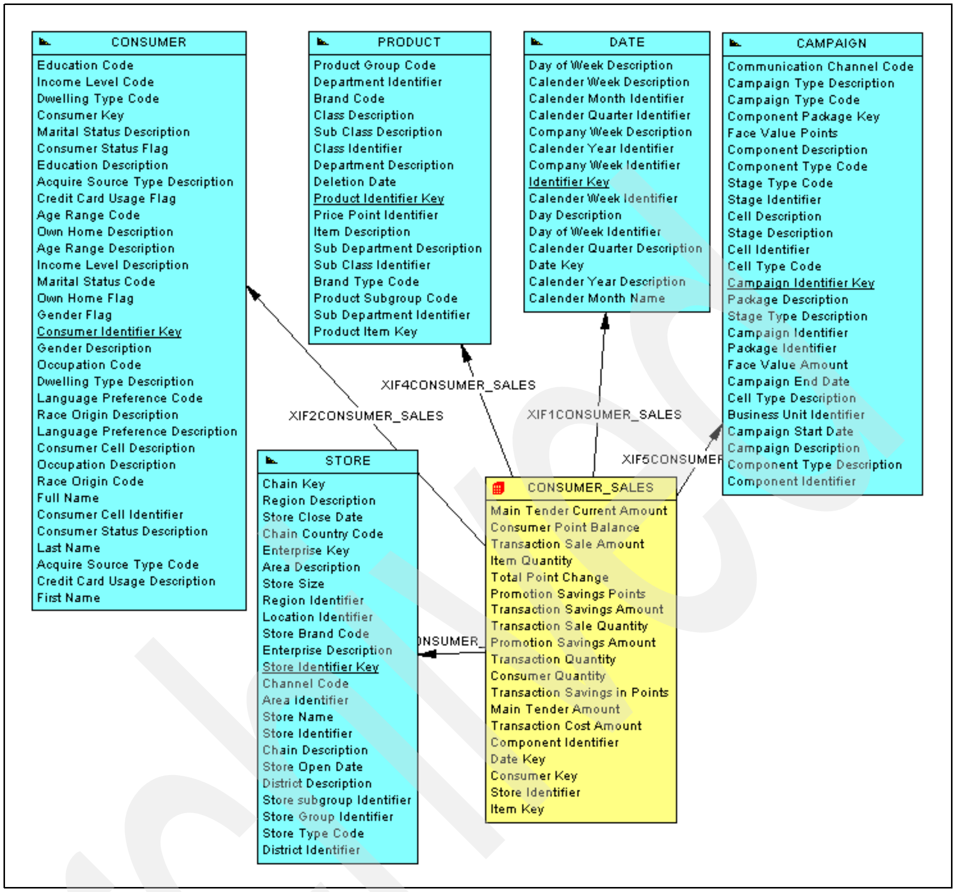


Figure 9-38 Logical view of the PowerDesigner PDM model

During the implementation of this data model in PowerDesigner, the dimensional modeling features of the PDM *physical diagram* were used. A dimensional type was specified on each table (**Fact** or **Dimension**) as shown in Figure 9-39.

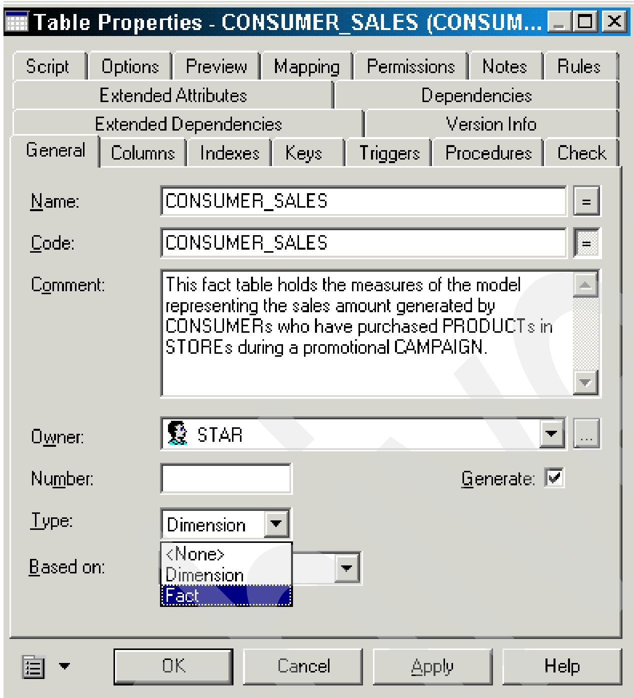


Figure 9-39 Specifying the tables' dimensional type

Documentation was also specified in this model, in the form of a **Comment** field on the objects as shown in Figure 9-40.

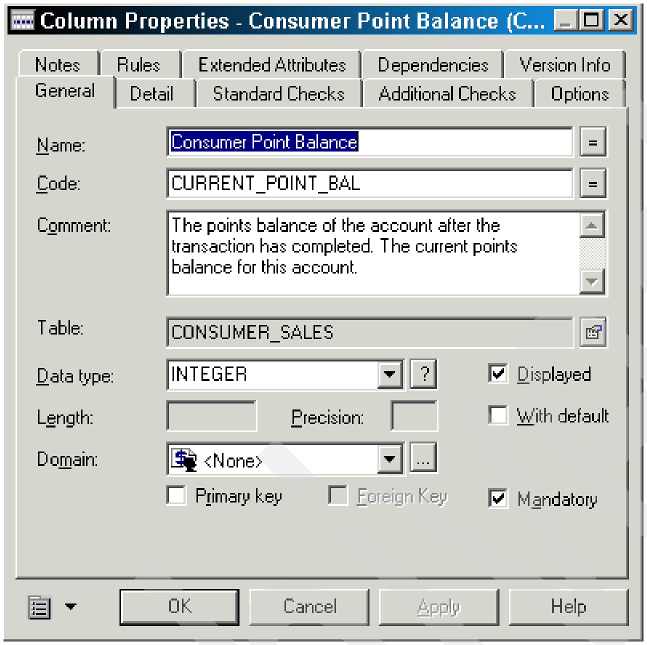


Figure 9-40 Adding documentation to the PowerDesigner model

2) Using PowerDesigner, generate the SQL DDL for this database

Once the model has been designed, the SQL DDL can be generated and the database created in DB2 UDB. In PowerDesigner, choose the menu **Database -> Generate Database** to generate the SQL script as shown in Figure 9-41.

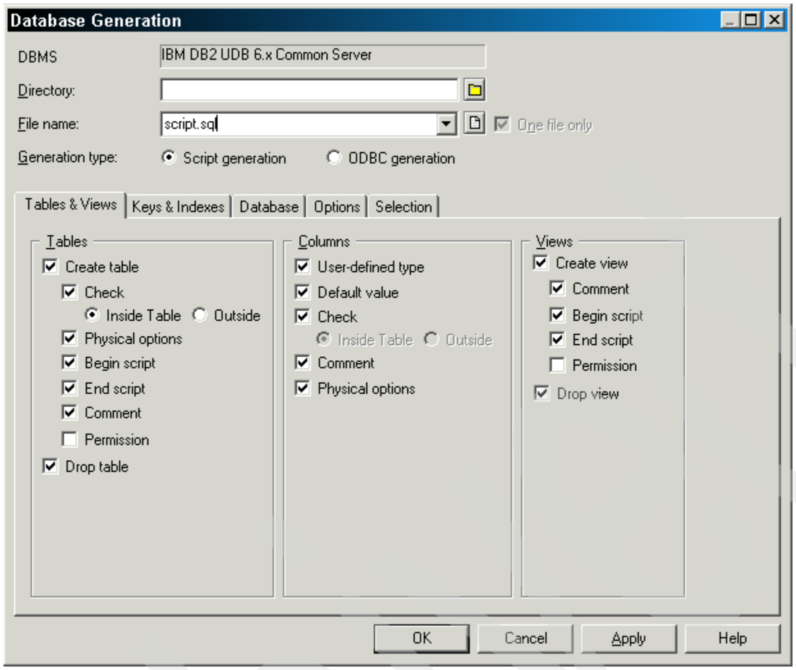


Figure 9-41 DB2 schema generation

3) Using DB2, create the tables and columns of this schema

This SQL script generated can be executed under DB2 Command Tool to create the DB2 tables as already discussed in "3) Using DB2, create the tables and columns of this schema" on page 356.

4) Using PowerDesigner, save the model as PDM XML

The next step of this process is to save the PowerDesigner model as a PDM XML file. The bridge will use this file as the input.

When the model is loaded in PowerDesigner, choose **Save As** from the File menu, select the **Physical Data Model** (xml) (*.pdm) format in the **Save as type list**, type the file name for the model you are saving in the **File name** text box and click **Save.**

> **Note:** PowerDesigner also allows sharing the definition of metadata (tables, views, relationships) across different models via the notion of shortcuts. If your model contains shortcuts to external objects defined in other models, the definition of the referenced objects may not be completely saved in the current PDM file.
>
> We recommend not using such external shortcuts for the purpose of metadata integration with DB2 Cube Views.

5) Using MIMB, convert PowerDesigner to DB2 Cube Views

Start the MIMB tool and select the import bridge labeled **Sybase PowerDesigner PDM 7.5 to 9.5**, and import your PowerDesigner PDM XML file, as shown in Figure 9-42.

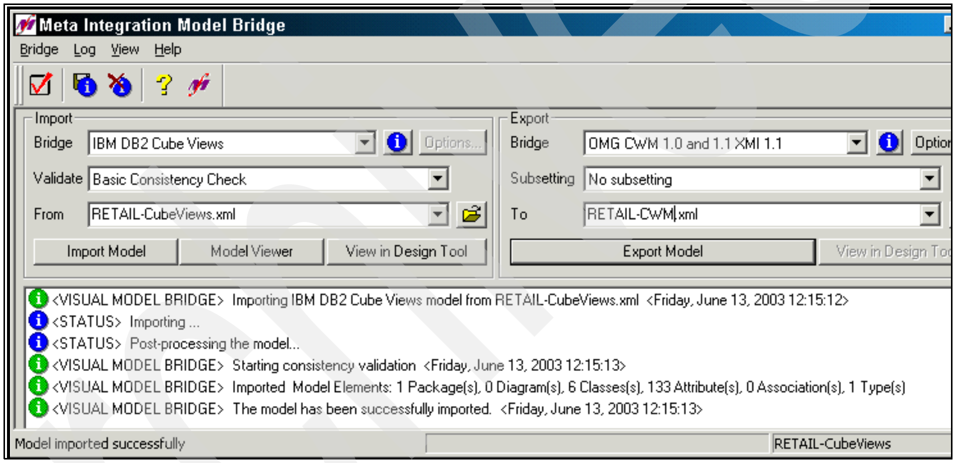


Figure 9-42 Importing the PowerDesigner model into MIMB

Select the export bridge labeled **IBM DB2 Cube Views** and click the **Options** button to specify the export parameters as shown in Figure 9-43.

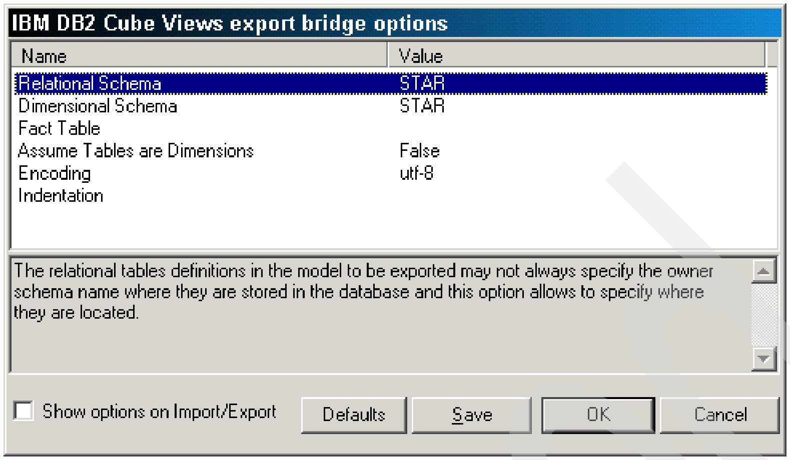


Figure 9-43 Specifying the export bridge parameters

The export parameters used in this scenario are as follows:

- The DB2 schema for the tables of the model is 'STAR', as the model may not always specify where each table is located.
- The cube model to be created will be located in the same 'STAR' DB2 schema.
- We specify that the encoding of the source model is utf-8.
- The other options are left with their default value.

Close this window, specify the name of the DB2 Cube Views XML file to be created, and click the **Export** button to get the display shown in Figure 9-44.

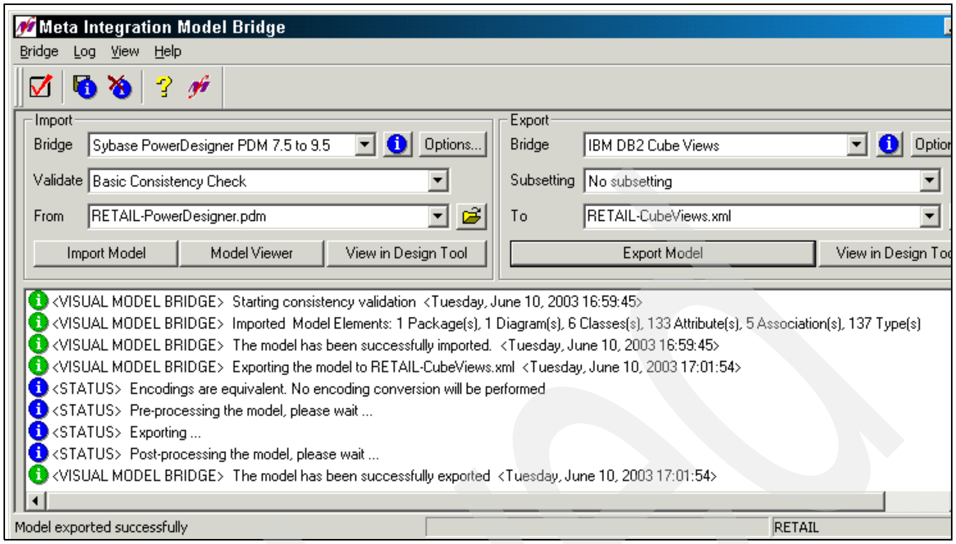


Figure 9-44 Exporting the model to DB2 Cube Views

6) Using DB2 Cube Views, import the DB2 Cube Views XML file

At this point, the cube model XML file has been created and is ready to be opened into the OLAP Center graphical tool. Just start OLAP Center, connect to your database, and choose **Import** in the OLAP Center menu to get the display shown in Figure 9-45.

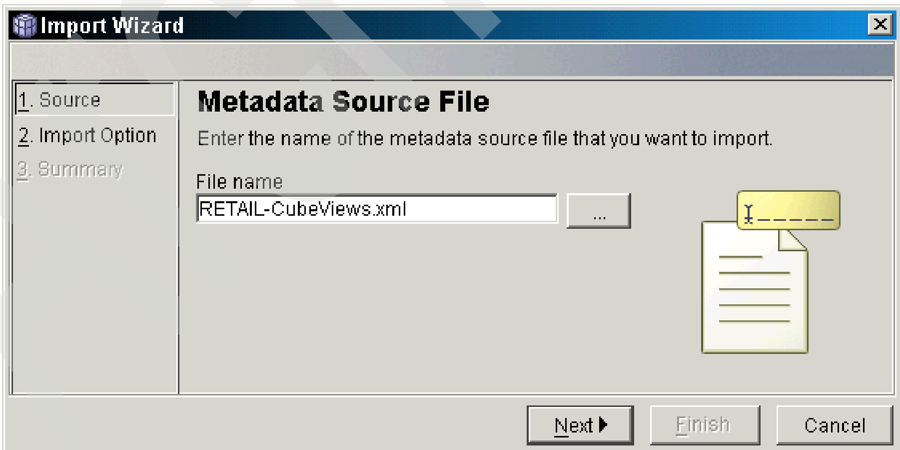


Figure 9-45 Specifying the XML file to import into OLAP Center

The content of the XML file is displayed in Figure 9-46, which allows controlling how this metadata should be imported, in case there is already some metadata in place and object name collision should occur:

- Either update the existing objects with the new imported version.
- Or keep the current version of the metadata.

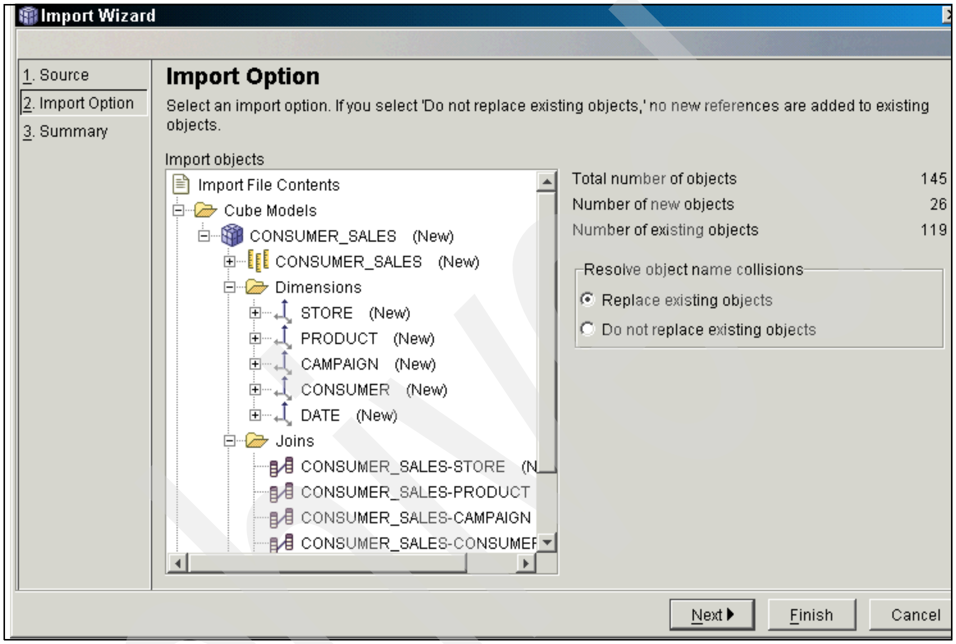


Figure 9-46 Controlling how the metadata is imported into OLAP Center

Finally, the PowerDesigner metadata converted and imported into OLAP Center will provide the DB2 cube model in Figure 2-9 on page 15.

The business names and descriptions defined in PowerDesigner are also converted to the cube model, as shown in Figure 9-47.

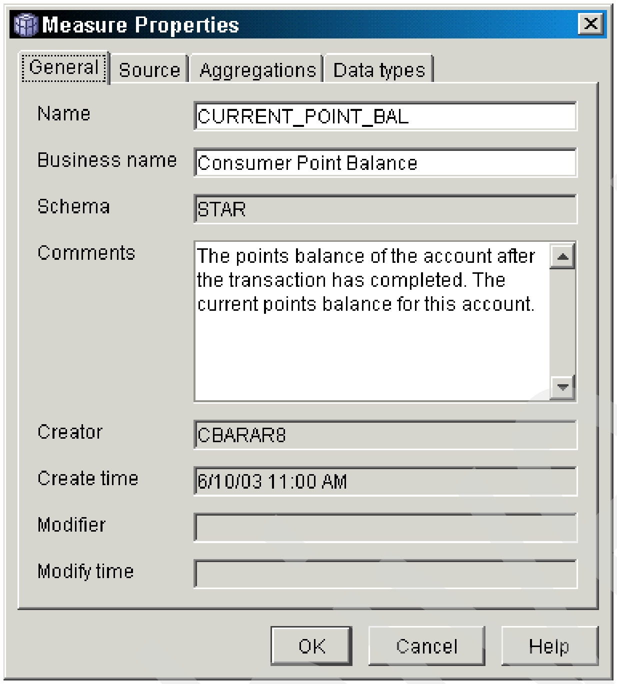


Figure 9-47 The PowerDesigner business names and descriptions are converted

Congratulations, the PowerDesigner star schema model was converted to DB2 Cube Views!

Reverse engineering from DB2 Cube Views to PowerDesigner

The goal of this scenario is to demonstrate how an existing DB2 Cube Views model can be converted into a PowerDesigner PDM physical diagram.

The overall process of this metadata conversion is as follows:

1. Using DB2 Cube Views, export your cube model as an XML file.
2. Using MIMB, convert this DB2 Cube Views XML file into an PowerDesigner XML file.
3. Using PowerDesigner, import this XML file.

Each step is detailed in the following paragraphs.

1) Using DB2 Cube Views, export your cube model as an XML file

The DB2 Cube Views model used in this scenario is the one shown in Figure 9-9 on page 339.

This step has already been detailed in “1) Using DB2 Cube Views, export your cube model as an XML file” on page 24. The DB2 cube model is saved into an XML file using OLAP Center in DB2 Cube Views.

2) Using MIMB, convert DB2 Cube Views into PowerDesigner

Start the MIMB software, select the import bridge labeled **IBM DB2 Cube Views** and import your model. Select the export bridge labeled **Sybase PowerDesigner PDM 7.5 to 9.5**, select the name of the export PowerDesigner PDM file and click the **Export** Model button to get the display in Figure 9-48.

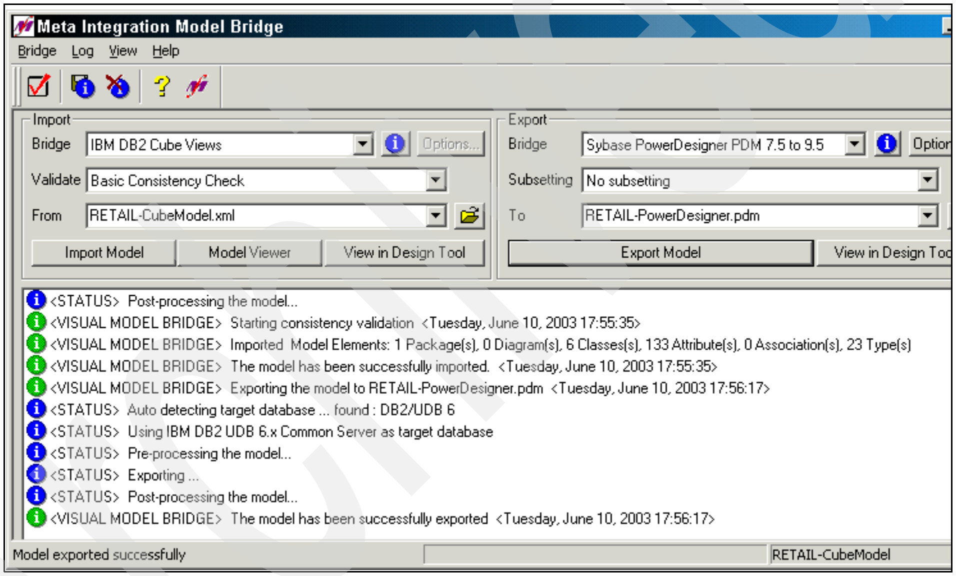


Figure 9-48 Converting the cube model XML file to an PowerDesigner XML file

3) Using PowerDesigner, import this XML file

At this point, you can open the generated PDM file into PowerDesigner using menu **File -> Open** as shown in Figure 9-49.

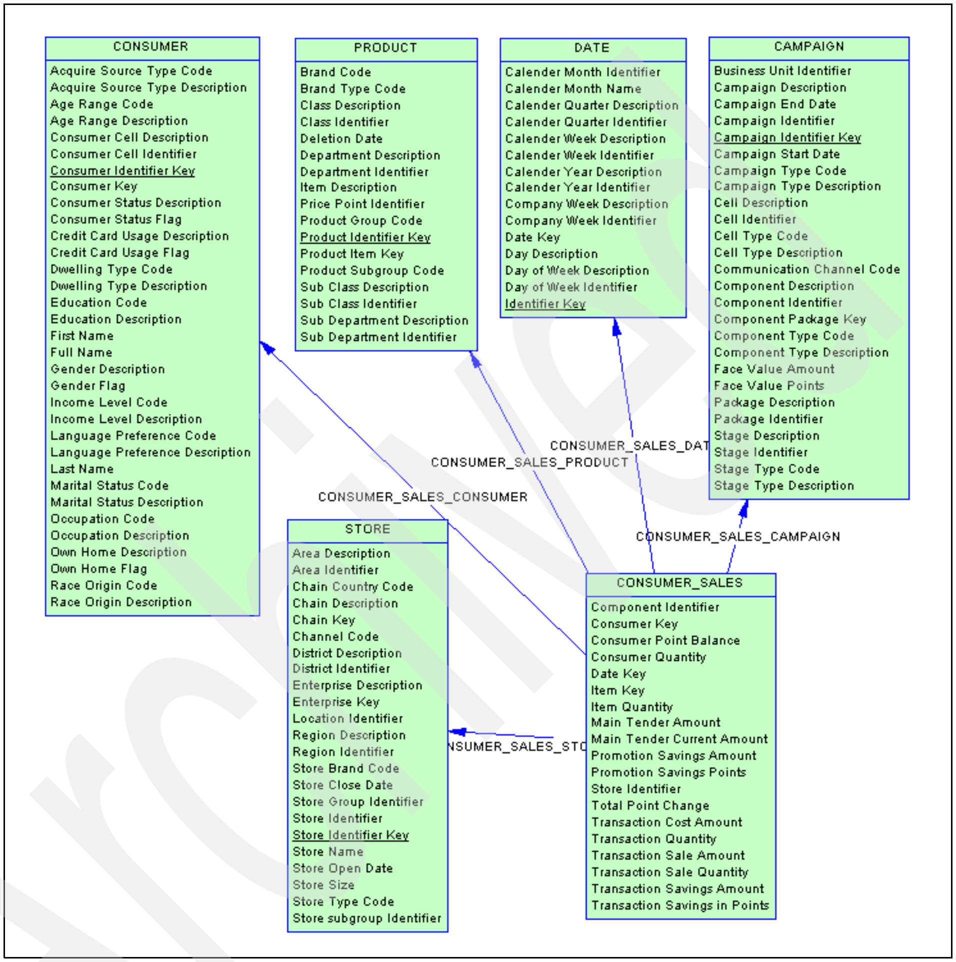


Figure 9-49 The cube model reverse engineered to PowerDesigner

The cube model converted to PowerDesigner contains the business names and descriptions as displayed in Figure 9-50.

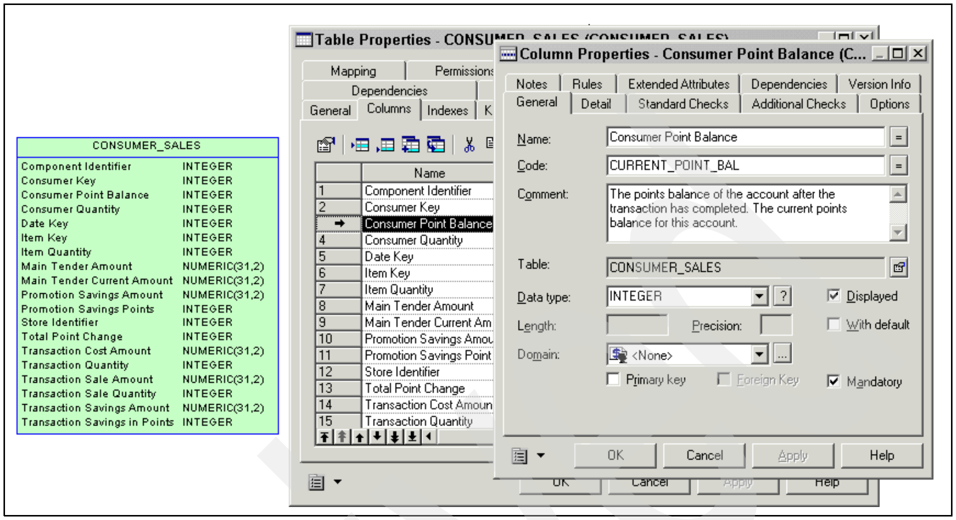


Figure 9-50 PowerDesigner business names and descriptions

Congratulations, the cube model was converted to PowerDesigner!

9.5.4 Metadata integration of DB2 Cube Views with IBM Rational Rose

IBM Rational Rose is one of the leading object modeling and data modeling tools. Rose can be used to design UML models for several target languages (C++, Java) as well as database schemas for DB2 UDB and many other database systems.

> **Note:** The Rose MDL file format is very widely used as a de facto standard means of exchanging UML metadata. Many design tools support it and therefore this scenario can also be used to interact and exchange metadata with them.
>
> A non-exhaustive list of such tools would include IBM Rational XDE, Microsoft Visual Studio 6 (Visual Modeler), Sybase PowerDesigner, Embarcadero Describe, Gentleware Poseidon and Casewise.

Forward engineering from Rational Rose to DB2 Cube Views

The goal of this scenario is to demonstrate how an existing Rose model can be converted to a DB2 cube model.

The overall process of this metadata conversion is as follows:

1. Using Rose, create the model.
2. Using Rose, generate the SQL DDL for this database.
3. Using DB2, run this SQL script to create the tables and columns of the database.
4. Using Rose, save the model as an MDL file.
5. Using MIMB, convert this Rose MDL file into a DB2 Cube Views XML file.
6. Using DB2 Cube Views, import this DB2 Cube Views XML file.

Each step of this process is described in the following paragraphs.

1) Using Rose, create the model

The Rose model used in this scenario is composed of an object model *(UML Class Diagram*) and a data model (*Database Schema diagram*). The object model in Figure 9-51 holds the logical definition of the tables (such as business names and descriptions).

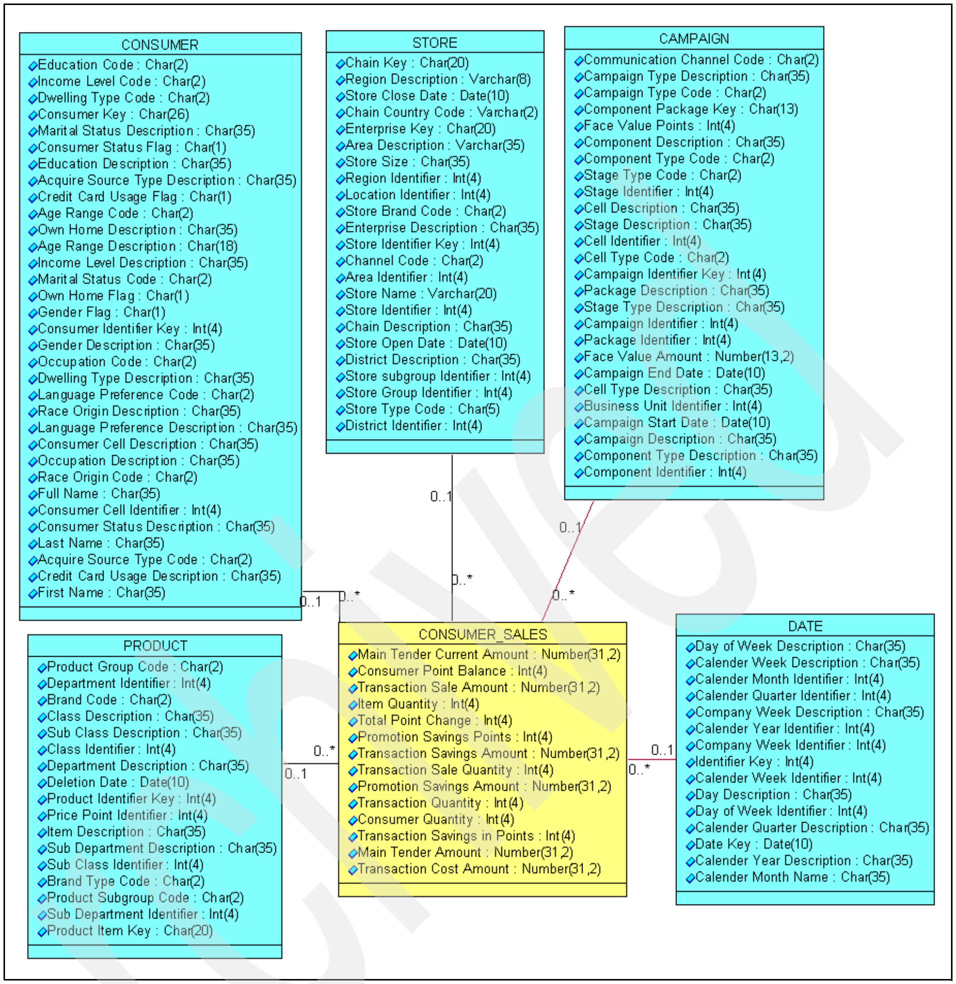


Figure 9-51 The Rose object model

The data model in Figure 9-52 holds the physical definition of the database (such as column names, primary keys, foreign keys)

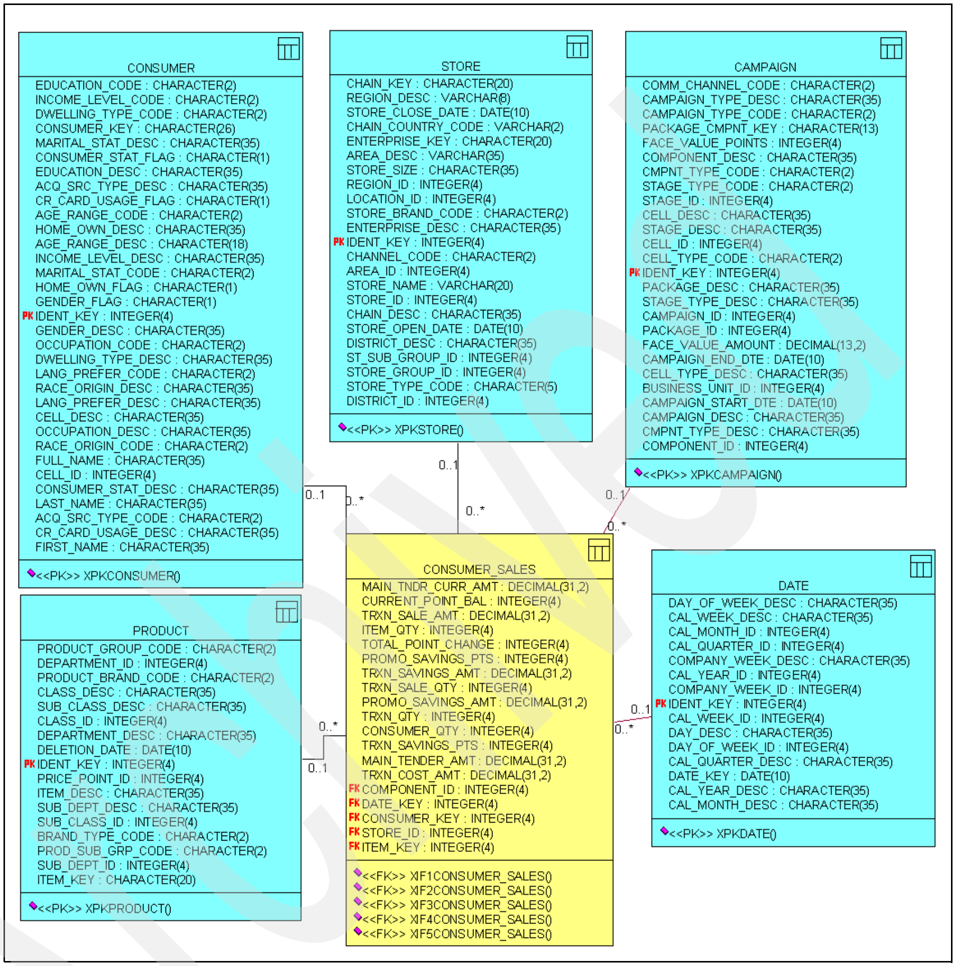


Figure 9-52 The Rose data model

To create this model in Rose, the UML object model was developed first, and was then transformed into a relational database schema, as shown in Figure 9-53, Figure 9-54, Figure 9-55, and Figure 9-56.

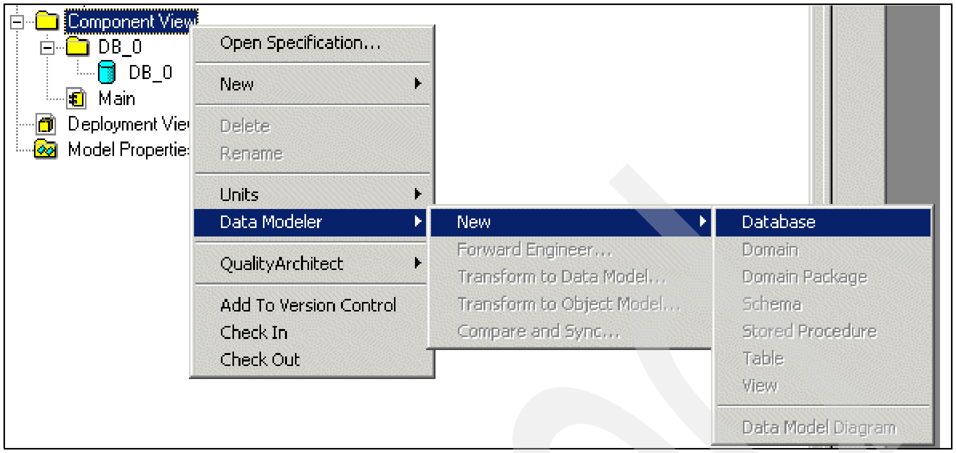


Figure 9-53 Create a new database

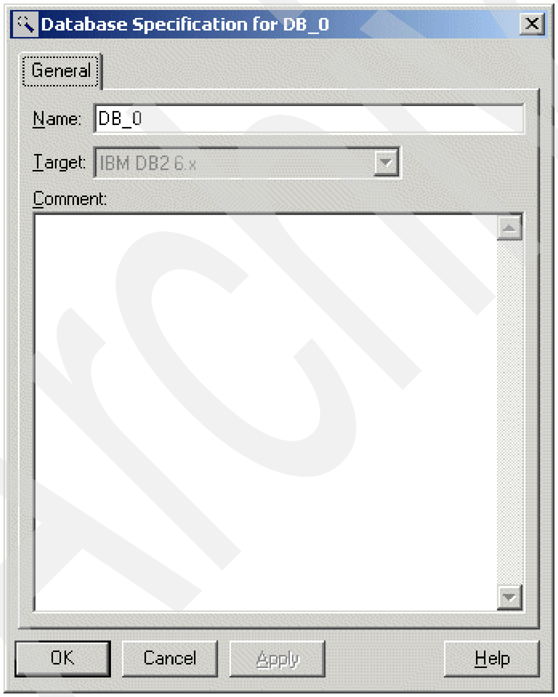


Figure 9-54 Define the database properties

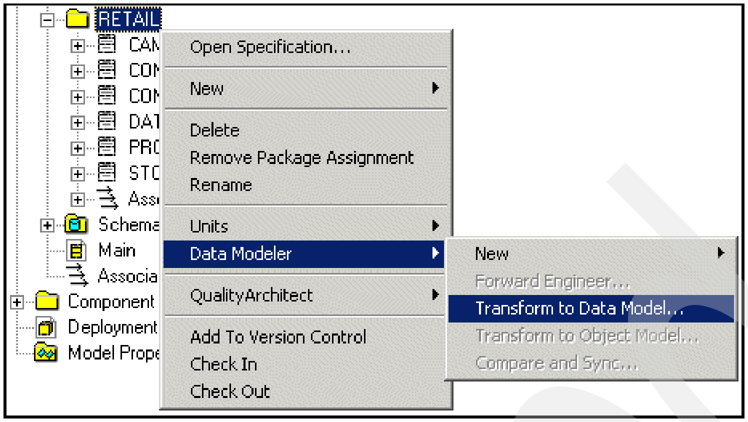


Figure 9-55 Transform to data model option

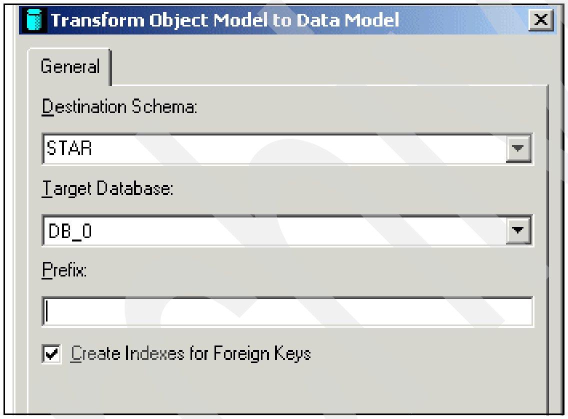


Figure 9-56 Transforming an object model into a data model in Rose

2) Using Rose, generate the SQL DDL

Once the model has been designed, the SQL DDL can be generated and the database created in DB2 UDB as shown in Figure 9-57.

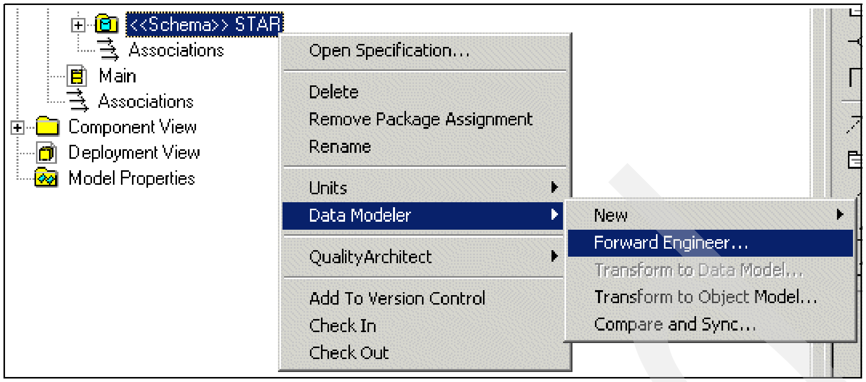


Figure 9-57 Generation of the SQL DDL in Rose

3) Using DB2, create the tables and columns of the database

This script can be executed to create the DB2 Tables using the DB2 Command Window tool (or any other tool) as described in “3) Using DB2, create the tables and columns of this schema” on page 356:

At this point, the database has been set up and it is ready to receive the cube model.

4) Using Rose, save the model as an MDL file

The next step of this process is to save the Rose model as an MDL file, the bridge will use this file as input.

When the model is loaded in Rose, choose **Save** from the **File** menu.

5) Using MIMB, convert Rose MDL into DB2 Cube Views XML

Start the MIMB tool and select the import bridge labeled **IBM Rational Rose 2000e to 2002**, and click the **Options** button to specify the import parameters as shown in Figure 9-58.

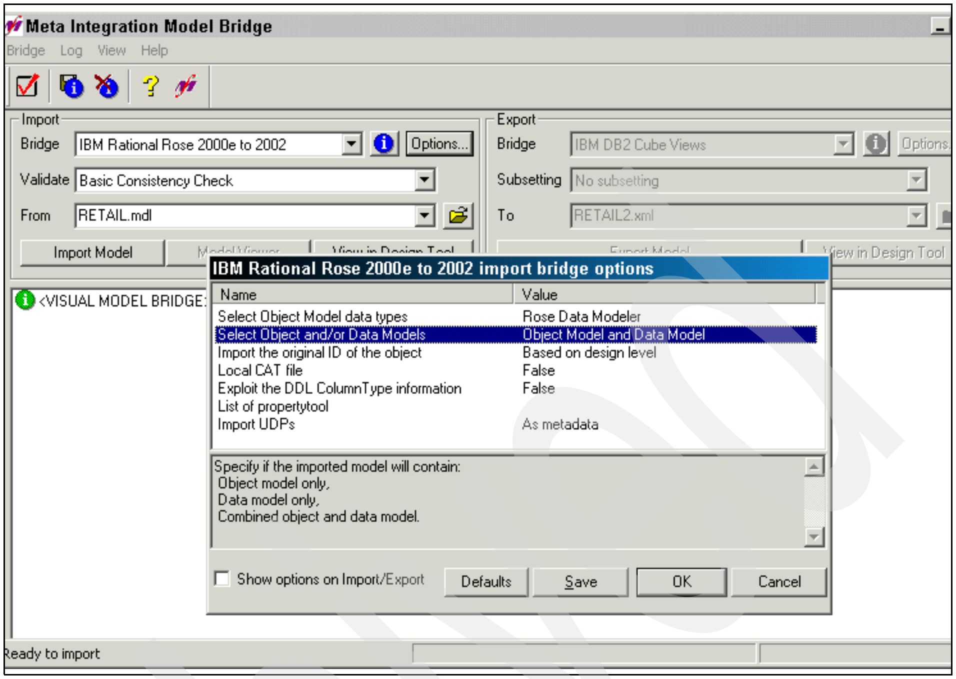


Figure 9-58 Specifying the import parameters

In this scenario, the import parameters are set as follows:

- The data types are imported as defined in the Rose data model.
- The Rose data model and its associated UML model are imported and integrated into logical and physical model
- The other options are left with their default values.

Then, we can import the Rose MDL file, as shown in Figure 9-59.

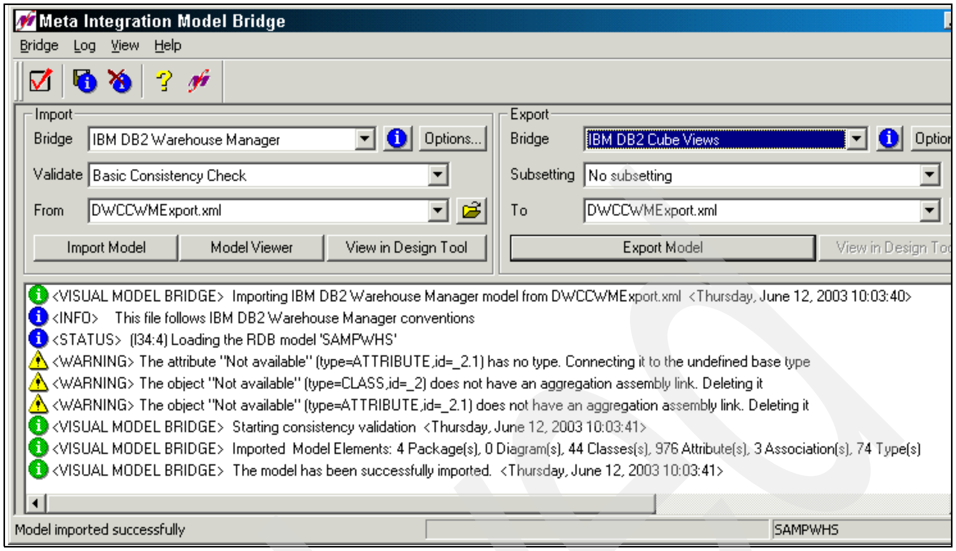


Figure 9-59 Importing the Rose model into MIMB

Select the export bridge labeled **IBM DB2 Cube Views** and click the **Options** button to specify the export parameters as shown in Figure 9-60.

Figure 9-60 Specifying the export bridge parameters

The export parameters used in this scenario are as follows:

- The DB2 schema for the tables of the model is *STAR*, as the model may not always specify where each table is located.
- The cube model to be created will be located in the same ‘STAR’ DB2 schema.
- Rose Data Modeler 2002 doesn’t support the notion of fact table and dimension tables yet. To work around this limitation, we can specify explicitly which table is to be considered as fact (*CONSUMER_SALES* in this case) and force the bridge to consider the other tables as dimensions.
- We should specify the encoding of the source model (locale encoding of the computer on which the Rose model was created), it is windows-1252 by default on Microsoft Windows.
- The other options are left with their default value.

Close this window, specify the name of the DB2 Cube Views XML file to be created, and click the **Export** button as shown in Figure 9-61.

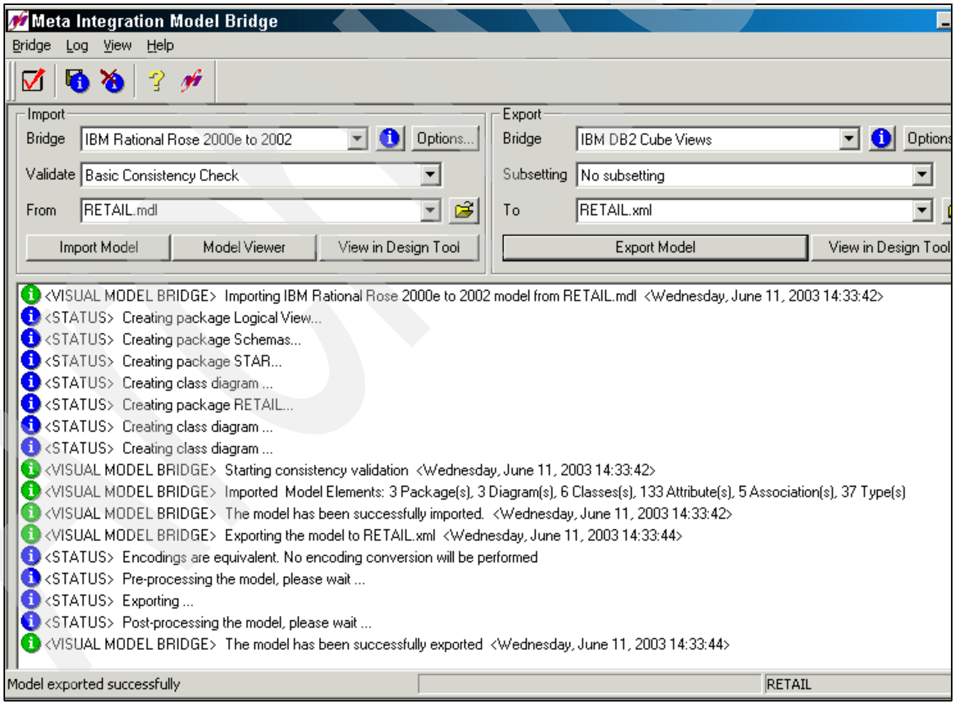


Figure 9-61 Exporting the model to DB2 Cube Views

6) Using DB2 Cube Views, import the DB2 Cube Views XML file

At this point, the cube model XML file has been created and is ready to be opened into the OLAP Center graphical tool. Just start OLAP Center, connect to your database, and choose **Import** in the OLAP Center menu to display Figure 9-62.

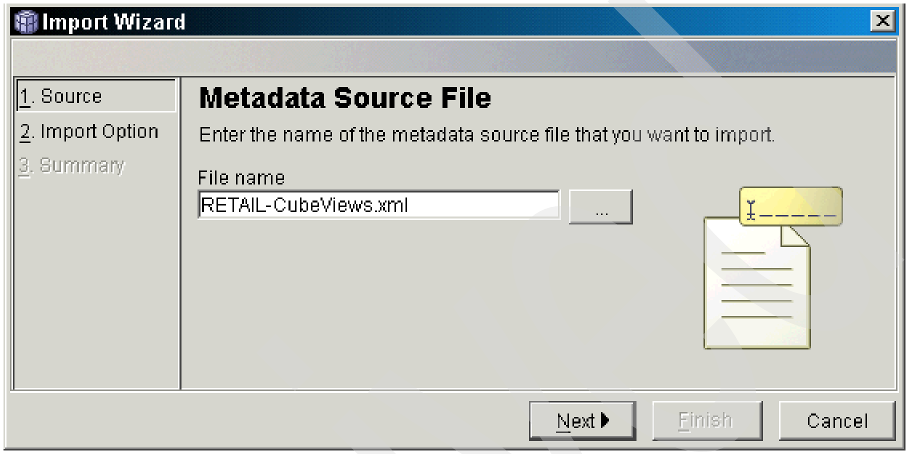


Figure 9-62 Specifying the XML file to import into OLAP Center

The content of the XML file is displayed in Figure 9-63, which allows controlling how this metadata should be imported, in case there is already some metadata in place and object name collision should occur:

- Either update the existing objects with the new imported version.
- Or keep the current version of the metadata.

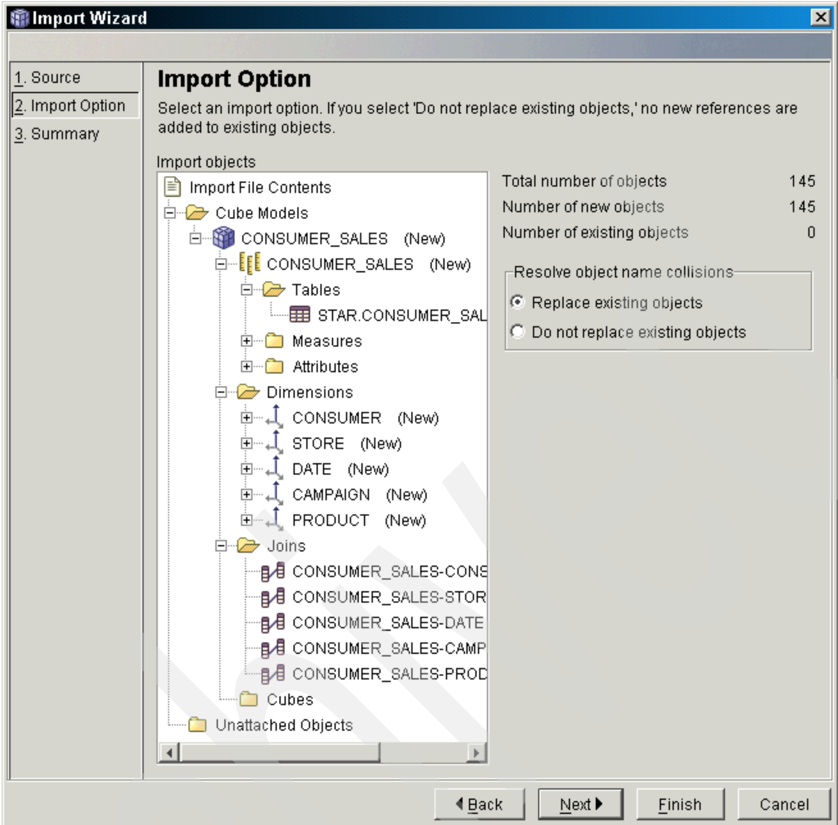


Figure 9-63 Controlling how the metadata is imported into OLAP Center

Finally, the Rose star schema metadata converted and imported into OLAP Center will provide the DB2 cube model in Figure 9-9 on page 339.

The business names and descriptions defined in Rose are also converted to the cube model, as shown in Figure 9-64.

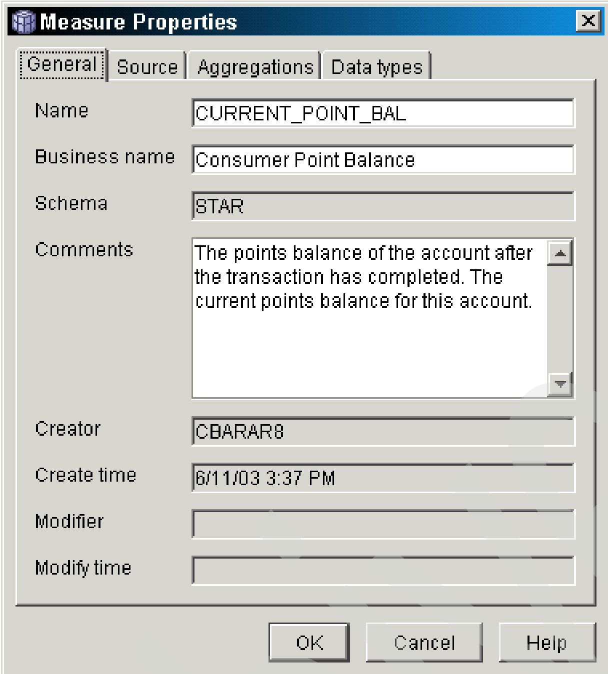


Figure 9-64 The Rose objects' business name and description are also converted

Congratulations, the Rose star schema model was converted to DB2 Cube Views!

Reverse engineering from DB2 Cube Views to Rational Rose

The goal of this scenario is to demonstrate how an existing DB2 Cube Views model can be converted into a Rose Model.

The overall process of this metadata conversion is as follows:

1. Using DB2 Cube Views, export your cube model as an XML file.
2. Using MIMB, convert this DB2 Cube Views XML file into a Rose MDL file.
3. Using Rose, import this MDL file.

Each step of this process is described in the following paragraphs.

1) Using DB2 Cube Views, export your cube model as an XML file

The DB2 Cube Views model used in this scenario is the one shown in Figure 9-9 on page 339.

This step has already been detailed in “1) Using DB2 Cube Views, export your cube model as an XML file” on page 24. The DB2 cube model is saved into an XML file using **OLAP Center > Export** in DB2 Cube Views.

2) Using MIMB, convert DB2 Cube Views XML into Rose MDL

Start the MIMB software, select the import bridge labeled **IBM DB2 Cube Views** and import your model. Select the export bridge labeled **IBM Rational Rose 2002**, select the name of the export Rose MDL file, and click the **Export Model** button as shown in Figure 9-65.

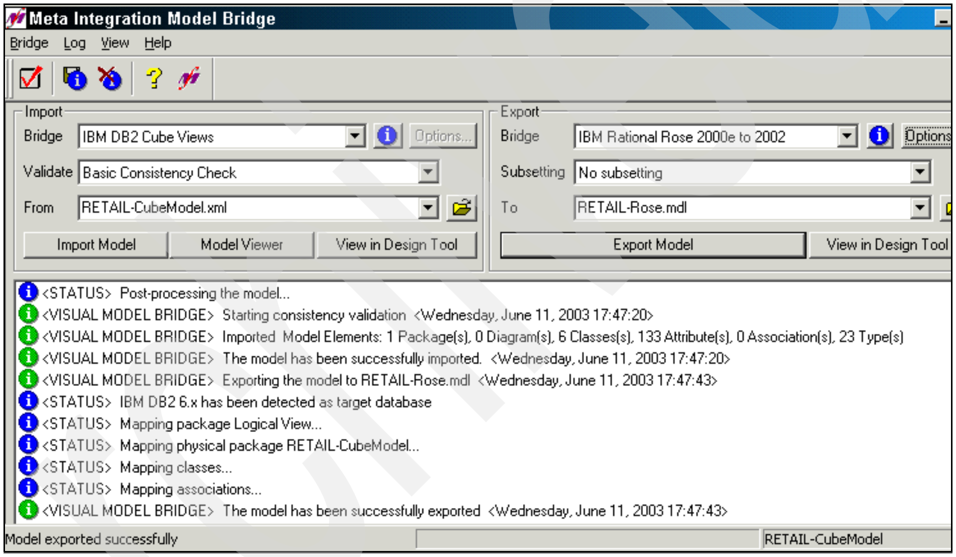


Figure 9-65 Converting the cube model XML file to a Rose MDL file

3) Using Rose, import this MDL file

At this point, you can open the generated MDL file into Rose using menu **File > Open**.

The data model after conversion is shown in Figure 9-66.

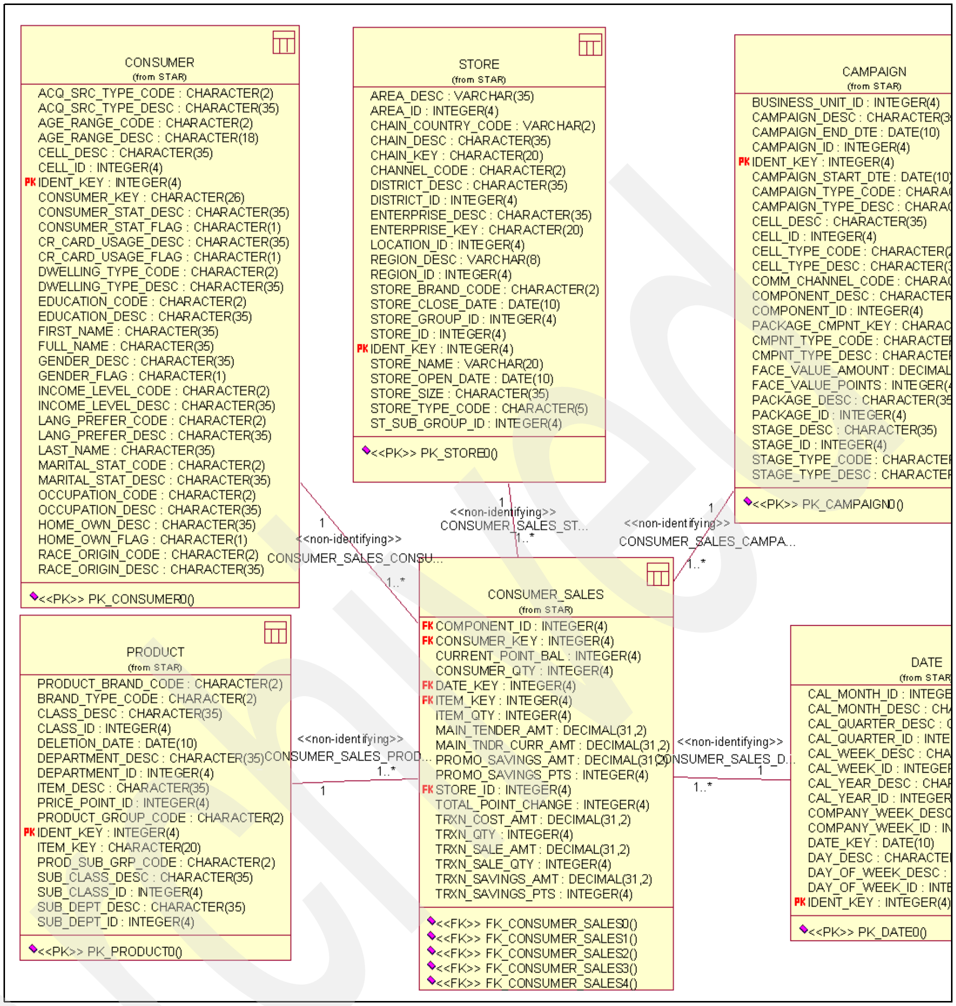


Figure 9-66 The cube model converted to Rose Data Modeler

The object model after conversion is shown in Figure 9-67.

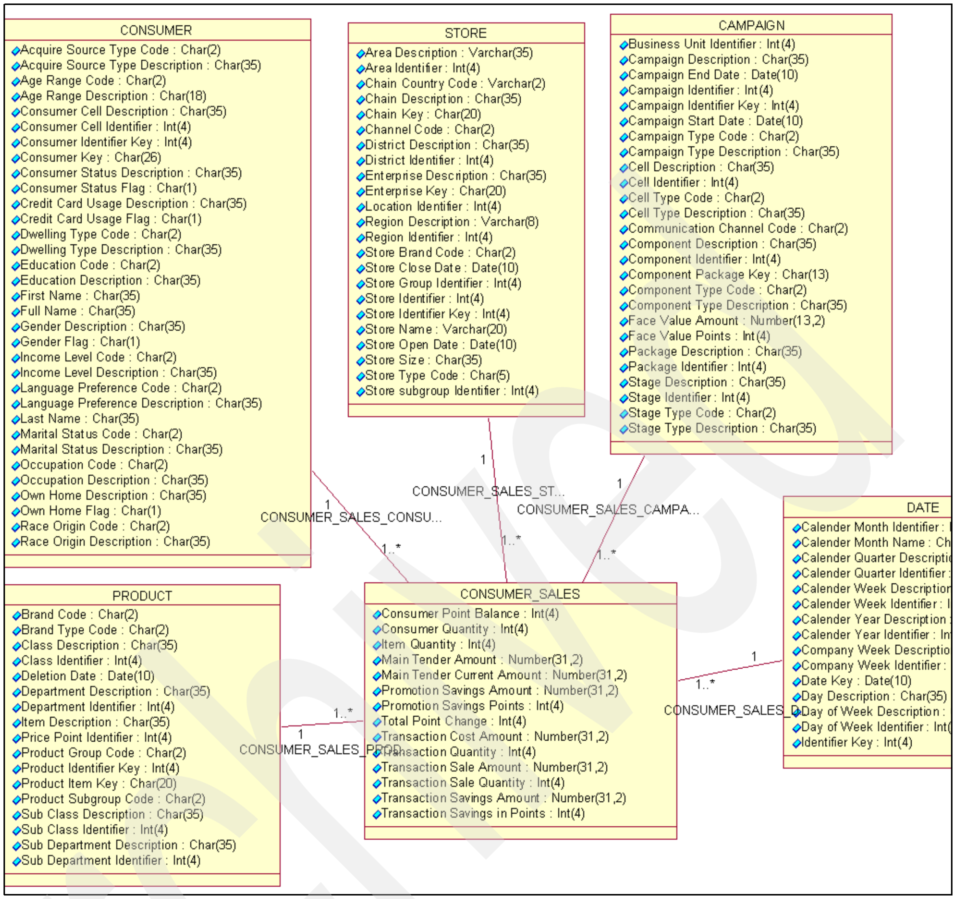


Figure 9-67 The cube model converted to Rose Object Modeler

Congratulations, the cube model was converted to Rose!

9.5.5 Metadata integration of DB2 Cube Views with CWM and XMI

The Object Management Group (OMG) Common Warehouse Metamodel (CWM) is an industry standard metamodel supported by numerous leading data and metadata management tools vendors. The CWM metamodel shown in Figure 9-68 is defined as an instance of the Meta Object Facility (MOF) meta-metamodel and expressed using the OMG Unified Modeling Language (UML) in terms of classes, relationships, diagrams and packages. Any model instance of the UML and CWM metamodel can also be serialized into an XML document using the XML Metadata Interchange (XMI) facility.

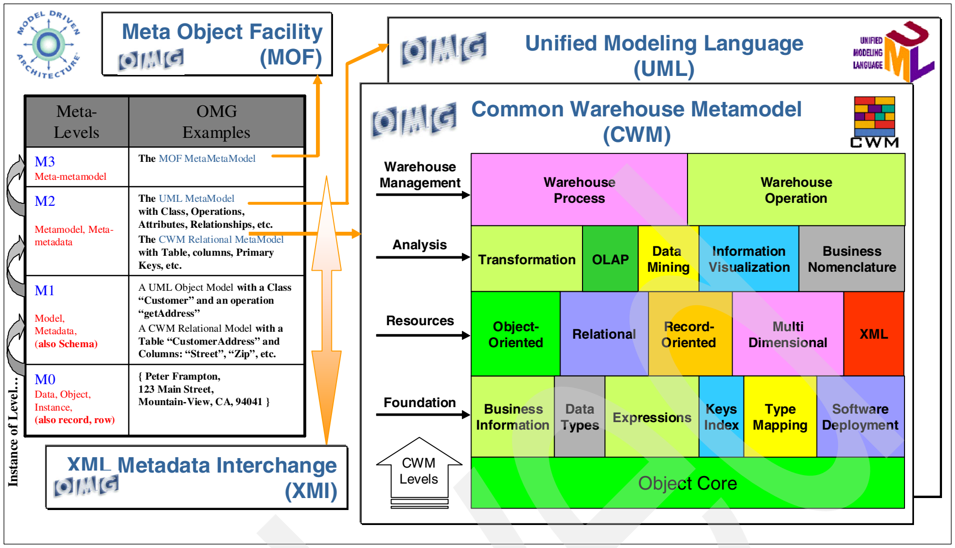


Figure 9-68 The OMG standards

Meta Integration Technology, Inc. (MITI) has been a strong supporter of the OMG CWM standard since 1999 and joined the OMG in 2000 as an influencing member. Since 2001, MITI became a domain member of the OMG focusing on XMI based metadata interchange. MITI is mostly working on the implementation and support of the CWM standard with other key OMG members such as Adaptive, Hyperion, IBM, Oracle, SAS, and Unisys. MITI is also actively participating to all OMG enablement showcases demonstrating bi-directional metadata bridges with many design tools, ETL tools and BI tools.

The April 28 - May 2, 2002 CWM Enablement Showcase at the Annual Meta Data Conference / DAMA Symposium, San Antonio, Texas is shown in Figure 9-69.

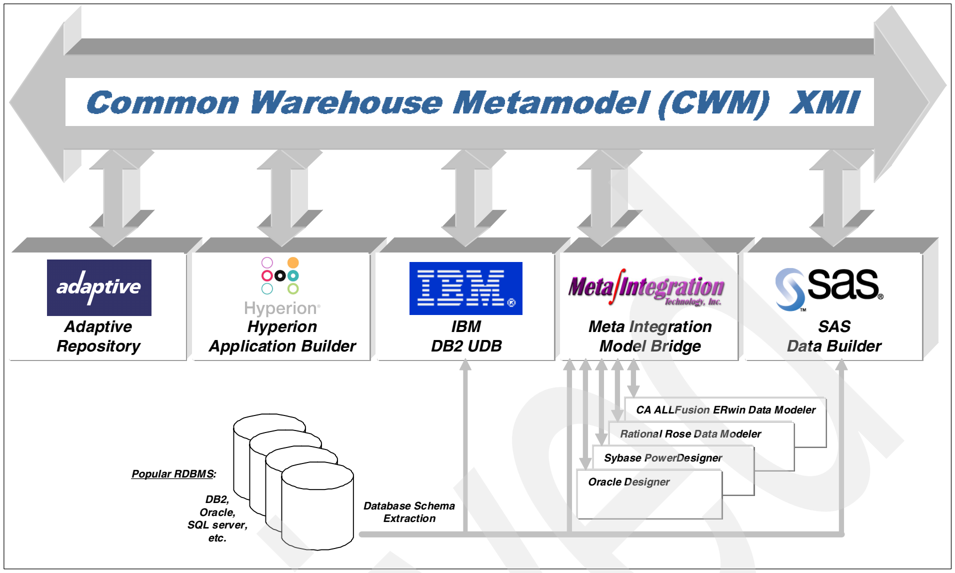


Figure 9-69 CWM Enablement Showcase

The metadata interchange and integration challenges using the CWM and XMI standards are due to multiple factors:

- The UML, CWM and XMI standards are evolving and each of them has multiple versions. A given instance metadata document is therefore a combination of versions of each of these standards.
- A testing suite or open source reference implementation is not yet available.
- Software vendors implementing import/export capabilities often need to extend the standard by using additional properties (TaggedValues), additional metadata resources (the CWMX extension packages) leading to specific CWM dialects.

Nevertheless, CWM XMI is the leading standard in metadata interchange and MITI has implemented a comprehensive support for it, including support for various versions of the metamodel and XMI encoding, and also many specific tool vendors' features. For more information, please read the following pages online:

```
http://www.metaintegration.net/Partners/OMG.html
http://www.metaintegration.net/Products/MIMB/Documentation/OMGXMI.html
http://www.metaintegration.net/Products/MIMB/SupportedTools.html
```

This scenario demonstrates how to export DB2 Cube Views metadata in the OMG CWM XMI format.

The next scenario will show how to import OMG CWM XMI metadata generated by DB2 Warehouse Manager into DB2 Cube Views.

The overall process of this metadata conversion is as follows:

1. Using DB2 Cube Views, create a cube model and save it in an XML file.
2. Using MIMB, convert this XML file into a CWM XMI file.

Each step of this process is described in the following paragraphs.

1) Using DB2 Cube Views, create a cube model and export it in XML

The DB2 Cube Views model used in this scenario is the one shown in Figure 9-9 on page 339.

This step has already been detailed in "1) Using DB2 Cube Views, export your cube model as an XML file" on page 24. The DB2 cube model is saved into an XML file using **OLAP Center > Export** in DB2 Cube Views.

2) Using MIMB, convert this XML file into a CWM XMI file

Start the MIMB software, select the import bridge labeled **IBM DB2 Cube Views** and import the cube model XML file as shown in Figure 9-70.

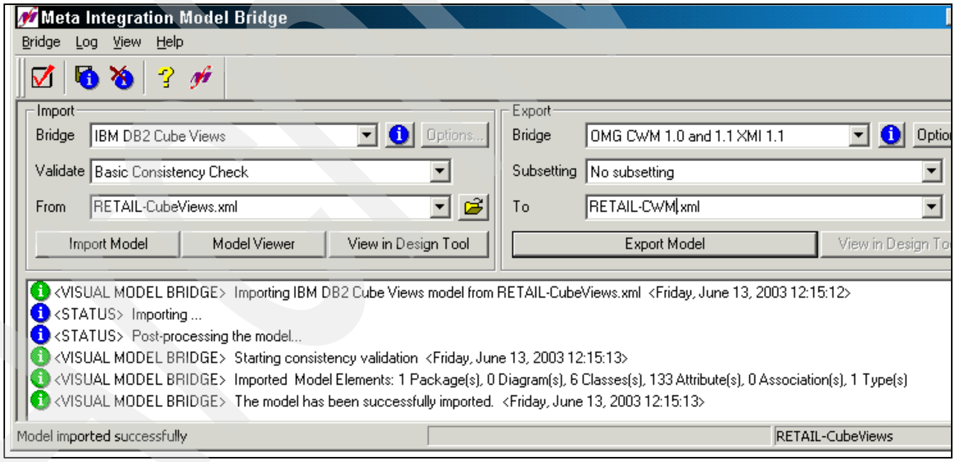


Figure 9-70 MIMB: Importing the cube model XML file

Select the export bridge labeled **OMG CWM 1.0 and 1.1 XMI 1.1** and type the name of the export file in the **To** field. Click the **Options** button to specify the export options.

The CWM export bridge has many parameters, which allow controlling how the CWM file should be created.

For example, in MIMB version 3.x, the export bridge **Model** option allows you to choose how the cube model's metadata should be mapped:

- As a relational model instance of the CWM:RDB package to represent the star-schema database information.
- As a UML software model instance of the CWM:ObjectModel package to allows software developers to import it in a UML-enabled design tool and develop their application taking advantage of all the business names and description defined in DB2 Cube Views.
- Or, both of these possibilities.

> **Note:** To follow up on enhancements for the export bridge with OLAP metadata, please check the following site regularly:
>
> http://www.metintegration.net/Products/MIMB

The model option is shown in Figure 9-71.

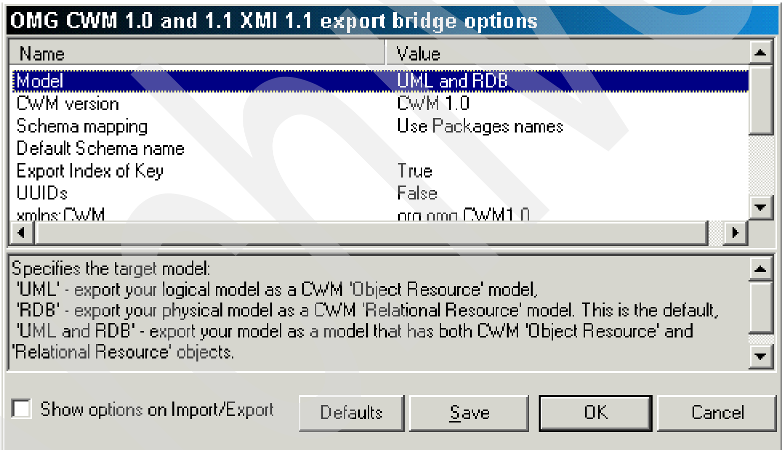


Figure 9-71 Specifying the export options: model

We also specify that the source encoding of the cube model XML file is utf-8 as shown in Figure 9-72.

Figure 9-72 Specifying the export options: encoding

Finally, click the **Export Model** button as shown in Figure 9-73 to create the CWM XMI file.

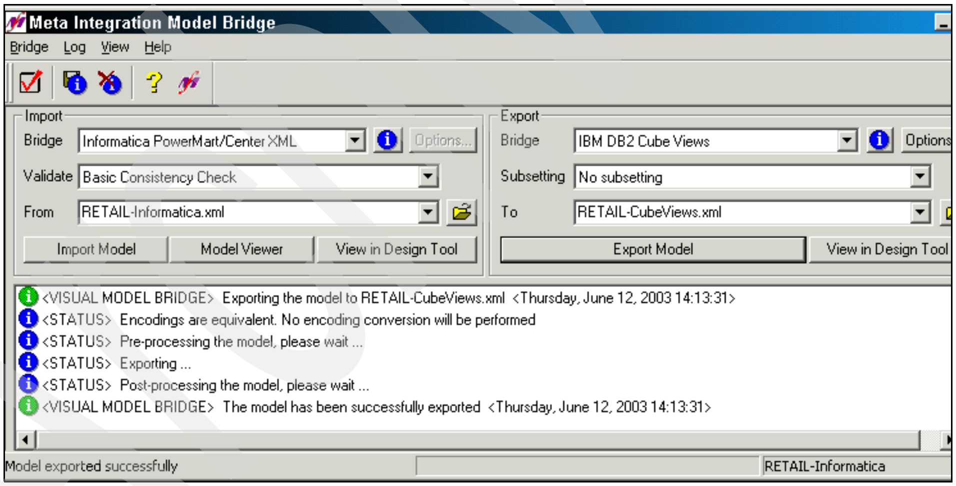


Figure 9-73 MIMB: exporting to CWM

Figure 9-74 is a sample of the exported CWM file.

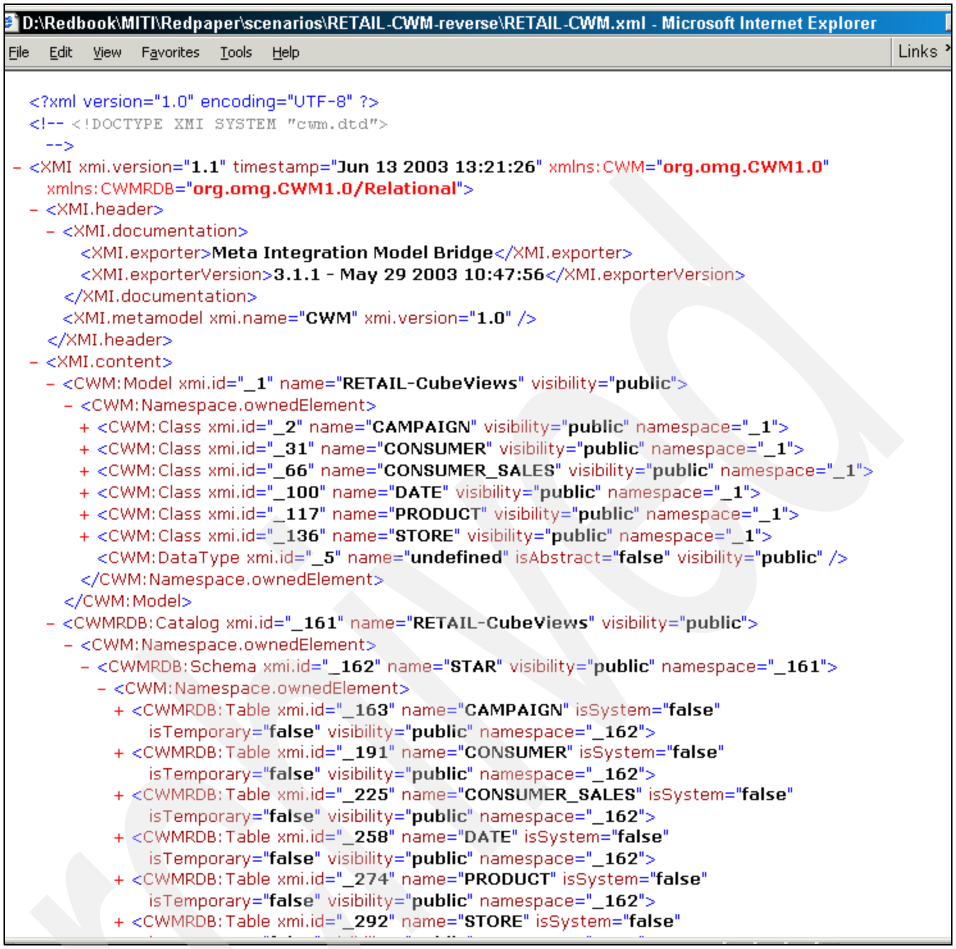


Figure 9-74 Sample CWM XMI file reverse engineered from a cube model

9.5.6 Metadata integration of DB2 Cube Views with DB2 Warehouse Manager

DB2 Warehouse Manager provides ETL and warehouse management functionalities to the DB2 platform. DB2 Warehouse Manager can be used to design a data warehouse or data mart, manage the different data sources populating it, and design the complex flow of data transformation between the source databases and the target data warehouse in a intuitive, GUI oriented way. The main user interface of this software is the DB2 Data Warehouse Center tool, and it supports the import and export of metadata via the OMG CWM XMI file format.

This scenario focuses on the exchange of metadata between DB2 Warehouse Center and DB2 Cube Views via the OMG CWM XMI file format. We will demonstrate how a datamart designed in DB2 Warehouse Center in the form of a star schema can be saved as a CWM XMI file, then converted to a DB2 Cube Views XML file using the MIMB utility, and finally, open it in DB2 Cube Views as a cube model.

The overall process of this metadata conversion is as follows:

1. Using DB2 Warehouse Center, create the star schema model.
2. Using DB2 Warehouse Center, save the model as a CWM XMI file.
3. Using MIMB, convert this CWM XMI file into a DB2 Cube Views XML file.
4. Using DB2 Cube Views, import this DB2 Cube Views XML file.

Each step of this process is described in the following paragraphs.

1) Using DB2 Data Warehouse Center, create the star schema model

This scenario uses the small "Beverage Company" star schema model shown in Figure 9-75.

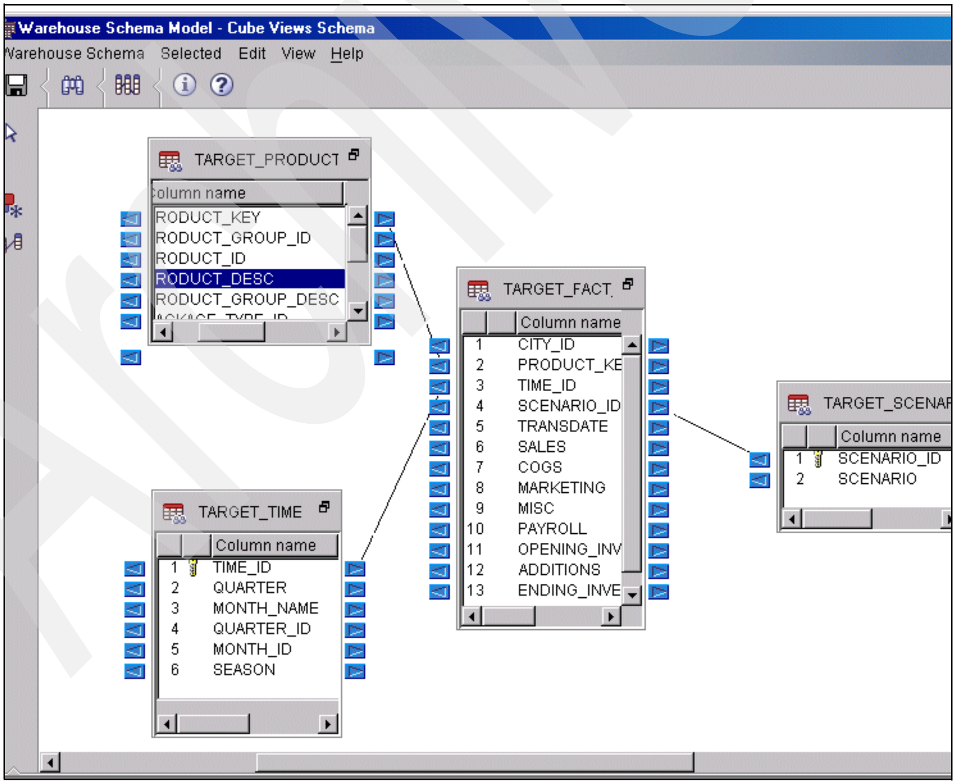


Figure 9-75 The Beverage company model in Data Warehouse Center

During the design of this model, a property has been set on each table to specify its role in the datamart (Fact or Dimension) as shown in Figure 9-76.

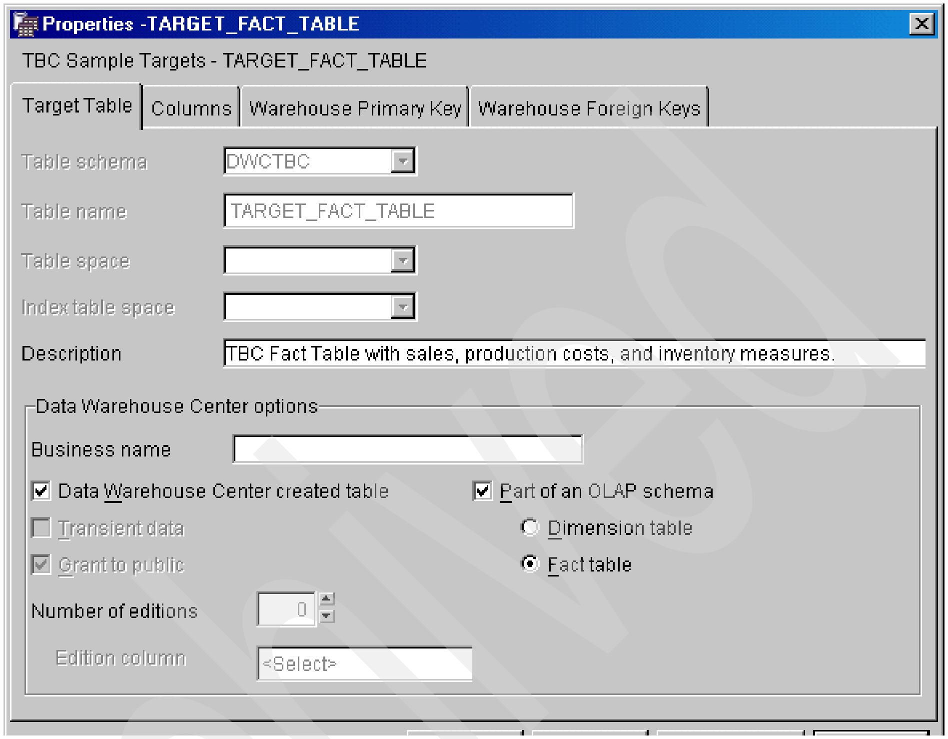


Figure 9-76 This is the fact table of the star schema

2) Using Data Warehouse Center, save the model as a CWM XMI file

From the DB2 Data Warehouse Center, export the metadata as shown in Figure 9-77.

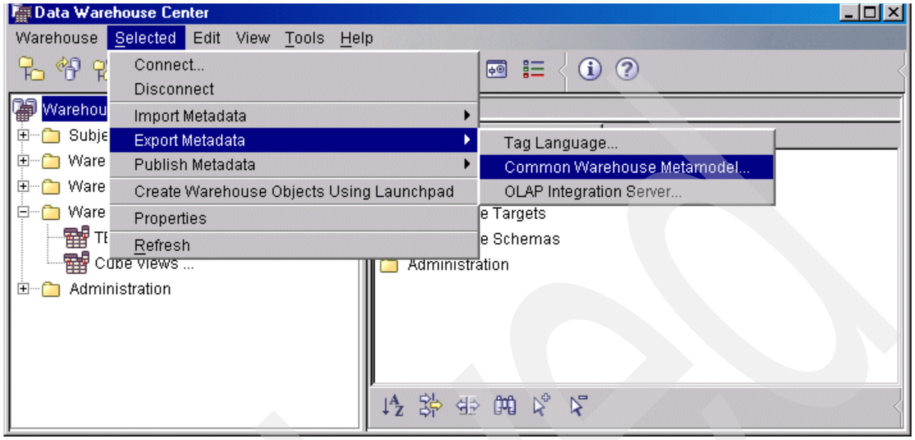


Figure 9-77 Starting the CWM export wizard from DB2 Data Warehouse Center

Then select the database to be exported as shown in Figure 9-78.

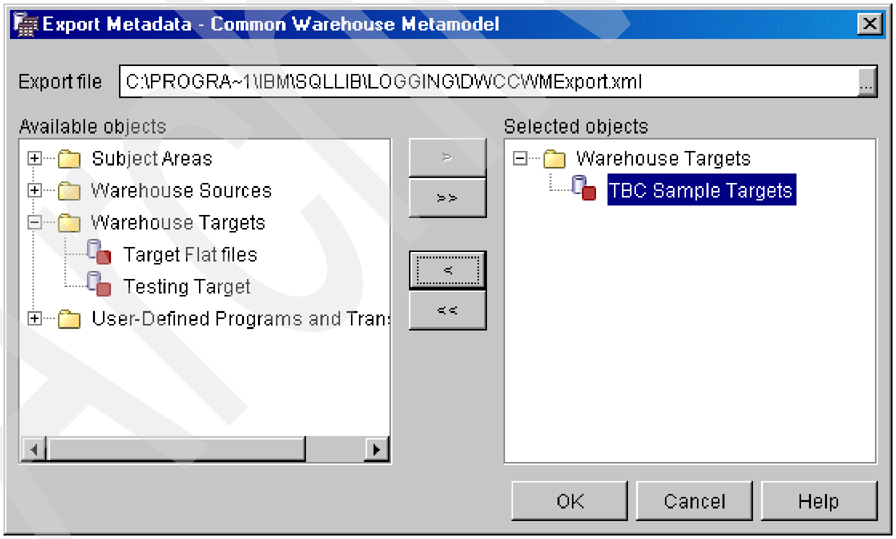


Figure 9-78 Selecting the database to be exported to CWM

Figure 9-79 is a sample of the exported CWM file.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <XMI xmi.version="1.1" timestamp="Wed Jun 11 12:22:17 MST 2003" xmlns:CWM="org.omg.CWM1.0"
    xmlns:CWMRDB="org.omg.CWM1.0/Relational">
  - <XMI.header>
    - <XMI.documentation>
        <XMI.exporter>XMI Application Framework</XMI.exporter>
        <XMI.exporterVersion>1.15</XMI.exporterVersion>
      </XMI.documentation>
    </XMI.header>
  - <XMI.content>
    + <CWM:SoftwareSystem xmi.id="_1" name="DWC">
    + <CWM:Class xmi.id="_2" name="Not available">
      <CWM:DataType xmi.id="_3" name="1" />
      <CWM:DataType xmi.id="_4" name="2" />
      <CWM:DataType xmi.id="_5" name="3" />
    - <CWMRDB:Catalog xmi.id="_6" name="SAMPWHS">
      + <CWM:ModelElement.taggedValue>
      - <CWM:Namespace.ownedElement>
        + <CWMRDB:Schema xmi.id="_6.10" name="ADMINISTRATOR" namespace="_6">
        + <CWMRDB:Schema xmi.id="_6.11" name="C360" namespace="_6">
        - <CWMRDB:Schema xmi.id="_6.12" name="DWCTBC" namespace="_6">
          - <CWM:Namespace.ownedElement>
            - <CWMRDB:Table xmi.id="_6.12.1" name="TARGET_FACT_TABLE" namespace="_6.12">
              + <CWM:ModelElement.taggedValue>
              - <CWM:Classifier.feature>
                + <CWMRDB:Column xmi.id="_6.12.1.16" name="CITY_ID" length="0" scale="0"
                    precision="0" isNullable="columnNoNulls" owner="_6.12.1" type="_25">
                + <CWMRDB:Column xmi.id="_6.12.1.17" name="PRODUCT_KEY" length="0"
                    scale="0" precision="0" isNullable="columnNoNulls" owner="_6.12.1"
                    type="_25">
                + <CWMRDB:Column xmi.id="_6.12.1.18" name="TIME_ID" length="0" scale="0"
                    precision="0" isNullable="columnNoNulls" owner="_6.12.1" type="_25">
                - <CWMRDB:Column xmi.id="_6.12.1.19" name="SCENARIO_ID" length="0"
```

Figure 9-79 The CWM XMI file rendered in a browser

3) Using MIMB, convert CWM XMI file into DB2 Cube Views XML file

At this point, start MIMB and select the **IBM DB2 Warehouse Manager** import bridge. This bridge is designed to understand the DB2 Warehouse Manager dialect of CWM. Then, import the CWM XMI file as shown in Figure 9-80.

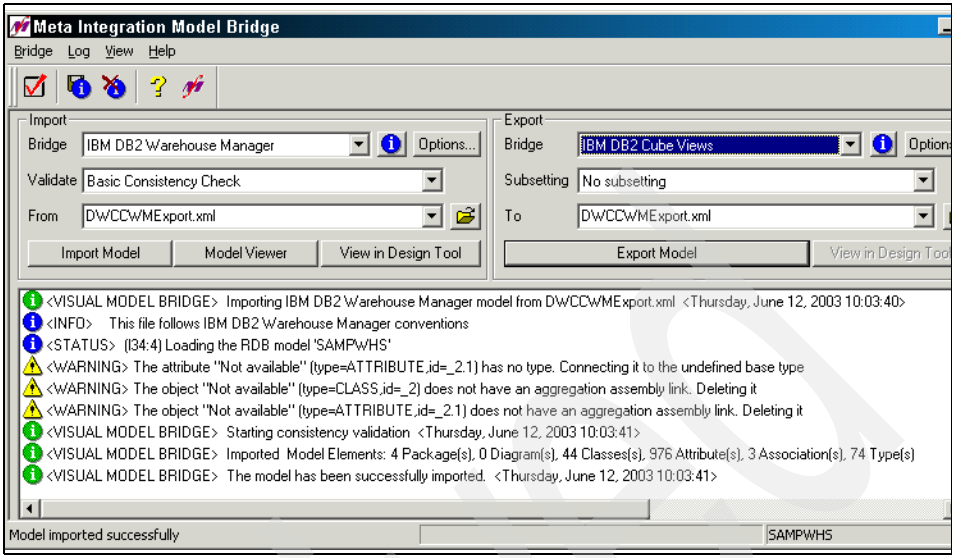


Figure 9-80 MIMB: importing the DB2 Data Warehouse Center CWM XMI file

Click the button labeled **Model Viewer** to review the imported metadata in Figure 9-81.

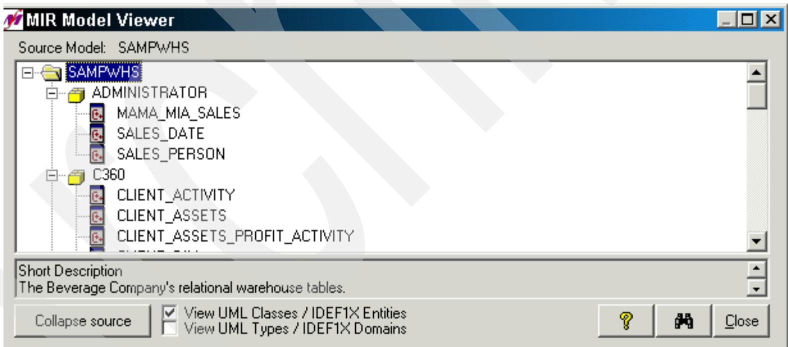


Figure 9-81 The sample warehouse Beverage Company imported from CWM

Select the export bridge labeled **IBM DB2 Cube Views** and click the **Options** button to specify the export parameters as shown in Figure 9-82.

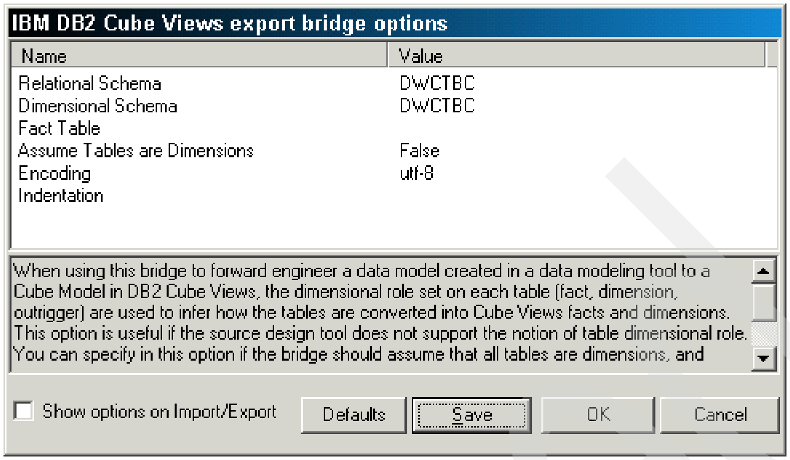


Figure 9-82 Specifying the export parameters

In this scenario, the star schema tables are located in a DB2 schema called *DWCTBC*, which we also use to store the OLAP metadata. We specify that the source encoding of the CWM file is utf-8. The other parameters are left to their default value.

We need to select the 4 tables to be exported as a cube model. As we have seen in the MIMB Model Viewer, there are many more tables in this warehouse, but we only need these 4 tables to be exported to the cube model.

Select the model subsetting option labeled **Specific Classes** as shown in Figure 9-83.

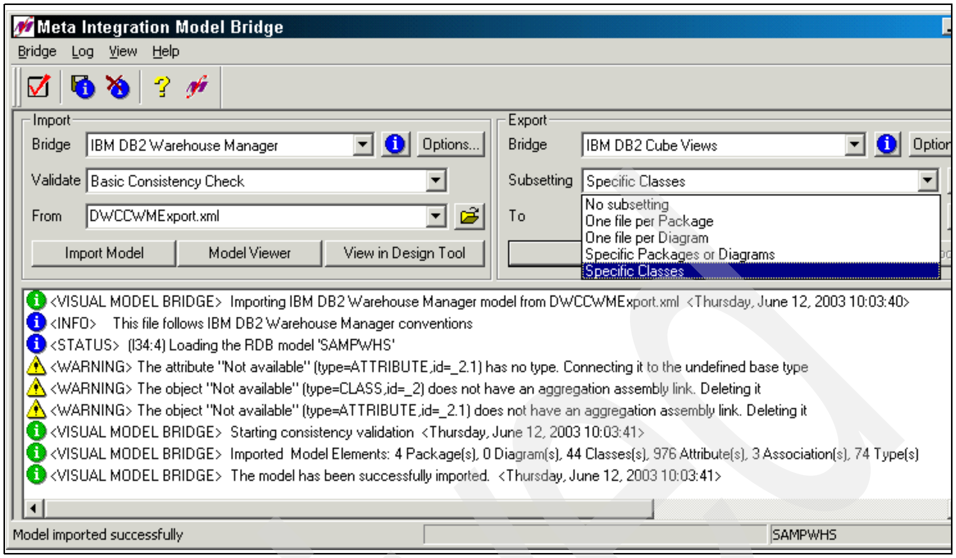


Figure 9-83   Choosing a subsetting mode

Drag and drop the 4 tables to be subsetted and click the button **Subset selected class(es)** as shown in Figure 9-84.

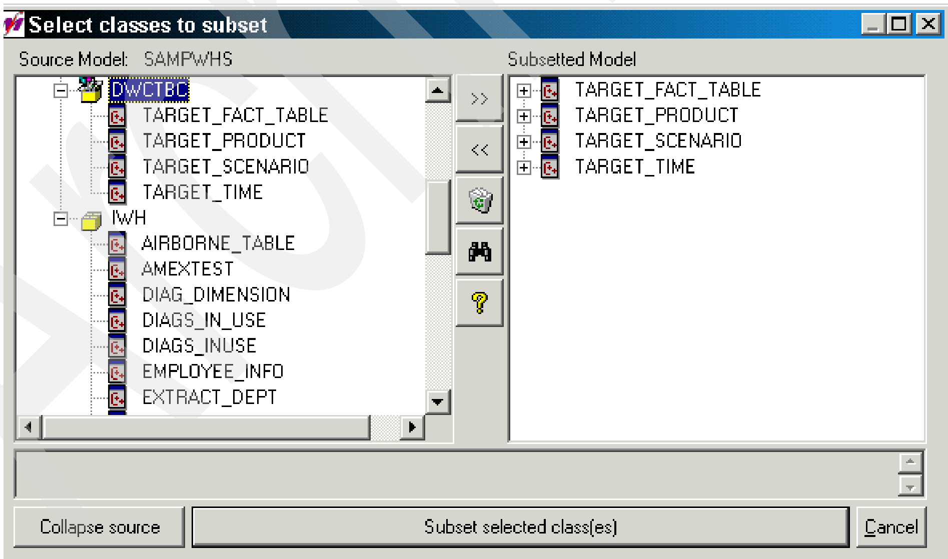


Figure 9-84   Subsetting the star schema model

Set the name of the final cube model XML file to be produced and click the button **Export elements**.as shown in Figure 9-85.

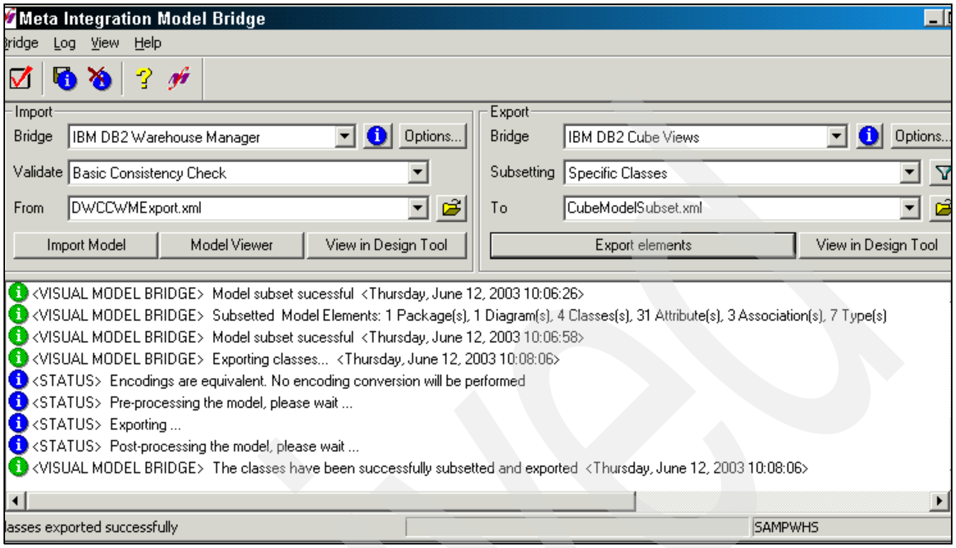


Figure 9-85 Exporting the cube model

The cube model has now been produced and is ready to be imported into DB2 Cube Views.

4) Using DB2 Cube Views, import this Cube Views XML file

Finally, the metadata is imported into OLAP Center as shown in Figure 9-86.

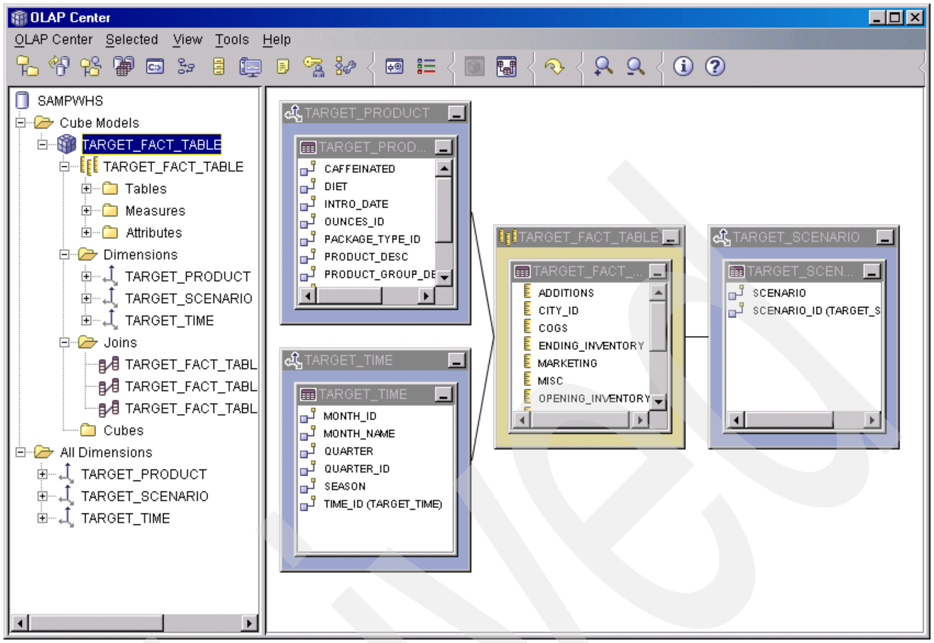


Figure 9-86 The Beverage Company star schema imported into DB2 Cube Views

Congratulations, you have imported into DB2 Cube Views a star schema designed in DB2 Warehouse Manager!

9.5.7 Metadata integration of DB2 Cube Views with Informatica

Informatica is one of the leading ETL tool vendors with tools such as PowerMart and PowerCenter that you can use to populate a DB2 data warehouse. The complex flow of data and transformations can be designed using PowerMart Designer. You can use Informatica PowerMart Designer to import and export of metadata via an XML file format.

This scenario demonstrates how to transform the metadata of a data warehouse designed in PowerMart 5.x and 6.x in the form of a star schema into a DB2 Cube Views cube model.

The overall process of this metadata conversion is as follows:

1. Using Informatica PowerCenter, save the metadata of the target data mart as XML.
2. Using MIMB, convert this Informatica XML file into a DB2 Cube Views XML file.

3. Using DB2 OLAP Center, import this DB2 Cube Views XML file.

Each step of this process is described in the following paragraphs.

1) Using PowerCenter, save target datamart metadata as XML

Using the Informatica PowerMart Designer tool, export definitions of target tables into an XML file as shown in Figure 9-87. Please refer to the Informatica documentation for details.


```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- This file was generated by Informatica XML Export Model Bridge from Meta Integration
Technology, Inc. (MITI)  -->
<POWERMART>
- <REPOSITORY NAME="Generated by MIMB">
  - <FOLDER NAME="Generated by MIMB">
    - <FOLDERVERSION NAME="Generated by MIMB">
      + <TARGET NAME="CAMPAIGN" BUSINESSNAME="CAMPAIGN" DESCRIPTION="A promotional
          CAMPAIGN is a limited period of time when some PRODUCTs are subjects to specific
          rebates. The promotional campaign is targetted to CONSUMERs via multiple
          communication channels and CONSUMERs purchasing PRODUCTs in our STOREs
          during this period can earn savings points." DATABASETYPE="DB2/UDB 6">
      + <TARGET NAME="CONSUMER" BUSINESSNAME="CONSUMER" DESCRIPTION="Holds the
          complete description of a CONSUMER purchasing PRODUCTs in our STOREs."
          DATABASETYPE="DB2/UDB 6">
      + <TARGET NAME="CONSUMER_SALES" BUSINESSNAME="CONSUMER_SALES"
          DESCRIPTION="This fact table holds the measures of the model representing the sales
          amount generated by CONSUMERs who have purchased PRODUCTs in STOREs during
          a promotional CAMPAIGN." DATABASETYPE="DB2/UDB 6">
      + <TARGET NAME="DATE" BUSINESSNAME="DATE" DESCRIPTION="This is the TIME dimension
          of the star schema" DATABASETYPE="DB2/UDB 6">
      + <TARGET NAME="PRODUCT" BUSINESSNAME="PRODUCT" DESCRIPTION="A PRODUCT can be
          purchased by a CONSUMER in a STORE at a particular DATE A PRODUCT can also be
          the subject of a time limited promotional CAMPAIGN" DATABASETYPE="DB2/UDB 6">
      + <TARGET NAME="STORE" BUSINESSNAME="STORE" DESCRIPTION="This STORE sells
          PRODUCTs to CONSUMERs" DATABASETYPE="DB2/UDB 6">
      </FOLDERVERSION>
    </FOLDER>
  </REPOSITORY>
</POWERMART>
```


Figure 9-87 The sample Informatica XML model

> **Note:** A copy of the Informatica software was not available during the writing of this chapter, so the XML file shown here was not directly generated by Informatica, but instead was forward engineered from an ERwin model to Informatica using the MIMB software. Nevertheless, the principles of this scenario are still relevant and the conversion process is the same.

2) Using MIMB, convert Informatica XML into DB2 Cube Views XML

Start the MIMB software, select the import bridge labeled **Informatica PowerMart/Center XML**, select the XML file to be imported, and click **Import Model** to import it as shown in Figure 9-88.

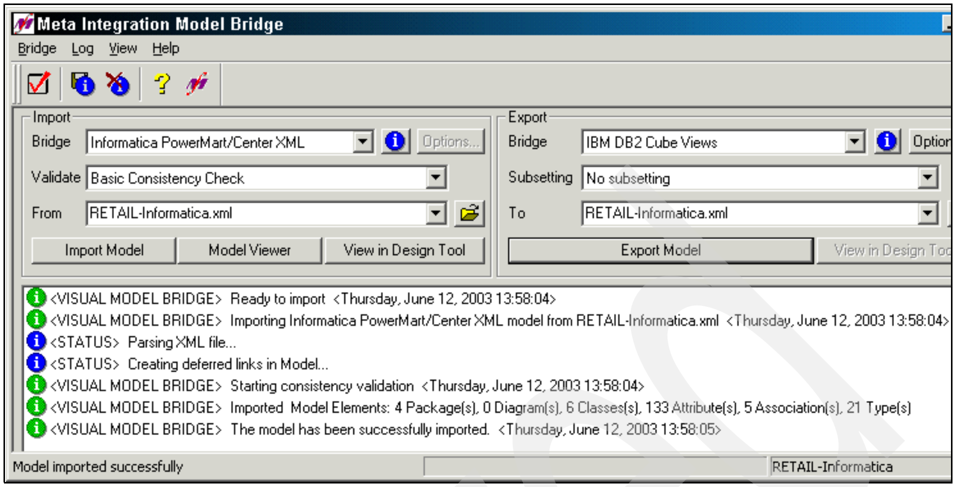


Figure 9-88 Importing the Informatica model

If the Informatica XML file contains the definition of tables that are not part of the target star schema, you can filter them out using the MIMB subsetting feature. Please refer to the MIMB documentation for details.

Then, select the export bridge labeled **IBM DB2 Cube Views** and press on the **Options** button to specify the export parameters as shown in Figure 9-89.

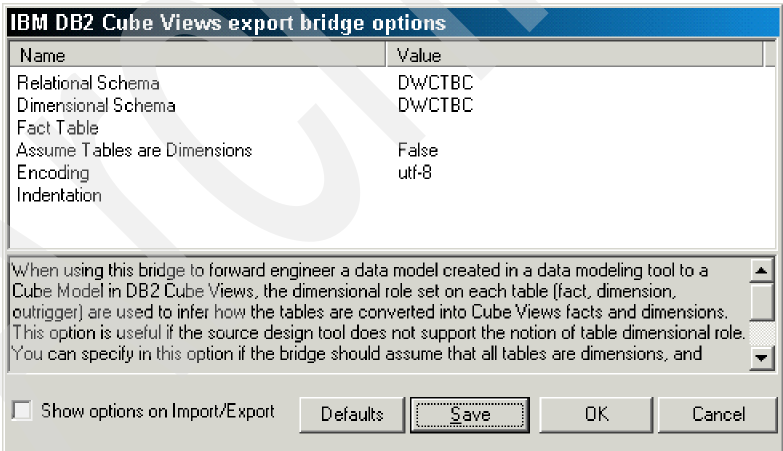


Figure 9-89 Specifying the export parameters

In this scenario, the tables are located in schema *STAR* and we will also create the OLAP objects in this schema. We have also indicated the name of the fact table and set the bridge to consider that the other tables are to be processed as dimensions.

> **Note:** The fact or dimension information on each table may not always be specified in the Informatica XML file. In this case, we can specify it this way.

We also specify that the source encoding of the Informatica model is utf-8.

At this point, you can export the model to the DB2 Cube Views XML file format as shown in Figure 9-90.

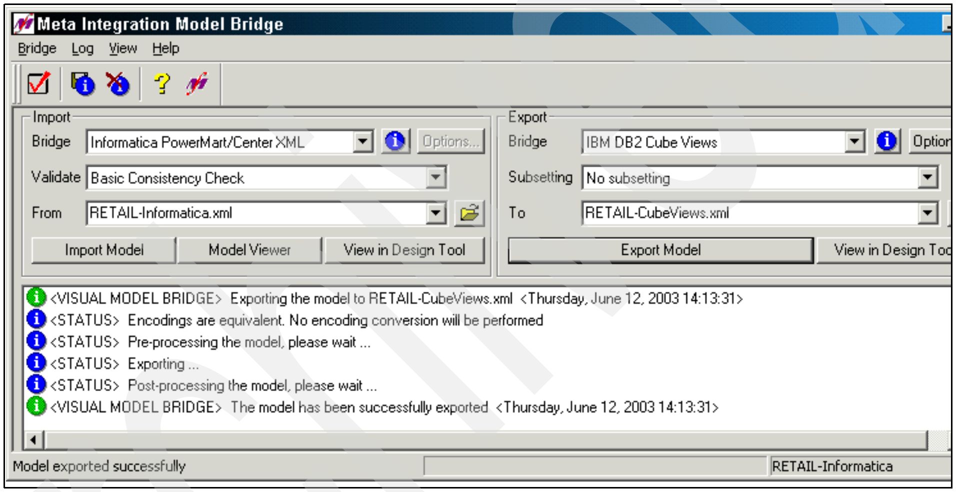


Figure 9-90 Exporting the model to DB2 Cube Views

3) Using DB2 Cube Views, import this DB2 Cube Views XML file

At this point, the cube model file is ready for importing. Select the **Import** item in the OLAP Center menu and use the wizard to import the file. You can see the result in Figure 9-91.

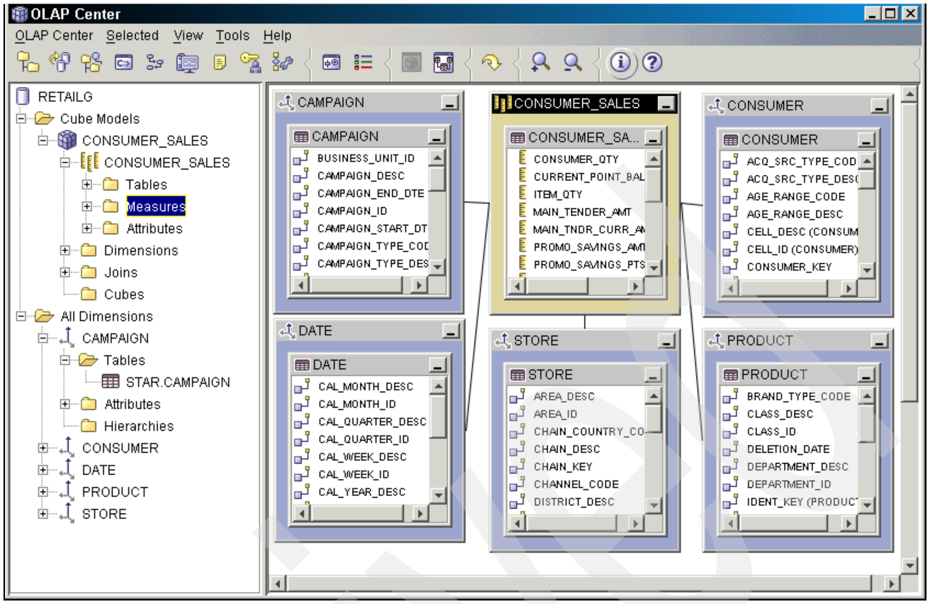


Figure 9-91 The cube model as imported in DB2 OLAP Center

Congratulations, you have imported into DB2 Cube Views a star schema designed in Informatica PowerMart/Center!

9.6 Refresh considerations

The metadata flows described above enable the forward engineering and reverse engineering of metadata between different tools (ERwin, PowerDesigner, Rose, DB2 Cube Views, DB2 Warehouse Center, PowerMart/Center) from different vendors (Computer Associates, Sybase, IBM, Informatica, OMG) using different metadata file formats (XML, ERX, PDM, MDL, CWM XMI) carrying metadata under different methodologies (Relational, OLAP).

The metadata conversion process between these tools, formats, and methodologies is complex and is sometimes more complicated in one direction than the other. For example, the definition of a relational foreign key can be used to create an OLAP join. However, when converting the other way, an OLAP join does not always represent a referential integrity constraint, and can be defined in the context of a specific business query.

Change in the enterprise is a reality and therefore, each of these tools have implemented metadata version and configuration management features to properly capture change and manage the versions of the enterprise metadata.

Most of these tools have also implemented their own metadata repository, which can store, compare and merge different versions of the metadata. For example, two versions of an ERwin model can be stored in the ModelMart repository and they can be compared and merged using the ERwin ‘Complete Compare’ feature; two versions of a PowerDesigner model can be stored in the PowerDesigner Repository and they can be compared using the ‘Compare Models’ feature.

Whether change occurs first in the database, or in a tool managing the database, and whether it is a small incremental update of a dramatic new version, it needs to be propagated to the other tools in the enterprise, and these tools also need to understand what has changed and how to handle this new version of the metadata.

The Meta Integration Model Bridge utility can extract the new version of the metadata from the source tool where the change happened, transform this metadata using sophisticated forward engineering and reverse engineering algorithms across vendors tools, formats, and methodologies, and publish the new version of the metadata into the destination tool.

To analyze the new version of the metadata in the destination tool and compare it to the current version of the metadata that may be already in place, it is recommended to use the version management features such as metadata comparator and metadata integrator in the destination tool, such as the ones implemented in most design tools, ETL tools and their underlying metadata repositories.

In case of DB2 Cube Views as a destination of a metadata flow, the version and configuration management features are available in the XML import wizard. They can be used to control how the current version of the metadata stored in the DB2 catalog can be replaced by the new version of the metadata in the XML file.

Using a third party metadata management suite such as a metadata repository equipped with advanced metadata versions comparison and integration tools could provide additional and complementary features in this regard.

For example, the Meta Integration Repository server (MIR) and Works client (MIW) suite of software is fully equipped for metadata version and configuration management, with a metadata repository manager, metadata comparator, metadata integrator, metadata mapper, in addition to all the metadata bridges also available in the Meta Integration Model Bridge (MIMB) utility (more than 40 of them as of summer 2003).

9.7 Conclusion: benefits

DB2 Cube Views simplifies and expedites business data analysis by presenting relational information as multidimensional objects.

To take advantage of these new features, you need to define this multidimensional metadata layer and configure these objects with business names, descriptions and more. You also need to understand the structure of the underlying database, such as the definition of the tables, their dimensionality, and the integrity relationships between them. The database stores a limited amount of this information in a very laconic and cryptic form. However, this structure and business information may be already defined in an ETL tool or a design tool in your company, and being able to share it with DB2 Cube Views would make the job of understanding and creating the multi-dimensional objects much easier.

MIMB does exactly this plus more. MIMB allows you to bring design and modeling information into DB2 Cube Views, and automates the creation process of the Cube Model and its related dimensions.

When you are done with your DB2 Cube Views design, you can also use MIMB to exchange your multidimensional model with your BI and reporting tools.

MIMB allows you to reuse the multidimensional objects you've created in DB2 Cube Views and populate this metadata in your BI and reporting tools.

Because understanding business data starts with a good understanding of the enterprise metadata, MIMB plays a key role in the metadata integration of DB2 Cube Views in the complete enterprise toolset. With MIMB, your design tools, ETL tools and BI tools are now compatible with each other and can exchange metadata with DB2 Cube Views.

10

Accessing DB2 dimensional data using Integration Server Bridge

This chapter describes certain deployment scenarios for using the OLAP Integration Server metadata bridge. It explains how to implement it for each metadata flow scenario and how and when DB2 OLAP Server benefits from DB2 Cube Views summary tables optimization.

10.1 DB2 OLAP Server and Integration Server bridge

IBM DB2 OLAP Server is a powerful OLAP engine that can be used to build analytic applications for fast, intuitive multidimensional analysis to fulfill a broad spectrum of business performance management needs. DB2 OLAP Server delivers more than 250 powerful calculation functions out of the box, and runs on each of IBM's @server™ platforms. Multi-user write back enables companies to model the effects of business variables on performance and hence develop plans more effectively.

DB2 UDB V8.1 is a close partner product to DB2 OLAP Server, and together these database products provide high level functionality and performance in order to enable business managers to analyze and more effectively manage business performance.

Figure 10-1 illustrates the DB2 OLAP Server functions in a relational database environment. At the bottom of the figure three types of database cubes are shown. On the right hand side there is the MOLAP cube where all data that is queried by the user is stored in the DB2 OLAP Server MOLAP file structure. In the middle there is again the MOLAP database, but in addition there is the ability for the user to drill through to the underlying relational database using Integration Server drill through. This relational database is represented by the box at the top left of the figure and is shown as a star schema model. Finally, on the left hand side, there is the hybrid database whereby higher levels of the hierarchies are held in the MOLAP database and lower levels are held in the relational database.

One of the components of DB2 OLAP Server is Integration Server as depicted in the top right hand box, and this may be used to:

- Define the OLAP database structure from the relational database. This may be for a MOLAP structure or for a HOLAP structure.
- Perform the data load in order to load data from the relational database to the OLAP database.
- Define the Integration Server drill through reports that are to be made available.

The metadata for Integration Server is also stored in a relational database, and this is represented by the box at the top left of the figure and labelled the IS metadata catalog.

Many end user tools are available to query the data in DB2 OLAP Server.

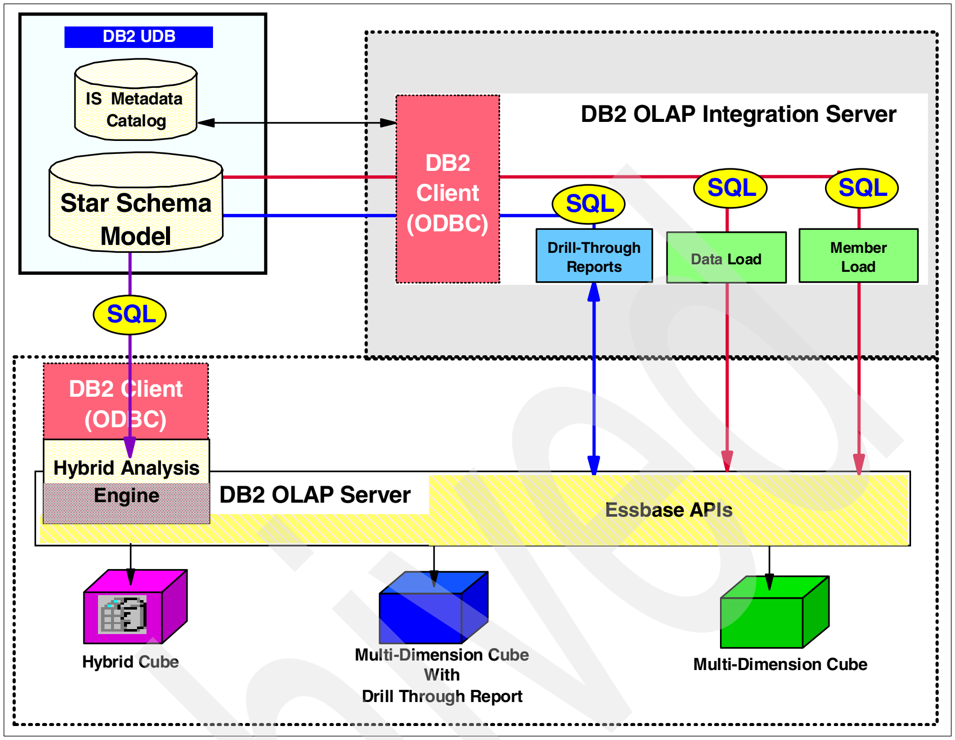


Figure 10-1   OLAP Server architecture

10.1.1 Integration Server

Integration Server provides the metadata layer for DB2 OLAP Server. It has a graphical user interface which can be used to describe the source relational database. This metadata is known as a model. Integration Server also describes the physical outline of the target DB2 OLAP Server database, this is known as a metaoutline. Given the understanding of the source relational database and the target multidimensional database, Integration Server is then able to extract the relevant column values from the relational data source in order to build the physical outline in DB2 OLAP Server. Having done that, it is then able to perform the data load from the relational data source.

In order to perform these tasks, the metadata in the Integration Server model has to map relational tables to dimensions in DB2 OLAP Server. It also has to describe how the hierarchies can be built from the relational data, and any transformations that are to take place. The metaoutline needs to specify the sequence of the dimensions that are required in the physical outline and select the hierarchies that are to be built from those defined the in model. Measures can be defined directly from the relational database, or complex measures can be built using the many function templates available in DB2 OLAP Server. The metaoutline is used to build the physical outline, and as such there is additional metadata than can be specified in the metaoutline that is specific to OLAP Server — for example, specifying whether the dimension is dense or sparse.

Figure 10-2 illustrates this process. Integration Server holds metadata that describes both the source relational database and the target DB2 OLAP Server database. With this information, Integration Server can generate and run the SQL that is required to both build the target outline, and perform the data load for the target DB2 OLAP Server database.

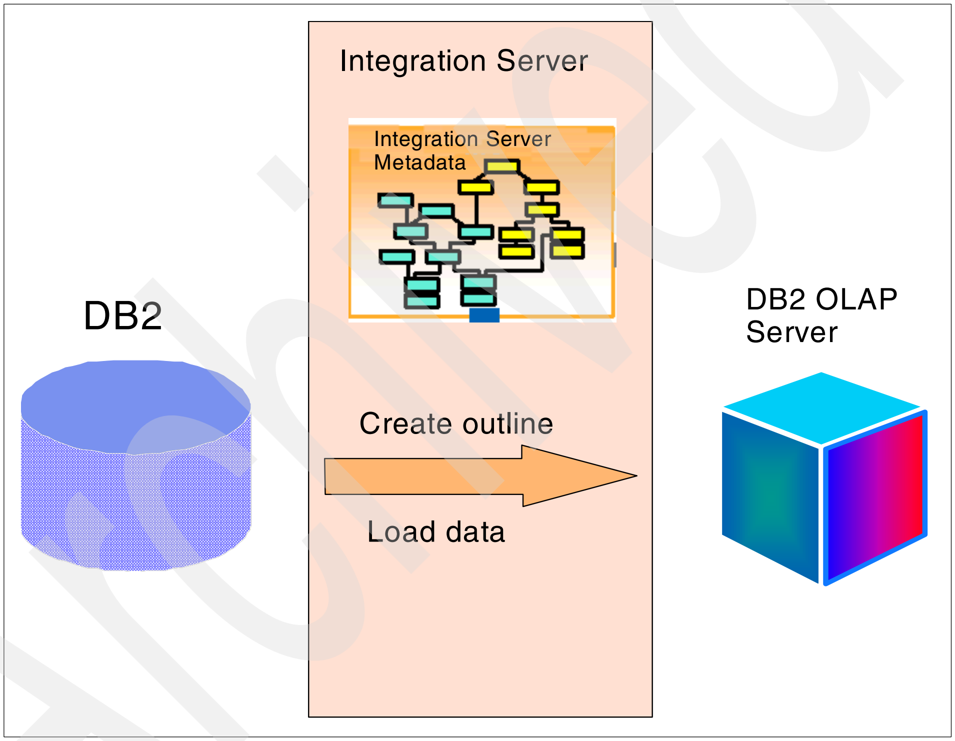


Figure 10-2 Integration Server

10.1.2 Hybrid Analysis

Hybrid Analysis enables a Hybrid OLAP environment for DB2 OLAP Server, whereby higher levels of the database are stored in the DB2 OLAP Server multidimensional storage structure, and lower levels of the database are stored in the relational database. The user is not aware that they are using two different file systems. The drill down from the multidimensional storage structure to the relational structure is seamless. They may, however, be aware of a change in performance levels as they move from simply fetching pre-computed blocks of data to issuing SQL queries across multiple tables, and this is an area which DB2 Cube Views can help, by providing MQTs to improve the performance of these queries.

Integration Server is required in order to implement Hybrid Analysis. The Integration Server metadata describes the hierarchy for a dimension and describes the cut-off point between what is in the multidimensional storage structure and what is in relational.

There are other features of DB2 OLAP Server available such as Enterprise Services, Administration Services, Spreadsheet Services and OLAP Mining. For more information on DB2 OLAP Server please refer to *DB2 OLAP Server V8.1: Using Advanced Functions*, SG24-6599 and go to the following Web site:

http://www-3.ibm.com/software/data/db2/db2olap/

10.1.3 Integration Server Bridge

There are elements of metadata that are common to both DB2 Cube Views and Integration Server. For example, DB2 Cube Views has a cube model and a cube, and Integration Server has a model and a metaoutline. Both have dimensions and hierarchies and measures. With this level of commonality between the two products, it is very useful to be able to export this common metadata from one product to the other product.

The Integration Server Bridge is a two-way bridge, meaning that metadata can be sent from DB2 Cube Views to Integration Server and also from Integration Server to DB2 Cube Views. However, you must always bear in mind that DB2 Cube Views metadata is designed for OLAP in general, whereas the Integration Server metadata is specific to DB2 OLAP Server. Therefore, there will be elements in both products that will not be able to be mapped when sent across to the other product.

This means that some metadata will be lost no matter which direction the metadata flows to or from. It is therefore not recommended that the bridge be used for *round-tripping*.

Table 10-1 shows the mapping that takes place between DB2 Cube Views and Integration Server

Table 10-1 Object mapping between DB2 Cube Views and Integration Server

Integration Server object	DB2 Cube Views object
Model	Cube Model
Fact	Facts
Numeric member of fact	Measure
Dimension	Dimension
Member	Attribute
Hierarchy	Hierarchy
Join	Join
Metaoutline	Cube
Non accounts dimension	Cube dimension
Accounts dimension	Cube facts
Metaoutline dimension hierarchy	Cube hierarchy

The Integration Server Bridge reads from and writes to XML files, and runs on the Windows platform only.

10.2 Metadata flow scenarios

The flow of metadata is either going to be from DB2 Cube Views to Integration Server or from Integration Server to DB2 Cube Views. If the starting point for implementation is a situation where neither DB2 Cube Views nor Integration Server are installed, then this is probably a straightforward scenario. Sometimes, however, one or more of the products may be well established, and the implementation of DB2 Cube Views needs to take into account existing factors.

This section suggests some of the issues to consider in the following scenarios:

- DB2 OLAP Server and DB2 Cube Views not installed
- DB2 OLAP Server with Integration Server installed, but DB2 Cube Views not installed
- OLAP Server installed, but Integration Server and DB2 Cube Views not installed
- DB2 Cube Views installed, but OLAP Server not installed

10.2.1 DB2 OLAP Server and DB2 Cube Views not installed

In this scenario we are starting from scratch with neither DB2 OLAP Server nor DB2 Cube Views installed. It is assumed that DB2 UDB V8.1 is already installed and that a multidimensional database has been designed and implemented using DB2.

In this scenario the following installation steps need to be performed:

- Install DB2 Cube Views
- Install DB2 OLAP Server including Integration Server
- Install the Integration Server bridge

At this point there is no DB2 Cube Views metadata describing the multidimensional DB2 database, and there is no metadata in Integration Server. As the Integration Server bridge is a two-way bridge, then the choice here is to either create the metadata in DB2 Cube Views and then export across the bridge to Integration Server, or to create the metadata in Integration Server and then export across the bridge to DB2 Cube Views.

The metadata in DB2 Cube Views is generic to OLAP, whereas the Integration Server metadata is specific to DB2 OLAP Server. Also, DB2 Cube Views requires unique names within object type across the whole model whereas Integration Server allows duplicate names in different contexts. For example, a hierarchy in one dimension can have the same name as a hierarchy in another dimension. DB2 Cube Views would not allow this. In general, therefore, the process flow may be to create the metadata in DB2 Cube Views and then export across the bridge to Integration Server. Once in Integration Server it can be further enhanced for DB2 OLAP Server specific functionality.

Furthermore, the metadata flow may need to take into account additional products other than just DB2 OLAP Server. The metadata flow may start with a *push* into DB2 Cube Views (across a bridge) from a data modelling tool or an ETL tool, for example. In this case the natural flow between DB2 Cube Views and Integration Server would again be from DB2 Cube Views to Integration Server.

Figure 10-3 provides an illustration of the scenario for metadata flow from DB2 Cube Views to Integration Server:

1. Create the metadata in DB2 Cube Views by any of the methods available.
2. Export the metadata to an XML file.
3. Process the XML file through the Integration Server Bridge.
4. Import the XML files that are produced into Integration Server.

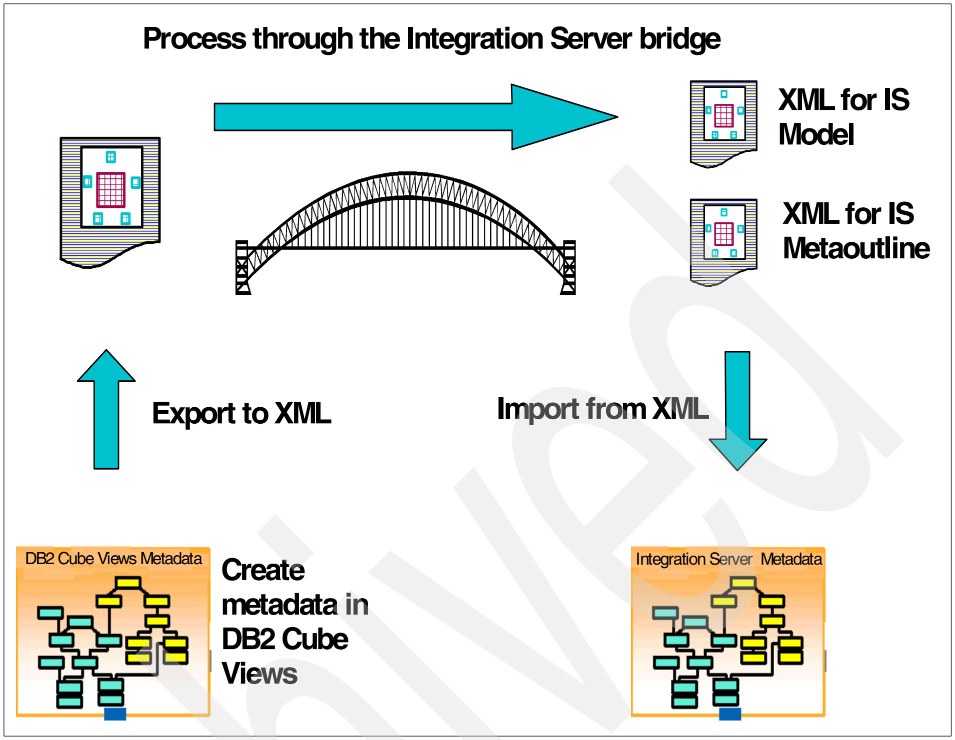


Figure 10-3 Metadata flow through the Integration Server bridge

The result of having imported the XML files that have been generated by the bridge into Integration Server will depend on whether the cube model, or the cube model and the cube, were exported originally from DB2 Cube Views. A cube model in DB2 Cube Views maps to an Integration Server model. A cube in Integration Server maps to a metaoutline in Integration Server. Figure 10-3 shows an input XML file with information from both a cube model and a cube being processed by the bridge to generate two XML files: one for the Integration Server model and one for the Integration Server metaoutline.

The metadata in DB2 Cube Views is generic to OLAP and not specific to Integration Server. Some metadata objects in DB2 Cube Views, such as aggregation scripts, have no equivalent in Integration Server, and for these objects it will not be possible to flow the metadata from DB2 Cube Views to Integration Server (IS).

10.2.2 DB2 OLAP Server and IS installed, but not DB2 Cube Views

In this scenario we have an existing installation of DB2 OLAP Server, including Integration Server, into which we are introducing DB2 Cube Views. It is assumed that DB2 UDB V8.1 is already installed and that a DB2 multidimensional database is being used to load data into the DB2 OLAP Server database via Integration Server.

In this case the only software product that needs to be installed is DB2 Cube Views.

Metadata will already exist in Integration Server to describe both the multidimensional databases that are in place and the OLAP Server applications and databases that have been created.

The task here, therefore, is to export the metadata from Integration Server to DB2 Cube Views in the reverse direction across the bridge, as can be shown in Figure 10-4. The Integration Server model and metaoutline are exported separately from Integration Server, and then the bridge combines these two XML files into a single XML file that can then be imported into DB2 Cube Views.

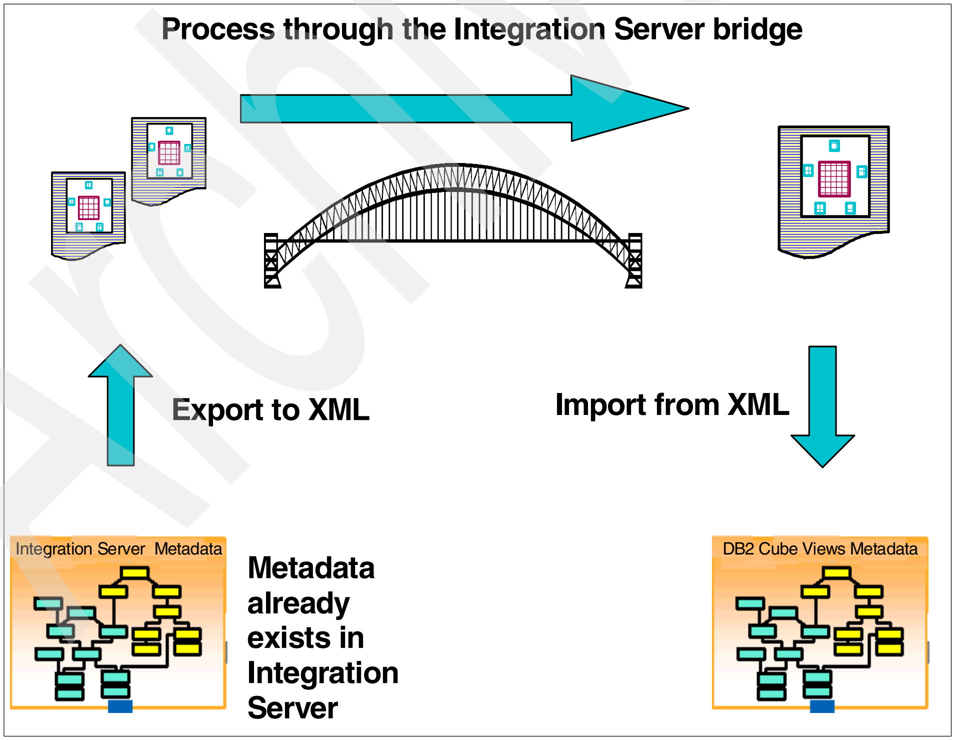


Figure 10-4 Reverse metadata flow through the Integration Server bridge

The metadata in Integration Server is specifically created in order to generate DB2 OLAP Server databases, the functionality of which cannot be totally replicated within DB2 UDB V8.1. Therefore, it may not be possible to flow the metadata for the more complex objects from Integration Server to DB2 Cube Views.

10.2.3 DB2 OLAP Server installed, but not IS and DB2 Cube Views

In this scenario we have an existing installation of DB2 OLAP Server, but until now Integration Server has not been implemented. Here we are introducing both Integration Server and DB2 Cube Views. It is assumed that DB2 UDB V8.1 is already installed and that a DB2 multidimensional database was already being used to load data into the DB2 OLAP Server database via data load rules files.

Here we have DB2 OLAP Server applications and databases, but no metadata to describe the relational database sources and the dimensions and hierarchies within those data sources.

It is not a straightforward process to reverse engineer DB2 OLAP Server metadata from existing DB2 OLAP Server databases. In this scenario the value of generating the metadata needs to be evaluated against the effort involved in generating this metadata.

It may well be that for some DB2 OLAP Server databases, the effort involved in generating the metadata outweighs the benefits, and a decision is taken not to generate metadata for those databases. For example, if those databases perform well and there are no plans to introduce Hybrid Analysis into those databases, or if those databases have no data load performance issues. Furthermore, if those databases are loaded from non-DB2 data sources, then metadata exchange with DB2 Cube Views will not be appropriate.

However, if the performance of data load or Hybrid Analysis needs to be improved then the effort may well be justified. Moreover if there are other tools in the environment with which to exchange OLAP metadata then this will further justify reverse engineering the metadata.

Metadata can either be created in DB2 Cube Views and put through the Integration Server bridge into Integration Server, or it can be created in Integration Server and the flow can then be from Integration Server, across the bridge into DB2 Cube Views. For the same reasons that were discussed in the initial scenario where neither DB2 OLAP Server nor DB2 Cube Views were installed, the flow of metadata may well be from DB2 Cube Views to Integration Server.

Once in Integration Server, a test DB2 OLAP Server application and database can be generated as a first pass attempt at trying to reproduce the production application/database. The metadata in Integration Server will then probably need to be enhanced in a series of reiterative steps until an exact duplicate of the production DB2 OLAP Server database has been generated in test. Once satisfied that the Integration Server metadata is an exact match for the DB2 OLAP Server database, then the live link to the production database can be established.

10.2.4 DB2 Cube Views installed, but not DB2 OLAP Server

In this scenario we have an existing installation of DB2 Cube Views and we are introducing DB2 OLAP Server, including Integration Server. The DB2 Cube Views implementation has created metadata in the form of one or more cube models, each having one or more cubes, to describe a DB2 multidimensional database.

This is a straightforward scenario and the metadata process flow illustrated in Figure 10-3 on page 424 can be followed. That means exporting the DB2 Cube Views metadata to XML, processing the XML through the Integration Server bridge, and finally importing the resulting XML file into Integration Server. The Integration Server metadata can then be further enhanced to take advantage of the DB2 OLAP Server functionality.

10.3 Implementation steps

The Integration Server bridge is going to be used to either move metadata from DB2 Cube Views to Integration Server or from Integration Server to DB2 Cube Views. This section discusses the use of the bridge in both of these directions.

One suggestion that is worth thinking about before starting with the Integration Server bridge has to do with organizing the output from each process. Use of the Integration Server bridge is going to generate a number of XML files. It is recommended that a naming convention be adopted to assist in identifying both the content of each XML file, and the process that generated each XML file. For example, was the XML file generated as a result of an export from DB2 Cube Views or as the result of a bridge process? If it was produced by the bridge, was the bridge being used to process the XML from DB2 Cube Views or from Integration Server? Having separate folders for each process may assist with the task of identifying both the content of an XML file and the process that was used to create it.

10.3.1 Metadata flow from DB2 Cube Views to Integration Server

The tasks that need to take place in order for metadata to flow from DB2 Cube Views to Integration Server are as follows:

1. Export the metadata from OLAP Center to an XML file.
2. Process the XML file through the Integration Server bridge to produce one or two output XML files.
3. Import the output XML file(s) into Integration Server.

Export from OLAP Center

The first step that is required is to export the metadata to an XML file. This can be done from within OLAP Center. From the menu bar, click **OLAP Center->Export** and you will be presented with the **Export Metadata Objects to XML File** window as shown in Figure 10-5.

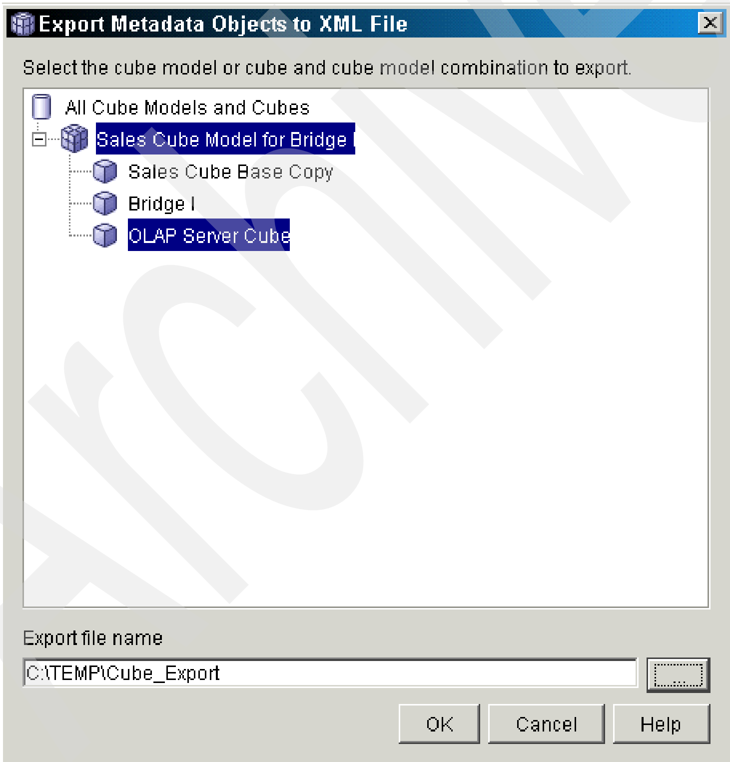


Figure 10-5 Export from DB2 Cube Views

In this window you are asked to select either the cube model or the cube that you wish to export. If you click a cube model, then only the cube model will be exported. In Integration Server terms, the cube model in DB2 Cube Views maps to a model in Integration Server.

If you click a cube, then the cube model that is associated with that cube will also be selected. A cube in DB2 Cube Views will map to a metaoutline in Integration Server. It is not possible to select a cube without a cube model because a cube model is required before a cube can exist. Moreover, a metaoutline without a model in Integration Server is not valid. So by selecting to export a cube in DB2 Cube Views, the result in Integration Server will be both a model and a metaoutline.

In DB2 Cube Views V8.1 FP2+, it is not possible to select more than one cube for export from within OLAP Center. Therefore if you have more than one cube in a cube model that you wish to export to Integration Server, you need to perform the export, bridge, import process multiple times.

Having selected the cube model or cube that you wish to migrate, you must then enter the full path name of the XML file that you wish to create. There is a browse button available to assist in selecting the appropriate drive and folder.

Finally, when you have entered the export file name, click the **OK** button. You should then receive an OLAP Center message informing you that your export has been successful.

Process the XML across the Integration Server bridge

The second step that is required is to process the XML file through the Integration Server bridge to produce one or two output XML files. If you exported the cube model only you will get one output XML file. If you exported the cube model and the cube you will get two XML files: one for the Integration Server model and one for the Integration Server metaoutline.

To launch the bridge, use the command line and run the ISBridge.bat file directly. By default the ISBridge.bat file is located in the SQLLIB\bin directory.

When launching the bridge, you will be presented with the **IS Bridge** window as shown in Figure 10-6.

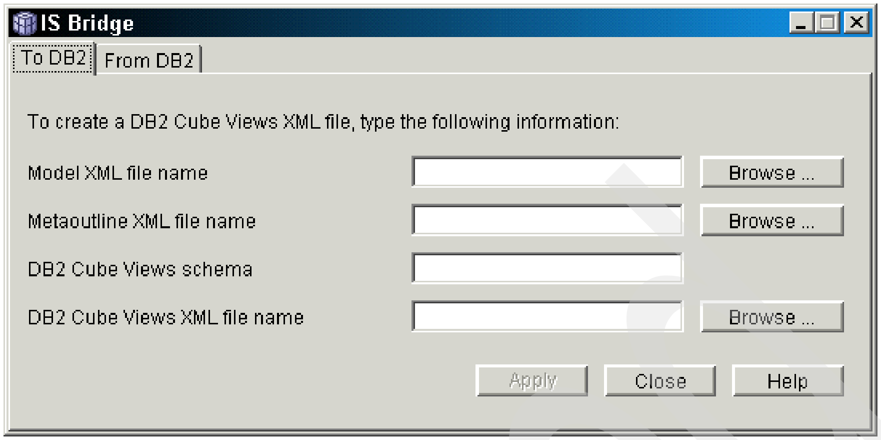


Figure 10-6 Integration Server Bridge window

The Integration Server bridge window contains two tabs. The **To DB2** tab is used if processing metadata from Integration Server to DB2 Cube Views. The **From DB2** tab is used if processing metadata from DB2 Cube Views to Integration Server. As this section is concerned with going from DB2 Cube Views to Integration Server, then at this point, you should select the **From DB2** tab and you will be presented with the window shown in Figure 10-7.

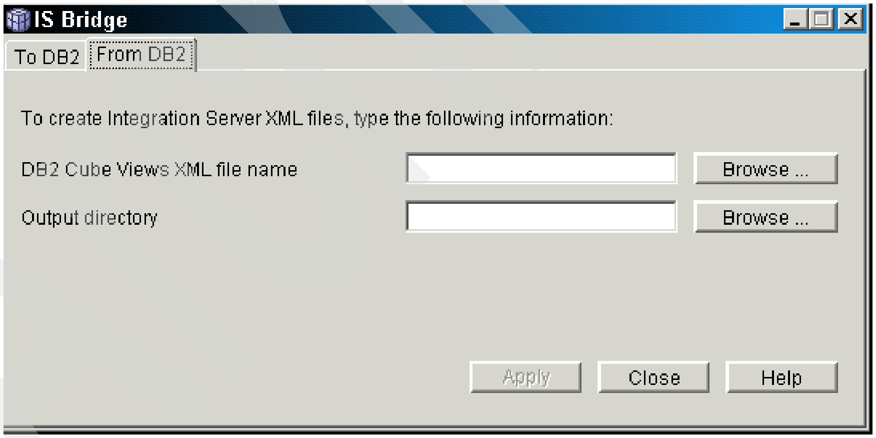


Figure 10-7 The IS bridge from DB2 Cube Views to Integration Server

In the **DB2 Cube Views XML file name** field, enter the full path name of the XML file that was created as the result of the export from DB2 Cube Views. There is a Browse button available to assist in finding the correct file and folder. In the **Output directory** field enter the full path name of the directory in which you want the bridge to create the output XML file(s). If the directory does not exist, the bridge will create it.

An example is shown in Figure 10-8.

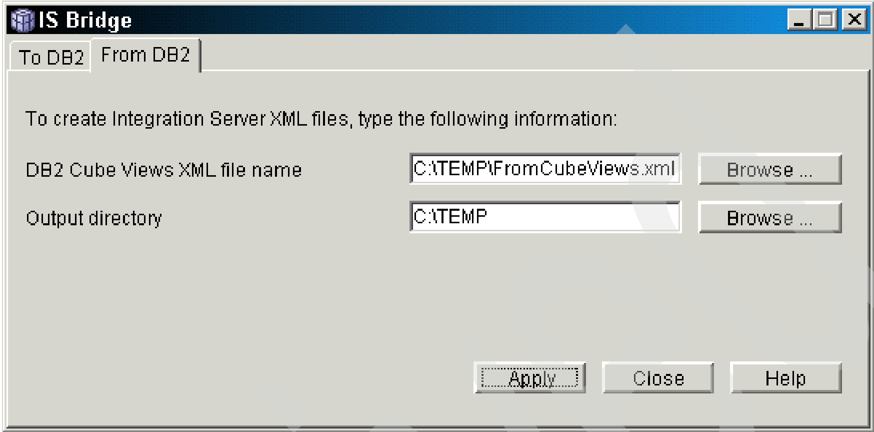


Figure 10-8 Use of bridge from DB2 Cube Views to Integration Server

When you click the **Apply** button, the bridge will process the input XML file and generate the output XML file(s) in the target directory specified. You should receive a successful completion message that will also inform you of the name of the output XML file which gets created in order to generate the Integration Server model, and the name of the output XML file which gets created in order to generate the Integration Server metaoutline (if applicable).

The bridge will generate a log file. The log file that is generated has a different name for each direction of the bridge. When processing from DB2 Cube Views to Integration Server the log file is called aislog.xml and by default is located in the directory where the ISBridge is started.

The log file will contain informational messages recording the names of the XML files that have been created. It will also detail any objects that could not be mapped. It is important therefore that you review this log as it is a record of what has not been mapped across into Integration Server. If any of these objects are required in Integration Server they will have to be created manually.

Table 10-1 on page 422 contains the mappings that take place. A list of the objects that cannot be mapped by the bridge can be found in the manual, *Bridge for Integration Server User's Guide,* SC18-7300.

If you have more than one cube in the cube model that you wish to take across to Integration Server, then you will have to repeat this process for each XML output file that was exported from DB2 Cube Views. When you process the subsequent files across the bridge the bridge will inform you that the model XML file already exists if you specify the same output directory. You then have the option to replace or not to replace. If you choose not to replace, then only one XML file (for the metaoutline) will be generated.

Import into Integration Server

Having generated the XML files that can be used by Integration Server, the next step is to launch Integration Server and import these XML files. From the Integration Server desktop console, click **File->XML Import/Export**. With the **Import** tab selected use the **Open XML File** button to navigate to the model XML file that was generated by the Integration Server bridge.

The window you get should look something like that displayed in Figure 10-9.

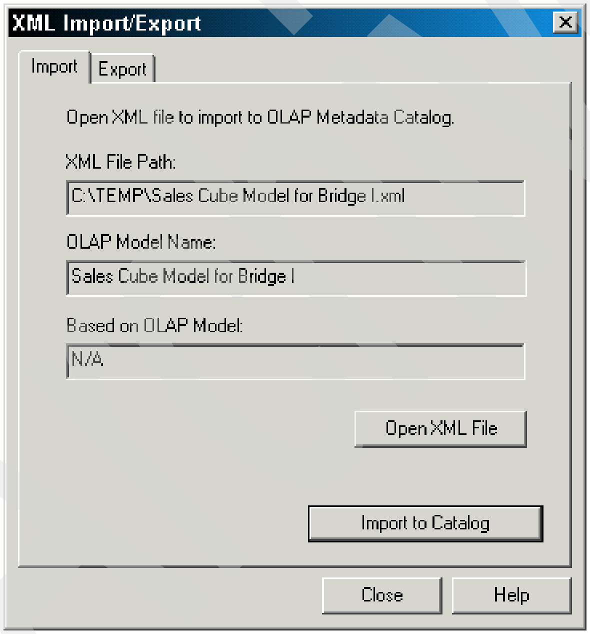


Figure 10-9 Import model into Integration Server

Click the **Import to Catalog** button to import the model into Integration Server.

If there is also a cube to import into Integration Server as a metaoutline, then repeat the process. You should see a window that looks something like that displayed in Figure 10-10.

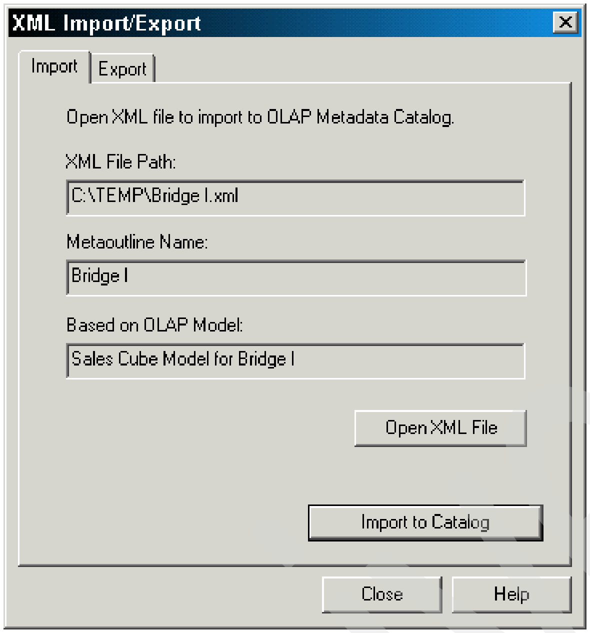


Figure 10-10 Import metaoutline into Integration Server

Once the import process has completed, you can go into Integration Server to view the results of the import. Figure 10-11 shows the model as it would appear in the Integration Server model.

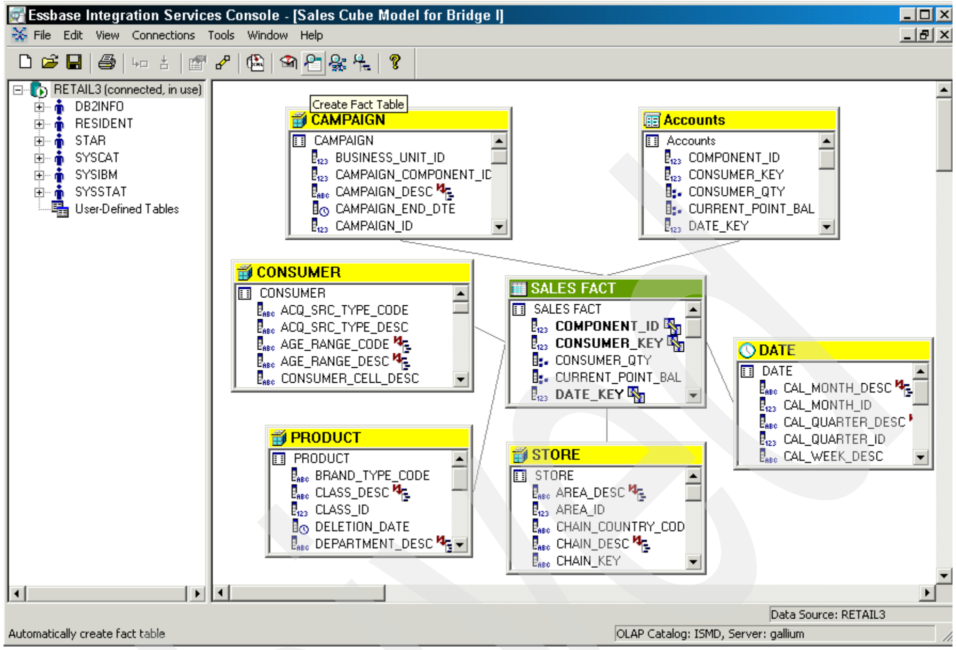


Figure 10-11 Result of import into Integration Server model

If you have more than one cube from the same cube model to bring across to Integration Server, then you should repeat the process for each cube. Selecting the additional cubes will also select the same cube model again for export. When you process this through the bridge, therefore, you will not only get the XML file that you need to import in order to create the second metaoutline, but also you will once again get the XML file for creating the Integration Server model. Furthermore, if you requested the same output directory, it will overwrite the original XML file that was generated from the previous import. Clearly, when importing into Integration Server, you only need to import the XML file for the additional metaoutlines.

Review the results in Integration Server

Having moved the metadata into Integration Server, it is then necessary to review the results and make any enhancements that may be required prior to building the physical DB2 OLAP Server outline.

Some of these considerations are discussed in this section:

- Sequence of dimensions
- Dense and sparse settings
- Time and Accounts dimensions

- Naming considerations
- Related attributes
- Calculated measures
- Adding back objects that do not exist
- Introducing DB2 OLAP Server functions

Sequence of dimensions and dense/sparse settings

The first thing you may notice when reviewing the metaoutline is that each of the dimensions are ordered in the same sequence that they were in the DB2 Cube Views cube, and that the *Accounts* dimension is placed as the last dimension in the metaoutline. In addition, each dimension is specified as being dense. You will therefore need to change your dense and sparse settings, and sequence the dimensions in the order that you require them for your physical outline.

Time and Accounts dimensions

DB2 Cube Views is able to flag a dimension as having a type of Time, and when this goes across the bridge to the Integration Server model, a dimension of type Time is also generated. However, within the metaoutline, the dimension is not tagged as being a Time dimension. This is because the metaoutline maps directly to a DB2 Cube Views cube, and in DB2 Cube Views a dimension in a cube does not have a type. Similarly, the accounts dimension that gets created in the Integration server metaoutline is not tagged as being of dimension type Accounts. You need to therefore to tag the accounts and time dimensions manually, if appropriate.

Naming considerations

In terms of names, you will notice that the name given to the metaoutline is the name of the cube within DB2 Cube Views. The accounts dimension is given a fixed name of *Accounts*. Moreover, in general, the names that you will see are the actual names from DB2 Cube Views, not the business names. There is no concept of comments in Integration Server, so there is nowhere to store any comments that have been documented in DB2 Cube Views.

The reason that the column names are used instead of the business names is to prevent problems arising should you want to combine one or more tables in the Integration Server model metadata, that is by dropping one table on top of the other on the right hand panel in the Integration Server model. In order to do this, the metadata column names need to be unique. It they are not, Integration Server will rename the second column in the model metadata by prefixing the column name with the table name and an underscore character. This is illustrated in Figure 10-12.

The top of this figure shows two tables called *MARKET* and *REGION* as they would be displayed in the Integration Server model if they were displayed separately. There is a column called *REGIONID* in both tables. The bottom of the figure shows the result of dropping the region table on top of the market table in the Integration Server model. The *REGIONID* from the second table is renamed to *REGION_REGIONID*.

It is in order to allow for this functionality that the Integration Server bridge utilizes the column names rather than the business names when moving metadata from DB2 Cube Views to Integration Server.

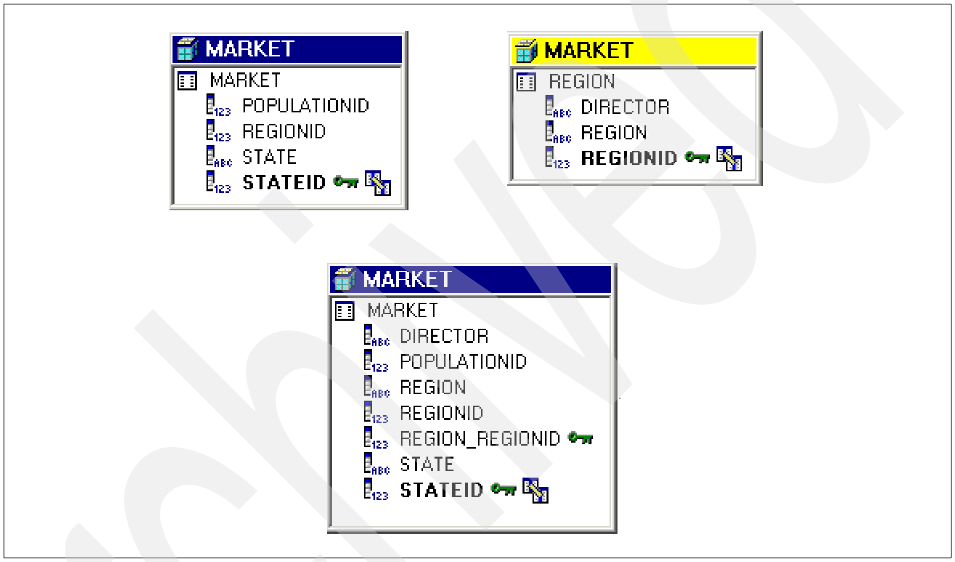


Figure 10-12 Integration Server column renaming in metadata

Related attributes

One of the DB2 Cube Views constructs that is not supported by the bridge is the attribute relationship. If a descriptive attribute relationship was defined in DB2 Cube Views you may wish to reassign that relationship in Integration Server by defining the descriptive column as an alias of the member that gets created in the metaoutline. Similarly an associated attribute relationship may equate conceptually in DB2 OLAP Server terms to an attribute. In this case you should manually create the attribute dimension in Integration Server.

Calculated measures

Some, but not all of the calculated measures can be mapped across to Integration Server. And those that do get mapped may appear differently in Integration Server.

In the cube model there is a calculated measure called *Profit*. It is defined in DB2 Cube Views as in Example 10-1.

Example 10-1 Profit: calculated measure

```
@Column(STAR.CONSUMER_SALES.TRXN_SALE_AMT) -
@Column(STAR.CONSUMER_SALES.TRXN_COST_AMT)
```

When this is mapped across to Integration Server, the measure is created in the Integration Server model in the *Accounts* dimension with a transformation rule, as is shown in Figure 10-13.

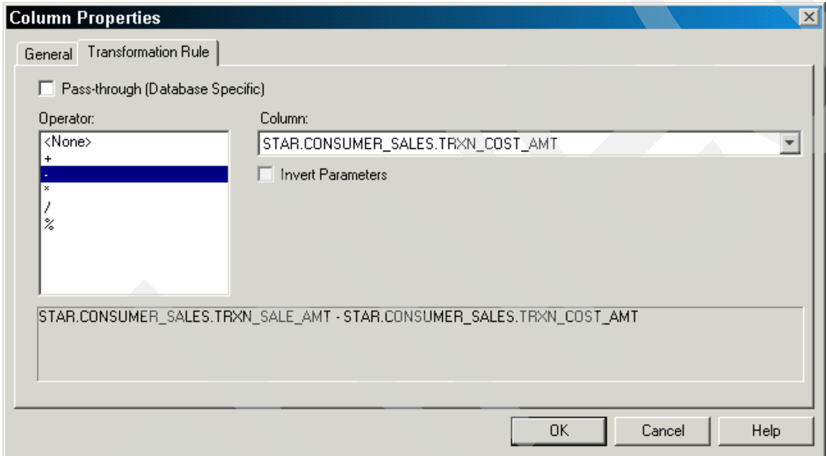


Figure 10-13 Integration Server column properties

For other measures, the Integration Server bridge will attempt to build a measure hierarchy in the Integration Server metaoutline. For example, in DB2 Cube Views a *Sales per unit* measure is defined as in Example 10-2.

Example 10-2 Sales per unit: calculated measure

```
@Measure(STAR.TRXN_SALE_AMT)/ @Measure(STAR.TRXN_SALE_QTY)
```

This will appear in Integration Server as shown in Figure 10-14.

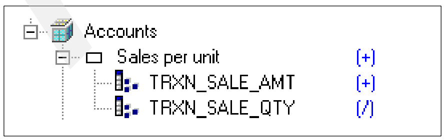


Figure 10-14 Integration Server measure hierarchy

If the bridge is unable to build a straightforward measure hierarchy, then the measure will be dropped. Example 10-3 shows a similar calculation to Example 10-2 in that a division is involved, however, in this example the value that is being divided is in itself an expression.

Example 10-3 Profit %: calculated measure

```
(@Measure(STAR.Profit)* 100.0)/ @Measure(STAR.TRXN_SALE_AMT)
```

In this case the measure will be dropped and will not appear in Integration Server.

The Integration Server bridge will not create any formula in the Integration Server metaoutline. What it will first try to do is to map a DB2 Cube Views measure to a measure in the Integration Server model as in the *Profit* example in Figure 10-13 on page 437. Only if it is not able to do this will it then attempt to map the measure to a measure hierarchy in the Integration Server metaoutline as shown in Figure 10-14 on page 437. The most likely reason for it not being able to map to a measure in the model in Integration Server is if the aggregation function is set to *None* in DB2 Cube Views.

When the aggregation function is set to *None* there is no aggregation line specified in the output XML when the export from DB2 Cube Views is performed. The Integration Server bridge sees that there is no aggregation statement in the XML and therefore attempts to map the measure to a measure hierarchy in the metaoutline, as measures in the metaoutline do not require aggregation.

Adding back objects that do not exist

One of the manual processes you may need to complete in Integration Server is adding back any columns into the Integration Server model that have been dropped by the bridge. For example, if you have a measure with multiple aggregations defined with an aggregation script in DB2 Cube Views, the bridge will not pass the attribute across as a member of the accounts dimension to the Integration Server metadata (this is because aggregation scripts are not supported in Integration Server). If you wish to include this column from the fact table in the accounts dimension, you will need to add it back manually. This can be done within the Integration Server model by going to **Table Properties** (for the fact table associated with the accounts dimension). Take the **Columns** tab, and from there is an option to add a column, as shown in Figure 10-15.

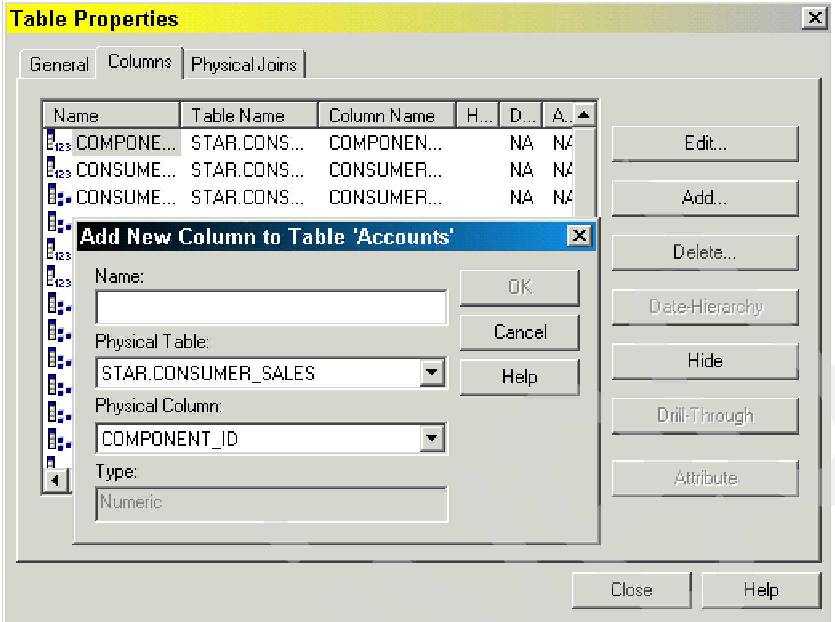


Figure 10-15 Add back missing columns in Integration Server

When considering the measures, you may choose to group measures differently in Integration Server and to change the consolidation attributes. Member properties such as the use of *two-pass calculation* and *dynamic calc* storage settings should also be reviewed.

Introducing DB2 OLAP Server functions

Once you are satisfied with the metadata within Integration Server, including adding any additional functionality specific to DB2 OLAP Server such as analysis functions or Hybrid Analysis, you can proceed with building the physical outline of the DB2 OLAP Server database.

When adding any new items into Integration Server, consider the effect on the resulting SQL that will be generated. If you plan to make use of an MQT, then clearly anything that you add must be able to be derived from the available MQT(s).

10.3.2 Metadata flow from Integration Server to DB2 Cube Views

The tasks that need to take place in order for metadata to flow from Integration Server to DB2 Cube Views are as follows:

1. Export the Integration Server model metadata to an XML file.

2. Export the Integration Server metaoutline metadata to an XML file.
3. Process these XML files through the Integration Server bridge to produce one output XML file.
4. Import the output XML file into DB2 Cube Views.

Export from Integration Server

The first step that is required is to export the Integration Server model metadata to an XML file. From the Integration Server desktop console click **File->XML Import/Export**. In the XML Import/Export window select the **Export** tab. You are then presented with a list of models and their associated metaoutlines.

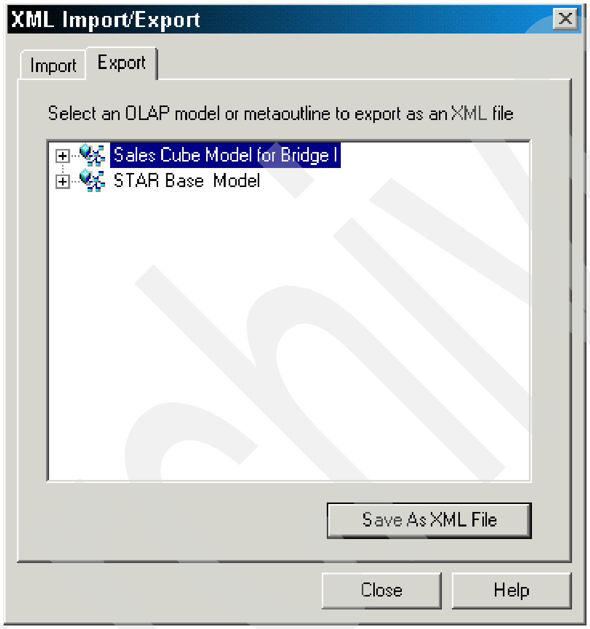


Figure 10-16 Integration Server export

Select the model that you wish to export and click the **Save as XML File** button.

When you have saved the model, repeat the process for the metaoutline.

At this stage you will have two XML files, one for the Integration Server model export and one for the Integration Server metaoutline export.

If the model has more than one metaoutline that you wish to take over to DB2 Cube Views, then you will also need to repeat the export process for each metaoutline.

Process the XML across the Integration Server bridge

As described in 10.3.1, “Metadata flow from DB2 Cube Views to Integration Server” on page 428, launch the Integration Server bridge and this time select the **To DB2** tab.

In this direction you are required to enter four items, as shown in Figure 10-17.

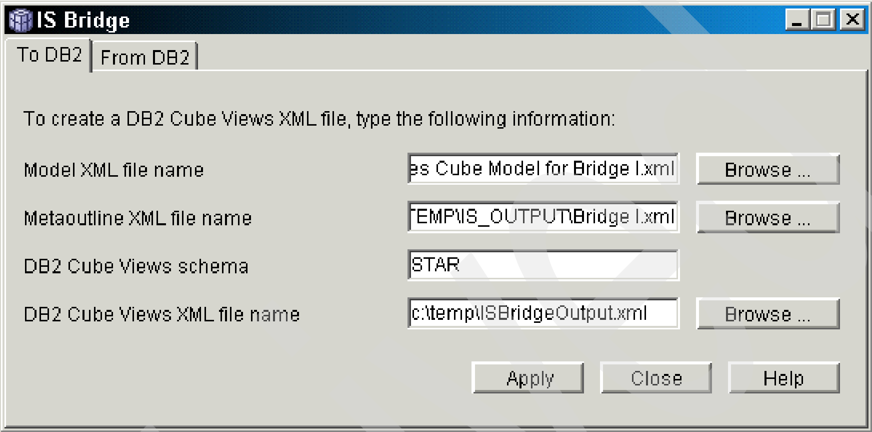


Figure 10-17 The IS bridge from Integration Server to DB2 Cube Views

You must specify the full path names of the model and metaoutline XML files that were exported from Integration Server. The **Browse** button is available to assist with locating these files. You are also required to specify the schema name of the database tables that relate to this metadata. Finally you should enter the full path name of the XML output file that you wish the bridge to create. It is important that you specify the .xml file type suffix when you enter the file name.

After you have clicked the **Apply** button, you should receive a successful completion message once the bridge process has finished. The output XML file should be created in the directory that you specified.

The bridge will generate a log file. The log file that is generated has a different name for each direction of the bridge. When processing from Integration Server to DB2 Cube Views the log file is called isalog.xml and by default is located in the directory where the ISBridge is started.

The log file will be empty if the bridge was able to map everything successfully. Objects that could not be mapped should be reported in the log. It is recommended therefore that you review this log to see those objects that have not been mapped across into DB2 Cube Views. It is possible at Fixpack 2+ of the product that some objects that were not able to be mapped do not appear in the log. A manual review in OLAP Center of the imported objects is therefore recommended. A list of the objects that cannot be mapped by the bridge can be found in the manual, *Bridge for Integration Server User's Guide*, SC18-7300. Table 10-1 on page 422 contains the mappings that do take place.

If you exported more than one metaoutline from Integration Server, then you will need to repeat this process for each metaoutline that you wish to bring across to DB2 Cube Views. Note that, for every metaoutline that you wish to process, you will also be required to specify the exported model XML file each time.

Import into DB2 Cube Views

Once the Integration Server bridge has generated the XML file, it can then be imported into DB2 Cube Views.

Start up OLAP Center and from the menu bar click **OLAP Center->Import** and you will be presented with the first screen of the Import Wizard as shown in Figure 10-18.

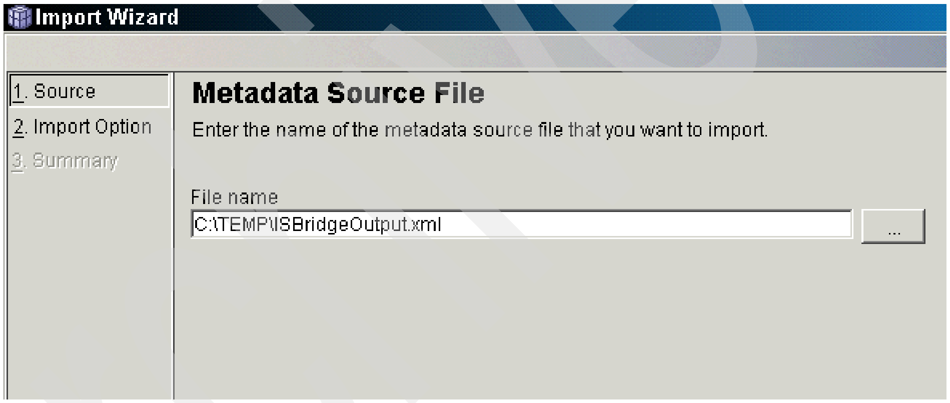


Figure 10-18 Import wizard screen 1

After having entered the full path name of the XML file that was generated by the Integration Server bridge, click **Next**, and you will be presented with the list of metadata objects that OLAP Center is going to import, as shown in Figure 10-19. Click the appropriate radio button to specify how the OLAP Center import should resolve existing names.

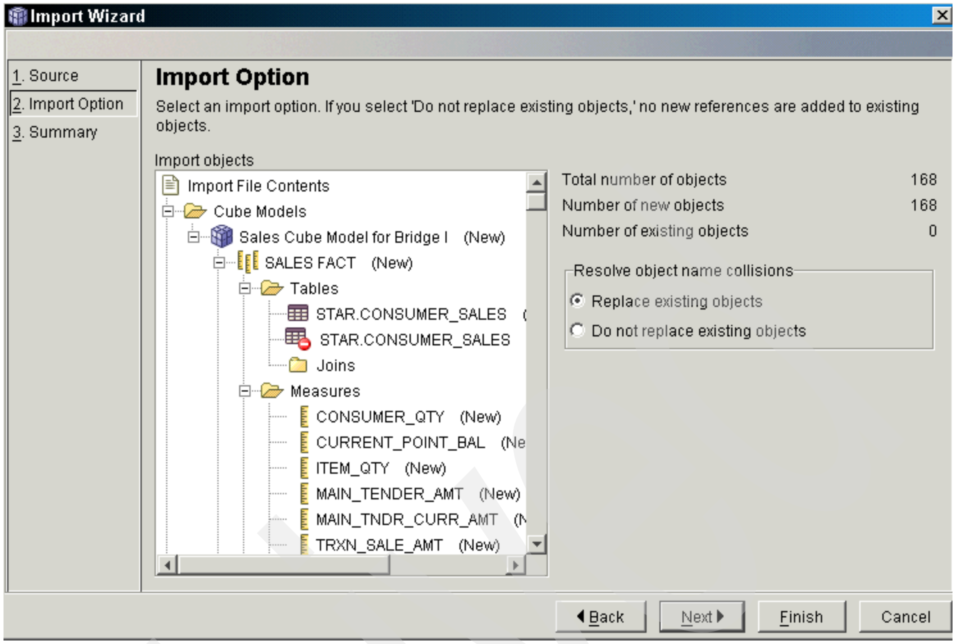


Figure 10-19 Import wizard screen 2

The import should then complete and the result can be viewed from OLAP Center.

If you are bringing across more than one metaoutline, you will have more than one XML file to import. When you import the additional XML file(s) that contain the additional migrated metaoutline(s) you will of course be including the model metadata each time in the XML file that you are importing. The import wizard will recognize that some of the objects that you are attempting to import exist already in the cube model, and when you get to the screen shown in Figure 10-19, the wizard will display the number of new objects (reflecting the new metaoutline) and the number of existing objects (reflecting the already imported model and metaoutline) that you are requesting to import. You can then select the appropriate radio button option to either replace or not replace the existing objects with the ones in the current import XML file.

Once you have imported the additional metaoutlines, the end result will be one cube model with multiple cubes, one cube per metaoutline.

Review the results in OLAP Center

Having moved the metadata into DB2 Cube Views, you can then view the result in OLAP Center and determine what enhancements you wish to add to complete the cube model and cube that has been generated.

Some of the considerations are discussed in this section:

- Bridge mapping
- Measures considerations
- Naming considerations
- Alternate hierarchies
- Hidden columns
- Automatically generated time dimension

Bridge mapping

The Integration Server model will have been mapped to a cube model in DB2 Cube Views, and the Integration Server metaoutline will have been mapped to a cube in DB2 Cube Views. The name of the Integration Server model will become the name of the DB2 Cube Views cube model. The name of the metaoutline will become the name of the Integration Server cube.

A dimension that is denoted as having a type of *Time* in the Integration Server model will also be tagged as a *Time* dimension in the DB2 Cube Views model. A dimension in a cube model is either a *Time* dimension or a *Regular* dimension.

Facts in the accounts dimension from the Integration Server model will attempt to be mapped back to measures in the cube model. Members of the accounts dimension in the metaoutline will attempt to be mapped back to measures in the DB2 Cube Views cube.

Measures considerations

In Integration Server the measures are usually a combination of measures that can be mapped straight back to a column in the fact table, and measures that are derived or calculated measures. Measures that can be mapped straight back to columns in the fact table will appear in both the Integration Server model and the metaoutline. This follows the architecture rules for DB2 Cube Views where measures that are in the cube must also appear in the cube model. For these types of measures the mapping from Integration Server to DB2 Cube Views is straightforward.

However, in addition to these types of measures, it is usual to also find a number of other measures in the Integration Server metaoutline that do not appear in the model. These are derived measures that are calculated either at calculation time or dynamically at query time. Such measures are usually defined by a formula that may comprise straightforward mathematical functions, or it may contain one or more of the many sophisticated functions provided with DB2 OLAP Server.

If a measure is defined using a formula in the Integration Server metaoutline, then it is not taken across to DB2 Cube Views in release one of the Integration Server Bridge. All measures that are defined using a formula in the metaoutline are dropped by the bridge currently.

For some measures that are defined in the metaoutline using a formula, it will be possible to recreate those measures in DB2 Cube Views by creating a calculated measure and building the appropriate SQL expression manually.

If a mathematical calculation is defined as a consolidation rather than as a formula, then this type of calculation may be taken across the bridge to DB2 Cube Views. For example, let us again consider a measure called *Sales per Unit* that is calculated by dividing *TRXN_SALE_AMT* by *TRXN_SALE_QTY*. *Sales per Unit* expressed in the metaoutline under the parent dimension name *Accounts* as a formula (*TRXN_SALE_AMT/ TRXN_SALE_QTY)* would be dropped by the Integration Server bridge because it is defined as a formula.

Whereas *Sales Per Unit* expressed as a consolidation rather than a formula in the metaoutline as in Figure 10-14 on page 437 may be passed across to DB2 Cube Views.

Whether or not this consolidation type of calculation is successfully passed across to DB2 Cube Views will depend upon whether or not the referenced measures are unique. That is to say:

- If *TRXN_SALE_AMT* or *TRXN_SALE_QTY* have already been defined in the metaoutline (either as measures in their own right or as measures used in a previous consolidation calculation), then the import into DB2 Cube Views will fail with a duplicate measure error.
- If *TRXN_SALE_AMT* and *TRXN_SALE_QTY* are unique, then the import into DB2 Cube Views will be successful and three measures will be created: *TRXN_SALE_AMT*, *TRXN_SALE_QTY*, and *Sales per Unit_UDM*. The Integration Server bridge adds the characters *_UDM* to indicate a User Defined Measure. *Sales per Unit_UDM* is mapped to the appropriate SQL expression, and the aggregation is set to *None*.

Naming considerations

The bridge takes into account the differences in requirements for the uniqueness of names in Integration Server and DB2 Cube Views. The rules are stricter in DB2 Cube Views than they are in Integration Server, and the bridge therefore performs some name changes of objects in order to avoid name collisions. So, for example, in Integration Server, a dimension in one metaoutline can have the same name as a dimension in another metaoutline for the same model. In DB2 Cube Views, dimensions referenced in different cubes for the same cube model cannot have the same name. In Integration Server, a hierarchy created for dimension A in a model can have the same name as a hierarchy for dimension B in the same model. In DB2 Cube Views, hierarchy names must be unique across the cube model.

Therefore, to avoid name collisions, the Integration Server bridge will perform name changes for dimensions and hierarchies when processing the metadata from Integration Server to DB2 Cube Views.

For dimensions, the name change occurs at the individual cube level; there are no name changes at the cube model level. At the cube level, the dimension name is changed such that it is prefixed with the cube name and a blank character. So, for example, if there were a metaoutline in Integration Server called *Sales Cube* that contained a dimension called *Customer*, then when taken across the bridge and imported into DB2 Cube Views, the resulting dimension name would be *Sales Cube Customer*.

Similar considerations are applied to hierarchy names. In fact, in Integration Server the hierarchies in the metaoutline do not have names as such. Hierarchies are generally defined and named in the model. When creating the metaoutline, the user drags the hierarchies from the left hand panel across to the right-hand panel where they are referenced not by name, but by the columns that are used in the structure of the hierarchy.

Furthermore, it is also possible in Integration Server metaoutlines to have hierarchies that do not reference back to the Integration Server model. In DB2 Cube Views, a cube hierarchy must always reference a cube model hierarchy. A cube hierarchy cannot exist in isolation. The Integration Server bridge therefore has to handle these differences.

The Integration Server bridge will attempt to map a metaoutline hierarchy back to a hierarchy in the Integration Server model. If it is unsuccessful, the bridge will create a cube model hierarchy for the cube hierarchy to reference. If the bridge needs to create a cube model hierarchy, it will use a naming convention of *NewDimensionName* with a suffix of *HIER*.

In the previous example of dimension name change, we had a metaoutline in Integration Server called *Sales Cube* that contained a dimension called *Customer*. When taken across the bridge and imported into DB2 Cube Views the resulting dimension name would be *Sales Cube Customer*. If the Integration Server bridge is unable to map a metaoutline hierarchy for *Customer* back to a hierarchy in the Integration Server model, then it will create a new hierarchy in the cube model called *Sales Cube Customer HIER*. If the Integration Server bridge is able to map a metaoutline hierarchy back to a hierarchy in the Integration Server model, then the existing model hierarchy will be used in the cube model in DB2 Cube Views and the existing name from Integration Server will be used without change.

When considering the hierarchy at the metaoutline level, the Integration Server bridge will always generate a name for the corresponding cube hierarchy in DB2 Cube Views. The naming convention used is *NewDimensionName* with a suffix of *CUBEHIER*.

Using the previous example, therefore, a cube hierarchy created by the Integration Server bridge for the Customer dimension would be given the name *Sales Cube Customer CUBEHIER*.

> **Note:** Be aware that any name changes that the Integration Server bridge performs are not logged in the isalog.xml file.

Alternate hierarchies

Another consideration regarding hierarchies is that in Integration Server a dimension in a metaoutline may have multiple or alternate hierarchies. This will not map to DB2 Cube Views because at the cube level only one hierarchy per dimension is permitted. If there is more than one hierarchy for any given dimension in a metaoutline, Integration Server will map the first hierarchy that is presented to it in the XML file that is exported from Integration Server. This may or may not be the first hierarchy that is presented to the user in the user interface.

Hidden columns

In the Integration Server model it is possible to flag that a column should be hidden. This is often used where there are many columns in the relational table that are not required for the OLAP Server database. The Integration Server bridge assumes that hidden columns are not required, and will therefore only take them across to DB2 Cube Views if they are required in a join. If a column is flagged as hidden and it is not required for a join, it will not be taken across to DB2 Cube Views as an available attribute.

Should you wish to include these hidden columns in DB2 Cube Views then you can add them back in OLAP Center once the import has completed.

Automatically generated time dimension

One of the functions available in Integration Server enables a *Time* dimension to be created where no actual time table exists in the star schema database. When creating an Integration Server model it is possible to tell Integration Server to create a *Time* dimension based on one of the available date columns in the fact table. Having selected a valid date column, a hierarchy can then be created in the Integration Server model using standard SQL functions. For example, the function *YEAR* will return the year from a valid date expression, the function *QUARTER* will return the quarter from a valid date expression, and the function *MONTH* will return the month from a valid date expression.

These three values can then be used to create a time hierarchy of *Year*, *Quarter*, *Month* (and of course many other time hierarchies can be created in the same way). When you export the model to XML you will see a dimension called *Time* in the XML file for which the physical table name that is given is the name of the fact table. The join between the fact table and the time dimension will physically be a self join on the fact table.

When the Integration Server metadata is imported into DB2 Cube Views (via the bridge), the Cube Model will reflect what was created in the Integration Server model. There will be a *Time* dimension that is based on the fact table, the attributes for *Year, Quarter and Month* will be mapped to SQL expressions as described above, and the hierarchy *Year, Quarter and Month* will be created. The join will be a self join on the fact table. By default in Integration Server the join on the fact table will be based on the date column that was selected. If this has not been changed then this will come across the bridge as an inner join on the date column with a cardinality of Many:1.

If left like this, an error will be received when trying to run the Optimization Advisor. The error will indicate that a primary key is not defined using the columns involved in the fact table self join. In order to optimize this type of cube model a primary key must be defined, the join cardinality must be 1:1 and the join type must be inner. An example of this is described in Chapter 6 of the *IBM DB2 Cube Views Business Modeling Scenarios manual,* SC18-7803.

Therefore, in order to optimize this cube model, you must ensure that a primary key is defined for the fact table, and that the column(s) referenced in the primary key are the ones used in the definition of the self join on the fact table in OLAP Center.

10.4 Maintenance

This chapter has looked at the use of the Integration Server bridge through the GUI interface. It is also possible to use the Integration Server bridge from a command line using the **`ISBridge`** command. The syntax and parameters for this command are detailed in the *Bridge for Integration Server User's Guide*, SC18-7300.

This means that in combination with the **`db2mdapiclient`** utility for import/export metadata to/from DB2 Cube Views (as described in Appendix D, “DB2 Cube Views stored procedure API” on page 673), and the Integration Server **`impexp.bat`** utility to import/export metadata to/from the Integration Server catalog, it is also possible to fully script the process that has been described in this chapter.

At Fixpack 2+ of DB2 Cube Views V8.1 Integration Server bridge, there is no support for incremental changes. Any change that needs to be reflected in either DB2 Cube Views or Integration Server will therefore require a complete refresh of the metadata from the incoming XML file.

The current Integration Server bridge only performs the mapping of metadata objects contained in the XML files that it is given. It relies totally on the import utility of both tools (DB2 Cube Views and Integration Server) to place the metadata in the catalogs.

In Integration Server, it is not possible to import changes to an existing model or metaoutline. Therefore, the existing model and metaoutline have to be deleted or renamed prior to importing the latest version that has come from DB2 Cube Views. This means that any manual enhancements that have been applied to the model or metaoutline in Integration Server will need to be re-applied.

When going the other way, from Integration Server to DB2 Cube Views, it is also recommended that a full refresh be performed.

10.5 DB2 OLAP Server examples and benefits

The benefits of using DB2 OLAP Server with DB2 Cube Views relate both to the interchange of metadata that the Integration Server bridge provides, and also to performance enhancements that may be gained from the use of MQTs where there are relational queries involved. The previous section discussed the exchange of metadata, this section looks at the performance considerations.

The interaction of DB2 OLAP Server with the relational database occurs in three areas:

- **Data load:** Loading the MOLAP database from the relational database
- **Hybrid Analysis:** Extending the MOLAP hierarchy into the relational database
- **Integration Server drill through reports**: Running relational reports from specific intersections in the MOLAP database

The use of an MQT can significantly improve the performance of relational queries in these situations. The benefit of having DB2 Cube Views is that the Optimization Advisor will save the DBA a significant amount of analysis time that might be spent trying to work out what MQTs to build. The Optimization Advisor will advise on the MQT to be built and provide the script required to build it.

10.5.1 Data load

In performing a data load, an SQL query is generated by Integration Server that will involve joining each of the dimension tables required for this database to the fact table, and potentially performing some aggregation of the data. This aggregation will depend on the level of granularity in the relational source compared to the level required for the OLAP database.

Sometimes a relational data source is purposely built for an OLAP database, and as such, it is at the same level of granularity as the OLAP database. Other times the OLAP database is a summary extraction from the relational source data, and as such, the data load query will involve a level of aggregation. Usually with larger databases and certainly if Hybrid Analysis or drill through reporting are enabled, then the MOLAP database will contain higher level data and will therefore require aggregations in the data load SQL.

Loading data from an MQT should be faster than loading data from the base tables. There will be no joins to perform and the data in the MQT should be an aggregation of the base data. The higher the level of aggregation, the smaller the MQT that the data load has to query and therefore the greater the potential performance benefit. However, there is a cost to consider, and that is the cost of building the MQT in terms of both time to build and storage space required.

If the level of granularity in the MOLAP database and the relational database are the same, then the hierarchies in the cube and cube model will be the same. When optimizing for extract, the Optimization Advisor will advise on an MQT based on the cube which will result in a large MQT that will basically be a result of the fact table joined to each of the dimension tables.

This type of MQT will probably take a long time to build. Additionally, if the MQT is at almost the same level as the fact table, then the greater the chance that the DB2 optimizer will choose not to select that MQT when deciding on how best to return the result set. In general, if the number of rows in the MQT is close to the number of rows in the fact table, then there will probably be little performance benefit.

If the level of granularity in the MOLAP database is at a higher level than the relational database, then the bottom level of the cube will be at a higher level than the bottom level in the cube model. When optimizing for extract, the Optimization Advisor will advise on an MQT based on the cube which in this case should result in a smaller MQT being built (relative to the size that it would be for the cube model). In general, it is more practical to consider the use of an MQT where the extraction is at a higher level than the level described in the cube model.

In terms of justifying the time to build, there are a number of things to consider. Clearly the more DB2 OLAP Server databases that get built from the one relational database the more likely it will be that the benefits of a reduced load time out weigh the cost to build the MQT. Moreover, as the MOLAP database is unavailable to users whilst the data load is taking place, then there are additional advantages related to end user availability to be had, by moving the workload away from the data load and into the relational database.

However, if the MOLAP user also requires access to the same relational database, then considerations will also need to be given to the scheduling of the MQT refreshes so as not to effect the end users. Similarly, the synchronization of data also needs to be considered in an environment where the user has access to both the MOLAP and relational data.

In this example a cube has been defined in DB2 Cube Views from the cube model example used in this book. The cube is defined at a higher level than the base fact table. For example the *DATE* dimension goes down to month instead of day, the *CONSUMER* dimension does not go down to individuals, the *STORE* dimension does not go down to individual store and the *STORE* hierarchy is only three levels deep. The cube as defined in DB2 Cube Views is shown in Figure 10-20.

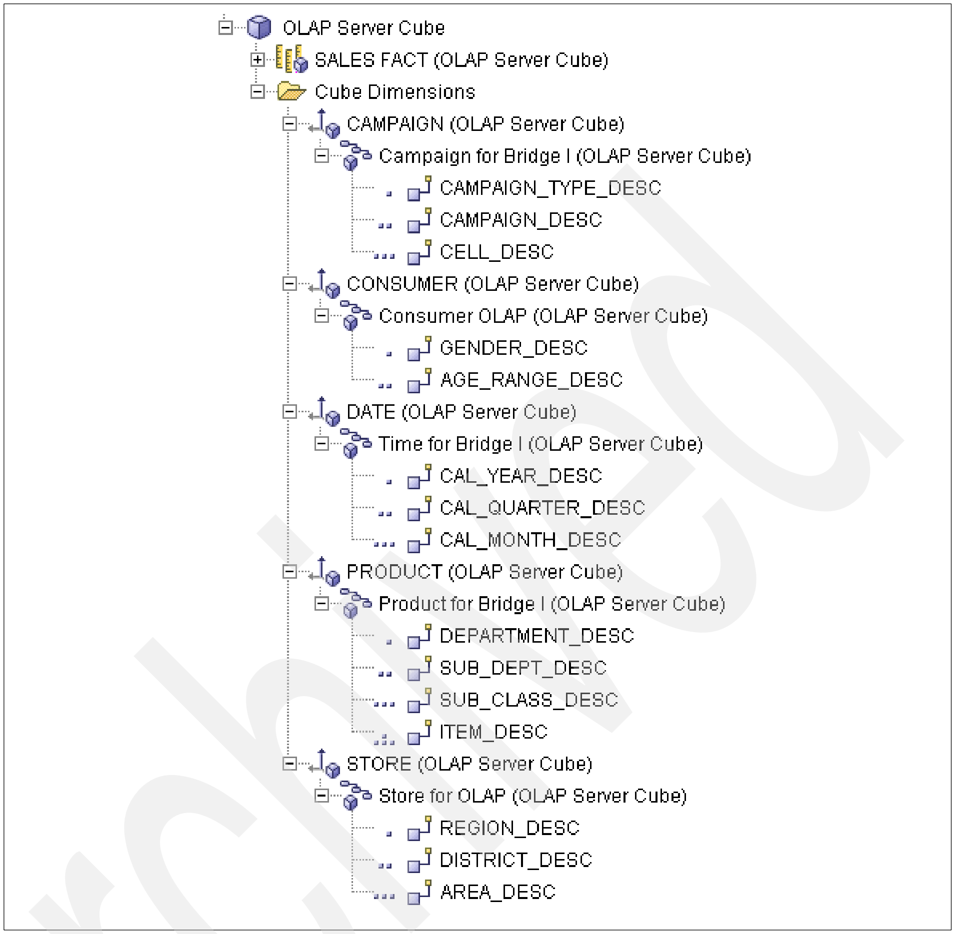


Figure 10-20 DB2 Cube Views cube

This cube was then exported from DB2 Cube Views, processed through the Integration Server bridge, imported into Integration Server, and the resultant metaoutline modified slightly such that a database can then be built in DB2 OLAP Server. Figure 10-21 shows the metaoutline in Integration Server.

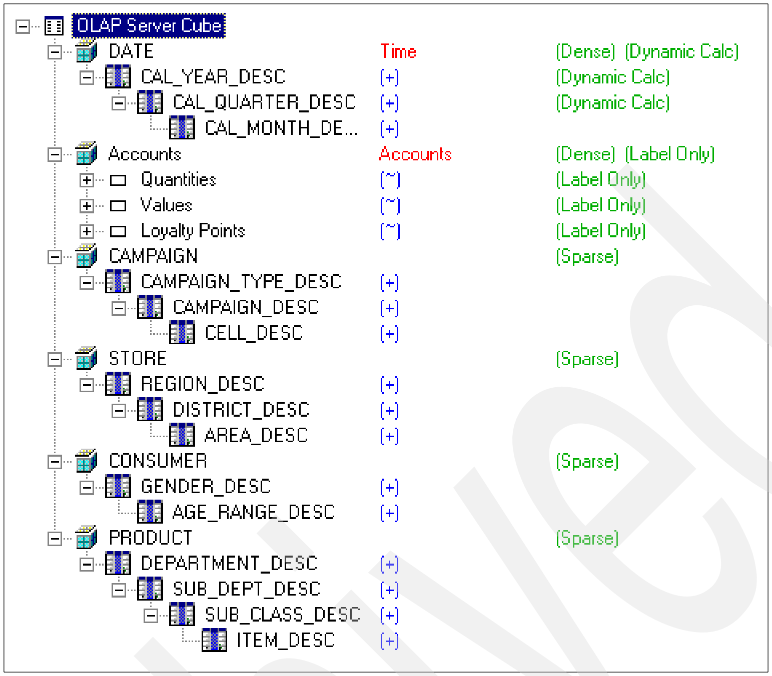


Figure 10-21 Integration Server metaoutline

The section “Review the results in Integration Server” on page 434 discussed some of the changes that might be done, having imported the metadata into Integration Server. Listed below are the changes that were made in this example:

- Changed the order of the dimensions, allocated *Time* and *Accounts* properties and specify the appropriate dense and sparse settings.
- Changed the name of the dimensions to remove the metaoutline suffix
- Changed the name of the measures to business names.
- Kept prefix members with their parent where appropriate in order to ensure unique member names.
- Specified *dynamic calc* for the higher level members of the *DATE* dimension.
- Changed consolidation properties for the measures and added three generation two members to the *ACCOUNTS* dimension in order to group the measures as either quantity, values or loyalty points measures.

- *Profit %* is lost as a measure when processed across the bridge. This was added back manually in Integration Server, specifying that the percentage be rounded to one decimal place. Figure 10-22 and Figure 10-23 show how this measure is defined in both DB2 Cube Views and Integration Server.

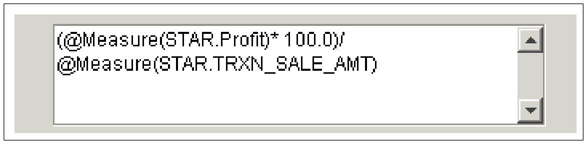


Figure 10-22 The measure in DB2 Cube Views

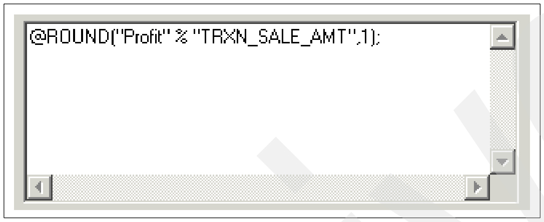


Figure 10-23 The measure in Integration Server

The Optimization Advisor was run against the cube model specifying a query type of *Extract*. The MQT script that was generated was then run to create the MQT. As this MQT is for extract, it is a straightforward summary table with a simple GROUP BY. The MQT aggregation is based on the cube. In the MQT create script the tables are tagged as in Figure 10-24. The actual columns in the MQT create GROUP BY clause are shown in Figure 10-25. By relating these back to the cube definition in Figure 10-20 on page 452 it is clear that the extract is matching the cube exactly.

```
FROM
"STAR"."CONSUMER_SALES" AS T1,
"STAR"."CAMPAIGN" AS T2,
"STAR"."CONSUMER" AS T3,
"STAR"."DATE" AS T4,
"STAR"."PRODUCT" AS T5,
"STAR"."STORE" AS T6
```

Figure 10-24 MQT script FROM clause

```
GROUP BY
T2."CAMPAIGN_TYPE_DESC",
T2."CAMPAIGN_DESC",
T2."CELL_DESC",
T2."COMPONENT_DESC",
T4."CAL_YEAR_DESC",
T4."CAL_QUARTER_DESC",
T4."CAL_MONTH_DESC",
T5."DEPARTMENT_DESC",
T5."SUB_DEPT_DESC",
T5."SUB_CLASS_DESC",
T5."ITEM_DESC",
T6."ENTERPRISE_DESC",
T6."CHAIN_DESC",
T6."REGION_DESC",
T6."DISTRICT_DESC",
T6."AREA_DESC",
T6."STORE_NAME",
T3."GENDER_DESC",
T3."AGE_RANGE_DESC",
T3."FULL_NAME")
```

Figure 10-25 MQT script GROUP BY clause

The data load SQL can then be copied and pasted into DB2 Explain to see whether the data load will in fact use the MQT. In order to access the data load SQL click **Outline->User Defined SQL** from the Integration Server metaoutline display. You will then see the window displayed in Figure 10-26. The SQL can then be copied from here.

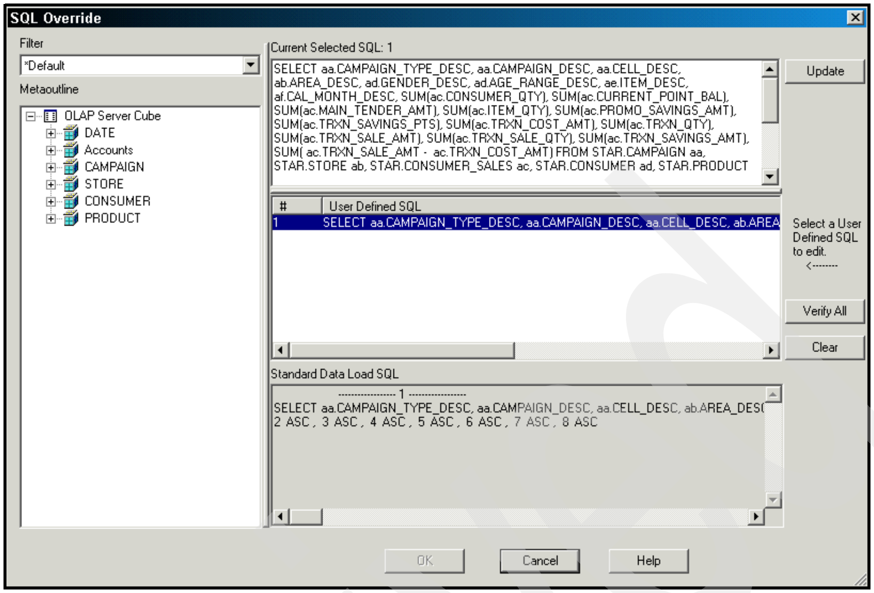


Figure 10-26 Integration Server data load SQL

Figure 10-27 shows the DB2 access plan graph from Visual Explain and this indeed verifies that a table scan of the MQT will be used for the data load instead of a join from the base tables.

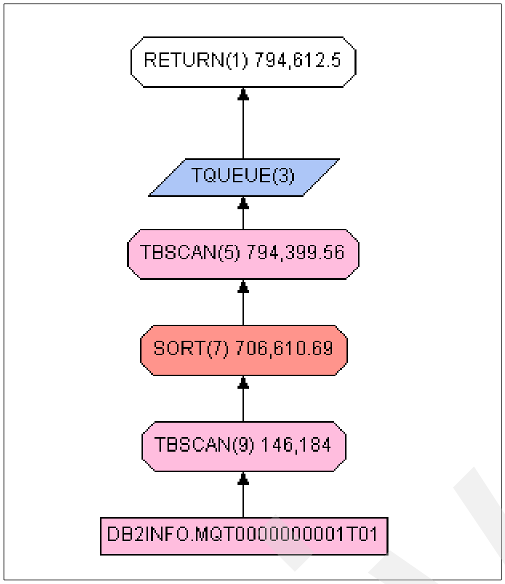


Figure 10-27 Load explain with MQT

The timerons cost taken from DB2 explain for the data load query with the MQT and without the MQT are shown in Table 10-2.

Table 10-2 Data load performance costs

Total cost timerons with MQT	794,612.5
Total cost timerons without MQT	2,706,616.75

A significant cost improvement can be seen when the query is re-routed to the MQT.

A second example of the potential benefits of DB2 Cube Views comes from a real customer installation. In this example the customer was a large enterprise who had implemented partitioning of DB2 OLAP Server databases, sourcing their data from a single database. Their source fact table was approximately a 300 million row table and from this they were able to build a 17 million row MQT to meet the data load requirements of the DB2 OLAP Server databases.

The time to just run the query to extract the data from the base tables was approximately 1.5 hours. This time does not include the time taken to write the blocks in the DB2 OLAP Server database. The time to run the query to extract from the MQT was just 2 minutes. Multiply this performance enhancement across each of their MOLAP partitions and this represents a highly significant improvement.

In this kind of environment where there is a number of similar OLAP Server databases, it may be worth spending some time thinking about the type of cubes to design in DB2 Cube Views. If a straight one-to-one mapping takes place from the Integration Server metaoutlines to DB2 Cube Views, then there will be the same number of cubes in DB2 Cube Views as there are databases in DB2 OLAP Server. When the Optimization Advisor is run in DB2 Cube Views, the script that is generated will create one MQT per cube.

This may not be the most optimal design. It may therefore be worth considering designing a fewer number of cubes in DB2 Cube Views that represent a super-set of the actual cubes that might otherwise be created. Certainly, in this particular example, only one MQT was created and was used for data load by each of the partitions in the partitioned database.

10.5.2 Hybrid Analysis

IBM DB2 OLAP Server Hybrid Analysis enables a hybrid OLAP environment whereby the lower levels of one or more dimensions can be located in the relational database whilst the higher levels remain in the MOLAP database. As the user drills down a hierarchy into the relational database, they are unaware that the database that they are querying has changed. They may, however, experience longer query times when the environment changes, because instead of retrieving pre-calculated blocks from the MOLAP database, they will be generating SQL queries against a relational database and potentially joining a number of dimension tables to a large fact table in order to produce the result set.

There are significant benefits to be achieved from implementing Hybrid Analysis. Removing the lower levels of one or more dimensions from the MOLAP database can significantly reduce the size of the MOLAP database and the time it takes to calculate the database. The outline that gets loaded into memory will also be reduced in size.

DB2 Cube Views enables Hybrid Analysis. If the user's query can be re-routed to an MQT that is an aggregation of the base data when it crosses the line from MOLAP to relational, then the user will experience better performance. The transition from the MOLAP database to the relational database will be a smoother one.

The more aggregation that has been achieved in the MQT, and therefore the fewer the number of rows relative to the fact table, the greater the benefit for the performance of the query.

From a DB2 Cube Views perspective, the cube that is defined should encompass the entire hybrid space. When running the Optimization Advisor select the query type of drill through. The Optimization Advisor will consider the cardinality of the dimensions and base the slices that it recommends on the dimension with the greater cardinality, on the assumption that this dimension is the most likely to be in the relational database.

The Optimization Advisor will attempt to create a low level slice of the data and then to add one or a few more slices of the data. A key factor governing the slices that are selected is the disk space limitation value that is entered when running the Optimization Advisor.

Figure 10-28 shows an extract from the script that was created for the drill through optimization for this cube (as is detailed in Figure 10-20 on page 452). The extract shows that in this example the Optimization Advisor has selected two slices of the database. The first is the low level slice which is identical to the extract MQT. For the second slice, the Optimization Advisor has identified the *PRODUCT* dimension as having the greater cardinality, and therefore as being the one that is most likely to be in the relational database. This second slice of the data goes right down to *ITEM_DESC* in the *PRODUCT* dimension.

```
GROUP BY GROUPING SETS (
(
T2."CAMPAIGN_TYPE_DESC",
T2."CAMPAIGN_DESC",
T2."CELL_DESC",
T3."GENDER_DESC",
T3."AGE_RANGE_DESC",
T4."CAL_YEAR_DESC",
T4."CAL_QUARTER_DESC",
T4."CAL_MONTH_DESC",
T5."DEPARTMENT_DESC",
T5."SUB_DEPT_DESC",
T5."SUB_CLASS_DESC",
T5."ITEM_DESC",
T6."REGION_DESC",
T6."DISTRICT_DESC",
T6."AREA_DESC"
),
(
T2."CAMPAIGN_TYPE_DESC",
T2."CAMPAIGN_DESC",
T5."DEPARTMENT_DESC",
T5."SUB_DEPT_DESC",
T5."SUB_CLASS_DESC",
T5."ITEM_DESC"
)))
```

Figure 10-28 MQT script GROUP BY GROUPING SETS clause

Having identified the *PRODUCT* dimension as the anchor point for what is most likely to be in the relational database, the Optimization Advisor will then evaluate a number of different options for identifying what else should be in the slice. In this example, this second slice includes the top two levels of the *CAMPAIGN* dimension.

Query workloads

The DB2 OLAP Server spreadsheet client was then used to perform a selection of queries to measure the performance achieved when a result set has to be fetched from the relational database. The queries that were run were simple queries chosen to demonstrate performance characteristics of extending out of the MOLAP database and into relational with MQTs, rather than to demonstrate any query functionality of the end user tool or any analytical functions with DB2 OLAP Server.

With each of the queries we experienced a performance improvement. The detailed performance results can be seen in Appendix B, “Hybrid Analysis query performance results” on page 661. A summary of the results is available in Table 10-8 on page 473.

Five different Hybrid Analysis scenarios were considered: Hybrid 1 to Hybrid 5. Within each of these Hybrid Analysis scenarios, two query types were considered.

- **Query 1**: In these queries, the non-hybrid dimensions are at a high level (generation two) of their hierarchies.
- **Query 2**: In these queries, the non-hybrid dimensions are at a low level (level zero) of their hierarchies.

Both query types were run for each Hybrid Analysis scenario. This generated a number of query workloads which are detailed below. The name for each query workload is the name of the query type prefixed with the name of the hybrid scenario. For most query workloads, there are variations. Each variation is named by using an alphabetic suffix.

Hybrid 1 (H1)

The dimension with the greater cardinality is *PRODUCT*, and with there being over 10,000 products in the dimension table, the greatest benefit in terms of reducing the size of the MOLAP database and the time it takes to calculate that database would be in putting the leaf level of *PRODUCT* into relational as is shown in Figure 10-29. In this figure, each of the dimensions and their hierarchies are represented. Those members of the hierarchy that are inside the area marked with the thick line are in MOLAP, and those members of the hierarchy outside of the area (just item in this case) are in DB2.

This and the following Hybrid diagrams only show the levels in each hierarchy that are contained within the MOLAP or HOLAP space. They do not show the additional lower levels that exist in the relational star schema itself.

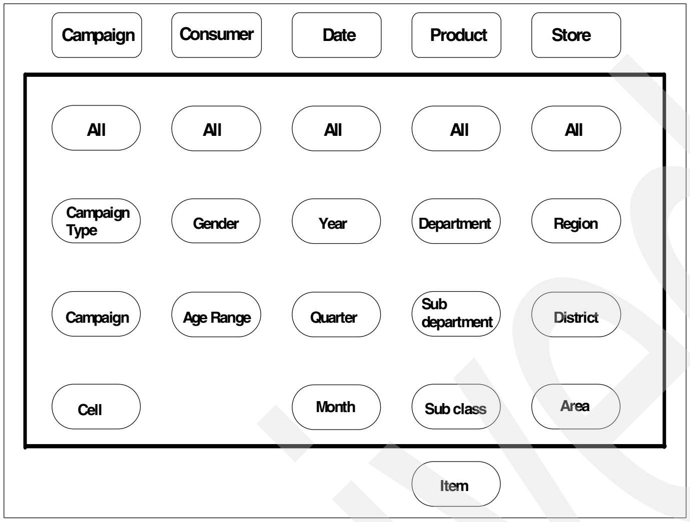


Figure 10-29 Hybrid 1

The queries that were run for the Hybrid 1 scenario are as follows:

- **H1_Query 1:**

  In this query, *ITEM_DESC* is in relational and other dimensions are at a high level.

  This query looks at the sale of shampoo products in the east region resulting from new product introduction campaigns in 2000 and 2001, comparing sales figures for females and males.

  The members selected from generation two of each of the other hierarchies are described in Table 10-3.

Table 10-3 H1_Query1

DIMENSION	MEMBER
CAMPAIGN	New Product Introduction
CONSUMER	Male, Female
DATE	2000, 2001
STORE	East

This query involves the user drilling down on *SHAMPOO* to retrieve the relational data. The measure used in each of the queries is sales.

Figure 10-30 shows the query as it would be in the spreadsheet client prior to the drill down on *SHAMPOO*.

New Product Introduction		Sales			
		2000		2001	
		Female	Male	Female	Male
East	SHAMPOO	4260.71	777.12	4185.48	1026.01
	HAIRCARE	64587.38	14126.55	64116.2	15056.03
	BODYCARE	104057.3	23101.55	102270.3	23615.66
	PRODUCT	746673.1	168491.4	744618.2	170266.4

Figure 10-30 H1_Query 1

- **H1_Query 2:**

  In this query, *ITEM_DESC* is in relational and other dimensions are at a low level.

  This query looks at the sales of luxury shower products across the cities in California resulting from a campaign targeting young single men. The report looks at the months in the first quarter of 2001 and breaks the sales down into three consumer groups, to see which products were purchased by the different consumer groups.

The members selected from level zero of each of the other hierarchies are described in Table 10-4.

Table 10-4 H1_Query 2 members

DIMENSION	MEMBER
CAMPAIGN	Luxury Shower Campaign_Young Single Guys
CONSUMER	Male_less than 19, Male_19-25, Male_26-35
DATE	January, February, March from 2001
STORE	Each of the cities in California: San Jose, San Francisco, Los Angeles, Sacramento, San Diego

For the *Product* dimension, six members from within the luxury shower sub-class were selected, these members would need to be retrieved from the relational database.

Figure 10-31 shows a subset of the report, it includes only one month from the *DATE* dimension and two cities from the *STORE* dimension.

Luxury Shower Campaign_Young Single Guys			Sales	
			January 2001	
		Male_less than 19	Male_19-25	Male_26-35
San Jose	O/L LUXURY SHOWER 2IN1	790	560	218
	O/L LUXURY SHOWER FOR MEN	942	989	764
	O/L LUXURY SHOWER LEMON	7	2	2
	O/L LUXURY SHOWER ORIGINAL	748	1212	921
	O/L LUXURY SHOWER SENSITIVE	231	227	154
	O/L LUXURY SHOWER SPORT	877	1032	576
San Francisco	O/L LUXURY SHOWER 2IN1	560	417	312
	O/L LUXURY SHOWER FOR MEN	1539	1927	1431
	O/L LUXURY SHOWER LEMON	12	6	7
	O/L LUXURY SHOWER ORIGINAL	936	992	1046
	O/L LUXURY SHOWER SENSITIVE	493	408	356
	O/L LUXURY SHOWER SPORT	1076	1597	877

Figure 10-31 H1_Query 2

Hybrid 2 (H2)

In this scenario, the bottom two levels of *PRODUCT* (item and sub class) are put into relational, as is shown in Figure 10-32.

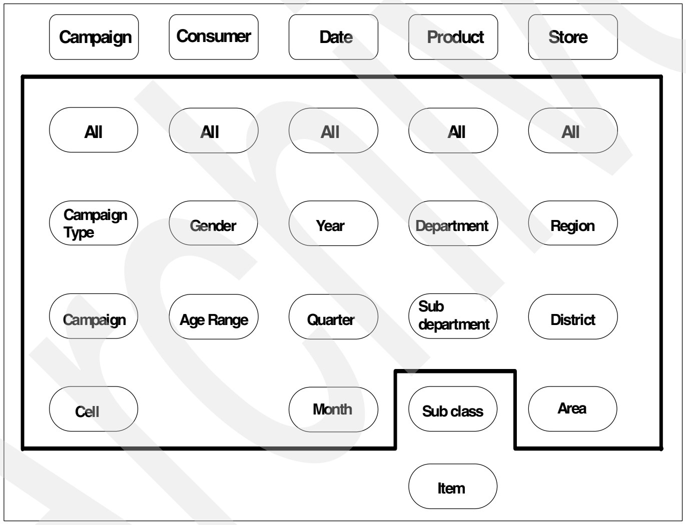


Figure 10-32 Hybrid 2

The queries that were run for the Hybrid 2 scenario are:

- **H2_Query 1:**

  In this query, *ITEM_DESC* and *SUB_CLASS_DESC* are in relational and other dimensions are at a high level.

  The members selected from generation two of each of the other hierarchies are as specified in Table 10-3 on page 461. The query is very similar to the one shown in Figure 10-30 on page 462.

This time, however, the query is split into two parts, as there are now two levels of the *PRODUCT* dimension in relational. H2_Query 1a involves a drill down on *HAIRCARE* and H2_Query 1b involves a drill down on *SHAMPOO* as before.

- **H2_Query 2:**

  In this query, *ITEM_DESC* and *SUB_CLASS_DESC* are in relational and other dimensions are at a low level.

  The members selected from level zero of each of the other hierarchies are as specified in Table 10-4 on page 462. The query is very similar to the one shown in Figure 10-31.

  This time again the query is split into two parts as there are now two levels of the *PRODUCT* dimension in relational. H2_Query 2a includes two level one members, *Stand Shower* and *Luxury Shower*, from the *PRODUCT* dimension. H2_Query 2b is identical to H1_Query 2.

  A subset of H2_Query 2a is shown in Figure 10-33. All of the cities are shown but again only one month is shown in the figure.

Luxury Shower Campaign_Young Single Guys				Sales
			January 2001	
		Male_less than 19	Male_19-25	Male_26-35
San Jose	STAND SHOWER	17	23	36
	LUXURY SHOWER	3872	4537	4312
San Francisco	STAND SHOWER	12	18	27
	LUXURY SHOWER	4879	5971	5562
Los Angeles	STAND SHOWER	13	14	7
	LUXURY SHOWER	4989	5978	5627
Sacramento	STAND SHOWER	6	9	4
	LUXURY SHOWER	1328	1542	1626
San Diego	STAND SHOWER	2	7	6
	LUXURY SHOWER	2748	2955	3062

Figure 10-33 H2_Query 2a

Hybrid 3 (H3)

In this scenario, the leaf level of two dimensions were placed outside of the MOLAP database. *PRODUCT* is the clear choice for one of the dimensions because of the number of items. The choice for the second dimension was not such an obvious one in this particular model, as the cardinality across the other dimensions was not so significant and was similar in each dimension. Usually there would be a clear choice for which dimension might next be enabled for hybrid. *STORE* was selected as an example of a second dimension, as is shown in Figure 10-34.

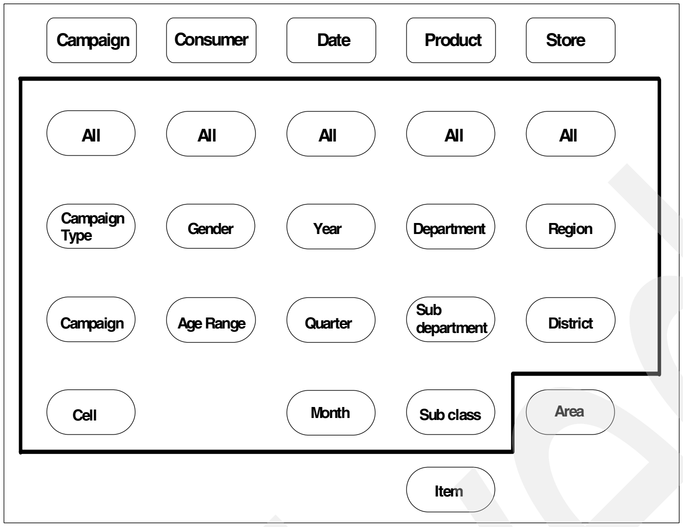


Figure 10-34 Hybrid 3

The queries that were run for the Hybrid 3 scenario are:

- **H3_Query 1**

  In this query, *ITEM_DESC* and *AREA_DESC* are in relational and other dimensions are at a high level.

  The members selected from generation two of each of the other hierarchies are as specified in Table 10-5. It is almost the same as before, the only change being for the *STORE* dimension.

Table 10-5 H3_Query 1

DIMENSION	MEMBER
CAMPAIGN	New Product Introduction
CONSUMER	Male, Female
DATE	2000, 2001
STORE	Florida

This query also now needs to have two parts, one being a drill down on the *PRODUCT* dimension and one being a drill down on the *STORE* dimension.

H3_Query 1a looks like Figure 10-35 prior to the drill downs.

New Product Introduction		Sales			
		2000		2001	
		Female	Male	Female	Male
Florida	SHAMPOO	1169.69	202.41	964.39	322.45
	HAIRCARE	15917.45	3547.23	16051.52	3664.21
	BODYCARE	25394.17	5701.56	25127.39	5770.65
	PRODUCT	182551.7	41659.74	182666	41996.07

Figure 10-35   H3_Query 1a

H3_Query 1a is very similar to H1_Query 1 in that it involves a drill down on *SHAMPOO*. However this time the *STORE* dimension member selected is Florida instead of East. A selection of shampoos are selected from those displayed, and then H3_Query 1b is run which is a drill down on *Florida*. The query prior to the drill down on *Florida* is shown in Figure 10-36. Some of the products listed were introduced in 2000 and some in 2001, hence the appearance on n/a in the report which stands for non-applicable.

New Product Introduction		Sales			
		2000		2001	
		Female	Male	Female	Male
Florida	O/L ALL-IN ONE SHAMPOO	121	46	n/a	n/a
	O/L BALANCE SHAMPOO	177	12	n/a	n/a
	O/L DIFFICULT SHAMPOO	n/a	n/a	54	2
	O/L DRY SHAMPOO	52	7	n/a	n/a
	O/L EXTRA SHAMPOO	n/a	n/a	79	15
	O/L NORM SHAMPOO	187	37	n/a	n/a
	O/L SPECIAL MOIST SHAMPOO	49	3	n/a	n/a
	O/L SUPER SHAMPOO	n/a	n/a	55	14
	O/L TREATMENT SHAMPOO	n/a	n/a	103	28

Figure 10-36   H3_Query1b

- **H3_Query 2**

In this query, *ITEM_DESC* and *AREA_DESC* are in relational and other dimensions are at a low level.

The members selected from level zero of each of the other hierarchies are as specified in Table 10-4 on page 462.

The query is exactly the same as H1_Query 2 as shown in Figure 10-31 because this already includes both *ITEM_DESC* and *AREA_DESC*.

Hybrid 4 (H4)

In this scenario, the bottom two levels of *PRODUCT* and the leaf level of *STORE* are put into relational as is shown in Figure 10-37.

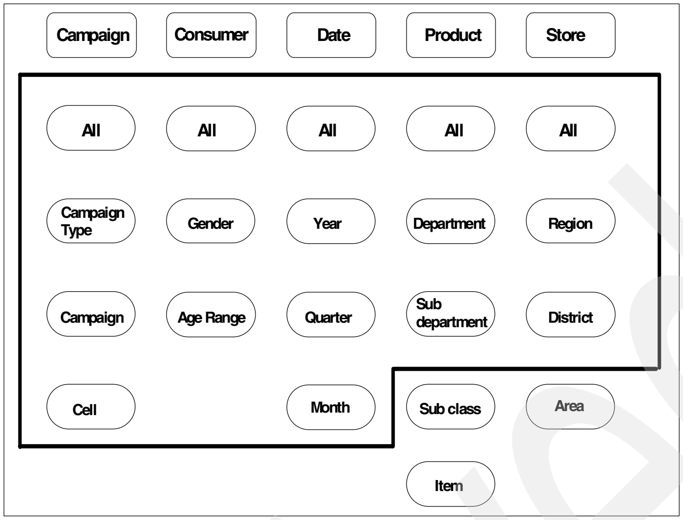


Figure 10-37 Hybrid 4

The queries that were run for the Hybrid 4 scenario are:

- **H4_Query 1:**

  In this query *ITEM_DESC* and *SUB_CLASS_DESC* from the *PRODUCT* dimension and *AREA_DESC* from the *STORE* dimension are in relational and other dimensions are at a high level.

  The query is again very similar to before, but this time three drill down queries will be performed. Review H3_Query 1a in Figure 10-35 on page 466. The initial report will need to be one level higher than this in the *PRODUCT* dimension. The three drill down queries that will be performed, therefore, will be:

  – Drill down on *HAIRCARE* (H4_Query 1a). Select only *SHAMPOO*.

  – Drill down on *SHAMPOO* (H4_Query 1b). Then select the shampoos as shown in Figure 10-36 on page 466 and finally

  – Drill down on *Florida* (H4_Query 1c).

- **H4_Query 2:**

  In this query *ITEM_DESC* and *SUB_CLASS_DESC* from the *PRODUCT* dimension and *AREA_DESC* from the *STORE* dimension are in relational and other dimensions are at a low level.

  Query H4_Query 2a is identical to H2_Query 2a as shown in Figure 10-33 on page 464. This includes *SUB_CLASS_DESC* and *AREA_DESC*.

Query H4_Query 2b is identical to H1_Query 2 as shown in Figure 10-31 on page 463 as this includes *ITEM_DESC* and *AREA_DESC*.

Hybrid 5 (H5)

In this scenario the bottom two levels of both *PRODUCT* and *STORE* are put into relational, as is shown in Figure 10-38.

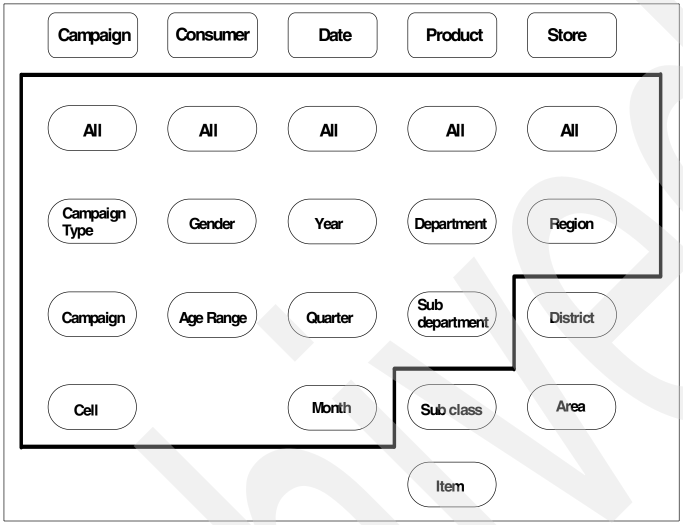


Figure 10-38 Hybrid 5

The queries that were run for the Hybrid 5 scenario are:

- **H5_Query 1:**

  In this query, *ITEM_DESC* and *SUB_CLASS_DESC* from the *PRODUCT* dimension and *AREA_DESC* and *DISTRICT_DESC* from the *STORE* dimension are in relational and other dimensions are at a high level.

  The query is again very similar to before, but this time four drill down queries will be performed. Review again H3_Query 1a in Figure 10-35 on page 466. The initial report will need to be one level higher than this in both the *PRODUCT* and *STORE* dimension. The four drill down queries that will be performed therefore will be:

  – Drill down on *HAIRCARE* (H5_Query 1a). Select only *SHAMPOO*.

  – Drill down on *SHAMPOO* (H5_Query 1b). Then select the shampoos as shown in Figure 10-36 on page 466.

  – Drill down on *East* (H5_Query 1c). Select *Florida*.

  – Drill down on *Florida* (H5_Query 1d)

- **H5_Query 2:**

  In this query *ITEM_DESC* and *SUB_CLASS_DESC* from the *PRODUCT* dimension and *AREA_DESC* and *DISTRICT_DESC* from the *STORE* dimension are in relational and other dimensions are at a low level.

  For this query, there will be four query workloads to take into account the introduction of *DISTRICT_DESC*.

  H5_Query 2a will be the same as H1_Query 2 shown in Figure 10-31 on page 463.

  H5_Query 2b will be the same as H2_Query 2a shown in Figure 10-33 on page 464.

  H5_Query 2c will be at the district level and will be as shown in Figure 10-39.

Luxury Shower Campaign_Young Single Guys			Sales	
			January 2001	
		Male_less than 19	Male_19-25	Male_26-35
California	O/L LUXURY SHOWER 2IN1	3599	2898	1097
	O/L LUXURY SHOWER FOR MEN	4710	4945	3834
	O/L LUXURY SHOWER LEMON	43	16	23
	O/L LUXURY SHOWER ORIGINAL	3749	6063	4612
	O/L LUXURY SHOWER SENSITIVE	1152	1139	784
	O/L LUXURY SHOWER SPORT	4382	5326	2947

Figure 10-39 H5_Query 2c

  H5_Query 2d will also be at the district level and will be as shown in Figure 10-40.

Luxury Shower Campaign_Young Single Guys				Sales
			January 2001	
		Male_less than 19	Male_19-25	Male_26-35
California	STAND SHOWER	50	71	80
	LUXURY SHOWER	17,816	20,983	20,189

Figure 10-40 H5_Query 2d

Query results

These query workloads were then run with each of the different levels of Hybrid Analysis enabled. The SQL that Hybrid Analysis generated was captured by using the Hybrid Analysis trace functionality within DB2 OLAP Server. A logging level of 2 was used to capture the SQL.

The SQL that Hybrid Analysis generates can be copied and pasted into DB2 Visual Explain to see which tables will be accessed.

For example, consider H1_Query 1, which involves a drill down on *SHAMPOO*. Hybrid Analysis will generate two queries for this, the first to discover the member names for the children of *SHAMPOO*, and the second to fetch the data for those members. These two queries are shown in Example 10-4 and Example 10-5.

Example 10-4 Lookup member names for H1_Query 1

```
SELECT DISTINCT aa.DEPARTMENT_DESC, aa.SUB_DEPT_DESC, aa.SUB_CLASS_DESC,
aa.ITEM_DESC
FROMSTAR.PRODUCT aa
WHERE(aa.SUB_CLASS_DESC = 'SHAMPOO' AND aa.SUB_DEPT_DESC = 'HAIRCARE' AND
aa.DEPARTMENT_DESC = 'BODYCARE')
ORDERBY 1 ASC, 2 ASC, 3 ASC, 4 ASC
```

Example 10-5 Fetch data for H1_Query 1

```
SELECT DISTINCT aa.CAL_YEAR_DESC, ab.CAMPAIGN_TYPE_DESC, ab.CAMPAIGN_DESC,
ac.GENDER_DESC, ad.DEPARTMENT_DESC, ad.SUB_DEPT_DESC, ad.SUB_CLASS_DESC,
ad.ITEM_DESC, ae.REGION_DESC, SUM(af.TRXN_SALE_AMT)
FROM STAR.DATE aa, STAR.CAMPAIGN ab, STAR.CONSUMER ac, STAR.PRODUCT ad,
STAR.STORE ae, STAR.CONSUMER_SALES af
WHERE af.DATE_KEY = aa.IDENT_KEY
  AND af.COMPONENT_ID = ab.IDENT_KEY
  AND af.CONSUMER_KEY = ac.IDENT_KEY
  AND af.ITEM_KEY = ad.IDENT_KEY
  AND af.STORE_ID = ae.IDENT_KEY
  AND (((((aa.CAL_YEAR_DESC IN ( '2000' , '2001'  )) ))))
  AND (((ab.CAMPAIGN_DESC = 'Luxury Shower Campaign' AND ab.CAMPAIGN_TYPE_DESC
= 'New Product Introduction')))
  AND((((ac.GENDER_DESC IN ( 'Female' , 'Male'  )) ))))
  AND(((ad.SUB_CLASS_DESC = 'SHAMPOO' AND ad.SUB_DEPT_DESC = 'HAIRCARE' AND
ad.DEPARTMENT_DESC = 'BODYCARE')))
  AND (((ae.REGION_DESC = 'East')))
GROUP BY  aa.CAL_YEAR_DESC , ab.CAMPAIGN_TYPE_DESC , ab.CAMPAIGN_DESC ,
ac.GENDER_DESC , ad.DEPARTMENT_DESC , ad.SUB_DEPT_DESC , ad.SUB_CLASS_DESC ,
ad.ITEM_DESC , ae.REGION_DESC
ORDER BY 9 ASC, 5 ASC, 6 ASC, 7 ASC, 8 ASC, 4 ASC, 3 ASC, 2 ASC, 1 ASC
```

The first query is a lookup from the *PRODUCT* table, and as such, the result set will be taken directly from that table. However it is the second query that should benefit from being re-directed to the MQT.

Without the MQT, the query accesses the fact table and each of the dimension tables and has to perform many joins. This can be seen in the main section of the access plan graph from DB2 Explain, which is shown in Figure 10-41.

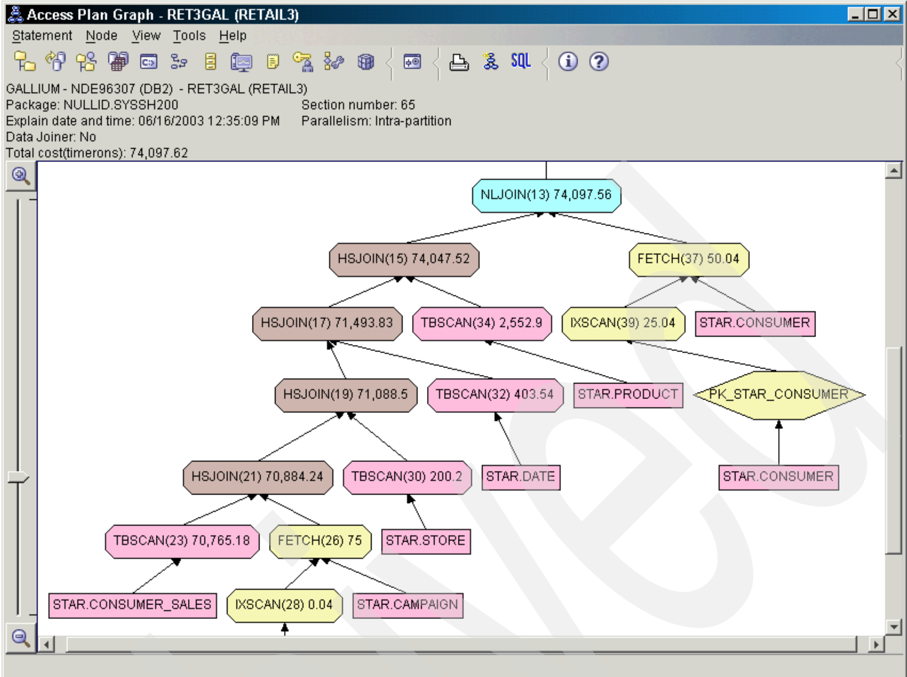


Figure 10-41 H1_Query 1 without MQT

With the MQT available to be used, the query is re-routed to the MQT. Figure 10-42 shows the bottom section of the DB2 access plan graph. The initial fetch was costed at 654.44 and after that there was very little cost involved, as the final cost for the query was reported as 654.5.

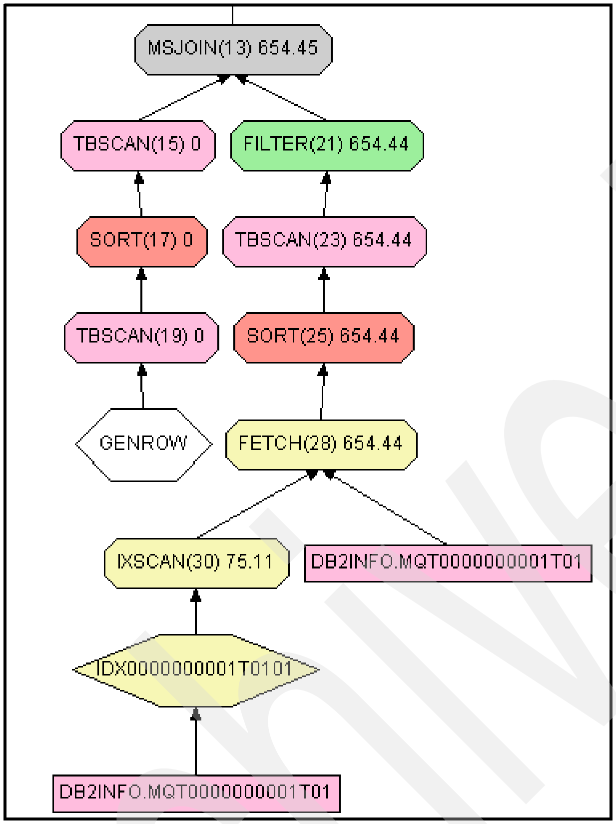


Figure 10-42 H1_Query 1 with MQT

The cost without the MQT is 74,097.62 timerons.

The cost with the MQT is 654.45 timerons.

The benefit of the MQT to the Hybrid Analysis user for this query is significant.

Each of the queries that have been described in this section were run and the performance results captured. The tests were not run under benchmark conditions, but they were run dedicated with no other jobs running at the same time. For each query Hybrid Analysis will generate two or more SQL statements. The performance results were recorded for each individual SQL statement (from here on referred to as *query*) within each Hybrid Analysis query (referred to in the charts as *query workload*). For each individual query the charts record the elapsed query time both with and without the MQT being available, and whether the query was re-routed to an MQT.

For example, consider H1_Query 1. This is a drill down on *SHAMPOO* into the relational data. Hybrid Analysis generates two queries in order to perform this operation: one to fetch the children of *SHAMPOO* and one to fetch the data. The performance results were charted and are shown in Table 10-6 and Table 10-7.

Table 10-6 H1_Query 1 without MQT

Query workload	Query ID	Elapsed time	Query re-routed?
H1_Query 1	35801	0.242	N
	35802	16.699	N

Table 10-7 H1_Query 1 with MQT

Query workload	Query ID	Elapsed time	Query re-routed?
H1_Query 1	35801	0.322	N
	35802	2.344	Y

As expected, the first query does not re-route, as it is just performing a query lookup on the *PRODUCT* table. However the second query does re-route and the elapsed time for the query is reduced from 17 seconds to 2 seconds.

The full results for each of the queries can be found in Appendix B, “Hybrid Analysis query performance results” on page 661.

The results in the appendix have been summarized and are presented in Table 10-8.

Table 10-8 Hybrid Analysis performance results

Query workload	Elapsed time without MQT	Elapsed time with MQT
H1_Query 1	16.941	2.666
H2_Query 1a	161.955	5.269
H2_Query 1b	12.416	0.827
H3_Query 1a	0.511	0.474
H3_Query 1b	0.901	0.526
H4_Query 1a	161.156	1.143
H4_Query 1b	11.77	0.867
H4_Query 1c	0.815	0.600

Query workload	Elapsed time without MQT	Elapsed time with MQT
H5_Query 1a	162.635	2.023
H5_Query 1b	12.023	0.939
H5_Query 1c	0.894	0.61
H5_Query 1d	0.841	0.52
H1_Query 2	3.727	5.03
H2_Query 2a	14.237	1.639
H2_Query 2b	1.093	2.248
H3_Query 2	2.092	3.449
H4_Query 2a	14.759	2.503
H4_Query 2b	2.424	3.462
H5_Query 2a	4.575	3.614
H5_Query 2b	13.597	2.652
H5_Query 2c	2.076	3.349
H5_Query 2d	1.125	2.097

Generally speaking, without MQTs, there is an expectation that the Query 2's will perform better than the Query 1's because of the level of aggregation required in the Query 1 queries (and of course the Query 2's also perform record selection). This is shown in the results for the majority of the query workloads. There are exceptions to this however because another factor that needs to be taken into account are the number of queries generated by Hybrid Analysis. For example, consider the H5 set of query workloads. H5_Query 1a requires the highest level of aggregation and without the MQT performs poorly even though only two queries are actually generated for this workload. However, for the other query workloads in H5 the Query 1's slightly outperform the Query 2's. For these other query workloads the level of aggregation is less in the Query 1 workloads and are less of a factor than the high number of queries being generated by these other Query 2 workloads.

The effect of the MQT here is to significantly improve the performance of the poorly performing Query 1's because some or all of the aggregation required is available in the MQT.

The very worst performing Query 1 queries (without the MQT) are H2_Query 1a, H4_Query 1a and H5_Query 1a. These are all 1a queries which means higher levels of aggregation, and they are all in the hybrid environments where more than one level of a dimension is in relational. Different indexes were applied and different results were achieved (with no MQT).

The point that was emphasized here was that with DB2 Cube Views the performance results were more consistent from the outset, without having to spend time doing performance analysis and creating numerous indexes in order to achieve the optimum results. With the MQT the performance for H2_Query 1a was improved from 161.955 seconds to 5.269 seconds. H4_Query 1a was improved from 161.156 seconds to 1.143 seconds and H5_Query 1a was improved from 162.635 seconds to 2.023 seconds.

When looking at the performance results in Table 10-8 on page 473 the key factor that comes across is the improvement in the consistency of the query response times when the MQT is available:

- Without the MQT query response times varied from 0.511 seconds to as much as 162.535 seconds.
- With the MQT the response time variation was reduced significantly with response times of between 0.474 seconds and 5.269 seconds. Note that the performance figures in table 10-8 relate to the query portion of the Hybrid Analysis query only and do not equate to the total response time experienced by the spreadsheet client user

The detailed results can be found in Table B-1 in Appendix B, “Hybrid Analysis query performance results” on page 661. Here it is interesting to see that for some of the Hybrid Analysis query workloads more than one of the SQL queries that get generated are able to be re-routed to the MQT. These queries are H2_Query 1b, H3_Query 1b, H4_Query 1b, H4_Query 1c, H5_Query 1b, H5_Query 1c and H5_Query 1d. Each of these query workloads generated two data fetch queries that could be re-routed to the MQT.

None of the Query 2 type queries generated SQL that involved re-routing more than once, although it is the Query 2 type queries that generate the most SQL statements. The more dimensions and levels that are in relational, the more queries Hybrid Analysis has to issue in order to confirm which table and column a data value relates to.

The worst performing Query 2 type queries (without the MQT) are H2_Query 2a, H4_Query 2a and H5_Query 2b. These are in fact all the same query, and they are also queries at the sub-class level of *PRODUCT* when both sub-class and item are in relational. With the MQT H2_Query 2a performance is improved from 14.237 seconds to 1.639 seconds. H4_Query 2a is improved from 14.759 to 2.503 seconds, and H5_Query 2b is improved from 13.597 to 2.652 seconds. Only the script that was generated by the Optimization Advisor was run, no additional database tuning was performed.

A few of the query workloads actually perform slightly worse with the MQT compared to without the MQT in this test. The reasons for this were not pursued at the time as the increase was only approximately 1 second. The query workloads that experienced this slight increase were all ones where higher numbers of queries were generated, were all Query 2 type queries, and were all ones at the lower levels of the hierarchies. A possible explanation for this could be related to the fact that the MQT in this example was larger than the fact table. The best results are usually achieved when the MQT is based on a slice of data at higher levels of the dimensions, resulting in an MQT that is smaller than the fact table. However in this example the MQT that was generated was positioned at a fairly low level in relation to the fact table, resulting in an MQT that was larger than the fact table.

DB2 OLAP Server calculation

By implementing Hybrid Analysis, the size of the outline is reduced, the size of the MOLAP database is reduced and the time taken to perform the calculation can be reduced dramatically.

For completeness, therefore, in terms of reviewing the performance benefits of the five Hybrid Analysis scenarios described here, the calculation times for each scenario was also recorded.

Table 10-9 summarizes these results.

Table 10-9 DB2 OLAP Server calculation times

	Calculation time (mins)	**Level zero blocks**	**Total blocks**	**Approximate database size**
Hybrid 1	27.2	377,097	1,746,773	2.7GB
Hybrid 2	5.0	64,619	289,679	700MB
Hybrid 3	11.8	181,554	712,020	1.3GB
Hybrid 4	1.6	26,529	108,229	300MB
Hybrid 5	0.7	11,072	38,122	120MB

A maximum batch window of 4 hours was specified for the calculation. The calculation of the database with everything in the MOLAP database (including item) did not complete within this time frame and was therefore canceled. The number of blocks in the database after the data load was already 3,634,703 which is significantly larger than each of the fully calculated databases with Hybrid Analysis enabled.

The calculation time was dramatically improved by placing item outside of the MOLAP database and into relational. Reducing the calculation time down to 27 minutes means that we have met our batch window objective comfortably without having to go any further.

10.5.3 Drill through reports

The drill through report scenario is very similar to the Hybrid Analysis one. With drill through a user is exiting the MOLAP cube and initiating an SQL query against the underlying relational database. The transition for the user however in this case is a clear one, in that the user has to request that a report be run from the list of reports that is presented to them. The performance expectation therefore may not be as high as for the Hybrid Analysis environment, because it is clear to the user that they are running a report as opposed to just drilling down through a dimension hierarchy. However, initiating a report from DB2 OLAP Server is still an interactive type of report and a user's patience only extends so far. If there are performance concerns with running drill through reports, then again DB2 Cube Views can help by providing MQTs that can be queried instead of having the query run against the base tables.

However, it is important to review carefully the content of the drill through reports that are being developed. Generally if the drill through to the relational database is to fetch lower levels of a hierarchy then this will probably, although by all means not necessarily, be implemented as a Hybrid Analysis solution rather than an Integration Server drill through report. Typically Integration Server drill through reports are written to access information that is outside of the hierarchy, for example to access text columns or additional dates.

If a drill through report accesses columns that are not available in the MQT then the MQT will not be able to be used. For example the MQT in our scenario does not go down to individual consumer or stores, therefore any report requesting data at these levels would not re-route to the MQT. Placing these lower levels in the cube in DB2 Cube Views in order to get them included in the MQT will significantly increase the size of the MQT. Similarly, if the drill through report requires a number of textual data columns to be included in the MQT, then again the size of the MQT may increase significantly. It is important to always consider the size increase implications of placing more columns and rows in the MQT.

The Optimization Advisor can be instructed to include additional columns in the MQT be defining them in DB2 Cube Views as related attributes or as additional lower levels of the hierarchy. Including them as another level in the hierarchy has significant implications for the size of the MQT and should be reviewed very carefully. The more practical implementation may be to include a few additional columns as related attributes.

When the Optimization Advisor is run, it will take into account any disk space limit that it has been given. If it estimates that to include the additional columns will result in this disk space limit being exceeded, then the script that it produces will not include one or more of these additional columns. It is important therefore to review the script that is produced by the Optimization Advisor to check what has been included.

In the following example, a drill through report is defined in Integration Server as shown in Figure 10-43. The definition means that when the user is at the intersection of *CELL_DESC* from the *CAMPAIGN* dimension and *SUB_CLASS_DESC* from the *PRODUCT* dimension, then they will have the opportunity to run a drill through report which will select item description, product brand code, the cell description about the campaign, and the component description (which specifies whether the campaign was a direct mail campaign, or a coupon or a floor discount and so on). The report that is produced will take as predicates the data values that come from the intersection point of campaign cell description and product sub class from which the user initiates the report.

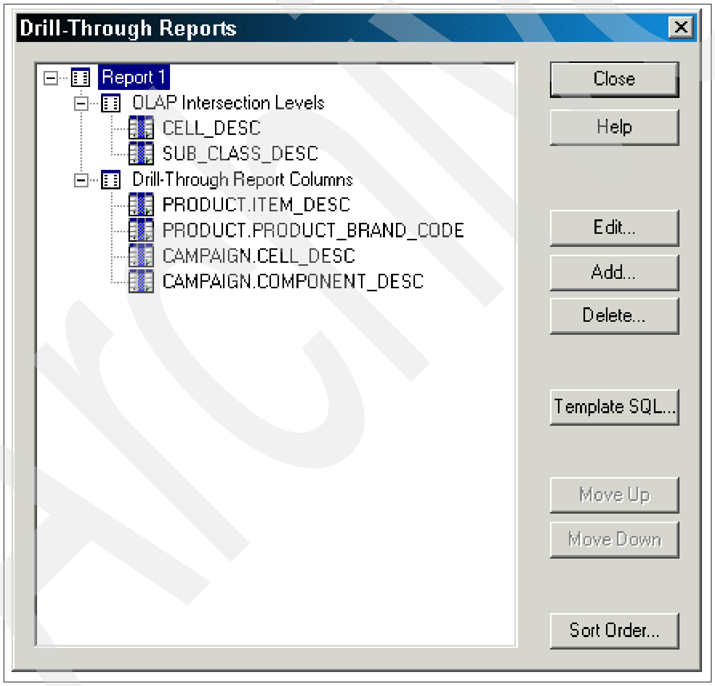


Figure 10-43   Integration Server drill through report definition

Neither the product brand code nor campaign component description are in the hierarchies defined for the cube in DB2 Cube Views. In order to have them included in the MQT these two columns were added as associated related attributes in DB2 Cube Views.

The Optimization Advisor should then be run and the new MQT created. In the Optimization Advisor there is no separate query type option to differentiate between Hybrid Analysis and drill through reports. For both types of queries the option that should be selected is drill through.

A subset of the report when it is run from the DB2 OLAP Server spreadsheet client is shown in Figure 10-44. In this example the user in the spreadsheet client has clicked on the cell intersection of *SHAMPOO* for the product subclass and of *Double Income No Kids* for the campaign cell. The resulting report lists the items within the shampoo subclass, the product brand code and the component description.

ITEM_DESC	PRODUCT_BRAND_CODE	CELL_DESC	COMPONENT_DESC
O/L ALL-IN ONE SHAMPOO	0	Double Income - No Kids	Mailer
O/L BALANCE SHAMPOO	0	Double Income - No Kids	Mailer
O/L DIFFICULT SHAMPOO	0	Double Income - No Kids	Mailer
O/L DRY SHAMPOO	0	Double Income - No Kids	Mailer
O/L EXTRA SHAMPOO	0	Double Income - No Kids	Mailer
O/L NORM SHAMPOO	0	Double Income - No Kids	Mailer
O/L SPECIAL MOIST SHAMPOO	0	Double Income - No Kids	Mailer
O/L SUPER SHAMPOO	0	Double Income - No Kids	Mailer
O/L TREATMENT SHAMPOO	0	Double Income - No Kids	Mailer
CORNFIELD BODY BUILD	1	Double Income - No Kids	Mailer
CORNFIELD DANDRUFF	1	Double Income - No Kids	Mailer

Figure 10-44 Integration Server drill through report sample

The template SQL that Integration Server generates for the drill through report can be accessed by clicking the **Template SQL** button in the **Drill-Through Reports** window. This button can be seen in Figure 10-43 on page 478. The template SQL can be copied into DB2 Visual Explain and modified to substitute the actual column names and data values for the template containers in the template.

Figure 10-45 shows the bottom section of the DB2 access plan graph when there is no MQT available. Here it is clear that the query is accessing the base tables.

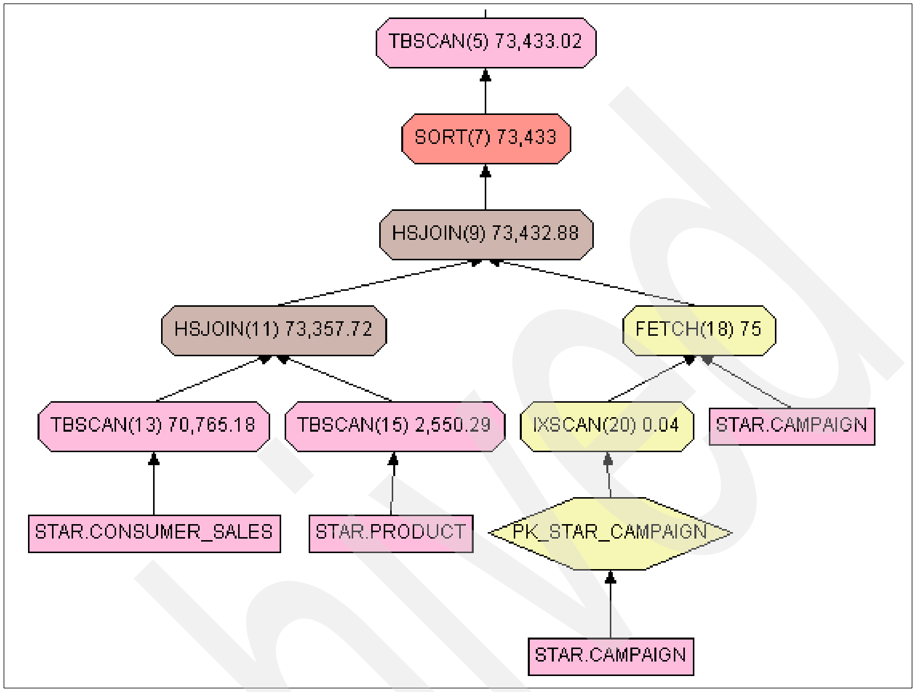


Figure 10-45 Integration Server drill through report without MQT

Figure 10-46 shows the bottom section of the access plan graph when there is an MQT available. The query is being re-routed to access the MQT.

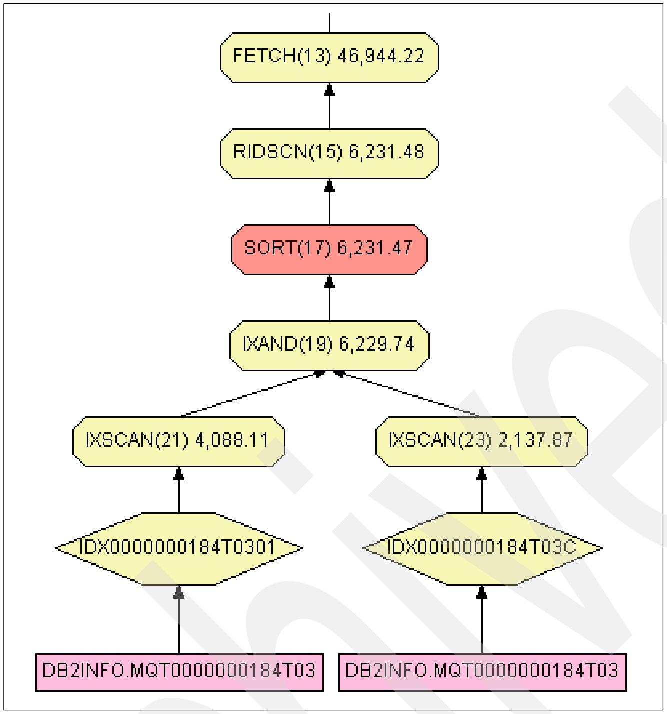


Figure 10-46 Integration Server drill through report with MQT

Table 10-10 shows the relative performance costs that were provided by DB2 Explain.

Table 10-10 Integration Server drill through report performance cost

Total cost timerons with MQT	46,944.27
Total cost timerons without MQT	73,433.12

Yet again a significant cost reduction is demonstrated when the query is re-routed to the MQT.

10.6 Conclusions

From the examples in this section, it is clear that DB2 Cube Views and DB2 OLAP Server work very well together. At all of the points where DB2 OLAP Server issues queries to DB2, performance benefits may be gained if the DB2 optimizer is able to re-route the query to an MQT. The fewer the rows in the MQT compared to the base fact table, the greater the gain in performance. The optimal slices of data for this MQT will be dependent upon the cardinality of the data.

Data load, Hybrid Analysis, and Integration Server drill through reports can all benefit from having their queries re-routed to an appropriate MQT.

Hybrid Analysis is of particular interest because there are two benefits. Firstly, implementing Hybrid Analysis results in a smaller database and potentially a dramatic decrease in calculation time. Secondly, the introduction of DB2 Cube Views can assist in improving the performance of the relational queries that Hybrid Analysis generates.

It may be that there is an existing DB2 OLAP Server MOLAP database for which there is a requirement to reduce the calculation time. By placing the lowest level of the dimension with the greatest cardinality in DB2, the calculation time can be significantly reduced and the Hybrid Analysis relational queries can be assisted with DB2 Cube Views.

Alternatively, it may be that there is a level of a dimension that users wish to include in the OLAP database. In the past it has not been possible to place this level in the MOLAP database as it would cause the size of the OLAP database to increase to an unmanageable size. With the assistance of DB2 Cube Views it may now be possible to include that additional data as a relational layer in a hybrid environment.

Or there may be an existing hybrid environment and DB2 Cube Views can assist by enabling an additional level in the same or another dimension to be included in the relational part of the hybrid space.

The considerations for Hybrid Analysis do not change with DB2 Cube Views, but DB2 Cube Views is an enabler for Hybrid Analysis and as such offers administrators greater flexibility in how they design their OLAP databases.

The ability to exchange metadata in both directions between DB2 Cube Views and Integration Server increases productivity. Once the metadata exists in one product, it can be sent across the bridge to the other product, thereby enabling a fast start in that second product.

11

Accessing DB2 dimensional data using Cognos

This chapter describes some scenarios for deployment and discusses benefits before showing how to implement and use the metadata bridge. The objective is to talk in a little more depth about the way the bridge carries out the mapping, so the reader can understand where and why they can benefit from the bridge, and know where the limits are.

11.1 The Cognos solution

Cognos Business Intelligence is integrated software that delivers all of the BI capabilities that organizations require to turn their data into meaningful information. One of its greatest values for both IT and business users is that, as information needs evolve, organizations can easily expand their use of business intelligence with one vendor. IT doesn't have to worry about integration issues, and users don't have to learn new systems.

Cognos Business Intelligence is easy to use, with all reporting and analysis capabilities accessible from one Web-based portal. Users can select reports, customize them, analyze information, and share information with the same facility as using the Web. For IT departments, Cognos BI is easy to deploy and administer, and is built for the demands of enterprise-scale environments.

Reporting and analysis connects users to the business. It gives them easy-to-use information for clear understanding of day-to-day operations and trends, while providing them a path to explore results to the necessary level of detail.

Cognos provides integrated business intelligence capabilities that deliver reports and analyses using the right format, delivery method, and environment to suit the particular business challenges a user faces.

11.1.1 Cognos Business Intelligence

Cognos Business Intelligence, shown in Figure 11-1, offers:

- Reporting and analysis
- Event detection
- Scorecards
- Analytic applications

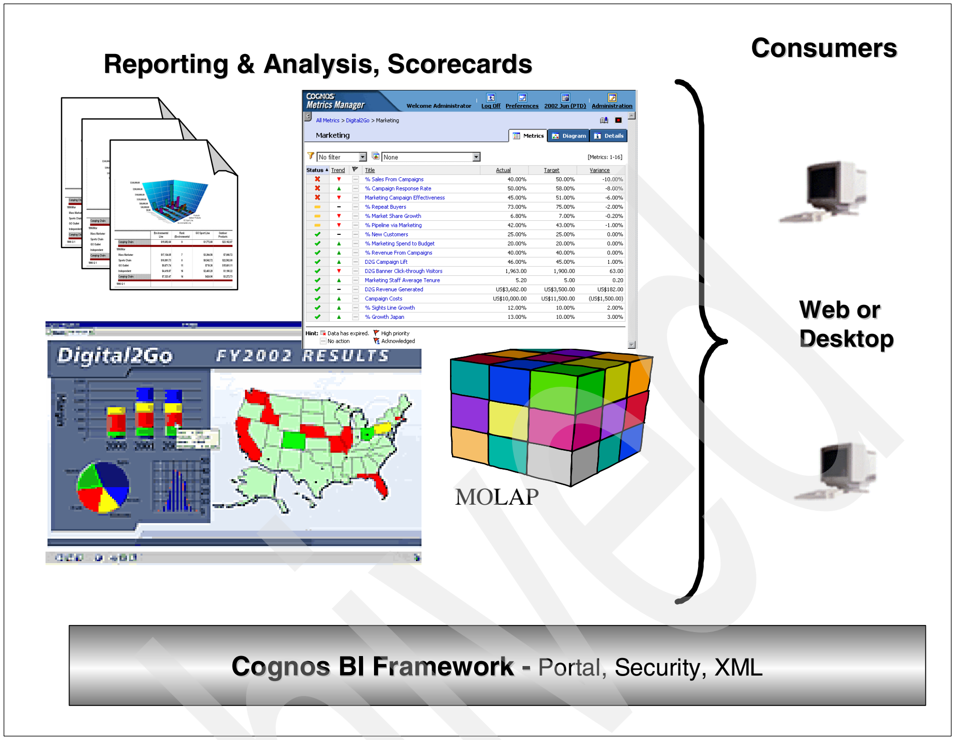


Figure 11-1 Cognos Business Intelligence

Reporting and analysis

Cognos Impromptu delivers the widest possible range of management reports to users in any large organization. Impromptu's advanced report formats, built-in report scheduling and bursting, and zero-footprint Web-based delivery of presentation-ready reports make it the software of choice for enterprise reporting.

Cognos PowerPlay delivers the world's leading Web-based online analytical processing (OLAP) capabilities. Users can analyze quickly and easily the dimensions and indicators on which success are measured in any business framework, for example, sales by week, month, or quarter, across any combination of region or sales channel.

Cognos Query lets novice and experienced users directly access corporate data resources for real-time data exploration. Using only a Web browser, users can navigate suites of published queries, saving and modifying them as required to meet their information needs.

Cognos Query balances ease of use and power. Query users and authors can service themselves, quickly and easily accessing published sets of queries using the fully integrated Cognos portal. Users can run and modify existing queries, design new ones, and optionally publish queries back to the portal for future use. Users can take advantage of advanced features, like query-to-query linking, prompts, and the ability to define sophisticated calculations in the browser environment.

Cognos Reporting and Analysis includes the advanced data visualization capabilities of Cognos Visualizer. Users can gain immediate understanding of business performance trends, issues, and opportunities by viewing complex metrics using highly visual, coordinated displays.

Cognos delivers enterprise dashboards through Cognos Visualizer Series 7, which uses powerful visual analysis (*visualizations*) to communicate complex business data sets quickly and intuitively. Cognos Visualizer dashboards use a diverse selection of maps and charts to display multiple measures simultaneously, enabling decision-makers to perform rich visual analysis. Cognos Visualizer dashboards leverage the enterprise-ready Cognos Series 7 framework, meaning easy installation, reliable secure business intelligence information, and broad enterprise deployment.

Event detection

In concert with Cognos' powerful reporting and analysis capabilities, Cognos NoticeCast provides the ability to push information to users, allowing them to focus quickly on what needs immediate attention. NoticeCast delivers personalized, high-value information based on defined events, providing automatic monitoring of performance management. Within delivered alerts, NoticeCast delivers business intelligence content and operational issues. Any user, anywhere across the organization or value chain, can monitor key events using e-mail notifications and alerts that push business-critical information to them.

Scorecards

Cognos Series 7 delivers enterprise scorecarding through Cognos Metrics Manager. Cognos Metrics Manager lets companies model plans or strategies as a set of inter-connected performance indicators. This can communicate goal-driven metrics to thousands of employees across your organization. Cognos Metrics Manager is next-generation scorecarding technology that is an essential component of corporate performance management. Your company can move from its plans, to monitoring performance, to drilling down on information to understand the issues causing the results.

The entire Cognos Series 7 solution is underpinned by the Cognos BI Framework, which delivers business intelligence via a customizable Web-based portal, with common security and metadata, and XML-based Web services for application integration and extension.

Analytic applications

Cognos offers an integrated set of analytic applications based on its Cognos Series 7 architecture. These applications come with pre-built reports, performance indicators, and connections with underlying data sources from ERP vendors. They package reporting and analysis, scorecarding, and planning capabilities for the areas of customer, supply chain, and financial/operational analysis.

Cognos Analytic Applications offer quick time to results, and a BI foundation that can be easily customized with Cognos business intelligence technology.

11.2 Architecture and components involved

The Cognos bridge, as shown in Figure 11-2, the DB2 Dimensional Metadata Wizard, comes with Cognos Impromptu and is part of the default installation. The bridge generates an Impromptu catalog, reports, and the Transformation Server model and Impromptu queries. The bridge will allow users to either build the Transformer files, the Impromptu files, or both. The bridge connects directly to DB2 Cube Views via the CLI interface, using the Cognos defined native gateway connection.

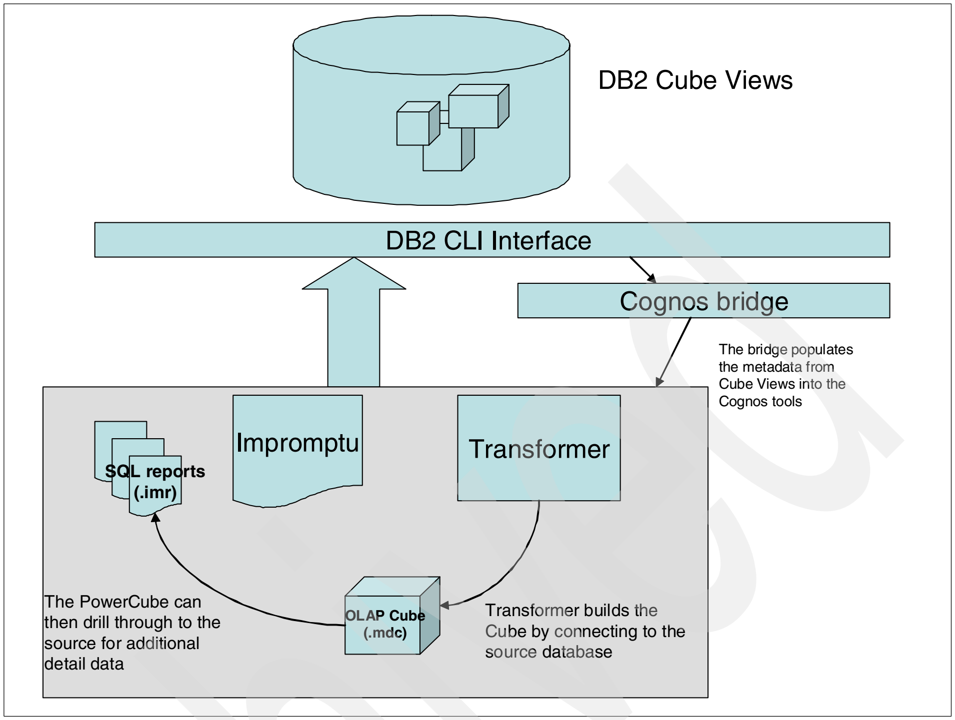


Figure 11-2 Architecture with DB2 Cube Views

Impromptu (see Figure 11-3) is SQL-based report writing tool and delivers the widest possible range of management reports to users in any large organization.

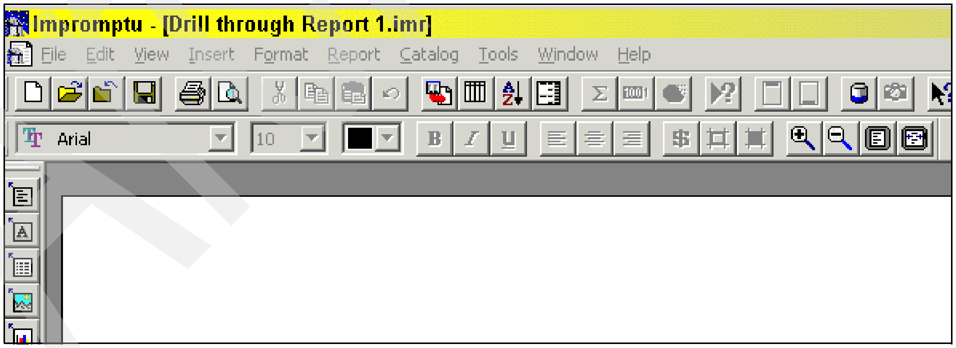


Figure 11-3 Impromptu window

Impromptu creates catalogs and reports. A catalog contains all the information necessary for Impromptu to access and retrieve information from a relational database. A catalog does not store data, but provides Impromptu with a business view of the data. A catalog also contains information about what database to access, where the database is stored, and how the tables in the catalog are joined. This information is organized in a business view of the reporting environment without the user having to have knowledge of the underlying source, by using:

- Folders: Meaningful groups of information representing columns from one or more tables
- Columns: Individual data elements that can appear in one or more folders
- Calculations: Expressions used to compute required values from existing data
- Conditions: Used to filter information so that only a certain type of information is displayed
- Prompts: Pre-defined selection criteria prompts that users can include in reports they create
- Other components: Such as metadata, join information, and user security.

An Impromptu report is a view of the current data in your database that is organized and formatted the way you want it. The data you see in your report depends on the data you can access from your catalog. A report is a focused answer to a business problem, based on SQL.

Cognos Impromptu delivers managed reporting for consistent, fact-based decision-making. Managed reporting enables report authors to create reports drawn from any data source. These reports can then be delivered to report consumers. Report authors use Impromptu to create business-context reports. Report authors can author virtually any report using Impromptu's superior frame-based reporting interface. Report data can come from any source, and reports can be deployed to Impromptu users across LANs and WANs, as well as to mobile users.

Cognos Transformation Server draws information from relational databases to model and build multidimensional PowerCubes. PowerCubes are data sets that can contain over a billion rows of data and 2 million categories (members). Business rules and calculations (for example, percentage growth and market share change) can be built right into them, and time series analysis is delivered automatically. PowerCubes and reports can be deployed to Web clients, or to Windows and Excel clients, all using the same application server.

Cognos PowerPlay's Transformation Server delivers OLAP designers an advanced environment for creating highly compact, fast, robust PowerCubes (MOLAP). Designers build Transformer models, which are a multidimensional representation of a business or enterprise comprising the structures and specifications for one or more PowerCubes. The information for a model is stored in a model file *.pyi* (binary) or *.mdl* (text). The model contains:

- Metadata of the data source(s)
- Dimensions: A broad grouping of descriptive data about a major aspect of a business, such as products, dates, or markets. Each dimension includes different levels of categories in one or more drill-down paths.
- Measures: Performance indicators that are quantifiable and used to determine how well a business is operating. Measures can be Revenue, Revenue/Employee, and Profit Margin.
- PowerCube(s) definitions: PowerCubes can contain all or portions of dimensions, measures, and other business rules.
- Other rules that define how the data will be shown to the users of the PowerCube(s)

Designers can create measure folders which group measures into business rules which allow users to navigate through various measure rollups and drill down to view the lower level measures in their OLAP reports. Designers can define multidimensional data structures visually, using standard drag and drop actions; define dimensions, levels, categories (members), or measures by dragging and dropping data items appropriately. This applies to advanced features like alternate drill downs, calculated measures, measure allocations, and calculated categories.

Designers can easily define date periods to analyze data across time, from years down to individual days. Designers can define their own custom time periods as required, and easily create relative time period definitions.

Cognos PowerPlay, users at any business or technical skill level in a company can perform their own multidimensional analysis, create reports, and share them for better decision-making: Cognos PowerPlay provides both a multidimensional data store and a feature rich end-user experience. PowerPlay supports many third-party OLAP servers such as DB2 OLAP. Its multi-server, load-balanced architecture is easy to install and manage, and it scales to thousands of users.

Cognos Metrics Manager, Visualizer, and Cognos Query, shown in Figure 11-4, can leverage the metadata imported by the bridge and enhanced with Impromptu and PowerPlay.

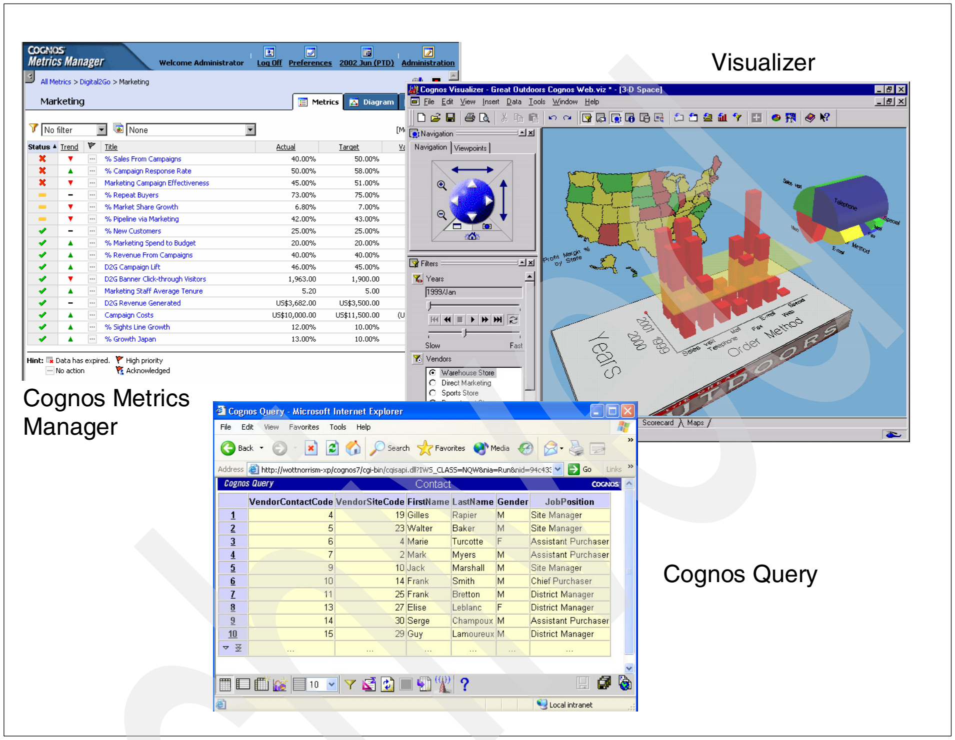


Figure 11-4 Cognos Metrics Manager, Visualizer and Cognos Query

The Cognos bridge, DB2 Dimensional Metadata Wizard, generates Impromptu catalogs and reports to give you a fast start in leveraging this dimensional metadata. The Impromptu Catalog can be used to build reports that will use the materialized query tables (MQTs), thus greatly improving performance.

You can generate Transformer models and Impromptu queries (.iqd) that reflect your dimensional design. This helps you build, more quickly and easily, PowerCubes that can provide drill-through access to the underlying DB2 data. This allows you to have a starting point from which to build and expand your business intelligence environment.

The Cognos DB2 Dimensional Metadata wizard imports a significant amount of the metadata defined within the cube model. This import significantly reduces the implementation time of Cognos into an environment using DB2 Cube Views. Business rules can be transferred to Cognos automatically, and any missing objects can be added easily. The Cognos tools have a GUI interface, which makes adding any additional functionality or business rules or missing objects from the import an easy task.

Terminology within the Cognos tool set closely matches the terms in DB2 Cube Views, making it an easy transition from one environment to the other. Resulting reports, catalog and model are easily updated and manipulated to meet the requirements of the MQTs.

Drill through to source from a PowerCube is automatically re-directed to the MQT when an upper level in a dimension is selected – providing the MQT exists! Leveraging the MQT(s) greatly reduces query execution time in Impromptu. As the user continues to drill down within the PowerCube, the automatic re-direction of the report to an MQT will not necessarily occur. Since the lower level detail transaction drill through reports will be filtered using a restricted SQL statement in their queries, their performance will be by default faster than if there was no filtering done. A middle ground is reached whereby the cost of building an MQT is balanced by the benefit of faster query execution.

11.3 Implementation steps

The major steps to implement the metadata bridge, described in Figure 11-5, are as follows:

- Step 1: Connect to DB2 Cube Views.
- Step 2: Import metadata and check the result.
- Step3: Open Transformer Model and build the PowerCube.

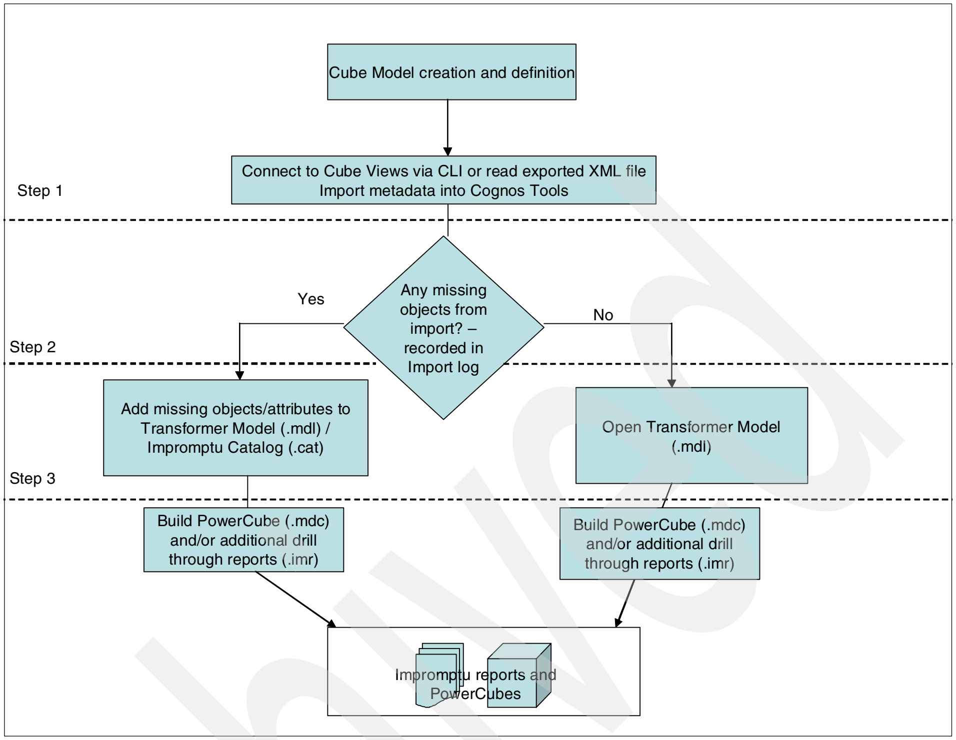


Figure 11-5 DB2 Dimensional Metadata wizard implementation steps

1. **Step 1:** Connect to DB2 Cube Views and import metadata.

   Connect to DB2 Cube Views, Cognos bridge. The DB2 Dimensional Metadata Wizard connects to DB2 Cube Views communicating via the CLI interface. CLI must be configured on the computer to allow a connection to the source DB2 Cube Views environment. Optionally, the bridge can import the metadata from a DB2 Cube Views exported XML file.

   Import metadata and check the result:

   a. Choose a data source as shown in Figure 11-6.

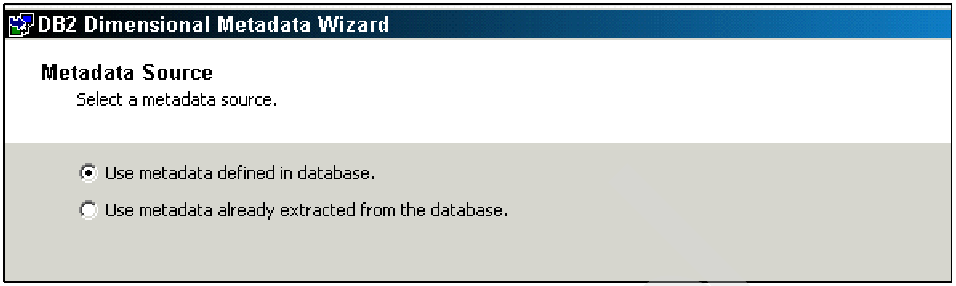


Figure 11-6 import metadata

The bridge prompts the user for a connection string that defines the DB2 Cube Views environment to be used. The user is then be prompted for authentication to DB2. Any access via the bridge to DB2 Cube Views is controlled by DB2 level security. The bridge then displays a list of DB2 Cube Views cube models that have been defined.

> **Note:** The Cognos bridge will only import the DB2 Cube Views cube models' metadata.

b. Logon to DB2 as shown in Figure 11-7.

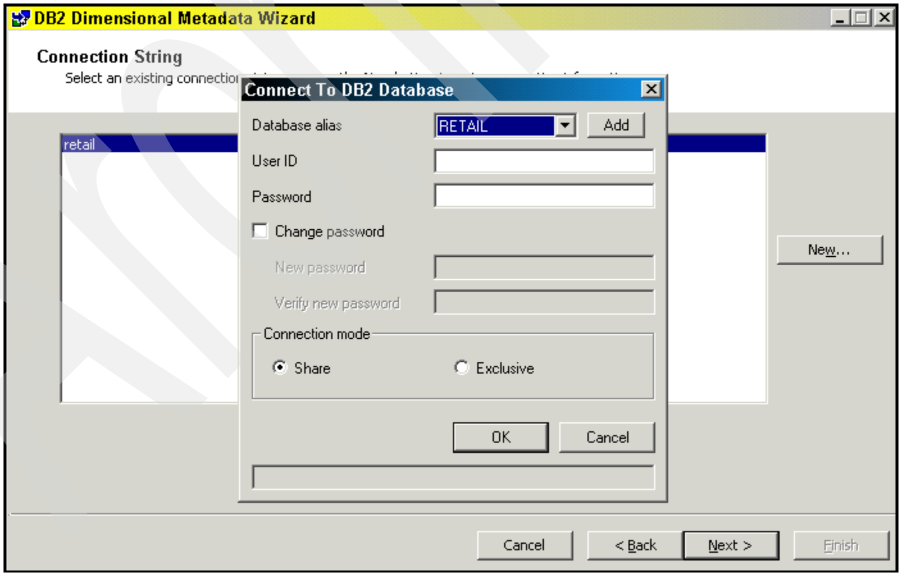


Figure 11-7 Logon to DB2

c. Once the particular cube model is chosen, select the Cognos target objects to be created: the Transformer model and PowerCube building SQL statements (*.iqd*), and/or the Impromptu Catalog and reports as shown in Figure 11-8.

   i. Impromptu catalogs (.cat) and reports (.imr):

      This option imports the metadata from DB2 Cube Views and generates an Impromptu catalog and default set of Impromptu reports, including drill through reports. These files provide reporting capabilities and assume a business presentation with context derived from the logical metadata defined within DB2 Cube Views.

   ii. PowerPlay Transformer models (.mdl) and Impromptu query definitions (.iqd to extract data):

      This option imports the metadata from DB2 Cube Views and generates the PowerPlay Transformer model and the query definitions. These files define the metadata, parameters and structure to build MOLAP PowerCubes.

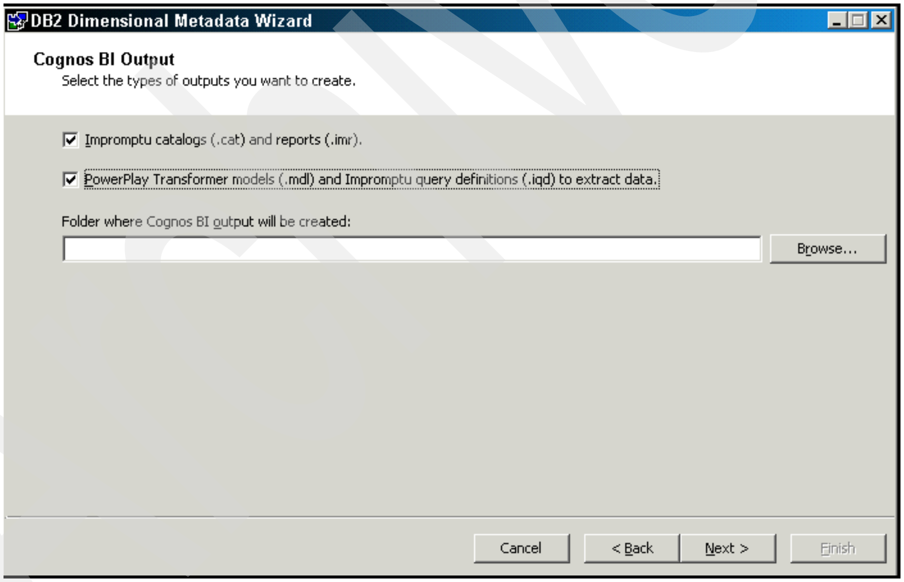


Figure 11-8 Cognos file options

2. **Step 2:** The bridge will parse the DB2 Cube Views metadata. This process is updated dynamically into a log file. The OLAP metadata objects are divided up into Cognos and non-Cognos objects. Cognos objects are built into the appropriate Cognos tool, either an Impromptu Catalog (.cat) or Transformer model (.mdl). The business rule names given to the objects in DB2 Cube

Views are maintained. Non-Cognos objects are items that do not have a direct link in Cognos metadata. These references are written to the log file as shown in Figure 11-9.

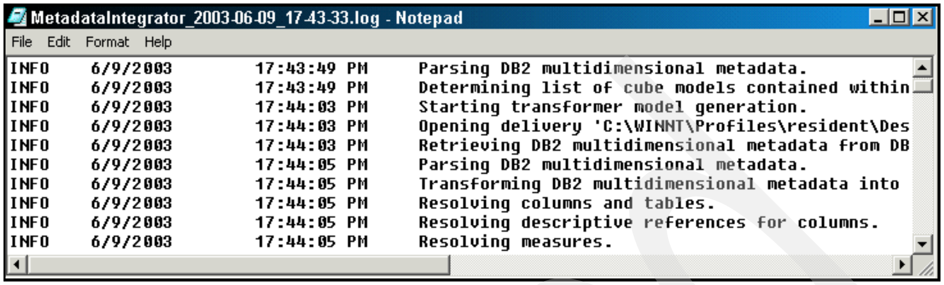


Figure 11-9 Metadata bridge log file

3. **Step 3:** Open Transformer Model and build the PowerCube
   – The Transformer model (.mdl) can be opened and the PowerCube (.mdc) built (see Figure 11-10).

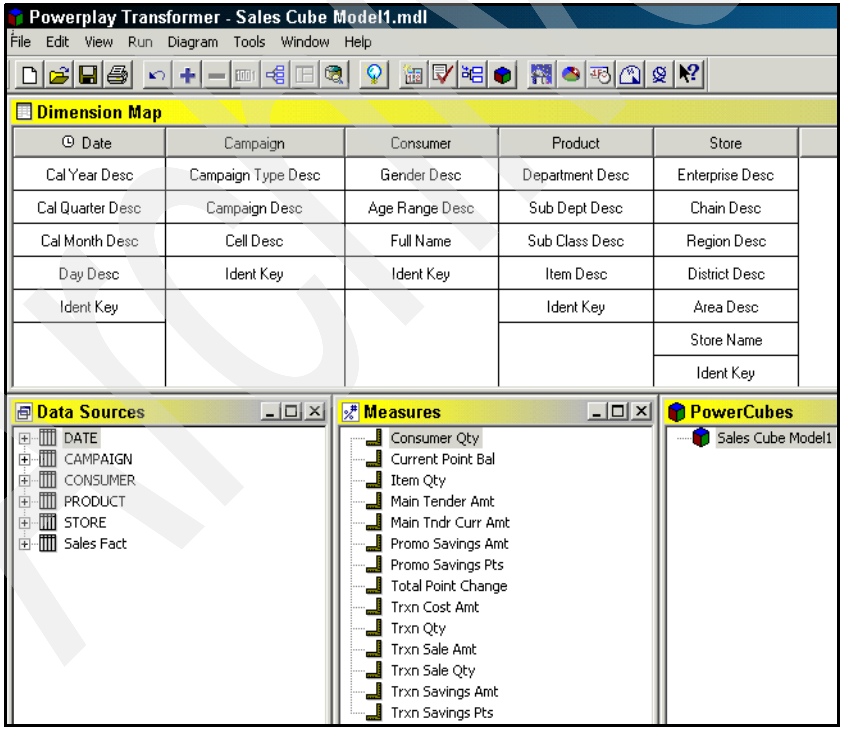


Figure 11-10 Transformer model default

Prior to building the PowerCube, further enhancements to the model structure as well as additions noted in the log file, can be added. Enhancements may include measure hierarchies, measure allocations, category counts, renaming of levels and dimensions, time-based partitions and other refinements available in Transformer.

Each measure within the model is automatically associated with the drill through report. This can be changed easily within the interface, by simply browsing to the preferred report.

– The Impromptu catalog (.cat) and Drill through report (.imr) as shown in Figure 11-11.

  The Impromptu catalog contains folders reflecting each table of the star schema. For example, the DB2 Cube Views model has a table called *Campaign*, containing *Campaign Type Desc*, *Campaign Desc*, *Cell Desc*, and *Campaign Ident Key*. These are imported with the same names and structure into the Impromptu Catalog (see above). Additional hierarchies are imported into folders called Additional columns folders, with all the associated columns. Each folder will contain a set of prompts which reflect the levels in the hierarchy(s). These prompts can be used in the building and execution of authored reports.

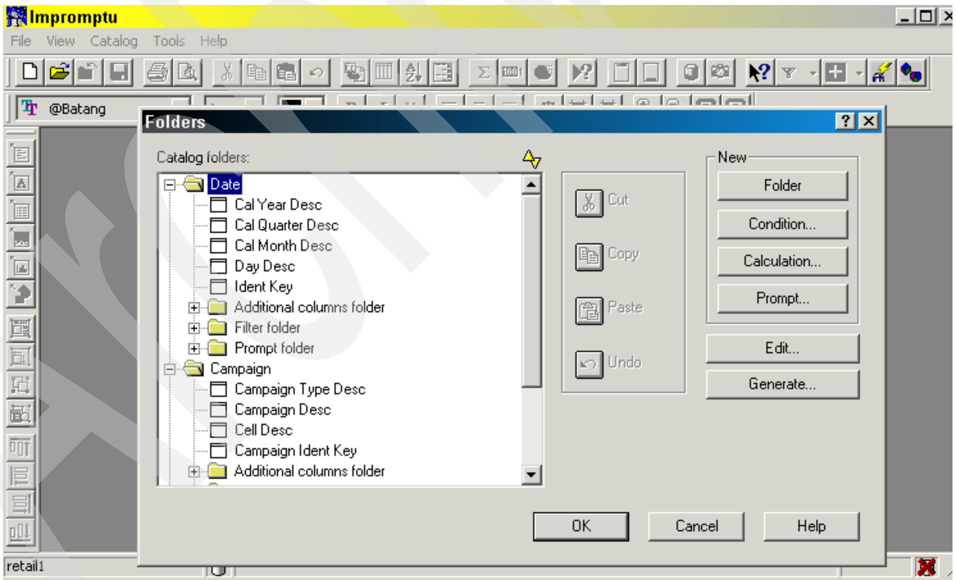


Figure 11-11 Impromptu default view

A set of reports is created by the bridge, based on each table of the star schema. Each report has groupings based on the columns of the table as well as the measures from the fact table. The first three columns of the table are grouped, and totals are inserted into footers, for each level of grouping. By doing this, these reports can leverage MQTs immediately.

A default template for these reports is included with the bridge. This template outlines the layout of the reports that the bridge generates. It can then be customized in Impromptu appropriately. Templates are a starting point from which reports can be built. They can contain information about column metadata, margin settings, page orientation, font choices for different report objects and so on. In many cases, users will create a default template that contains the corporate or group logo, colors and fonts as per company standards. The template report called libcogmdi.imt is stored in:

```
<rootlocation>\cer3\bin\ where <rootlocation> is the directory into which
the Cognos software was installed
```

The drill through report is based on the fact table of the star schema. By default, it is linked to each measure within the Transformer model. For further information about the default reports, please see section 2.4.

11.4 Implementation considerations

There are some differences in the naming conventions used by DB2 Cube Views and Cognos. Table 11-1 contains the mappings between the terms used in DB2 Cube Views, how they correspond to the Cognos tools, and if they are imported by the bridge, or must be defined after the import in Cognos.

Table 11-1 Metadata bridge mapping

DB2 Cube Views object	Cognos terminology	Bridge or defined in Cognos
Cube model	Transformer Model (.MDL)	Bridge
Cube	PowerCube (.mdc)	Define subset from Transformer Model and execute build
Hierarchy	Drill-Down Path (hierarchy)	Bridge
Multiple hierarchies	Alternate Drill-Down Paths (hierarchy)	Represented in Impromptu as a sub folder of the first hierarchy. Define within Transformer Model.
Ragged hierarchy	Drill-Down Path (hierarchy)	Transferred to Impromptu as a sub folder of the first hierarchy. Defined within Transformer Model
Measure	Measure	Bridge

DB2 Cube Views object	Cognos terminology	Bridge or defined in Cognos
Complex or calculated measure	Calculated measure	Define within Transformer Model
Table	Table	Bridge – viewed as a Folder or a Table in Impromptu
Attribute	Level	Bridge
Attribute relationship		Defined in Transformer Model
Join	Join	Bridge
Dimension	Dimension	Bridge

Figure 11-12 is a cube model example containing several dimensions, measures, and tables, as well as multiple hierarchies, and complex measures.

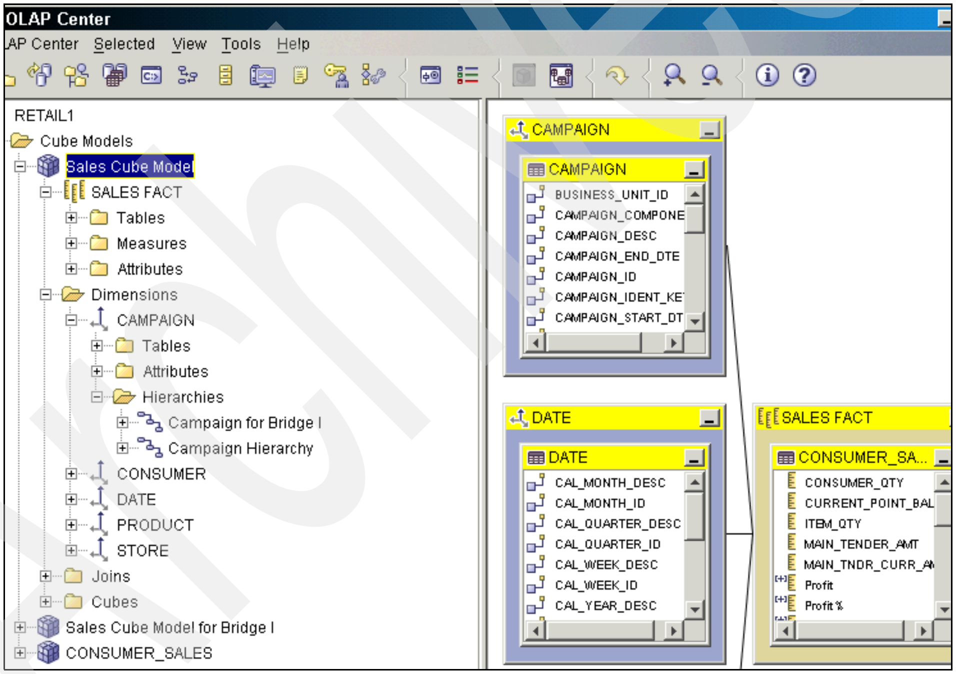


Figure 11-12 Default DB2 Cube Views cube model

Figure 11-13 shows screenshots of how the Cognos environment can match the cube model. These are screenshots of the fully modified Transformer model that matches the cube model with the dimensions, measures, hierarchies, and complex measures. Additionally, Cube Groups, Measure folders, and extra dimensions have been created in Cognos, providing additional business value.

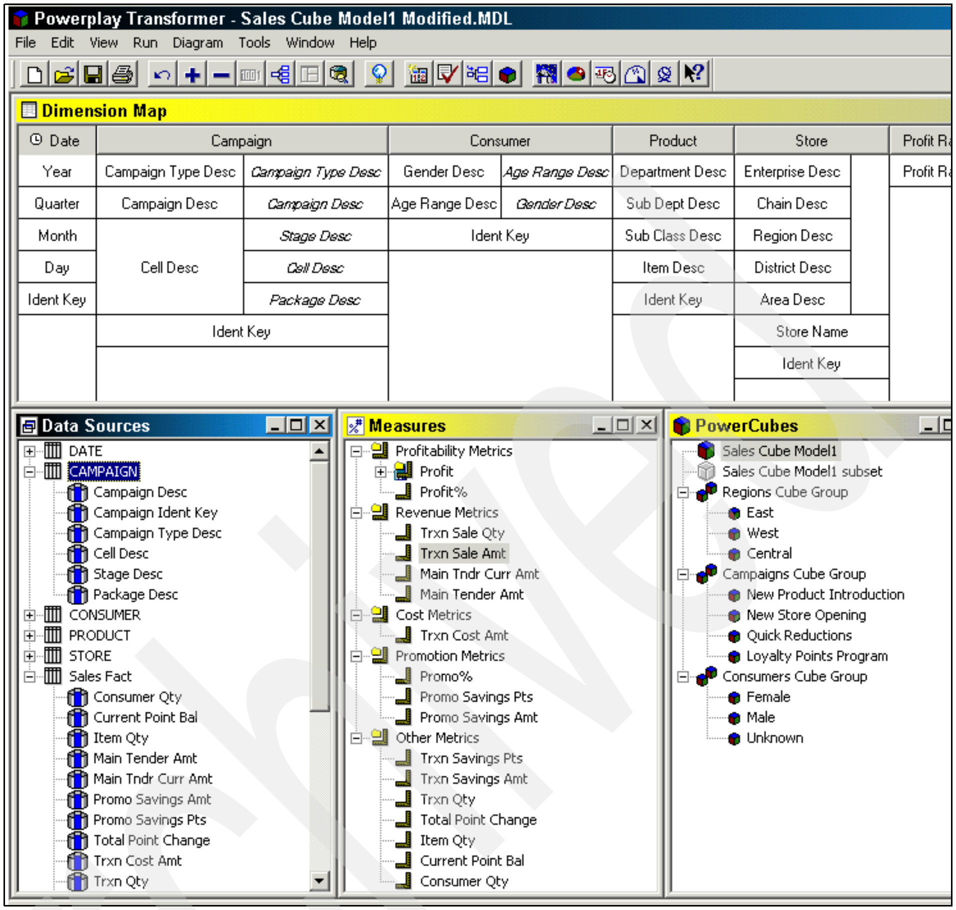


Figure 11-13 Transformer model

Figure 11-14 shows the hierarchies imported into Impromptu.

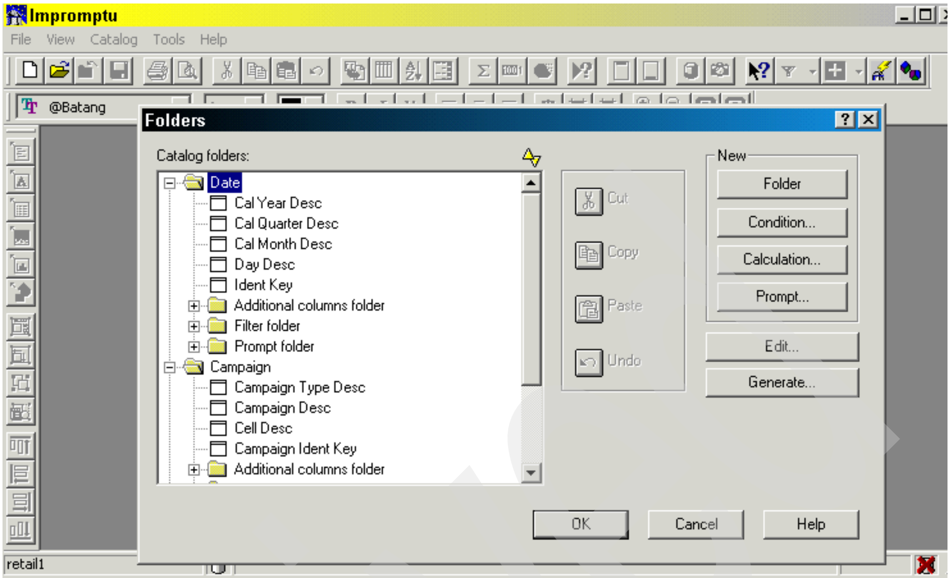


Figure 11-14 Impromptu default view

The first hierarchy listed in the cube model will be the first folder, and any additional hierarchies will be listed *within* the folder called 'Additional columns folder'. As with the Transformer, further business rules can be added such as additional calculations, prompts and filters to the Impromptu Catalog, enhancing the business value of the users' reporting capabilities.

Some of the Cognos OLAP features that can enhance the BI from DB2 Cube Views include:

- PowerPlay Cube Groups: A set of cubes built relating to a single level within one dimension of the model. For instance, PowerCubes can be created on *Regions*, or *Campaigns* (see Figure 11-13 on page 500)
- Measure formatting: Applying formatting options to the measure values so they appear for the consumers consistently.
- Measure folders: Group measures into business rules which allow users to navigate through various measure rollups and drill down to view the lower level measures in their OLAP reports.
- PowerCube level security: Restricting access to portions of data in a PowerCube to certain members of the user community
- Security in Cognos: Imbed the user ID for drill through access with the user id: user will not be prompted when accessing the PowerCube or Impromptu Report.

- Mobile or desktop PowerCubes: Allow users to have a local copy of the PowerCube and continue to use the MOLAP database disconnected from the network.

11.4.1 Optimizing drill through

Cognos defines drill through as an action that enables Impromptu, PowerPlay users to view transaction-level details in an Impromptu report. You can use drill-through reports to create a report that better meets the needs of its primary audience while still providing an easy to access link to related information. A report user who wants to see related or more detailed information selects a value in the report and drills through to an associated report.

By default, Transformer will use a transactional query as its drill through source. This works well for users who drill to the bottom of cube dimensions, then wish to drill through to DB2 for more details. The cube model in DB2 Cube Views will already have been defined to represent the complete star schema and the cubes based on business needs too. Optimization Advisor would have already been performed to recommend drill through MQT, assuming that the cube model is reasonably optimized.

However, in order to optimize drill-through queries that occur at higher levels in the cube, additional drill-through queries can be authored in Impromptu to use MQTs to maximize query performance.

For example, if an MQT exists that is grouped on the *Customer*, *Campaign* and *Store* dimensions, at the *Age_Range*, *Campaign_Type* and *Region* levels within these dimensions, then:

1. Author an appropriate report in Impromptu.
2. Ensure that the resulting SQL is in the format of Example 11-1.

Example 11-1 SQL example

```
select T1."AGE_RANGE_DESC" "c1" , T2."REGION_DESC" "c2" ,
T3."CAMPAIGN_TYPE_DESC" "c3" , sum(T4."TRXN_SALE_QTY") "c4" ,
sum(T4."TRXN_SALE_AMT") "c5"
 from "STAR"."CONSUMER_SALES" T4, "STAR"."CONSUMER" T1, "STAR"."STORE" T2,
"STAR"."CAMPAIGN" T3
 where T4."CONSUMER_KEY" = T1."IDENT_KEY" and T4."STORE_ID" = T2."IDENT_KEY"
and T4."COMPONENT_ID" = T3."IDENT_KEY"
 group by T1."AGE_RANGE_DESC", T2."REGION_DESC", T3."CAMPAIGN_TYPE_DESC"
 order by 1 asc , 2 asc , 3 asc
```

3. Test that this report uses the MQT when it executes. For example, run the report and save the SQL (using the “Save As SQL” option), then run the SQL using the “Explain” option in DB2 Control Center.
4. Then add the query to the Transformer model as a drill-through source. This can be added at the overall cube level, or for a specific measure, as shown in Figure 11-15.

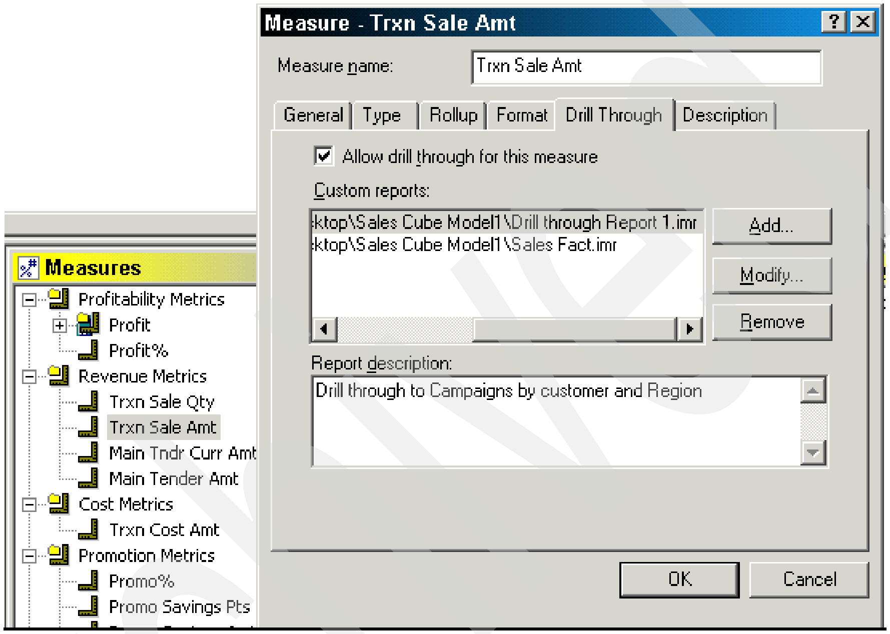


Figure 11-15 Transformer model measure drill through setup

When the user clicks **Drill Through** in PowerPlay, a number of drill through options are displayed, for example:

- Drill through to campaigns by customer and region (see Figure 11-16)
- Drill through to Products by Region, Area, and Campaign

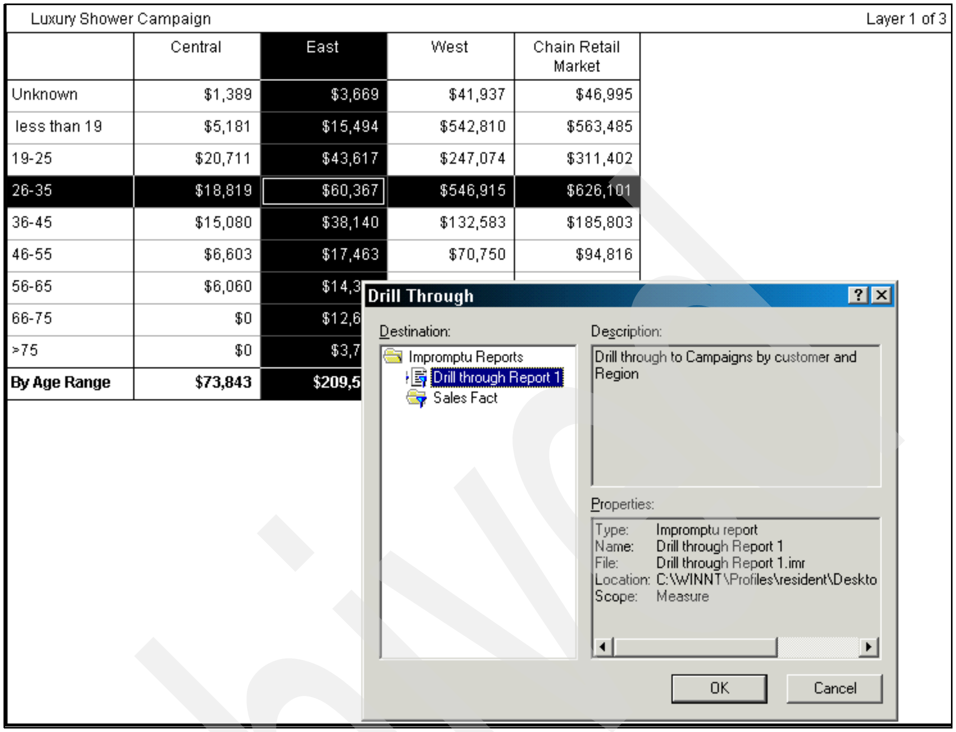


Figure 11-16 Drill through to campaigns by customer and region

Figure 11-16 will drill through to provide the results shown in Figure 11-17.

Drill Through Filter: Region Desc = East and Campaign Desc = Luxury Shower Campaign and Campaign Type Desc = New Product Introduc

Department Desc	Sub Dept Desc	Region Desc	mpaign Type De	Campaign Desc	ender Des	Age Range Desc	Trxn Savings Amt
BODYCARE	BATHCARE	East	New Product Introduction	Luxury Shower Campaign	Female	26-35	$6,881.43
					Male	26-35	$326.39
					Unknown	26-35	$1.91
	HAIRCARE	East	New Product Introduction	Luxury Shower Campaign	Female	26-35	$43,769.24
					Male	26-35	$1,957.26
					Unknown	26-35	$59.99
	SKINCARE	East	New Product Introduction	Luxury Shower Campaign	Female	26-35	$20,060.75
					Male	26-35	$1,019.39
					Unknown	26-35	$45.52
HOMECARE	GARDEN	East	New Product Introduction	Luxury Shower Campaign	Female	26-35	$64,195.65
					Male	26-35	$2,951.78
					Unknown	26-35	$74.40
	GIFTS	East	New Product Introduction	Luxury Shower Campaign	Female	26-35	$1,363.50
					Male	26-35	$87.20

Figure 11-17 Drill through result

When PowerPlay drills to an Impromptu report (Figure 11-18), it passes the drill context to Impromptu, which is potentially applied to query items in the report.

> **Note:** Filtered dimensions that are not included in the drill through query will simply be ignored, ensuring that the MQT is still used. Users have the option of adding to the query any additional information they would like to include once they have drilled through, or even drilling from this query to another, more detailed, report (assuming report to report drill paths have been defined).

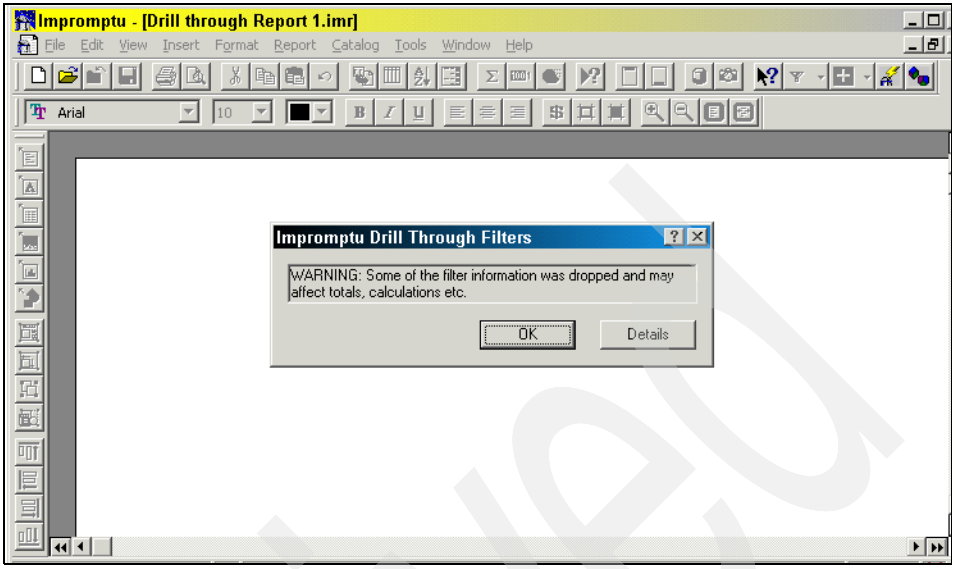


Figure 11-18 Impromptu drill through reports

The resulting SQL from this drill through report can be run in the DB2 Cube Views Control Center, to verify that it is using the MQT (see Figure 11-19).

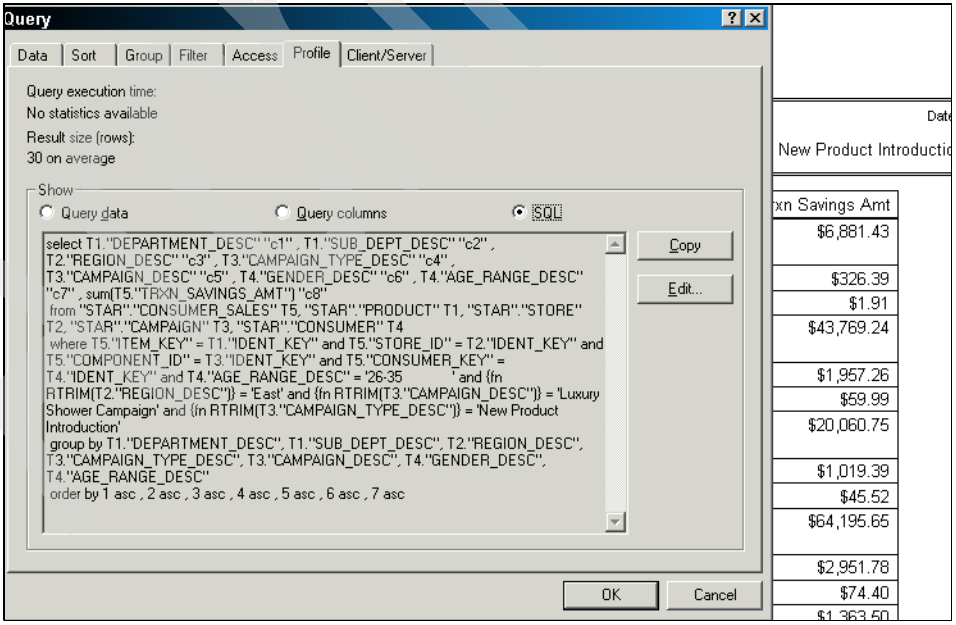


Figure 11-19 the drill through SQL

In fact, in this example the drill-through uses the MQT as shown in Figure 11-20.

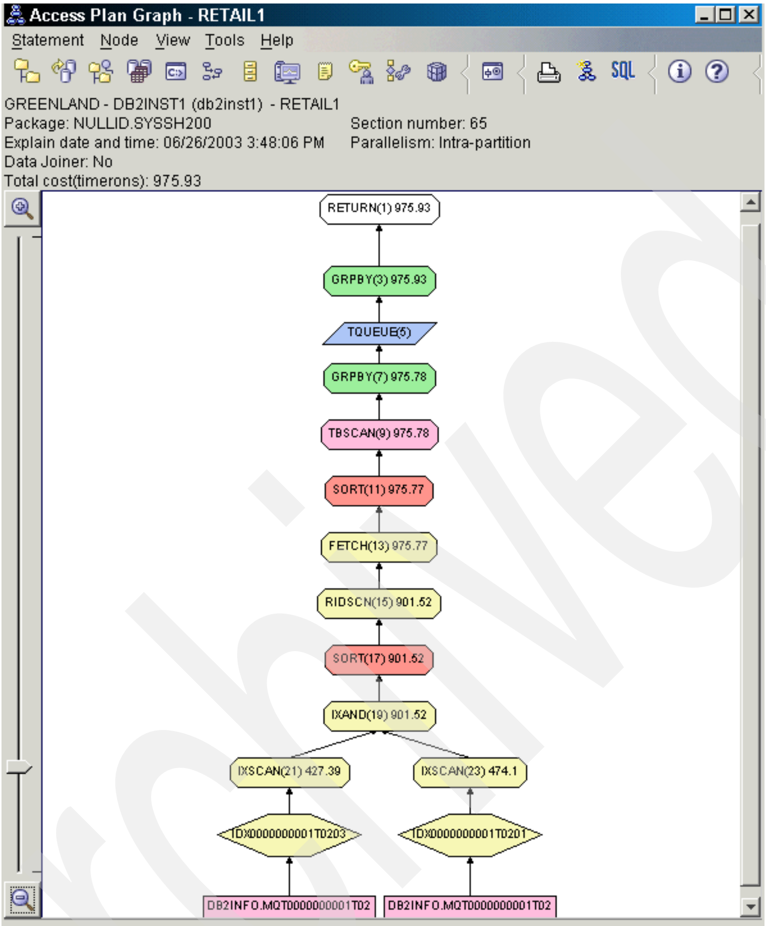


Figure 11-20 Drill through query DB2 explain: using MQTs

Note: Drill-through reports authored to take advantage of MQTs will show significant performance improvements. For example, a similar drill-through to a report which runs against the base star schema instead of the MQT produces query results as shown in Figure 11-21 and in Figure 11-22.

The DB2 Explain access graph for the same query without MQT is shown:

- For the lower part of the Explain graph, in Figure 11-21.

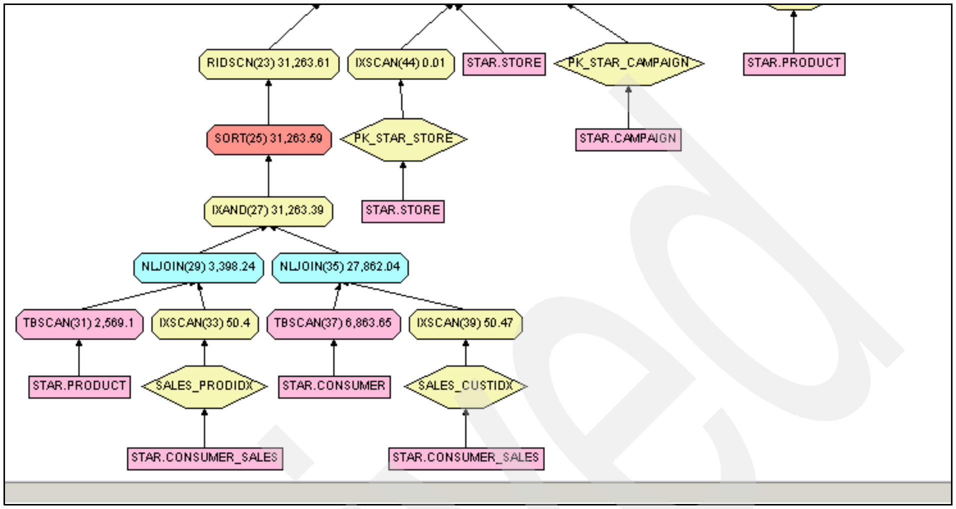


Figure 11-21 Drill through DB2 explain: without MQT (lower level access graph)

- For the upper part of the Explain graph, in Figure 11-22.

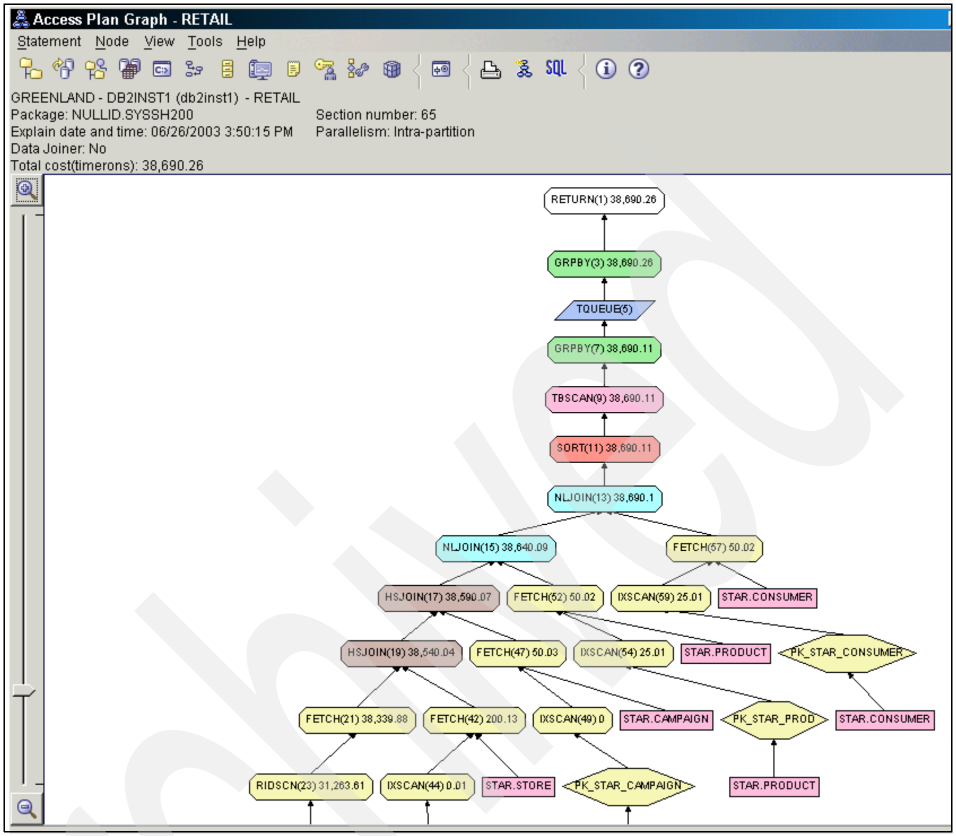


Figure 11-22 Drill through DB2 explain: without MQT (upper level access graph)

In this example, the drill through which leveraged an MQT index for its query cost 975.93 timerons. An equivalent drill through query against the fact and dimension tables, rather than an MQT, cost 38,690.26 timerons. This MQT query executed in 3.55 % of the time of the non-MQT query -96.45% faster! Table 11-2 summarizes the results.

Table 11-2 Drill-through performance result

	Drill-through report: timeron
Without MQT	38,690.26
With MQT	975.93

11.4.2 Optimizing Impromptu reports

The Cognos Metadata bridge will create default Impromptu reports, one for each dimension in the DB2 Cube Views. By default, these reports will not take advantage of any MQTs that may be available. However, it is simple to modify, or reproduce, these reports to leverage the MQTs, as described below:

1. Using the Impromptu Query Wizard (see Figure 11-23), add the levels of a specific dimension, and the measures you would like to display (which are located in the fact folder).

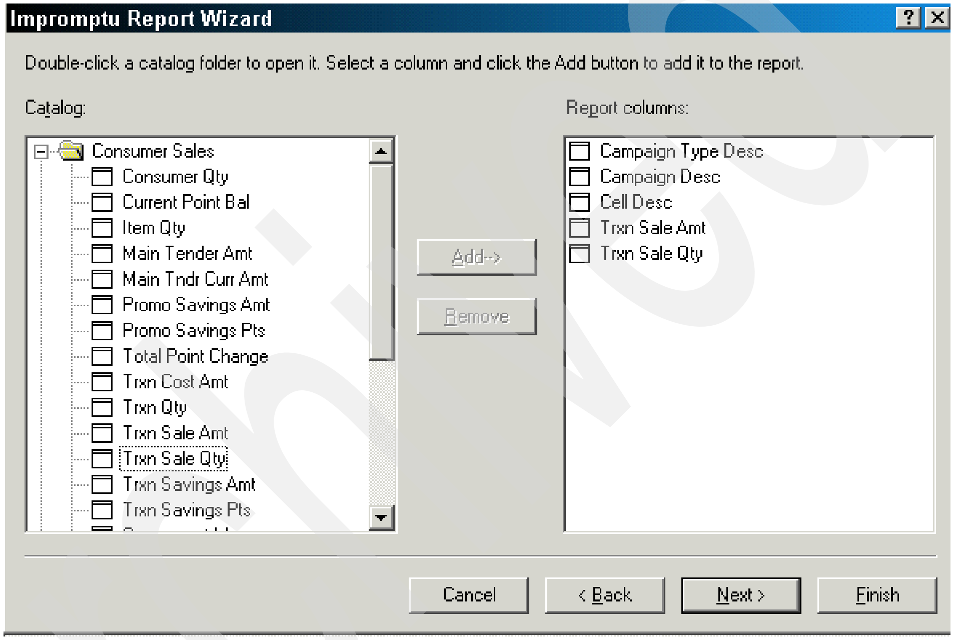


Figure 11-23 Impromptu Report wizard

2. This will generate a simple **`SQL SELECT`** query, returning the transaction detail rows from the fact table as in Figure 11-24, and descriptive information from the associated dimension tables (in this case the Campaign dimension table).

Campaign Type Desc	Campaign Desc	Cell Desc	Trxn Sale Amt	Trxn Sale Qty
New Product Introduction	Luxury Shower Campaign	Double Income - No Kids	2.89	1
New Product Introduction	Luxury Shower Campaign	Senior Citizens	4.58	2
New Product Introduction	Luxury Shower Campaign	Senior Citizens	7.71	5
New Product Introduction	Luxury Shower Campaign	Senior Citizens	16.80	7
New Product Introduction	Luxury Shower Campaign	Senior Citizens	6.92	6
New Product Introduction	Luxury Shower Campaign	Senior Citizens	15.65	3
New Product Introduction	Luxury Shower Campaign	Senior Citizens	10.51	5
New Product Introduction	Luxury Shower Campaign	Senior Citizens	16.80	7
New Product Introduction	Luxury Shower Campaign	Senior Citizens	6.48	2
New Product Introduction	Luxury Shower Campaign	Senior Citizens	2.99	1
New Product Introduction	Luxury Shower Campaign	Senior Citizens	10.51	5
New Product Introduction	Luxury Shower Campaign	Senior Citizens	5.47	4

Figure 11-24 Transaction detail rows from the fact table

3. From the Task bar in Impromptu, select **Report / Query, Group** tab as shown in Figure 11-25. Group on the Dimension Columns that you are reporting on. This is to ensure that the query will return aggregated data, rather than the individual transaction details.

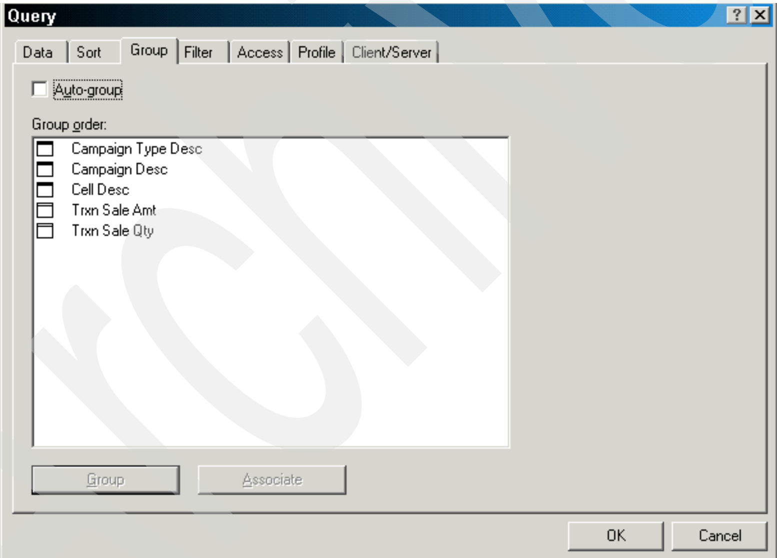


Figure 11-25 Query properties

4. Choose the **Data** tab, select each measure column in turn, and add an aggregation, for example, total (which generates a **SQL SUM**) for each measure column (data column from the fact table) as shown in Figure 11-26.

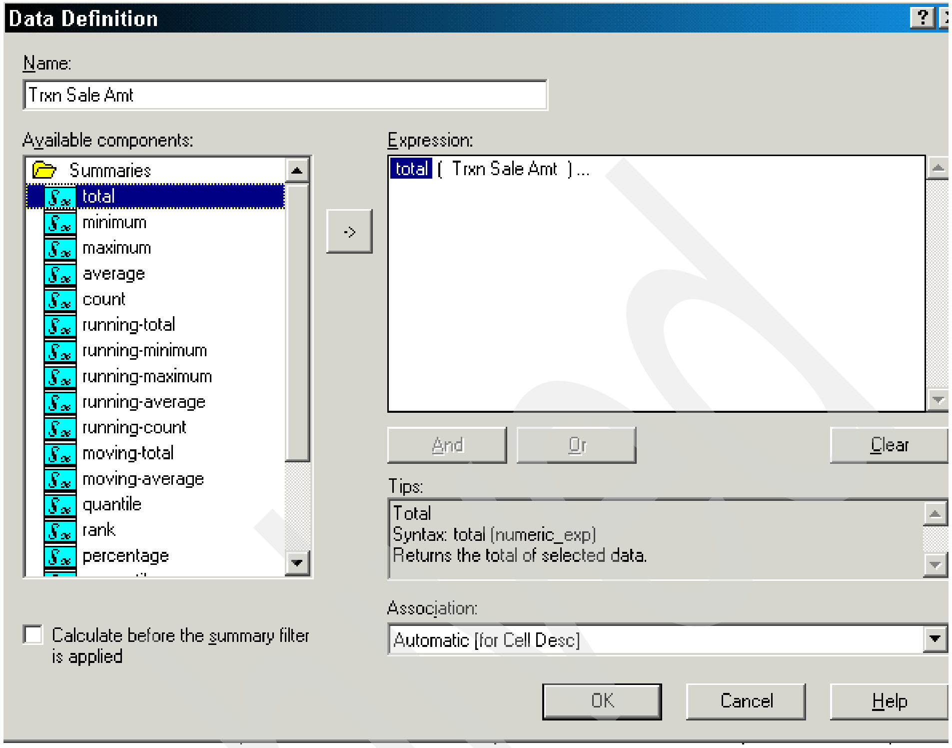


Figure 11-26 Data definition

5. Once all measure columns are defined as aggregates, click **OK**. The query returns aggregate data (see Figure 11-27), via the MQT.

Campaign Type Desc	Campaign Desc	Cell Desc	Trxn Sale Amt	Trxn Sale Qty
Loyalty Points Program	Frequent Buyer Price Reduction	Competitive Shoppers	1783047.06	1305246
New Product Introduction	Luxury Shower Campaign	Double Income - No Kids	5071322.11	2679755
		Family Shoppers	2078555.08	1527378
		Senior Citizens	2409049.28	1401251
		Single Women 41-55	1188633.40	868647
		Young Single Guys	1209698.51	877629
	Skincare Product Launch	Single Women 18-40	2078313.11	1523243
New Store Opening	Grand Opening Sale	Bargain Shoppers	2375195.06	1740966
		Gold Customers	1791830.44	1313788
		Low-Volume Customers	296922.71	218060
		Single Men 30-55	1187981.71	869186
	Store Brand Promotion	Brand Loyal	591335.80	433348
		Brand Switchers	2678162.82	1965161

Figure 11-27 Results on aggregate data

Table 11-3 summarizes the results on such queries based on the DB2 Explain access graph.

Table 11-3 Impromptu report performance result

	Impromptu report: timeron
Without MQT	757,449.69
With MQT	18,822.72

11.4.3 Implementation considerations: mappings

If DB2 Cube Views contains measure calculations or alternate hierarchies, these will need to be reproduced in the Cognos metadata. This is a simple process.

Calculated measures

Calculated measures defined in the OLAP Center will need to be reproduced in the Cognos environment. Figure 11-28 shows, for example, a measure “Profit% in DB2 Cube Views.

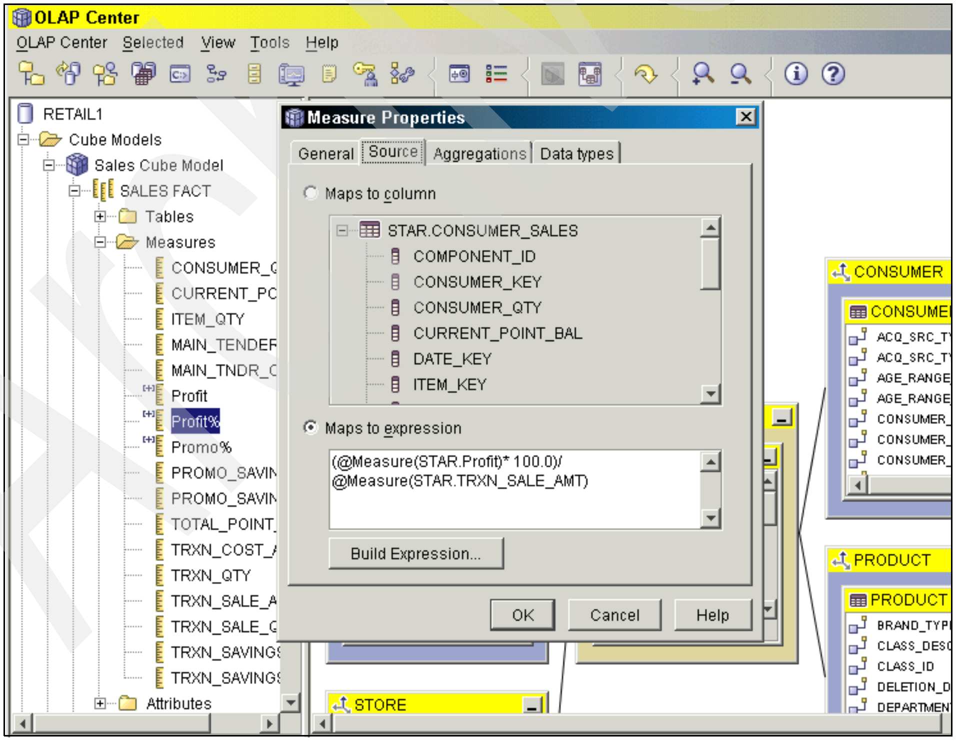


Figure 11-28 Calculated measure in Db2 Cube Views

Cognos allows for the creation of calculated measures in either the Impromptu Catalog, or within the Transformer Model.

To reproduce these calculations in Impromptu, create them in the Folders option under the Catalog menu item as shown in Figure 11-29.

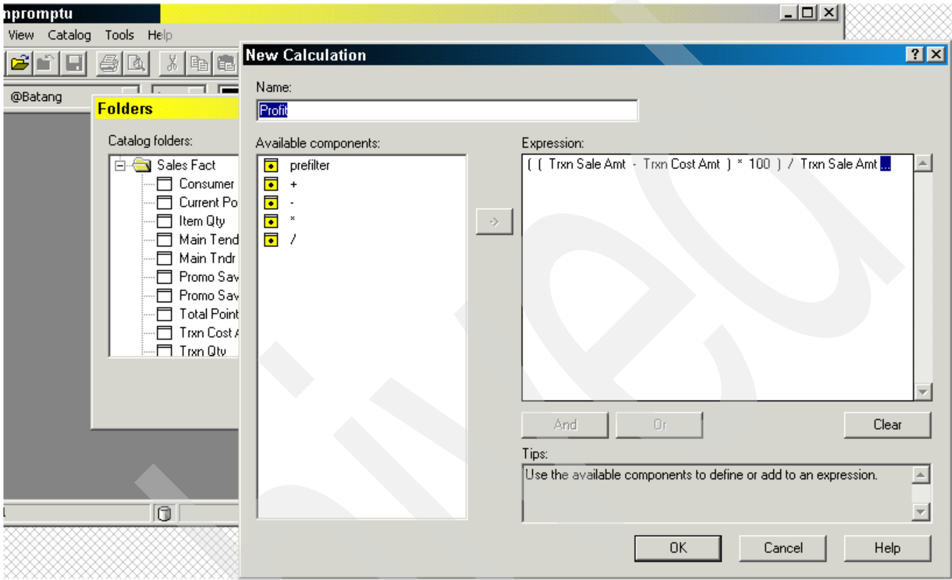


Figure 11-29 Calculated measure in Impromptu

Alternate hierarchies

Alternate hierarchies for a dimension as shown in Figure 11-30 are not transferred.

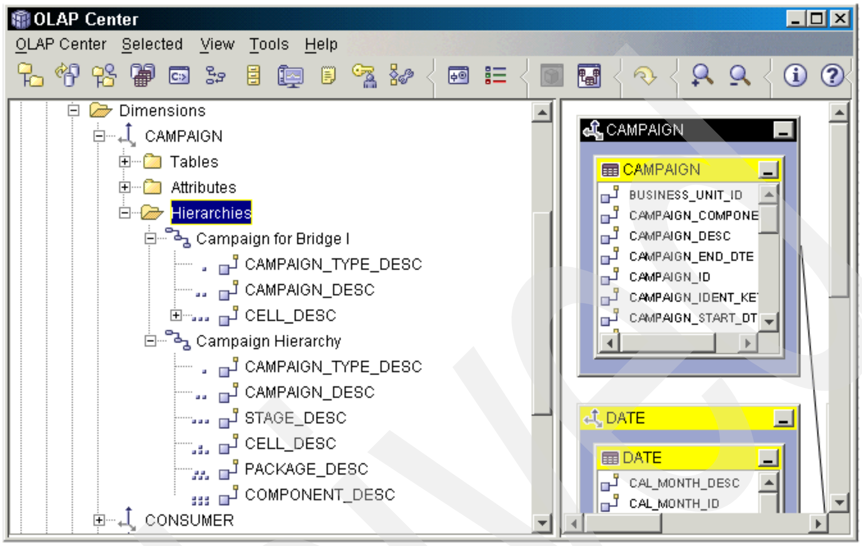


Figure 11-30 Alternate hierarchies in DB2 Cube Views

These can be reproduced in the Transformer Modeling environment as shown in Figure 11-31.

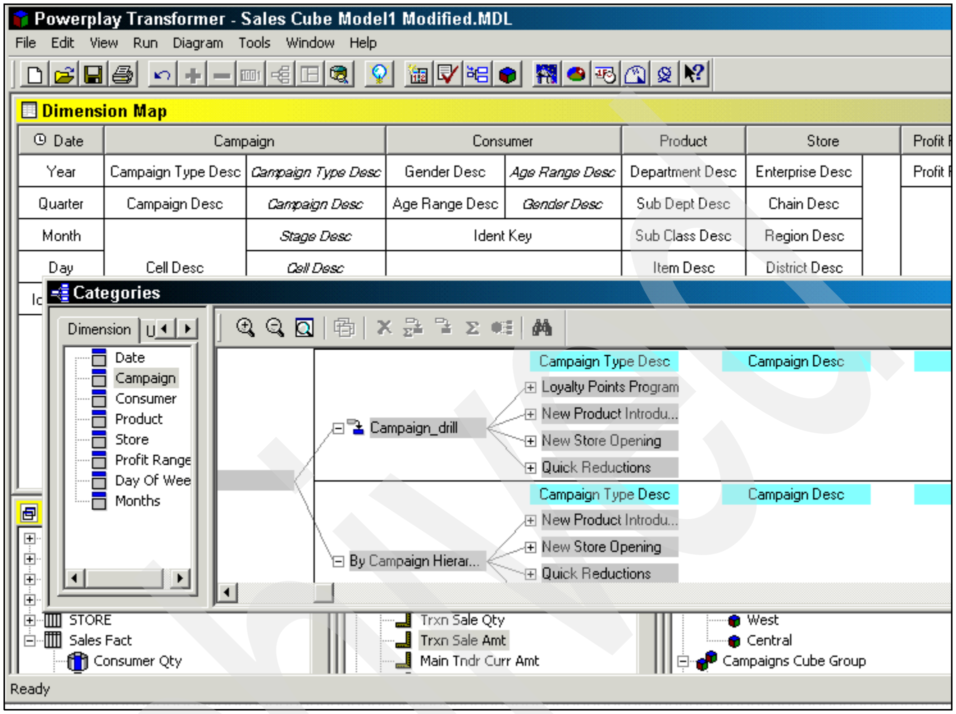


Figure 11-31 Alternate hierarchies in PowerPlay Transformer

Simply drag columns from the Data Sources window into the Dimension Map, to reproduce the alternate hierarchies required as shown in Figure 11-32.

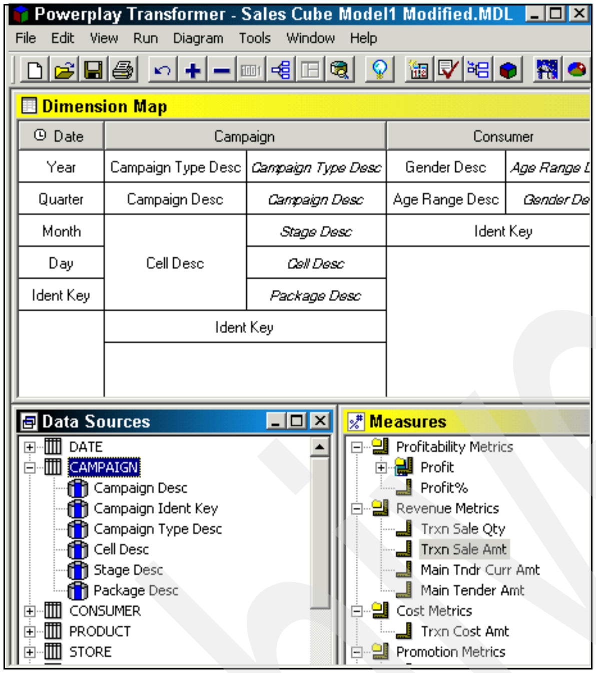


Figure 11-32 Reproduce the alternate hierarchies in DB2 Cube Views

However, note that any additional attributes used as levels in an alternate hierarchy in the cube model will require modification to the Impromptu Query (IQD) used as the source for that Dimension in Transformer. In this example, attributes *Stage Desc* and *Package Desc* need to be added.

Create a Query in Impromptu that returns the original columns shown in Figure 11-33 for that query source in Transformer (in this case *Campaign Desc*, *Campaign Ident Key*, *Campaign Type Desc* and *Cell Desc*) plus any new columns required for the alternate hierarchy:

Campaign Query for Transformer

Campaign Desc	Campaign Ident Key	Campaign Type Desc	Cell Desc	Stage Desc	P
Grand Opening Sale	5	New Store Opening	Low-Volume Customers	Stage #150	Prom
Frequent Buyer Price Reduction	13	Loyalty Points Program	Competitive Shoppers	Stage #180	Prom
Grand Opening Sale	16	New Store Opening	Single Men 30-55	Stage #150	Prom
Overstock Clearance	17	Quick Reductions	Bargain Shoppers	Stage #130	Prom
Luxury Shower Campaign	1	New Product Introduction	Young Single Guys	Stage #110	Prom
Luxury Shower Campaign	2	New Product Introduction	Single Women 41-55	Stage #110	Prom
Grand Opening Sale	4	New Store Opening	Gold Customers	Stage #150	Prom
Luxury Shower Campaign	7	New Product Introduction	Double Income - No Kids	Stage #110	Prom
Luxury Shower Campaign	8	New Product Introduction	Family Shoppers	Stage #110	Prom
Store Brand Promotion	10	New Store Opening	Brand Switchers	Stage #160	Prom
Overstock Clearance	12	Quick Reductions	Single Men 18-29	Stage #130	Prom
Holiday Clearance	15	Quick Reductions	Family Shoppers	Stage #140	Prom
Luxury Shower Campaign	3	New Product Introduction	Senior Citizens	Stage #110	Prom
Grand Opening Sale	6	New Store Opening	Bargain Shoppers	Stage #150	Prom
Store Brand Promotion	9	New Store Opening	Brand Loyal	Stage #160	Prom
Skincare Product Launch	11	New Product Introduction	Single Women 18-40	Stage #120	Prom
Overstock Clearance	14	Quick Reductions	Young Single Guys	Stage #130	Prom

Figure 11-33 Impromptu query

Save this new query as an .iqd, point the Transformer model to this iqd, then match up the columns in Transformer and add the new columns to the model as shown in Figure 11-34.

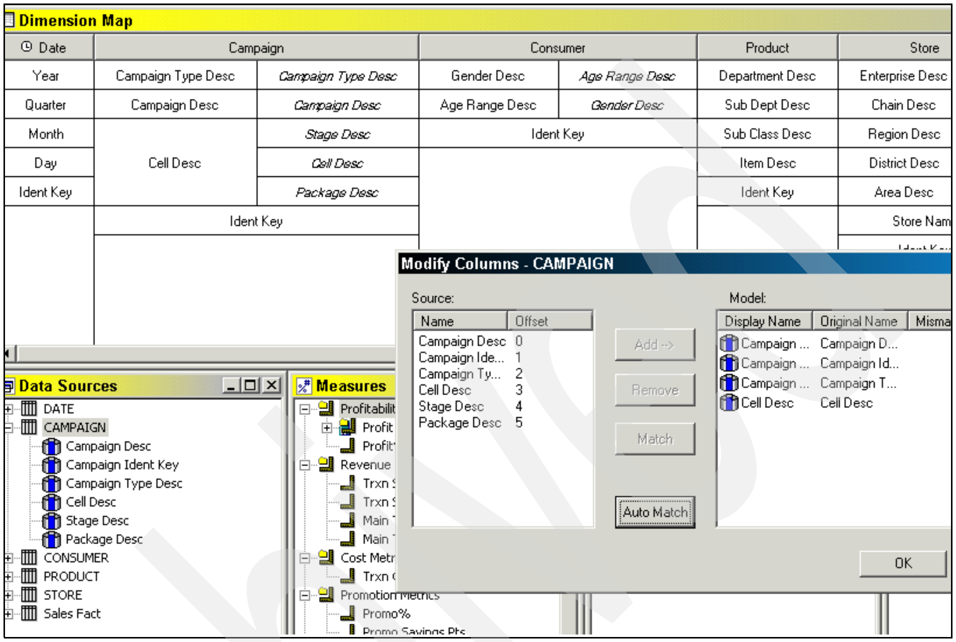


Figure 11-34 transformer: adding alternate hierarchy

Finally, create the alternate hierarchy for the Dimension as shown in Figure 11-35.

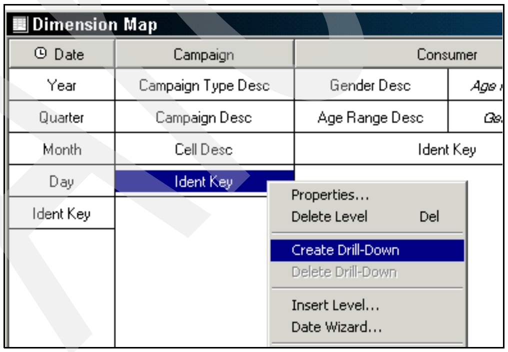


Figure 11-35 Create alternate drill down

11.4.4 Enhancing the DB2 cube model

The Cognos tools integrate the metadata defined with the DB2 cube model and allows for the quick implementation of this definition. Cognos also provides the capacity to take this cube model to the next level, allowing for the definition of further business rules and addition metadata to enhance the user's reporting capabilities — for example:

- Date dimension
- Measure formatting

Enhancing the Date dimension

A good practice would be to ensure that the Date dimension in DB2 contains a valid date field (for example, YYYYMMDD). This will allow the PowerPlay cube to leverage Transformer's powerful relative time capabilities, without having to maintain multiple relative time columns (such as Current Month, YTD and so on) in DB2 Cube Views. By simply dragging a date column from the Date Dimension Query into Transformer's dimension map, we automatically get a Year/Quarter/Month drill down, and many powerful relative time options as shown in Figure 11-36.

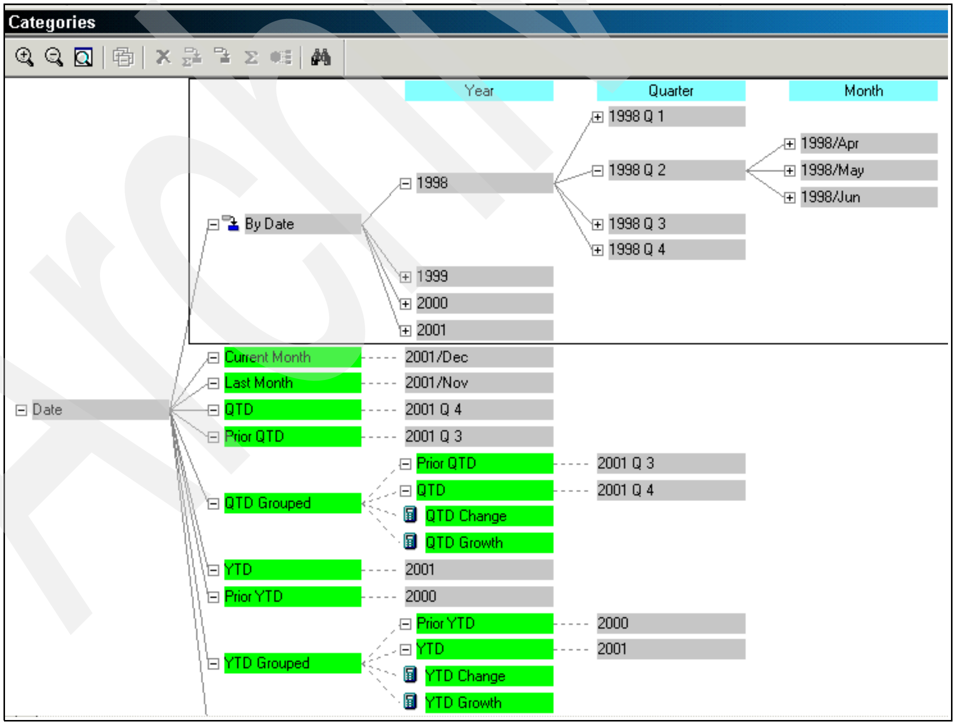


Figure 11-36 Transformer Model relative time

We can also add more dimensions to the cube design, for example in Figure 11-37 adding dimensions for Day of Week or Month.

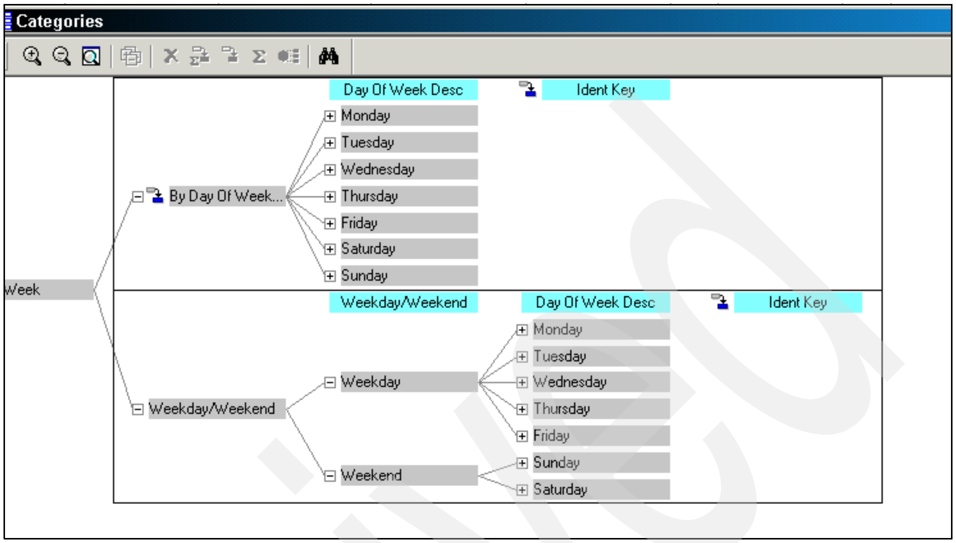


Figure 11-37 Transformer model Day of Week

This is particularly useful for allowing analysis of seasonal trends such as monthly trends as shown in Figure 11-38.

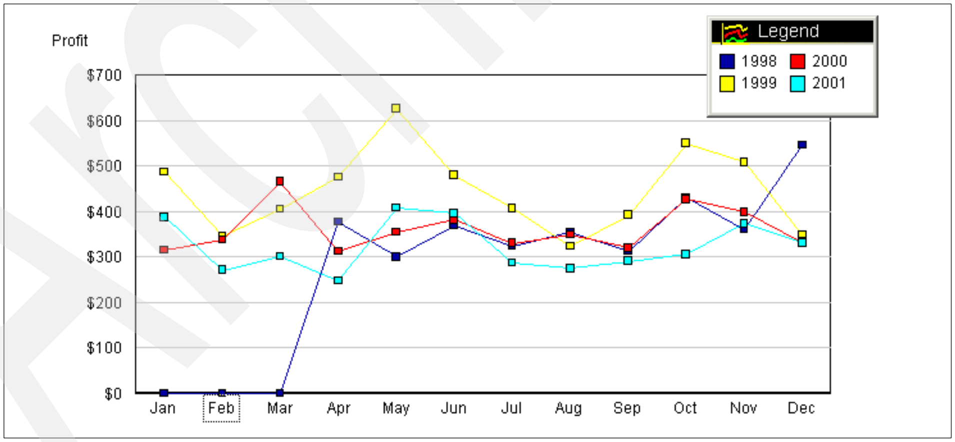


Figure 11-38 PowerPlay seasonality

Another possibility is to use *manual groupings*. Manual levels allow us to create groupings that do not exist in the source data. For example, if we wanted to group our major customers into a *Major Customers* group, where no such grouping exists in the *Customers* dimension table.

So, if we need to analyze the trend of sales or profitability throughout the week, and compare weekdays to weekends, we can add a *manual level* to the dimension in Transformer: *Weekday/Weekend*. Figure 11-39 shows *% Profit* by *Year* and *Day of Week*.

		1998	1999	2000	2001	Date
Weekday	Monday	45.41%	35.58%	56.86%	58.71%	46.85%
	Tuesday	61.65%	39.68%	61.85%	57.88%	51.72%
	Wednesday	55.15%	39.45%	55.54%	56.56%	49.09%
	Thursday	44.43%	39.47%	57.93%	57.11%	49.06%
	Friday	40.13%	33.71%	60.66%	59.15%	45.79%
	Weekday	**48.07%**	**37.54%**	**58.56%**	**57.87%**	**48.44%**
Weekend	Sunday	51.73%	32.84%	60.07%	60.29%	47.04%
	Saturday	46.85%	30.80%	56.22%	60.54%	44.15%
	Weekend	**49.12%**	**31.81%**	**58.25%**	**60.41%**	**45.59%**
Weekday/Weekend		**48.39%**	**35.62%**	**58.47%**	**58.57%**	**47.57%**

Figure 11-39 PowerPlay seasonality: another example

Measure formatting

Measure formatting can be applied within Transformer (see Figure 11-40), to ensure that measures are formatted appropriately (for example, $, %) when displayed to the user in PowerPlay. The measure formatting properties are stored in the PowerCube, and are the default settings for the measure.

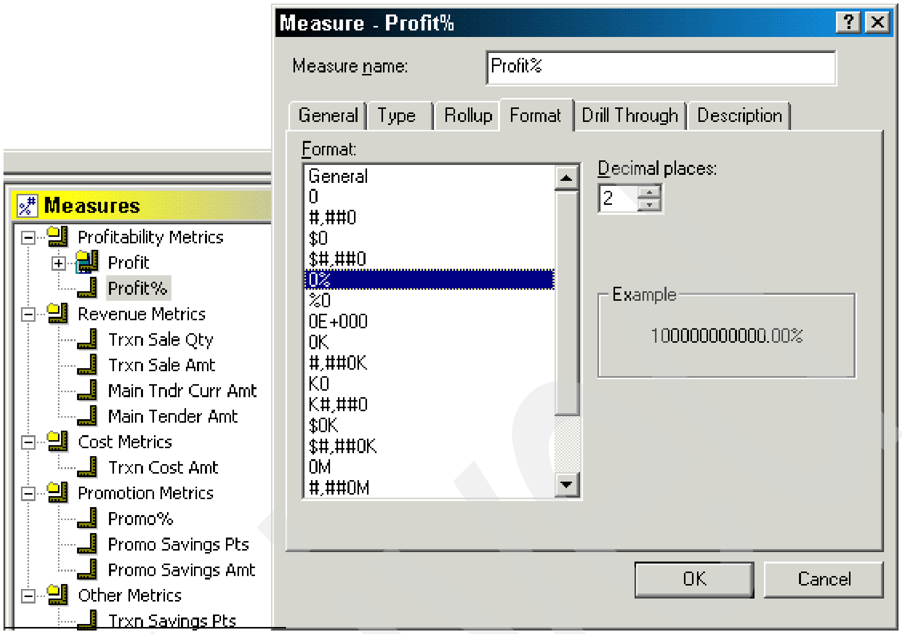


Figure 11-40 Transformer Model measure formatting

11.5 Cube model refresh considerations

The Cognos DB2 Dimensional Metadata Wizard is currently a one-way communication link. It is strongly recommended to back up the existing Cognos Transformation Model, and the Impromptu catalog and drill through reports before using the bridge to ensure all the changes you have made to the Cognos files have been maintained. With the documentation outlining the changes that have been made to the cube model, evaluate:

- What changes are to be made to the Cognos metadata?
- What changes from the Cognos metadata are to be transferred to the cube model?
- What changes from the cube model metadata are to be brought over to the Cognos environment?
- Based on the mapping list in 11.4, “Implementation considerations” on page 498, some of the new objects will have to be manually added.

11.6 Scenarios

This section develops two business analysis scenarios:

- One oriented toward sales: sales analysis scenario
- One oriented toward finances: financial analysis scenario

11.6.1 Sales analysis scenario

This section presents a basic scenario for sales analysis using the Cognos end-user environment. Typical sales analysis is of the form “Who?”, “What?”, “When?, “Where?”, giving you actionable information about your sales function.

In other words:

- Who is buying? (*Customer* dimension)
- What are they buying? (*Product* dimension)
- When are they buying? (*Date* dimension)
- Where are they buying? (*Location* dimension)
- How much? (*Revenue* and *Units Sold* measures)

You can see customer buying patterns, needs, answer important business questions and track sales performance metrics. For example, rank and compare sales volumes and values by customer type over time, or evaluate the effectiveness of sales resources.

Following are some example business questions that can be answered with the Cognos reporting and analysis tools:

What has been the trend of sales (revenue, profit)?

The full business question to answer is:

- What has been the trend of sales (revenue, profit across campaigns, and how has this changed over time?

To solve this business issue, the steps in PowerPlay are:

1. Open PowerPlay, with the *Date* dimension as columns and *Campaigns* as rows. Select *Sales Amount* as the active measure. (Locate the *Sales Amount* measure in the *Measures* folder, then click the **Filter** toolbar button) as shown on Figure 11-41.

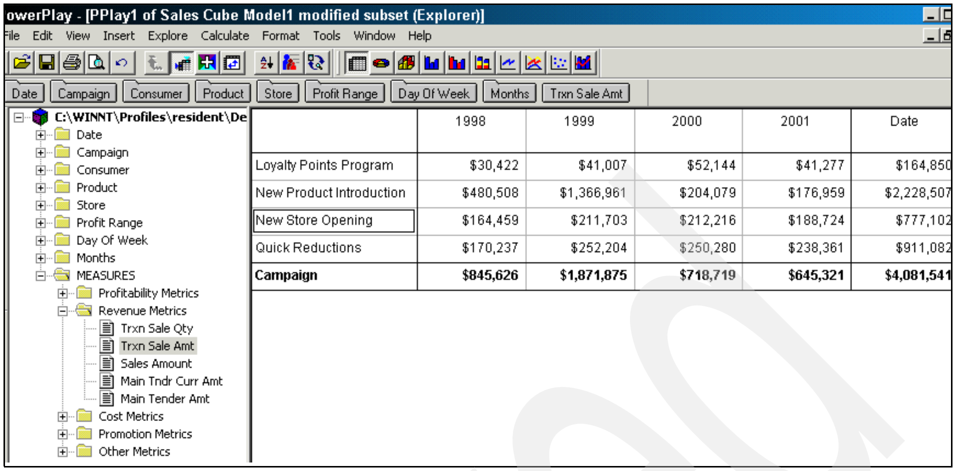


Figure 11-41 Scenario 1: report example 1

2. To more easily see the trend, switch to a clustered bar graph view (either via the **Clustered Bar Graph** toolbar button, or using the **Change Displays** option under the **Explore** menu option) (see Figure 11-42).

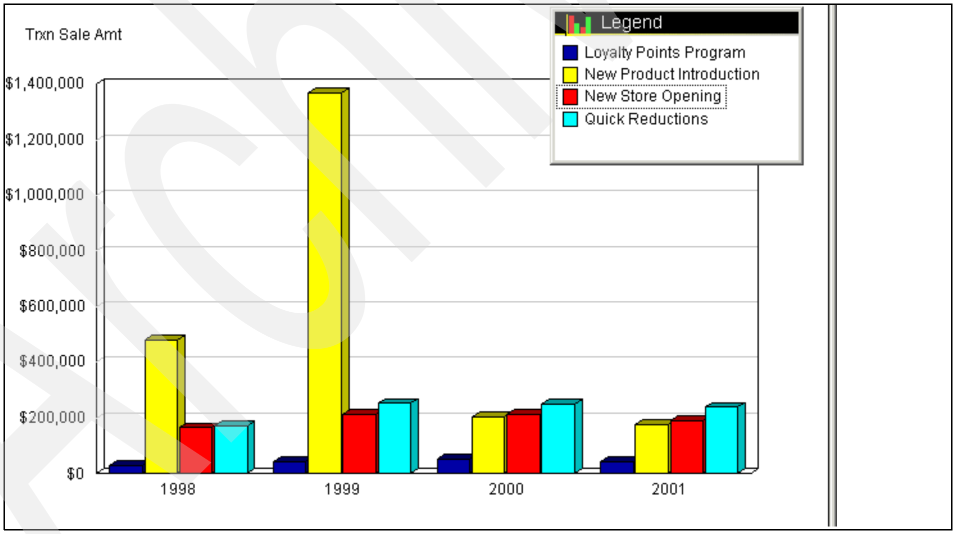


Figure 11-42 Scenario 1: report example 2

3. We can now clearly see that the sales for *Campaign New Product Introduction* were much higher in 1999. Drill down into *New Product Introduction* and drill into *Year 1999* as shown in Figure 11-43.

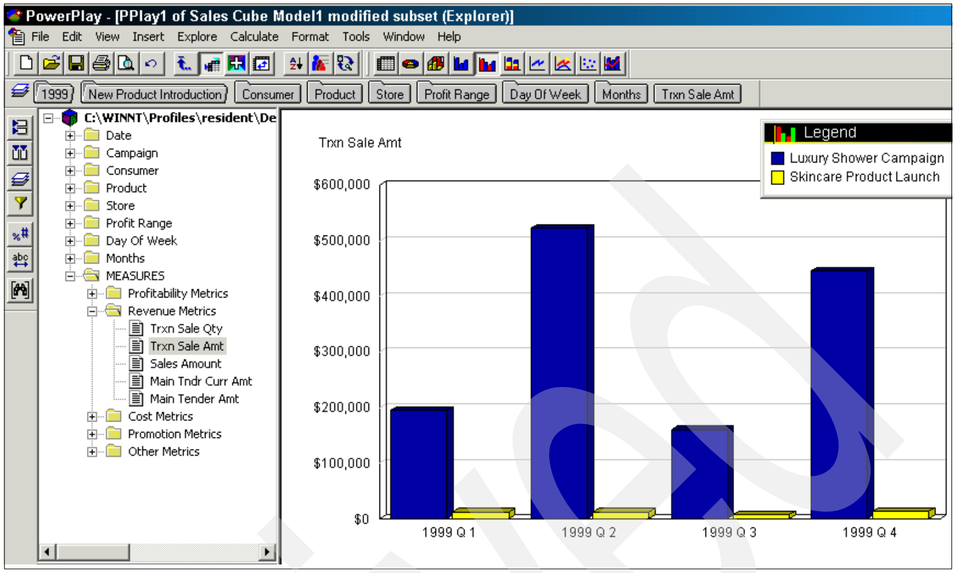


Figure 11-43 Scenario 1: report example 3

4. To see which days of the week these sales are occurring, drag the *Day of Week* dimension over one of the date labels for example, *1999 Q2*. Drag the *Consumer* dimension into the legend (see Figure 11-44).

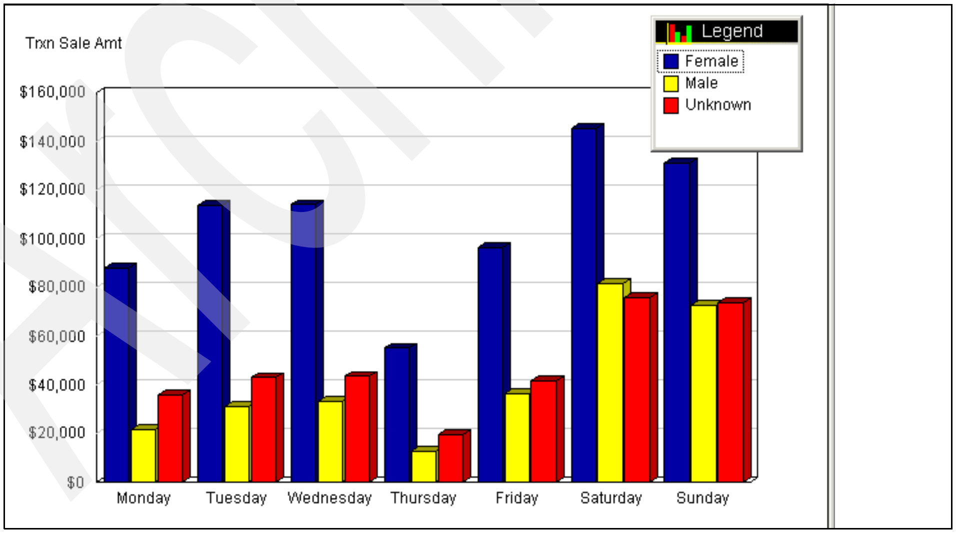


Figure 11-44 Scenario 1: report example 4

5. You should now see that sales are higher at the weekend, and that female consumers generate most of the sales. To more easily see the proportion of sales made by female consumers, switch to a stacked bar graph and display values as *% of bar total*. as shown in Figure 11-45.

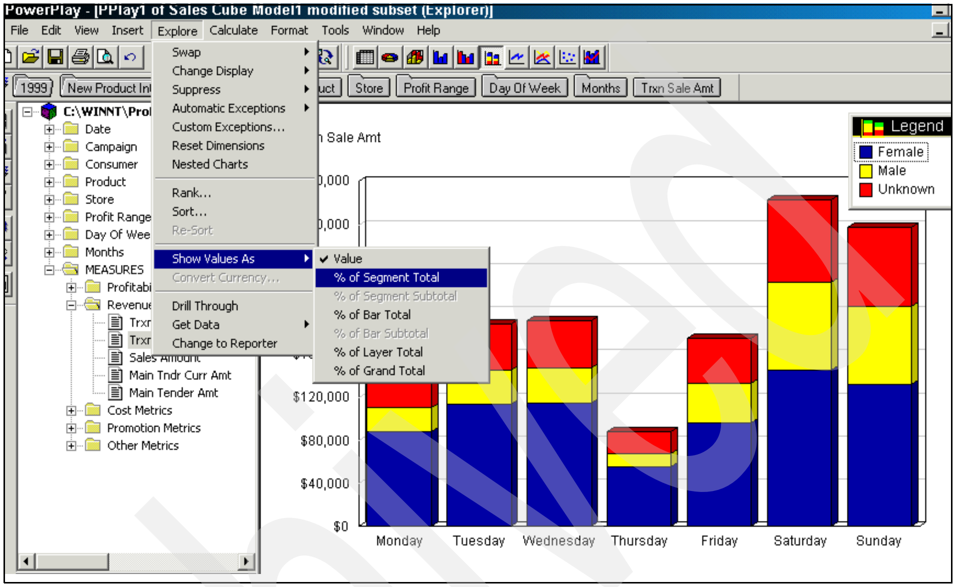


Figure 11-45 Scenario 1: report example 5

6. Hide the *Unknown* category (right-click the category and select **Hide**) You should see in Figure 11-46 that, even though most purchases are being made by female consumers, the proportion of males increases at the weekend.

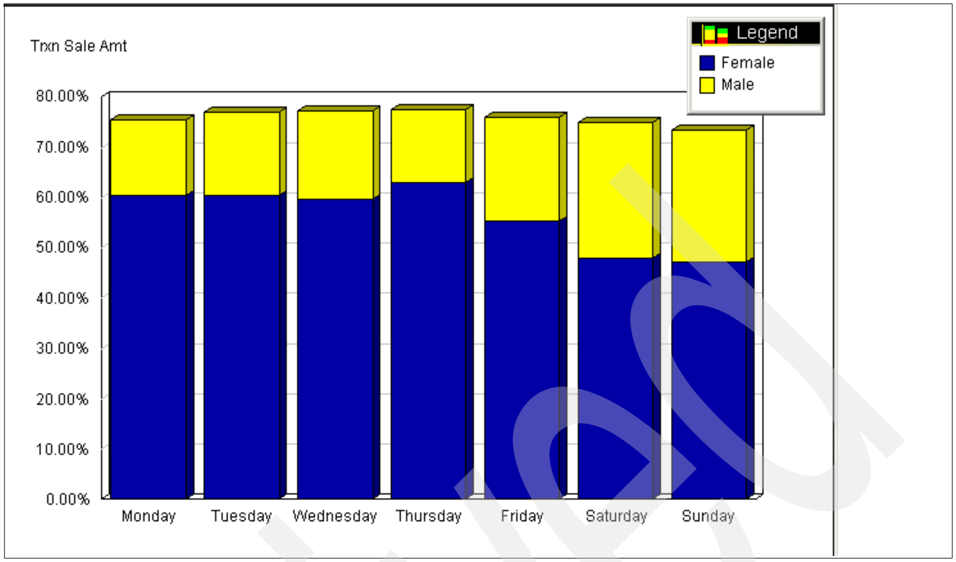


Figure 11-46 Scenario 1: report example 6

7. Drill into *Female* consumers and switch back to displaying numbers as values. We can see in Figure 11-47 that the 26-35 *Age Range* is generating the most sales.

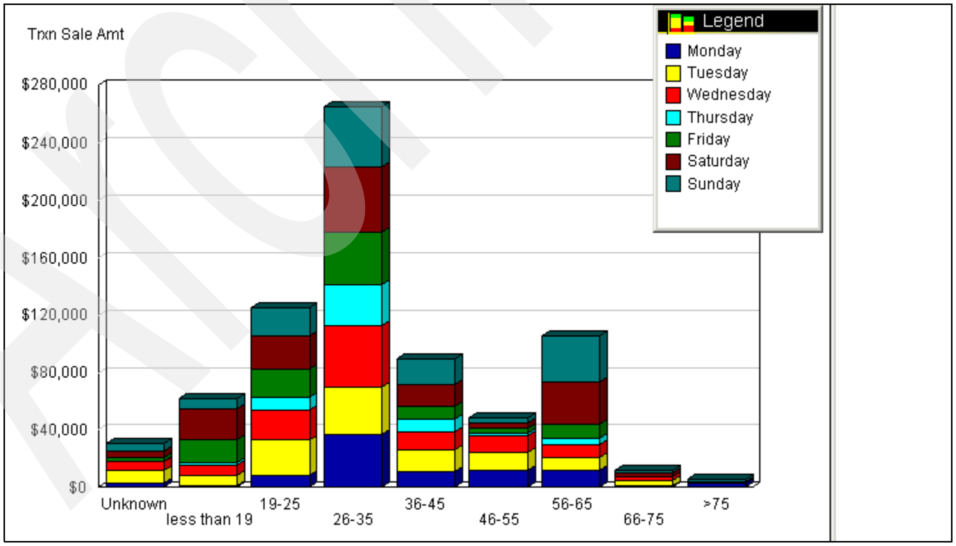


Figure 11-47 Scenario 1: report example 7

8. Now that we have analyzed our sales cube and discovered an interesting trend, we may like to drill through to perform a more detailed query against the underlying data. For example, select the 26-35 category, and click the **Drill Through** icon. You will be prompted to select a drill-through report from the list as shown in Figure 11-48.

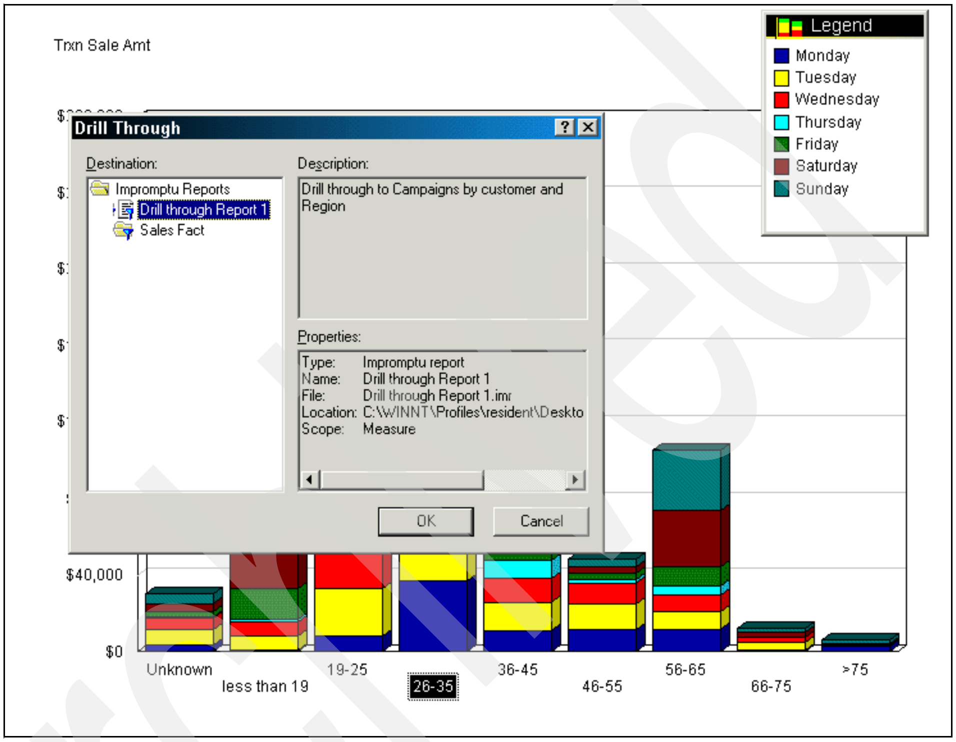
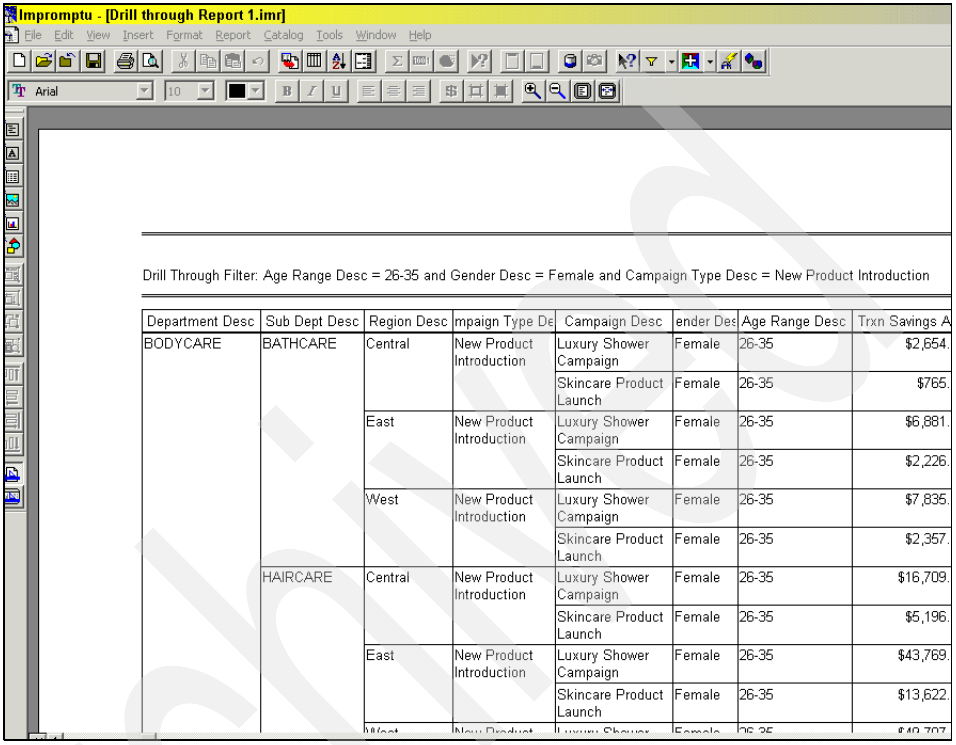


Figure 11-48 Scenario 1: report example 8

9. Select a drill-through report. You will be prompted to log into DB2. Enter your username and password. When the report runs, you will see the report in Figure 11-49, filtered on the categories selected in PowerPlay.



Drill Through Filter: Age Range Desc = 26-35 and Gender Desc = Female and Campaign Type Desc = New Product Introduction

Department Desc	Sub Dept Desc	Region Desc	mpaign Type De	Campaign Desc	ender Des	Age Range Desc	Trxn Savings A
BODYCARE	BATHCARE	Central	New Product Introduction	Luxury Shower Campaign	Female	26-35	$2,654.
				Skincare Product Launch	Female	26-35	$765.
		East	New Product Introduction	Luxury Shower Campaign	Female	26-35	$6,881.
				Skincare Product Launch	Female	26-35	$2,226.
		West	New Product Introduction	Luxury Shower Campaign	Female	26-35	$7,835.
				Skincare Product Launch	Female	26-35	$2,357.
	HAIRCARE	Central	New Product Introduction	Luxury Shower Campaign	Female	26-35	$16,709.
				Skincare Product Launch	Female	26-35	$5,196.
		East	New Product Introduction	Luxury Shower Campaign	Female	26-35	$43,769.
				Skincare Product Launch	Female	26-35	$13,622.

Figure 11-49 Scenario 1: report example 9

What are the top 10 revenue generating stores?

Top-10/bottom-10 reports are very common in sales analysis. For example:

- Which stores have the highest costs?
- Which regions are performing the best against plan?

To illustrate, we will analyze which are our top 10 stores for revenue for the organization, based on YTD growth over the same period last year.

To solve this business issue, the steps in PowerPlay are:

1. Select the alternate hierarchy for the *Stores* dimension, *All Stores*, as rows. Select the relative time category *YTD Grouped* as columns. Select *Sales Amount* as the active measure. You should see something similar to Figure 11-50.

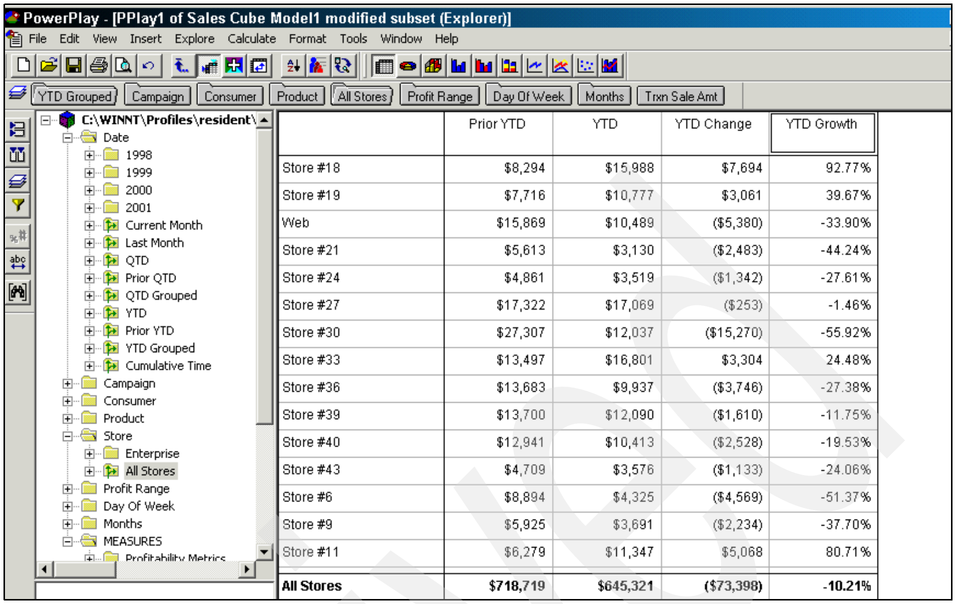


Figure 11-50 Scenario 2: report example 1

2. Using the **Rank** toolbar icon (or by selecting **Rank** from the **Explore** menu), rank on *YTD Growth*, and select *Top 10* shown in Figure 11-51.

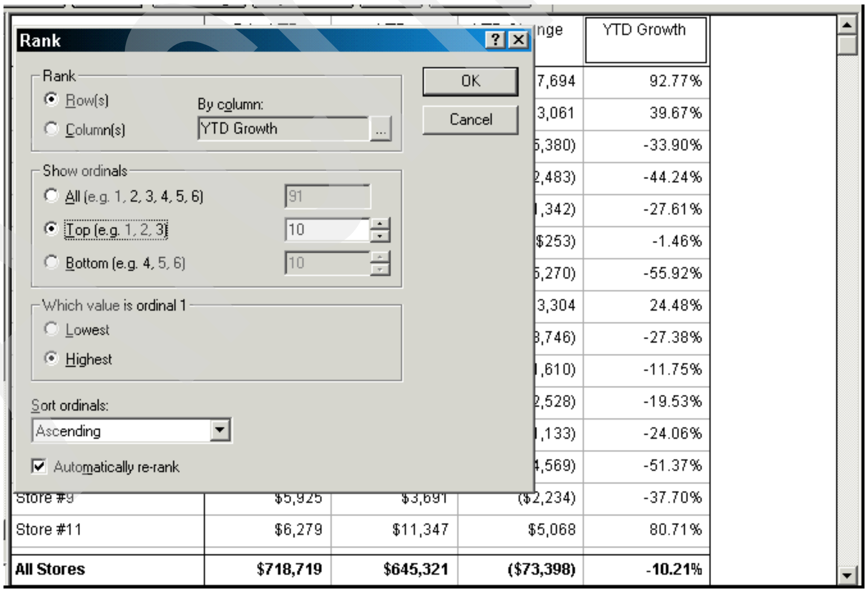


Figure 11-51 Scenario 2: report example 2

3. The result is the report in Figure 11-52.

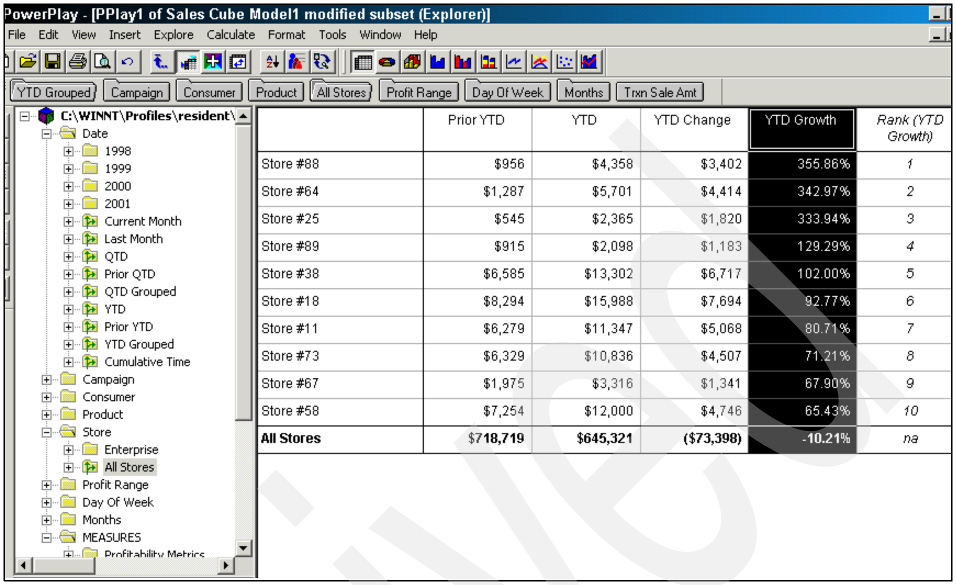


Figure 11-52 Scenario 2: report example 3

Where are we getting most of our high profit margin?

In order to understand which sales transactions were highly profitable, then understand where they are occurring, we would want to enhance the metadata to capture margin ranges, then add this to our Cognos cube model as an *Exception Dimension* which can then be used for reporting.

To do this:

1. We would first need to add the margin range as a calculation in the Impromptu catalog as shown in Figure 11-53.

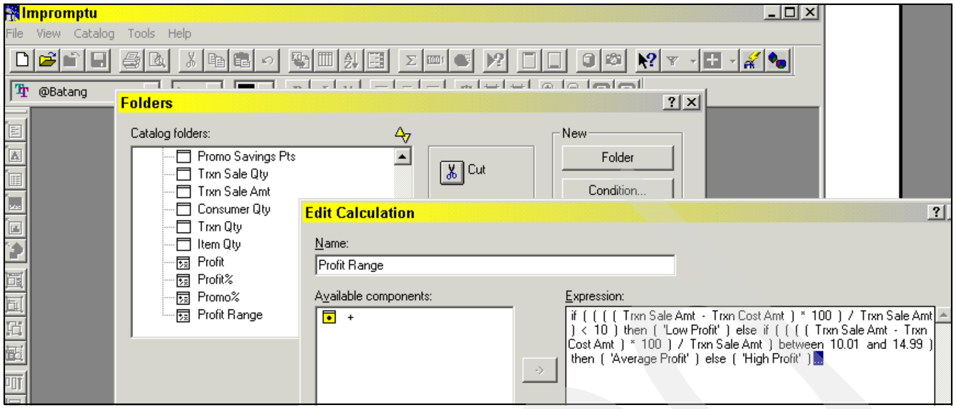


Figure 11-53 Scenario 3: add a new calculation

2. We then add this calculation to the Fact iqd, and finally to the Transformer Model. We now have this margin calculation available as a dimension. For example, to answer the question:
   - How many high profit transactions occurred, by region, in 2001?

     We built the graph in Figure 11-54.

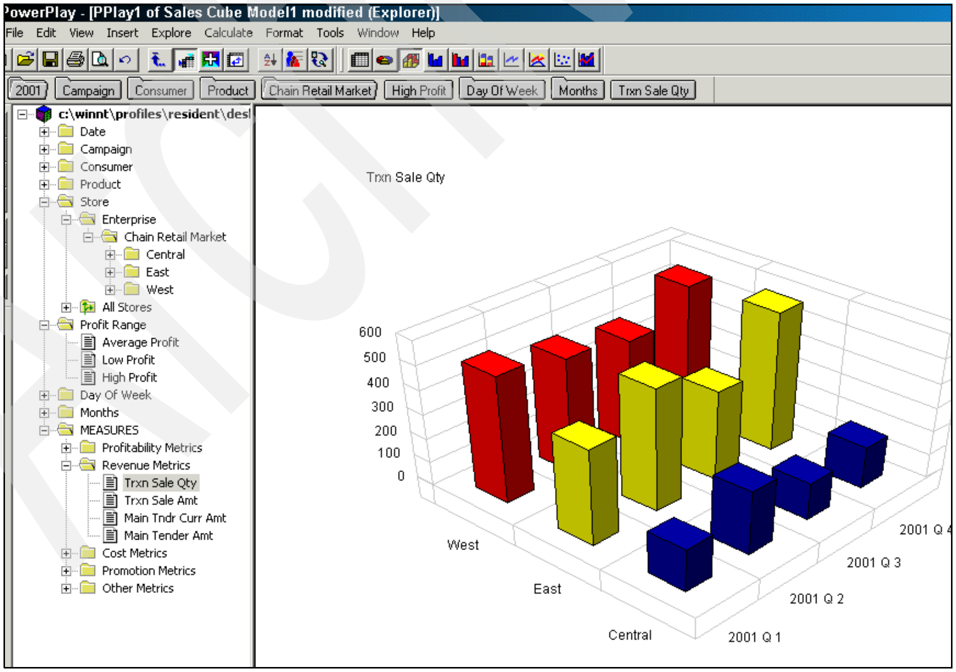


Figure 11-54 Scenario 3: the graph

11.6.2 Financial analysis scenario

This typically takes the form of profitability analysis over time, and may include forecasting. For example, financial analysis often asks for profit growth this month versus last month, this quarter versus last quarter, or this quarter versus the same quarter last year.

In this section we focus on the following typical financial issues:

- How have sales related profit margins changed over time?
- How can we forecast the sales for the upcoming year?

How have sales related profit margins changed over time?

A possible path to answer this business question may be as follows:

1. To view the profit generated by sales, select the *Profitability Metrics* as columns, and *Day of Week* as rows. We can see in Figure 11-55 that weekends generate the most $ profit, however Monday-Thursday is where we gain our best *% Profitability*.

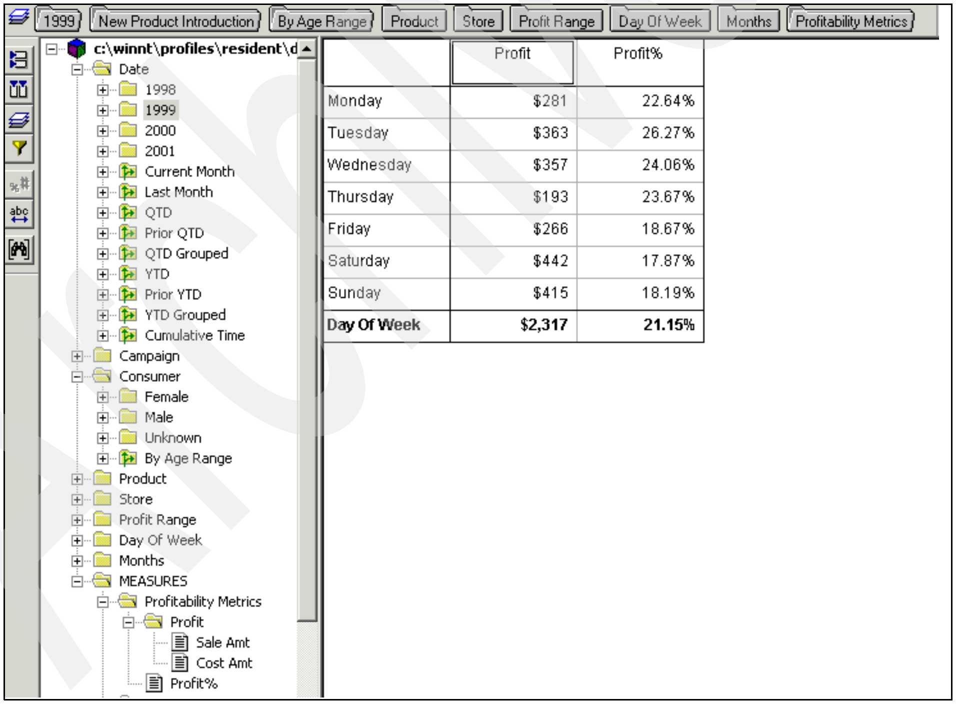


Figure 11-55 Financial scenario: report example 1

2. Let us look at this *YTD*, compared to the same period last year. Drag in *YTD Grouping* from the *Date* dimension. We get quite a different picture as shown in Figure 11-56.

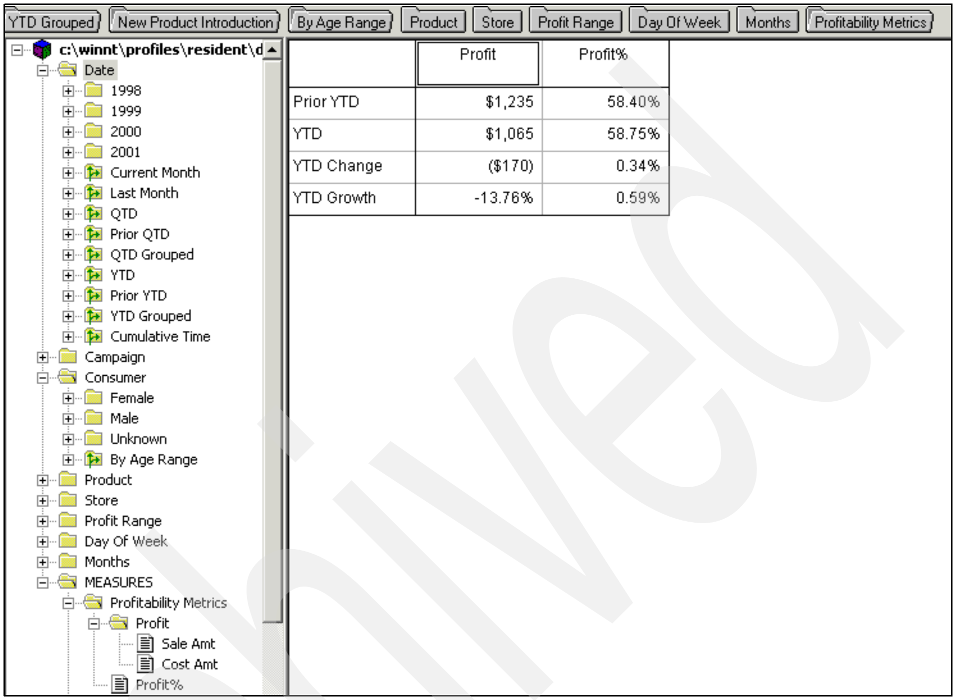


Figure 11-56 Financial scenario: report example 2

3. Drill into the Profit Measure, to see the source measures it is derived from. We can see in Figure 11-57 that our *Costs* have reduced, but our *Sales* have reduced by more, hence the reduction in profit compared to last year.

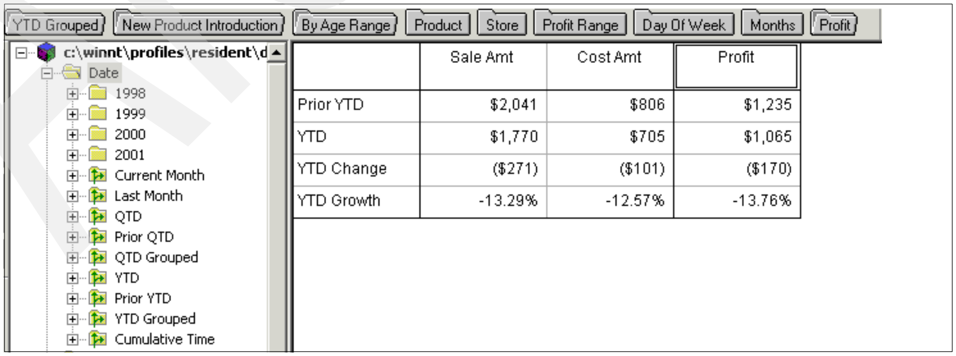


Figure 11-57 Financial scenario: report example 3

How can we forecast the sales for the upcoming year?

Suppose we now want to forecast our sales for the upcoming year, for *Grand Opening Sales* (*Campaign* dimension) of *Homestyle* products (*Product* dimension), in *Utah* (*Stores* dimension).

These are the steps:

1. Switch to a Line chart, filter on *Grand Opening Sales*, *Utah*, *Homestyle.*
2. Select *Sales Amount* as the active measure, and drag in *Years*.
3. Choose the forecast method as shown in Figure 11-58.

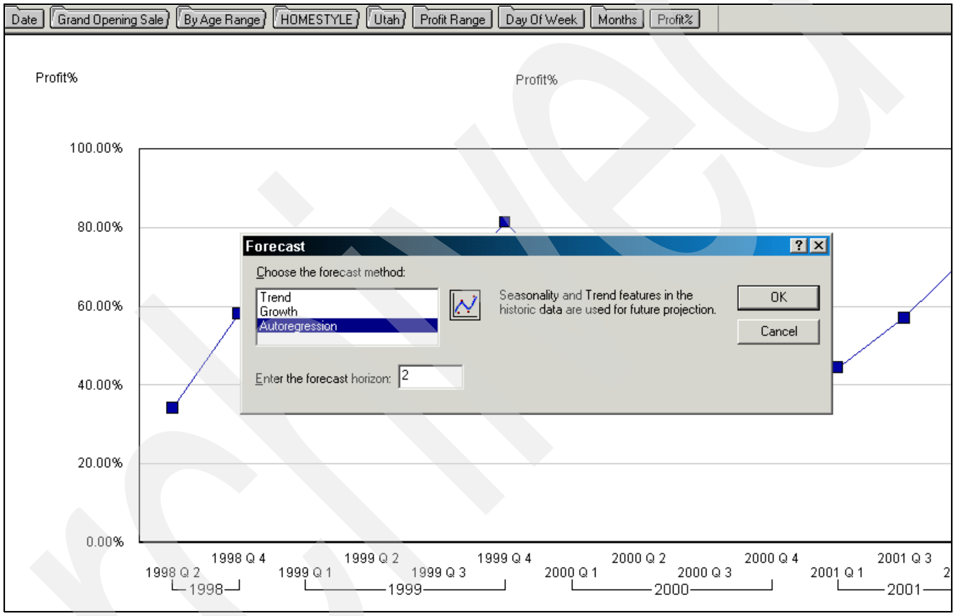


Figure 11-58 Forecasting scenario: Forecast option

4. We now have a simple sales forecast (see Figure 11-59), taking into account seasonality in the historic data (these sales projection numbers could then be saved out, for example to feed back into a Sales Force Automation (SFA) system).

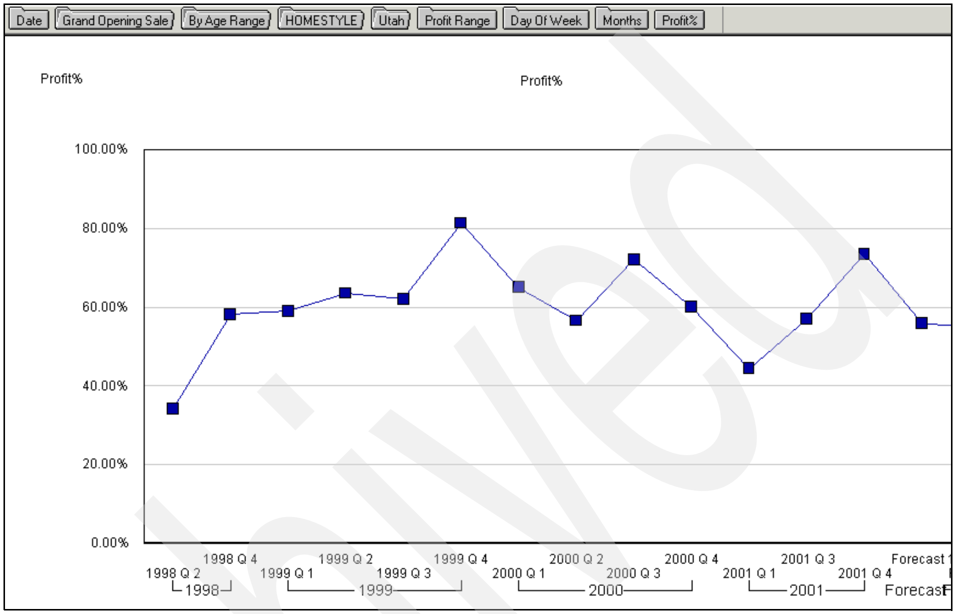


Figure 11-59 Forecasting scenario: result

5. Finally, suppose the user needs to disconnect from the network and travel to a meeting, but would like to perform further analysis during the flight. Simply select **Save As PowerPlay Cube**, as shown in Figure 11-60, and a mobile sub-cube will be created at the point the user is in the cube (for example, *Grand Opening Sales* campaigns, *Homestyle* products, *Utah* stores)

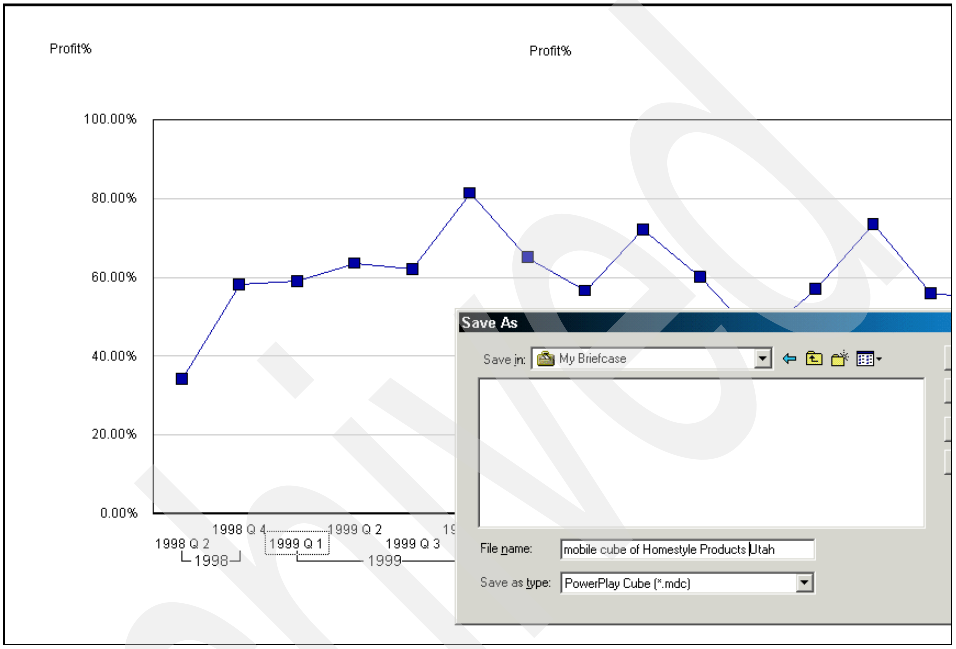


Figure 11-60 Create a mobile sub-cube

6. This cube can then be opened up as shown in Figure 11-61 and further analysis performed, even though the user is disconnected from the network.

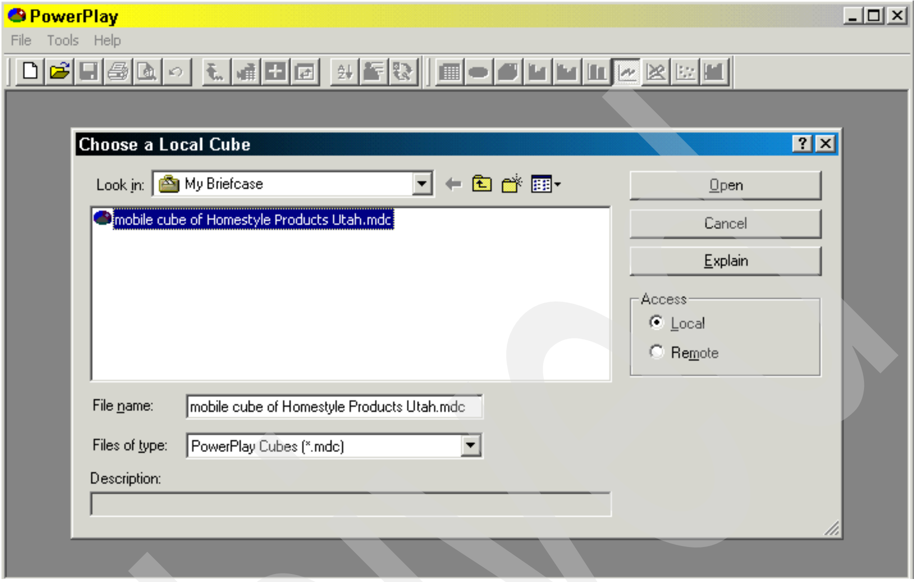


Figure 11-61 Open the PowerPlay sub-cube saved

The basis for the scenarios we used came from the Cognos Analytic Applications. These Applications combine software and services expertise from Cognos with the best practices, thinking, and experience of business experts. The result is business intelligence software that comes with built-in reports, key metrics, and integrated business processes. Customers can be up and running (and gaining value) from business intelligence technology, quickly and easily.

The hundreds of pre-built reports, metrics, and information and process connections that come with Cognos Analytic Applications enable customers to fully realize their investments in CRM, ERP, and other operational data systems. Cognos Analytic Applications draw from these operational data sources to provide the reports, analysis, and scorecards that bring real business value.

11.6.3 Performance results with MQT

Most of the reports created and ran for the two scenarios used the MQTs.

Table 11-3 summarizes some of the grouping combinations tested with Cognos and the results observed when using the DB2 Explain access plan graph.

Table 11-4 Impromptu report performance result

Cognos query type examples	DB2 explain results: timerons	
	Without MQT	**With MQT**
Drill-through	38690.26	975.93
Group by region area	758,072.81	25.18
Group by age range and region	764,946.94	13,822.72
Group by gender, age range	763,935.25	13,822.72
Group by age range, region and campaign type	767,485.88	13,822.72
Group by gender, age range, region, campaign type and cell	772,457.12	13,822.72

Both Cognos PowerPlay for drill through and Cognos Impromptu catalog for other SQL reports may benefit from MQTs.

The drill through report is basically a SQL report that a report author would create. Optimally, as explained in 11.4.1, "Optimizing drill through" on page 502, that report would be created with the knowledge of the MQTs and therefore built to leverage the MQT. The PowerPlay Cube allows users to navigate the data without having to connect to the database. They would perform OLAP analysis in PowerPlay until they get to a point where they wanted additional data that was not in the PowerCube or a list style report with invoice numbers, for example, on it that they would drill through. When the user *drills through*, he is using the Impromptu report which has been already defined and designed to leverage the MQT. The end user who performs the drill through does *NOT* build their own report from scratch. The navigation they perform in the PowerCube will be passed on in a filter to the Impromptu before it is executed against the database (sent to the Optimizer).

So we could say that *ALL* reports in Impromptu have the capability to use the MQTs.

A good practice when designing star schema and cube model to leverage MQT performance, would be:

- Understand the business rules and business questions that the users want to ask.
- Find out what are the most common types of inquiries the users want to perform and then design the star schema around that.
- Define DB2 Cube Views cube model and cubes to meet those expectations.

11.7 Conclusion: benefits

The Cognos BI Solution is able to derive immediate benefit from the DB2 Cube Views. The Cognos DB2 Dimensional Metadata Wizard connects directly to DB2 DB2 Cube Views, and extracts the metadata, mirroring it to the Cognos BI Solution. Dimensions, measures, and hierarchies are replicated, providing extensibility to the DB2 Cube Views, and the further capability to enhance and build upon the business value presented by the metadata.

To build the reports in "Scenarios" on page 524, Cognos ran several SQL based reports leveraging the MQTs and the metadata from DB2 Cube Views. Dramatic improvements in performance were realized with these reports. The architecture of the Cognos BI Solution leverages the improved query performance with the use of SQL based Impromptu reports, via drill through and direct query. Maximum benefit of the MQTs is realized when the SQL reports themselves are built using the intelligence of the metadata in combination with the definitions of the MQTs. Optimal performance of the Impromptu report is realized.

The metadata import greatly facilitates the designing and creation of effective and meaningful MOLAP PowerCubes. PowerCube data population uses transaction level data, and as MQTs are generally built at summary levels, this process will not necessarily leverage MQTS. Users navigate large volumes of summarized data with sub-second response times using PowerCubes.

At the time of this publishing, Cognos was set to release Cognos ReportNet. Cognos ReportNet was designed and developed to meet the requirements of all areas of enterprise reporting (ad hoc reporting, managed reporting, production reporting). ReportNet leverages DB2 Cube Views, providing complete reporting access.

Cognos offers the widest spectrum of BI capabilities in the industry. Reporting, analysis, ad hoc query, scorecarding, visualization, and event detection are all delivered in a seamless BI environment that allows users to navigate integrated information intuitively and manage business performance from a central portal. Combined with the metadata management, high speed aggregations, summary and optimizations of DB2 Cube Views, the complete Cognos-IBM Solution is sure to meet and exceed customer expectations. Cognos BI embraces and enhances DB2 Cube Views, extending business value by providing robust and complete reporting and analysis, scorecarding, planning and monitoring capabilities.

Cognos is the world leader in business intelligence (BI) and performance planning software for the enterprise. Cognos solutions let companies drive performance with enterprise planning and budgeting; monitor performance with enterprise scorecarding; and understand their performance with the reporting and analysis of Enterprise business intelligence. To help its customers maximize the value of their investment in Cognos software, the company offers award-winning support and services available around the world, through support centers located in North America, Europe and Asia/Pacific.

Founded in 1969, Cognos serves more than 22,000 customers around the world.

12

Accessing DB2 dimensional data using BusinessObjects

This chapter describes certain deployment scenarios for the BusinessObjects Universal Metadata Bridge. It explains how to implement and to use the Universal Metadata Bridge, and discusses benefits of its use. The objective is to provide more detailed information about the way the bridge carries out mapping. This will enable the audience to understand how they can benefit from using the bridge, as well as providing an understanding of its limits.

12.1 Business Objects product overview

The launch of BusinessObjects Enterprise 6 brings the entire BusinessObjects product line under the umbrella of a single integrated product suite.

BusinessObjects Enterprise 6 is the industry's leading suite of integrated business intelligence products. It enables organizations to track, understand, and manage enterprise performance. Unique to the market, Enterprise 6 provides the Industry's best Web query, reporting, and analysis; the most advanced and complete suite of analytic applications; a broad set of connectors to applications; and is the most integrated BI suite.

12.1.1 BusinessObjects Enterprise 6

BusinessObjects Enterprise 6 contains data integration products, a business intelligence platform, and enterprise analytical applications as shown in Figure 12-1.

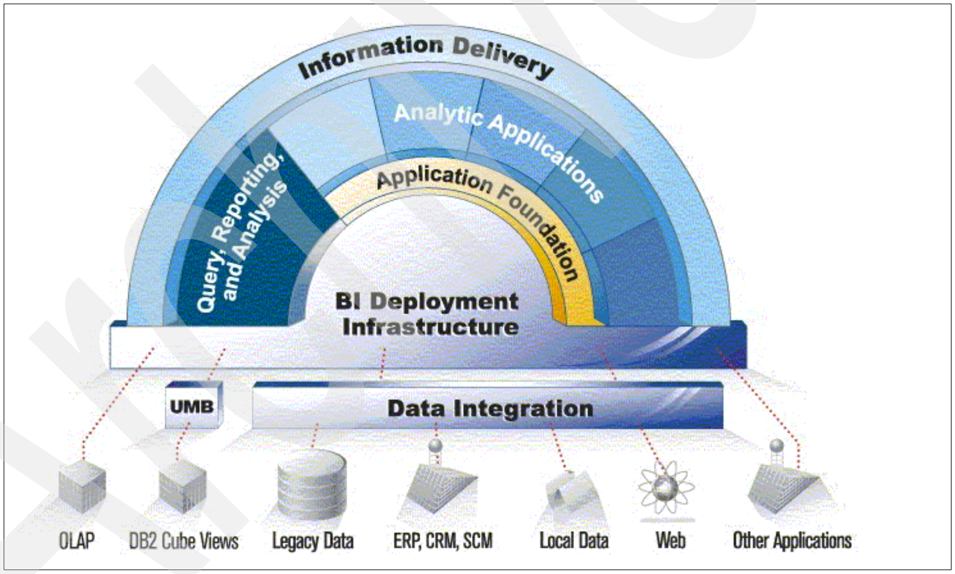


Figure 12-1 BusinessObjects Enterprise 6 product family

In this section we will only discuss on Business Intelligence platform. For more information on BusinessObjects product line, please check the Web site:

`http://www.businessobjects.com`

The business intelligence platform is sub-divided into three areas:

- Query, reporting, and analysis
- Information delivery
- BI deployment infrastructure

Query, reporting, and analysis

BusinessObjects Enterprise 6 provides query, reporting, and analysis products to meet the needs of all users with both Windows and Web-based interfaces.

BusinessObjects provides integrated query, reporting, and analysis for the enterprise. It is a Web-enabled full-client product that allows users to easily query, report, analyze, and share the wealth of information stored in multiple data sources within and beyond the enterprise.

WebIntelligence is the industry's best query, reporting, and analysis solution for the Web. WebIntelligence is a thin-client solution that enables users to query, report, analyze, and share corporate data using a simple browser as their interface, while maintaining tight security over data access.

BusinessQuery for Excel opens up the power and ease of use of BusinessObjects data access to users of Microsoft Excel. It offers users a simple way to retrieve data from corporate databases, then combine and analyze it in their favorite spreadsheet.

Information delivery

BusinessObjects Enterprise 6 meets information delivery requirements through the combination of a BI portal and powerful broadcast capabilities.

BusinessObjects InfoView is a Business Intelligence Portal (BIP) that collects and consolidates a company's BI information and presents it in a secure, focused, and personalized view to users inside and outside an organization. InfoView lets users personalize how they view, manage, and distribute BI content. It is both a standalone Business Intelligence Portal (BIP), as well as a BI content provider for Enterprise Information Portals (EIPs).

BusinessObjects Broadcast Agent delivers timely and personalized information via multiple devices to hundreds of thousands of users, through intelligent and cost effective delivery mechanisms. It provides a scalable solution to drive the quick delivery of business driven alerts and mission critical information that decision makers need, however, whenever, and wherever they need it.

BI deployment infrastructure

BusinessObjects Enterprise 6 provides a full business intelligence deployment infrastructure including administrative tools to set up and manage an enterprise BI deployment, and development tools to customize and extend the BI deployment.

BusinessObjects Designer is a graphical design tool used to create the rich "semantic layer" (metadata) that makes Business Objects products so intuitive for non-technical users. Designer can also work with existing metadata in your enterprise.

BusinessObjects Universal Metadata Bridge leverages existing metadata to automatically create universes.

BusinessObjects Developer Suite is a development toolkit that allows customers and partners to customize, integrate, and extend BusinessObjects Enterprise 6. The Developer Suite modules provide everything a developer would need (object models, documentation, and samples) to customize and integrate Enterprise 6 into an existing IT infrastructure.

12.2 BusinessObjects Universal Metadata Bridge overview

The BusinessObjects Universal Metadata Bridge provides seamless integration between DB2 Cube Views and BusinessObjects products. It allows you to create BusinessObjects universes using metadata from DB2 Cube Views repositories that can be exported in DB2 Cube Views XML files. It also allows you to call a COM module and integrate bridge functionalities in your own program, creating a complete bridge between DB2 Cube Views and BusinessObjects Designer.

By providing easy access to DB2 Cube Views' metadata, you leverage your investment in existing technology, increase the efficiency and effectiveness of BusinessObjects universe management, and optimize your BusinessObjects reports' queries when using DB2 Cube Views' MQTs.

There are three metadata exchange modes available:

- Application mode, which includes easy-to-use user interface for creating an universe.
- API mode, using bridge-specific Application Program Interface (API) functions placed in your own applications.
- Batch mode, a silent, automatic mode of application execution, allows you to create several universes at once.

Figure 12-2 shows how the BusinessObjects Universal Metadata Bridge uses metadata from a DB2 Cube Views repository to create BusinessObjects universes.

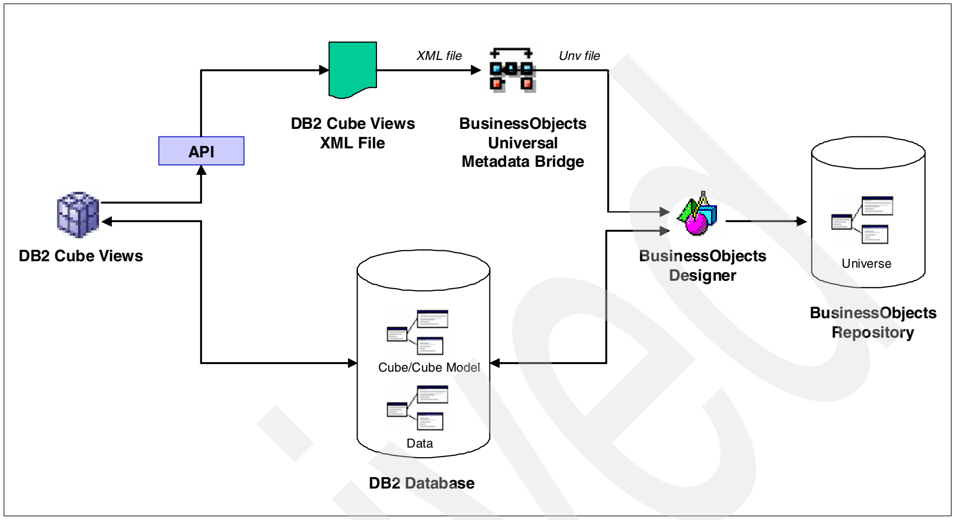


Figure 12-2 Metadata flow

The DB2 Cube Views metadata is exported in a DB2 Cube Views XML file. This file can be imported by BusinessObjects Universal Metadata Bridge to automatically create BusinessObjects universe file according to mapping rules.

The BusinessObjects Universal Metadata Bridge makes the universe creation process simple and convenient.

12.2.1 Metadata mapping

This section explains how BusinessObjects Universal Metadata Bridge maps a Cube or Cube Model to a BusinessObjects Universe.

BusinessObjects classes and objects are derived from the DB2 Cube Views objects according to the mapping listed in Figure 12-3 and in Figure 12-4.

DB2 Cube Views Object		BusinessObjects Object	
Cube Model/ Cube	Name	Short name (file name)	Universe
	Business name	Name	
	Comments	Description	
Cube Dim	Business Name	Name	Class of dimensions
	Comments	Description	
Cube Facts		Name = “Measures”	Class of Measures
	Comments	Description	
	List of Measures	Measures Objects of class	
Cube Hierarchy Or Hierarchy	Business Name	Name	Custom Hierarchy
	List of Attributes	Objects of the hierarchy (ordered)	

Figure 12-3 Metadata mapping

DB2 Cube Views Object		BusinessObjects Object	
Attribute	Business Name	Name	Dimension or Detail type object
	Comments	Description	
	SQL Expression Template	Select	
	Datatype name	Type <*Character*>	
		List of Values: associate default list of values	
Measure	Business Name	Name	Measure type object
	Comments	Description	
	SQL Expression Template	Select	
	Datatype	Type <*Character*>	
	First Aggregation of List of Aggregations	Function <*Sum*>	
Attribute Relationship	LeftAttribute	Change RightAttribute-related object qualification to Detail and associate it to LeftAttribute-related object	
	RightAttribute		

Figure 12-4 Additional metadata mapping

Note: The values between the < > marks are default values to assign to BusinessObjects objects properties that cannot be Null and when the corresponding value is missing in DB2 Cube Views.

You can visualize these mappings in the following numbering on the diagram in Figure 12-5:

1. Measure class is created in the BusinessObjects universe for the list of the measures defined in the fact.

2. For each measure of the Cube/Cube Model, a measure is created in the BusinessObjects universe with its business name.
3. For each dimension of the Cube/Cube Model, a class is created in the BusinessObjects universe with its business name.
4. For each attribute of the Cube/Cube Model, an object is created in the BusinessObjects universe with its business name

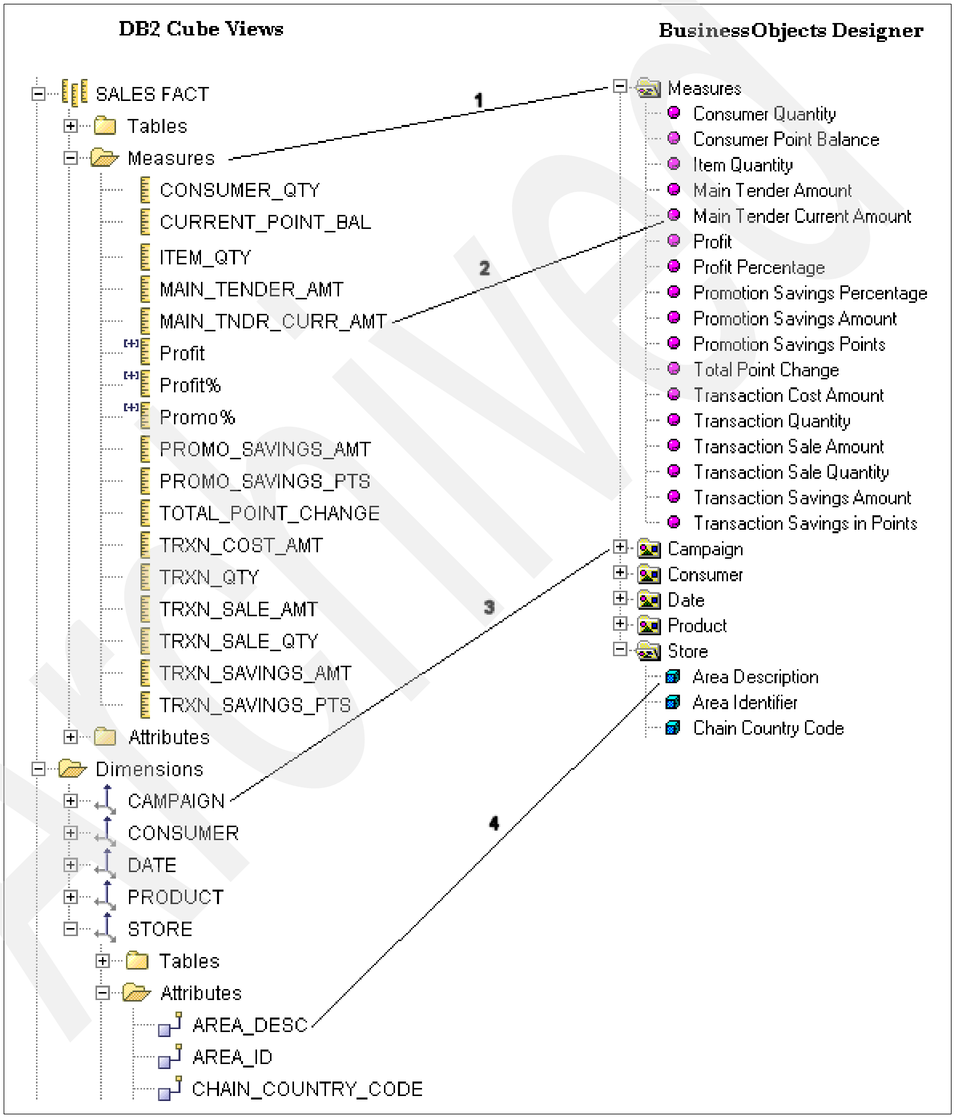


Figure 12-5 Cube model to universes mapping

Additional mapping is providing in Figure 12-6:

5. For each hierarchy of the Cube/Cube Model, a custom hierarchy is created in the BusinessObjects universe with its business name.
6. For each level of a hierarchy, an ordered object is created in its associated BusinessObjects custom hierarchy.

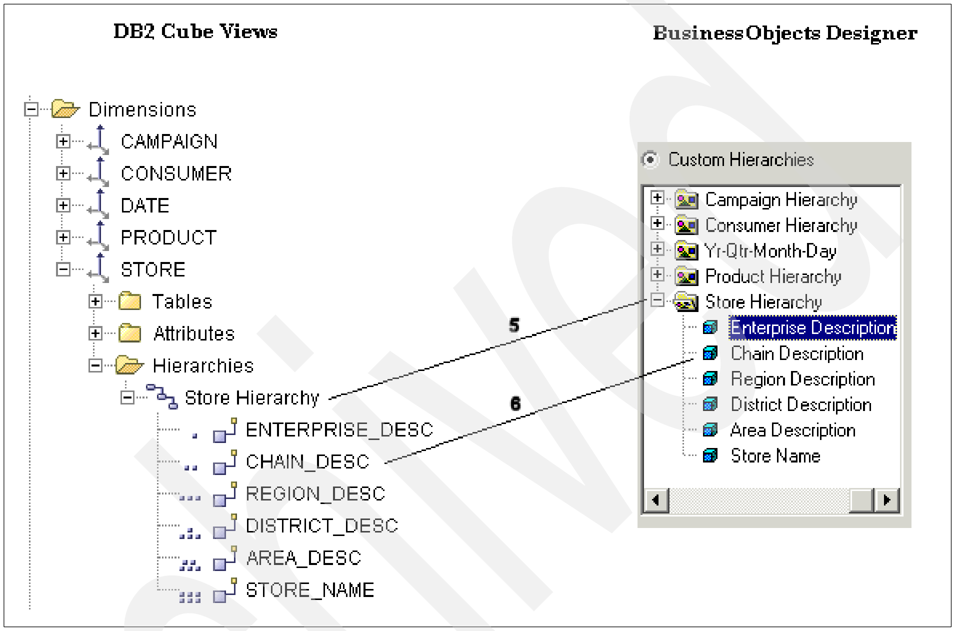


Figure 12-6 Hierarchies mapping

7. For each join of the Cube/Cube Model, a join is created in the BusinessObjects universe. The join type, operator and cardinality are mapped as in Figure 12-7.

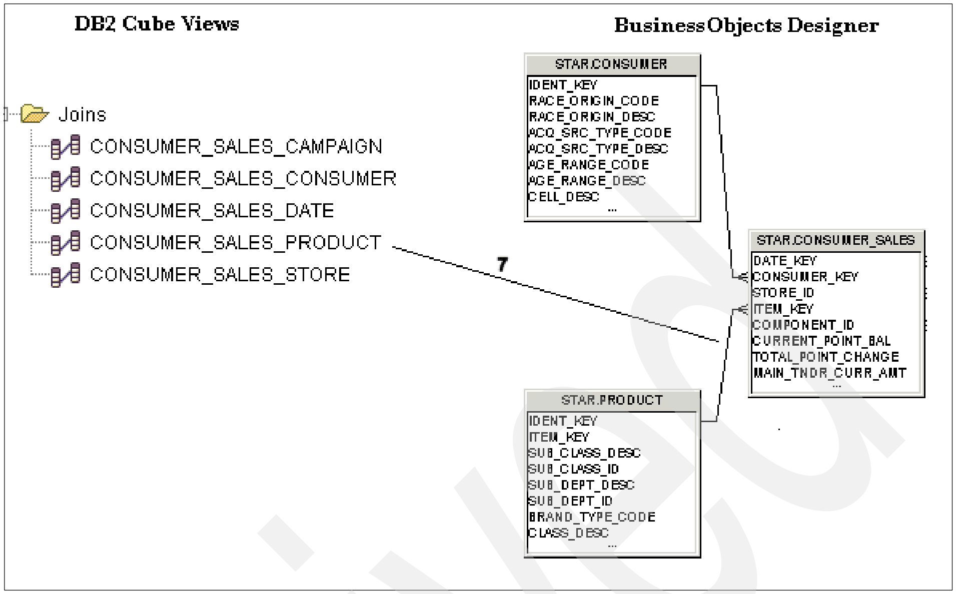


Figure 12-7 Joins mapping

8. For each descriptive attribute relationship, an associated dimension within a dimension/detail relationship is created in the BusinessObjects universe with its business name.
9. For each descriptive attribute relationship, a detail within a dimension/detail relationship is created in the BusinessObjects universe with its business name.

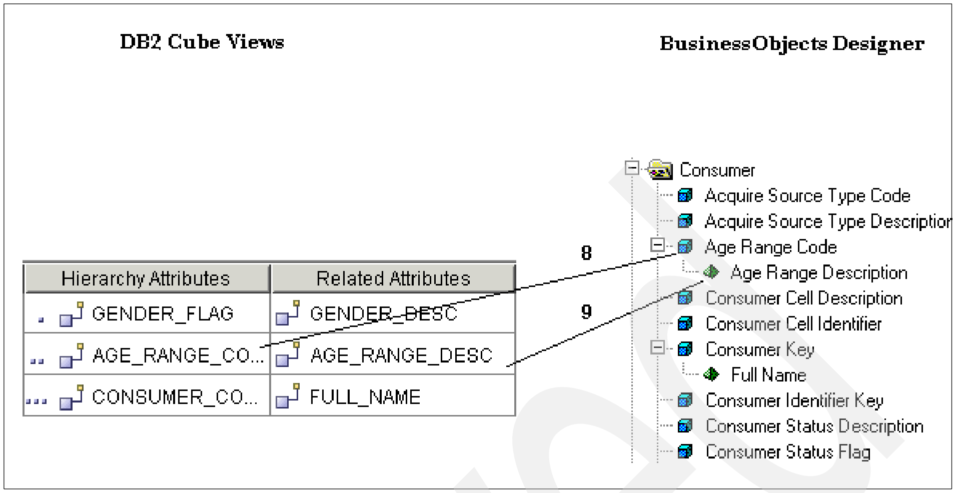


Figure 12-8 Descriptive values mapping

12.2.2 Complex measure mapping

Some DB2 Cube Views complex measures contain more than one aggregate function.

Before mapping each measure, the BusinessObjects Universal Metadata Bridge determines the number of aggregate functions, and then creates a measure for each one.

The name of the BusinessObjects measure is composed of the following parts:

- Business name
- Dimensions involved in this aggregation (listed in dimensionRef property of the aggregation) in the DB2 Cube Views XML file.

The name of the measure respects the following rule:

```
<measure name> by <dimension>
```

The exception is where the dimensionRef property is empty. In this case the name is:

```
<measure name> by other dimension
```

> **Note:** An empty space in a dimension name is turned into underscore "_" in the measure name. For example: the Sales Rep dimension becomes Sales_Rep.

The aggregate function of the BusinessObjects measure matches the aggregate function in DB2 Cube Views.

The other information used by the BusinessObjects Universal Metadata Bridge to create a BusinessObjects measure is as follows:

- The SQL expression of the Select property of BusinessObjects measure is deduced from SQLExpression in DB2 Cube Views for simple measures.
- The description of the BusinessObjects measure is composed of the original description from DB2 Cube Views and additional comment describing the measure dimension context:

```
Applied on dimensions: dim1, dim2, dim3
```

Example of a mapped complex measure

When a complex measure with multiple aggregations exists, the BusinessObjects Universal Metadata Bridge creates measures for each aggregation. For example, the Inventory measure in Figure 12-9 has three aggregations: AVG, MAX, and SUM.

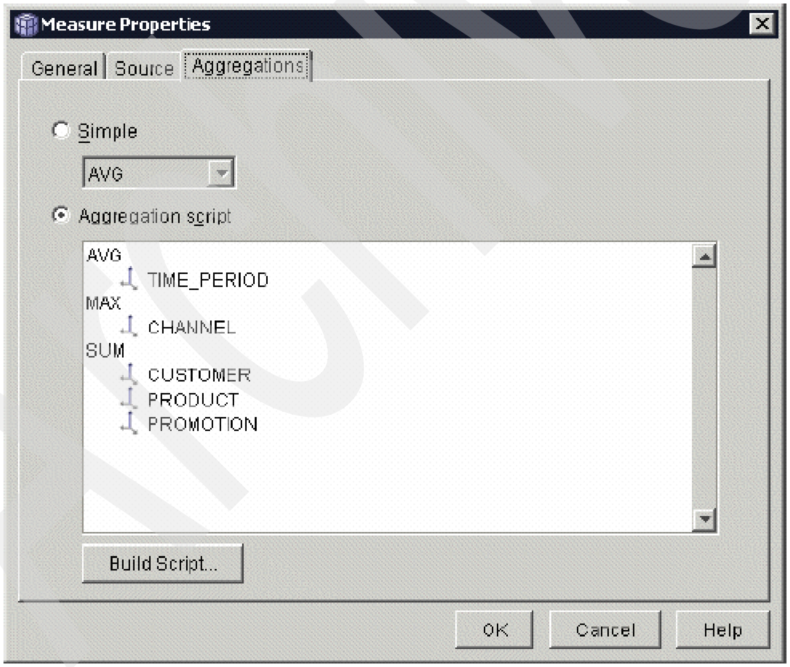


Figure 12-9 Complex measure with multiple aggregations

The DB2 Cube Views XML file is shown in Figure 12-10.

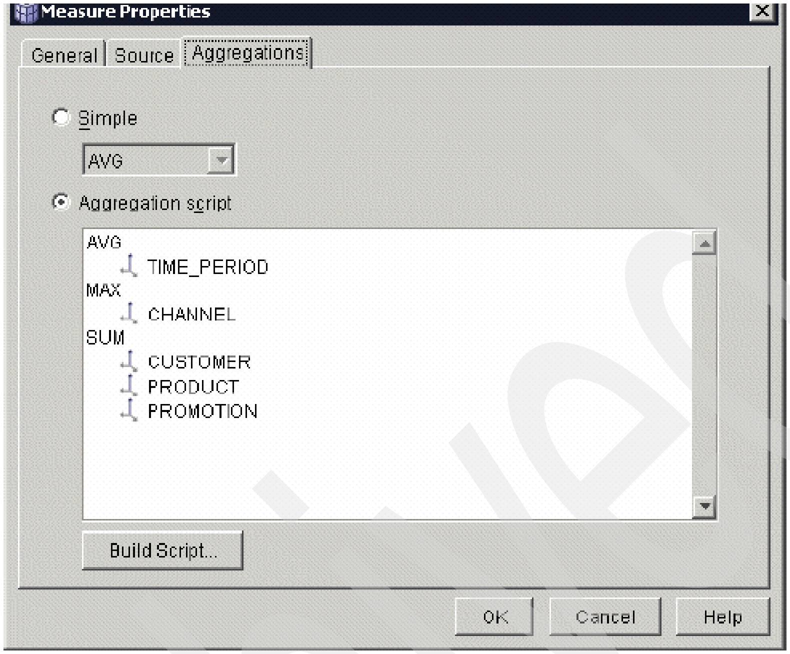


Figure 12-10   DB2 Cube Views XML file for multiple aggregations

When the BusinessObjects Universal Metadata Bridge reads the XML file, it converts the aggregations to measures.

The AVG aggregation becomes Inventory by Time_Period as shown in Figure 12-11.

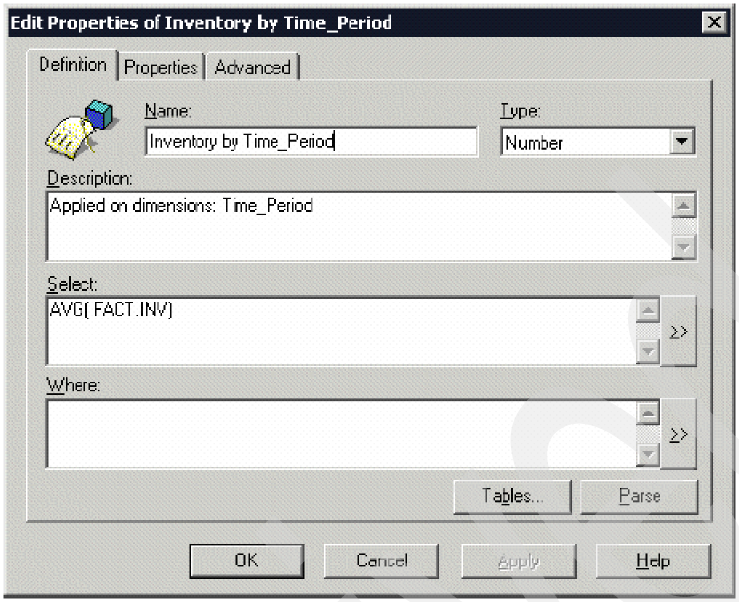


Figure 12-11 AVG aggregation example

The MAX aggregation becomes Inventory by Channel as in Figure 12-12.

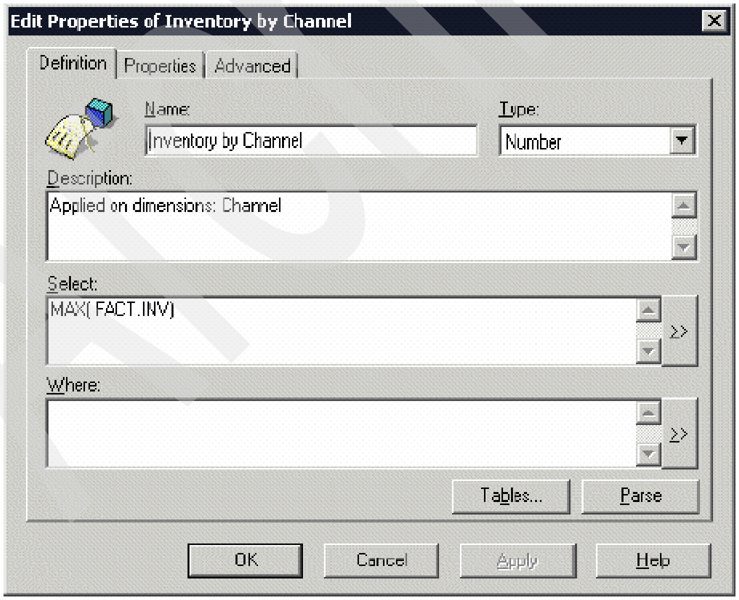


Figure 12-12 MAX aggregation example

The SUM aggregation becomes Inventory by other dimensions as in Figure 12-13.

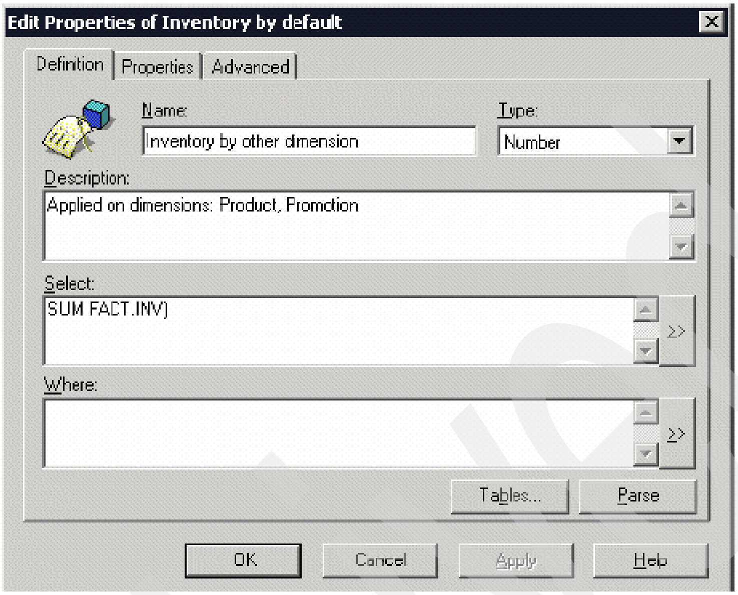


Figure 12-13 SUM aggregation example

The universe list includes the new measures as shown in Figure 12-14.

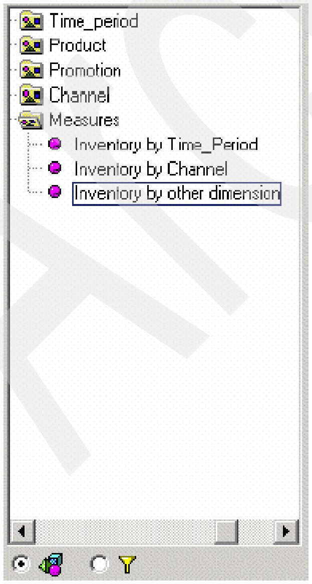


Figure 12-14 Universe result

12.2.3 Data type conversion

BusinessObjects objects support four data types:

- Date
- Character
- Number
- Long text

Figure 12-15 lists the internal data types and their equivalent in BusinessObjects objects.

DB2 Datatype	BusinessObjects Datatype
CHARACTER	CHARACTER
CLOB	LONG TEXT
DATE	DATE
DECIMAL	NUMBER
DOUBLE	NUMBER
FLOAT	NUMBER
INTEGER	NUMBER
LONG VARCHAR	LONG TEXT
NUMERIC	NUMBER
SMALLINT	NUMBER
TIME	DATE
TIMESTAMP	DATE
VARCHAR	LONG TEXT

Figure 12-15 Data types

12.3 Implementation steps

As already mentioned in 12.2, “BusinessObjects Universal Metadata Bridge overview” on page 546, there are three metadata exchange modes available:

- Application mode as described in Figure 12-16.

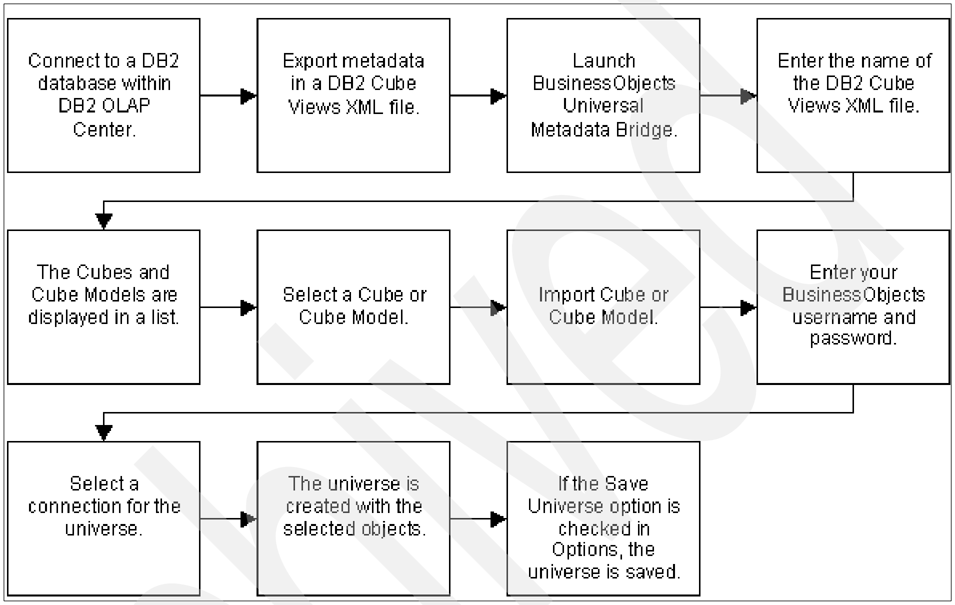


Figure 12-16 Application mode process

- API mode as described in Figure 12-17.

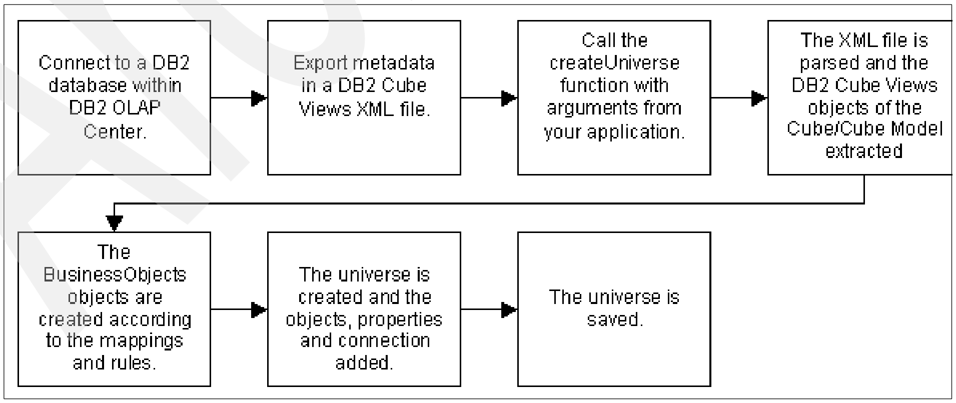


Figure 12-17 API mode process

- Batch mode as described in Figure 12-18.

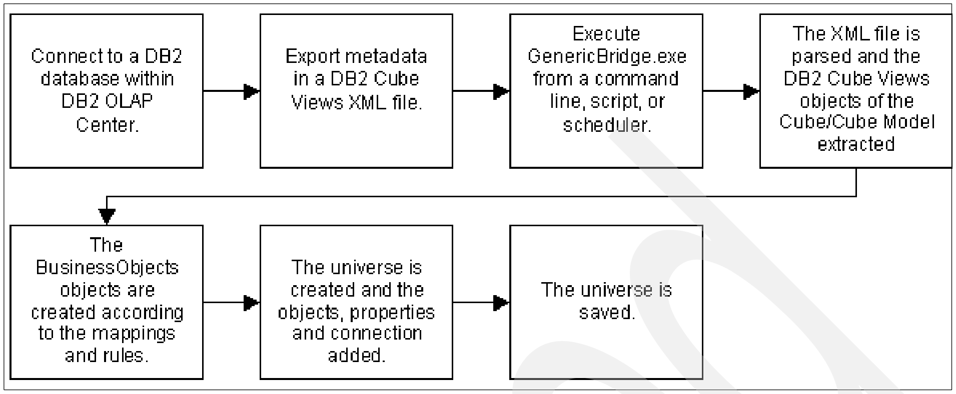


Figure 12-18 Batch mode process

These are the steps to implement the metadata bridge:

- Export the metadata from DB2 OLAP Center.
- Map the metadata in the BusinessObjects based on the mode process chosen.

12.3.1 Export metadata from DB2 OLAP Center

Before launching BusinessObjects Universal Metadata Bridge, you must first export DB2 Cube/Cube Model metadata from DB2 OLAP Center (see Figure 12-19).

1. To connect to a DB2 database:
   a. Click \ from the OLAP Center main window. The Database Connection window opens.
   b. In the database connection window, enter the following information:
      - In the Database name field, select the database to which you want to connect.
      - In the User name field, type the user ID for the database that you specified.
      - In the Password field, type the password for the user ID that you specified.
   c. Click OK. The metadata objects in your connected DB2 are displayed in the OLAP Center object tree.

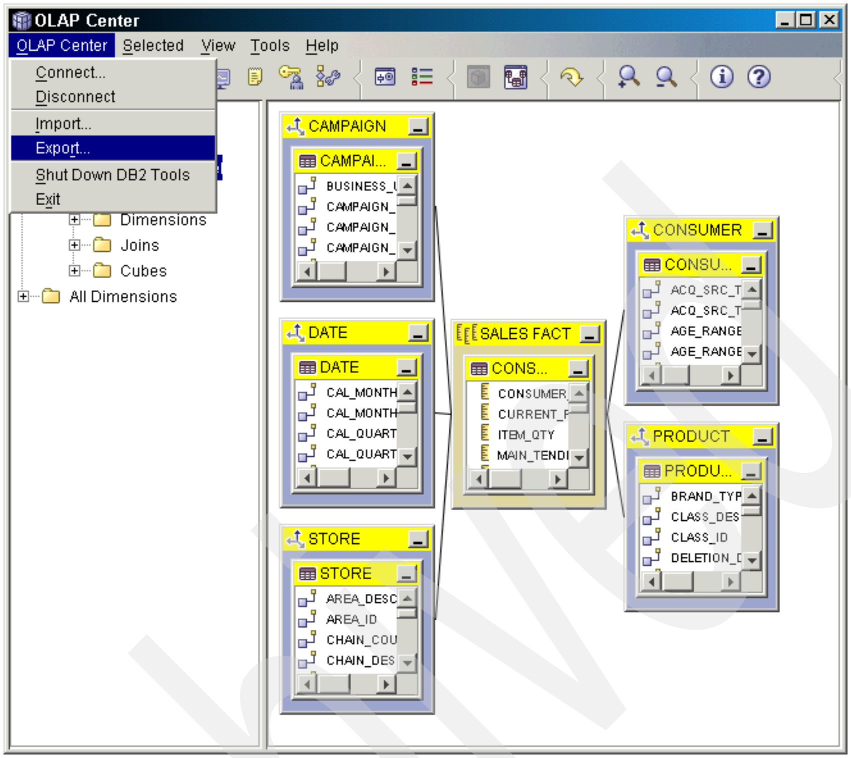


Figure 12-19 Export metadata from DB2 OLAP Center

2. To export metadata:
   a. From the OLAP Center main window, click **OLAP Center --> Export**. The Export window opens.
   b. Select either one cube model or one cube model and one cube to export. You cannot export a cube without its corresponding cube model.
   c. Specify an export XML file name or browse for an XML file to overwrite.
   d. Click **OK**. The Export window closes, and a DB2 Cube Views XML file is created containing information about the metadata objects that you specified.

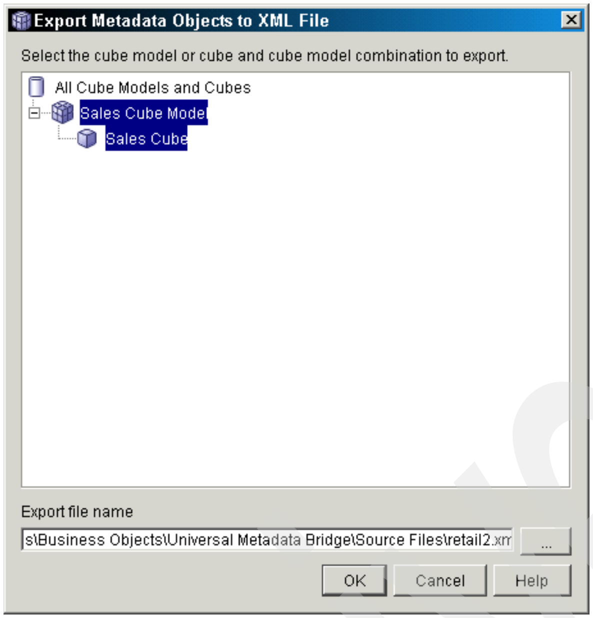


Figure 12-20 Export metadata to XML file

12.3.2 Import the metadata in the universe using Application Mode

Once the DB2 Cube Views XML file has been created, you can launch BusinessObjects Universal Metadata Bridge to import this XML file.

BusinessObjects Universal Metadata Bridge analyses the content in the XML file to extract metadata information. It then creates a BusinessObjects universe including classes, objects, tables, columns, custom hierarchies, aggregation functions and joins.

> **Attention:** The FULL OUTER join is not supported by Designer. A FULL OUTER join in DB2 Cube Views model is transformed into a LEFT OUTER join in the BusinessObjects universe.

To create the Universe:

1. Select **Start > Programs > BusinessObjects Universal Metadata Bridge > BusinessObjects Universal Metadata Bridge**.

   The BusinessObjects Universal Metadata Bridge panel opens as in Figure 12-21.

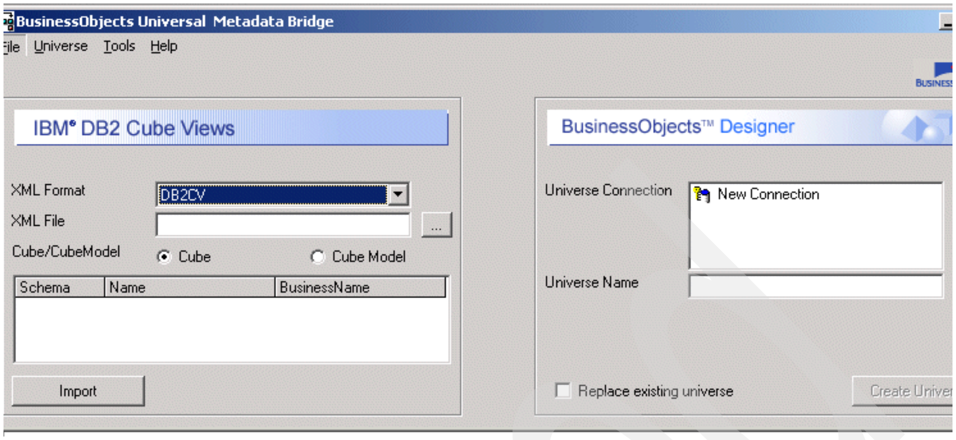


Figure 12-21 Universal Metadata Bridge panel

2. In the XML section of the screen, select the XML file location by either typing in the path to the file, or clicking the button next to the XML File text box as in Figure 12-22 and Figure 12-23.

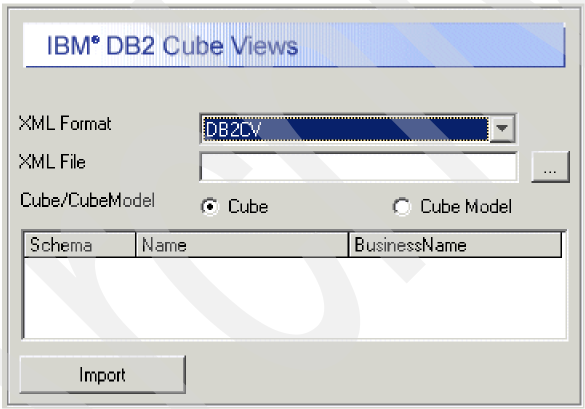


Figure 12-22 Choose the XML file to import

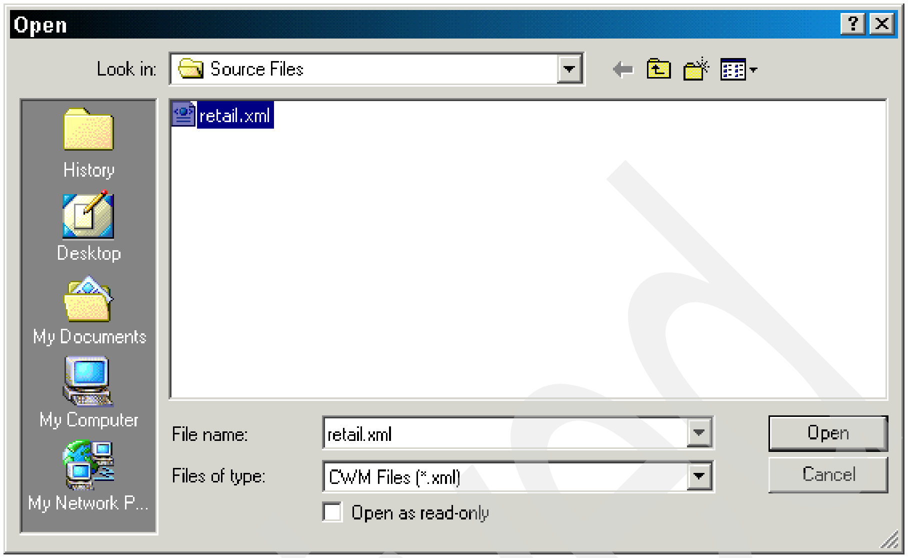


Figure 12-23 Browse the XML file

3. Cube is the default option button selection and the available cube schemas appear in the list box. If you would rather use a cube model, click **Cube Model**.
4. Select the schema you want to use to create a universe, and click **Import**. The schema appears in the panel as shown in Figure 12-24.

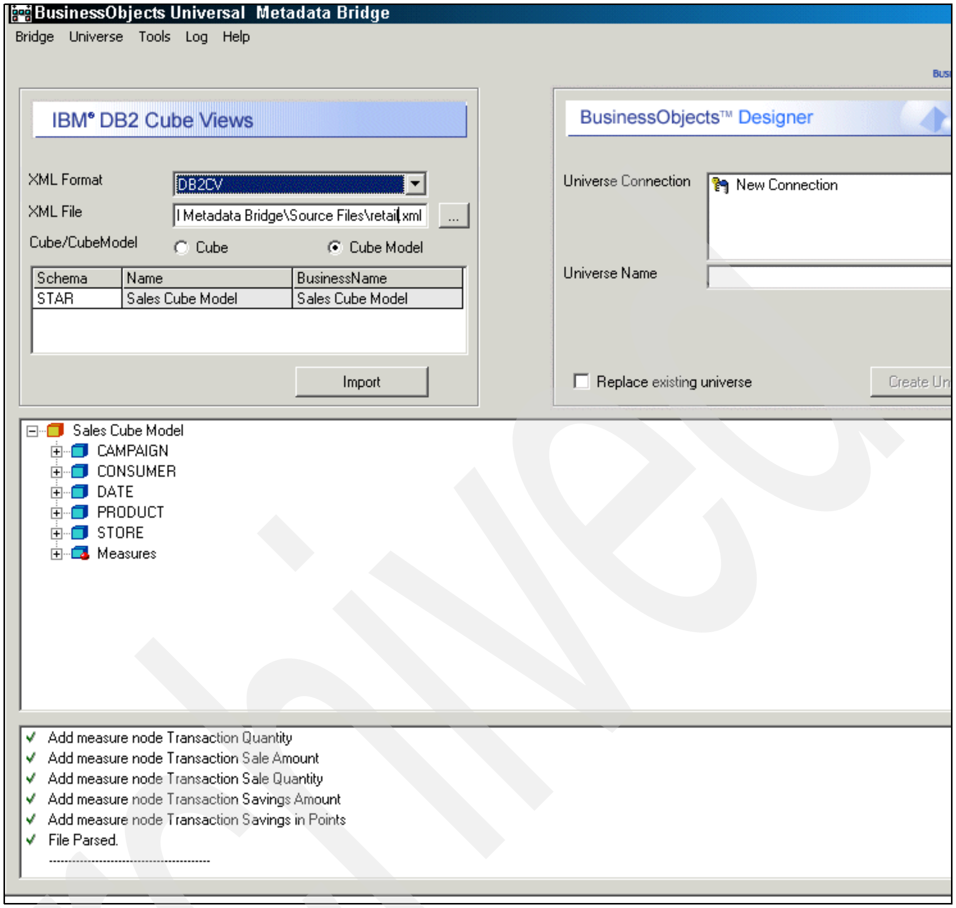


Figure 12-24 Specify the cube model and/or cube

5. Select the schema you want to use to create a universe, and click **Import**. The schema appears in the pane as shown below.

> **Note:** You expand the object tree in BusinessObjects Universal Metadata Bridge by clicking the + next to the object group name, as shown in Figure 12-25.

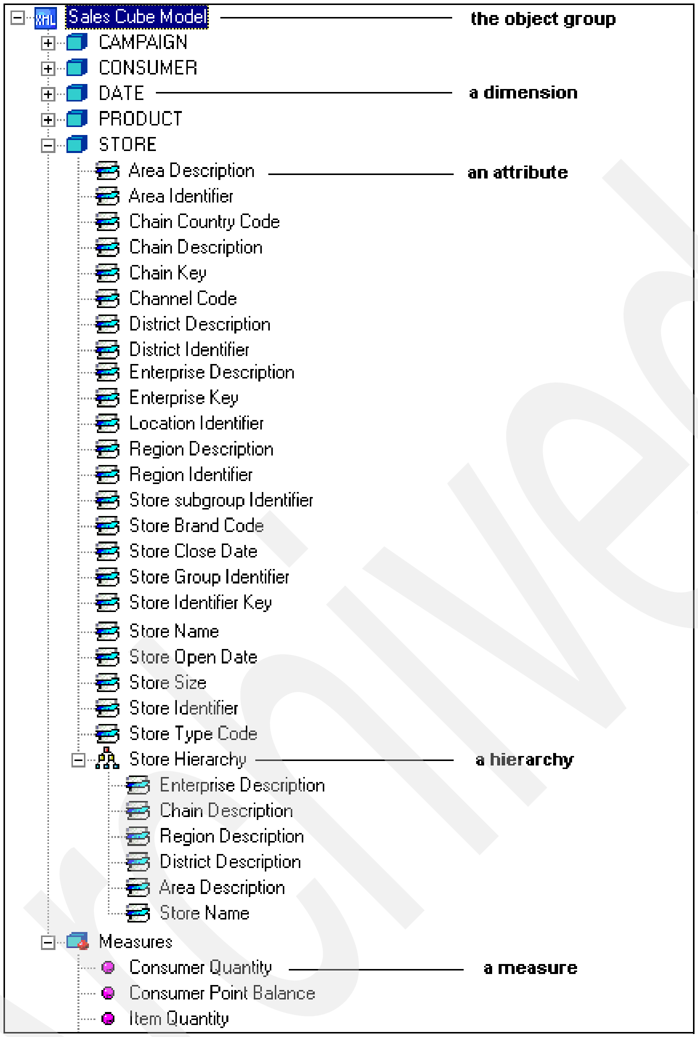


Figure 12-25 Object tree

Notice that the object group contains dimensions, attributes, hierarchies, and measures.

6. Enter the universe name.
7. Select a universe connection in the Universe Connection panel.
8. If you want to replace an existing universe, click the **Replace existing universe** check box.

9. Click **Create universe**. The new universe opens in Designer as shown in Figure 12-26.

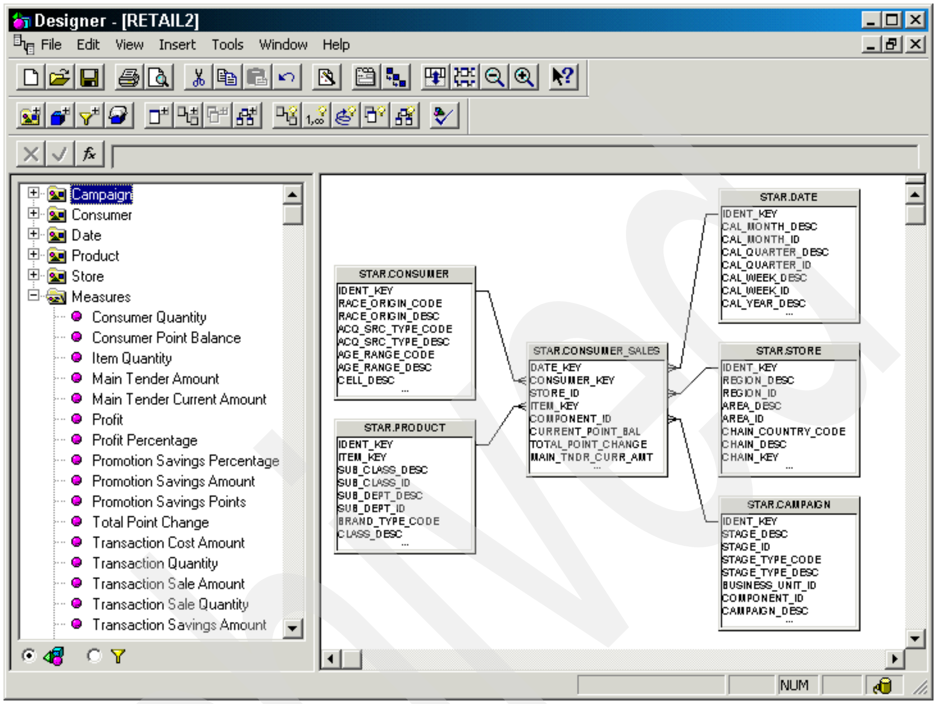


Figure 12-26 The universe created

12.3.3 Import the metadata in the universe using API mode

If you already have an application, and you want to add the bridge functionality, BusinessObjects Metadata Universal Bridge provides a Component Object Model (COM) API for object-based interoperability with various programming languages. The component is universalBridgeComApi.dll.

To create the Universe, using the function createUniverse, the following arguments are necessary:

- DB2 Cube Views XML file name
- XML format ("DB2CV")
- Keep or replace universe option (REPLACE or KEEP)
- Designer instance name
- Name of cube/cube model
- Schema name
- Cube/cube model option (Cube or CubeModel)

12.3.4 Import the metadata in the universe using the batch mode

The BusinessObjects Universal Metadata Bridge allows you to execute functions in Batch mode.

This mode is most useful when you need to:

- Create several universes
- Schedule the universe creation at a certain time

All parameters needed for batch mode during execution are entered as arguments of the executable when it is called.

To create a universe using batch mode, one or more XML files containing metadata must be available.

Batch mode can be called from a command line, script, or Scheduler. Batch mode produces a log file containing errors and warnings encountered during execution of the batch file. A batch file is composed of:

- Batch files sequences
- Batch file arguments

Batch file sequences

There is one creation sequence you can use in your batch file:

```
GenericBridge -f DB2CV -c <XML file> -o <Cube or CubeModel> -n <cubename>
-h <schema name> [-g] [-u <designer user>] [-w <designer password>] [-s
<security domain>] [-x <connection name>] [-k <universe option>]
```

An example of universe created without a repository will be:

```
GenericBridge -f DB2CV -c "%BOGENERICBRIDGE%\Source Files\tbc_exported.xml"
-o "CubeModel" -n "TBC Model" -h "TBC" -g -x "SAMPLE" –k "replace"
```

An example of universe created with a repository will be:

```
GenericBridge -f DB2CV -c "%BOGENERICBRIDGE%\Source Files\tbc_exported.xml"
-o "CubeModel" -n "TBC Model" -h "TBC" -u "super" -w "s" -s "BOMain" -x
"SAMPLE" -k "replace"
```

Batch file arguments

Figure 12-27 is an explanation of the arguments used in the batch file creation sequences.

	Argument	Description	Mandatory/ Optional
-f	<file format>	Aurora	Mandatory
-c	<XML file name>	The Aurora XML file name used to create the universe.	Mandatory
-o	<cube type>	CUBE or CUBEMODEL. The cube type defines if the universe is created from a cube or a cube model.	Mandatory
-n	<cube name>	The name of the cube or the cube model	Mandatory
-h	<schema name>	The schema name of the cube or cube model.	Mandatory
-u	<designer user>	The user name that allows access to Designer.	Mandatory
-w	<designer password>	The password name that allows access to Designer.	Mandatory
-s	<security domain>	The security domain name.	Mandatory
-x	<connection name>	The connection name to the Aurora database. The default is "DefaultConnect".	Optional
-k	<universe option>	KEEP or REPLACE universe if it already exists. The default is KEEP.	Optional
-g	<workgroup mode>	Indicates that the universe is in workgroup mode	Optional

Figure 12-27 Batch file arguments

12.3.5 Warning messages

During the universe creation process, BusinessObjects Universal Metadata Bridge can detect potential inconsistency within the input XML file, listed in Figure 12-28 and Figure 12-29.

Warning Message	Description
Warning: the XML file contains no cube for the cube model	This warning occurs if no cube is defined for the cube model in the XML file.
Warning: the XML file contains no dimension for the cube	This warning occurs if no dimension is defined for the cube in the XML file.
Warning: the XML file contains no attribute for the dimension	This warning occurs if no attribute is defined for the dimension in the XML file.
Warning: the XML file contains no hierarchy for the dimension	This warning occurs if no hierarchy is defined for the dimension in the XML file.
Warning: the XML file contains no measure for the fact	This warning occurs if no measure is defined for the fact table in the XML file.
Warning: the XML file contains no level for the custom hierarchy	This warning occurs if no level is defined for the hierarchy in the XML file.
Warning: the type of the join is set to LEFT OUTER	This warning occurs if the type of the join is FULL OUTER in the XML file. The join is created in BusinessObjects universe with LEFT OUTER type.
Warning: the file: <*universename*>.unv renamed to <*universename_date*>.unv	This warning occurs if the "replace existing universe" option is selected, and if the universe already exists.

Figure 12-28 Warning messages

Warning Message	Description
Warning: the XML file contains no cube or cube model	This warning occurs if no cube or cube model is defined in the XML file. The universe is not created.
Warning: the XML file contains no schema	This warning occurs if no schema is defined in the XML file. The universe is not created.
Warning: the XML file contains no table	This warning occurs if no table is defined in the XML file. The universe is not created.
Warning: the XML file contains no column for the table	This warning occurs if no column is defined for the table in the XML file.

Figure 12-29 Additional warning messages

12.4 Reports and queries examples

To demonstrate how the MQTs built using DB2 Cube Views Optimization Advisor improved BusinessObjects queries performance we used a simple business scenario:

- Query1: Who are the most profitable consumers?
- Query 2: What do they buy?

To check if the report query is optimized by DB2 Cube Views through the Optimization Advisor, we used the following method:

1. In BusinessObjects, launch SQL Viewer from the Query Panel (see Figure 12-30), and copy the SQL statement.

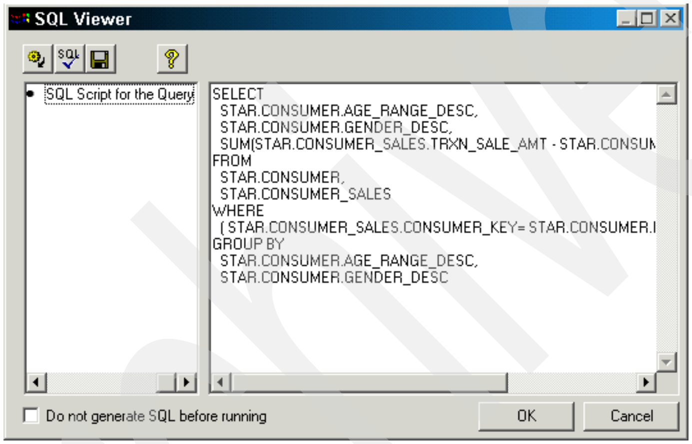


Figure 12-30 Get the SQL statement from SQL Viewer

2. In DB2 Control Center, launch Explain SQL Panel, and paste the SQL statement.

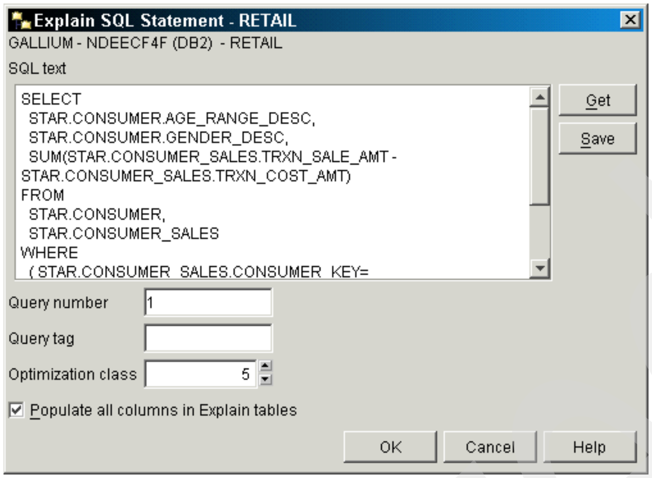


Figure 12-31 Launch the explain

3. In DB2 Control Center, analyze the access plan graph from DB2 Explain to check if MQTs is used.

> **Note:** Depending on the type of BusinessObjects reports that you want to create, you can run Optimization Advisor from DB2 OLAP Center to define MQTs (Materialized Query Tables) that can be created. This allows your report query to be optimized by MQTs with a shorter response time.

4. Check the response time under BusinessObjects Data Manager (see Figure 12-32).

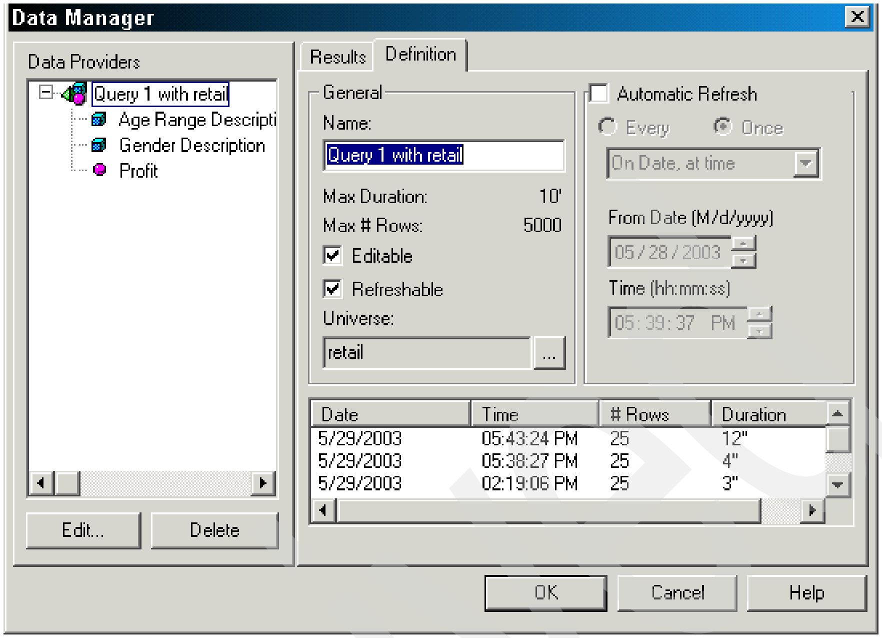


Figure 12-32 Check the response time

The following examples show reports created on top of the universe that has been previously built with the BusinessObjects Universal Metadata Bridge. It can be seen that query response times are improved by MQTs.

We will present for each example:

- The report
- The SQL generated by BusinessOjects
- The query performance
- The data access result using the access plan graph from DB2 Explain

12.4.1 Query 1

Query 1 addresses the business question:

> What are the top five most profitable consumer groups?

The report

The report is shown in Figure 12-33.

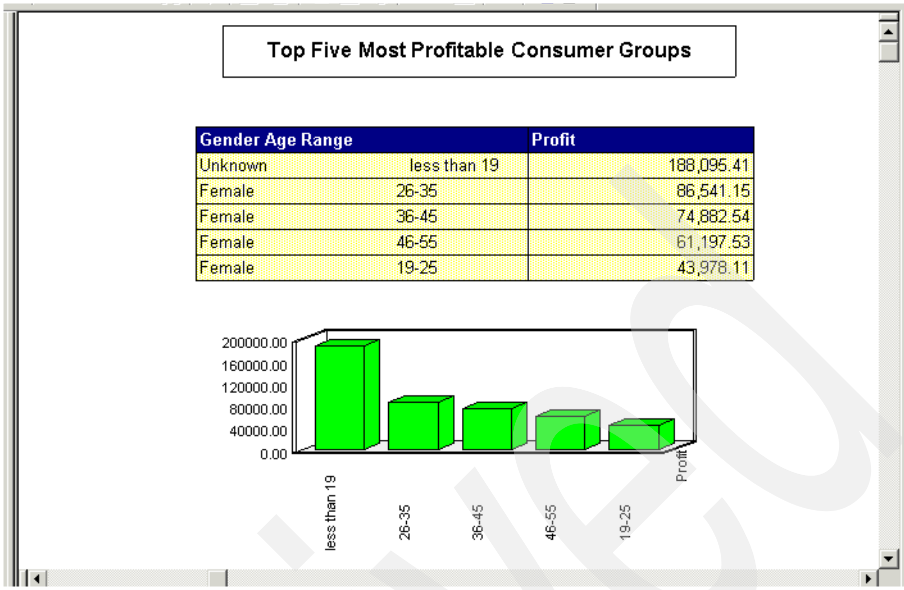


Figure 12-33   Report 1

The SQL

The SQL is shown in Example 12-1.

Example 12-1   SQL 1

```
SQL
SELECT
  STAR.CONSUMER.AGE_RANGE_DESC,
  STAR.CONSUMER.GENDER_DESC,
  SUM(STAR.CONSUMER_SALES.TRXN_SALE_AMT -
STAR.CONSUMER_SALES.TRXN_COST_AMT)
FROM
  STAR.CONSUMER,
  STAR.CONSUMER_SALES
WHERE
   ( STAR.CONSUMER_SALES.CONSUMER_KEY= STAR.CONSUMER.IDENT_KEY  )
GROUP BY
  STAR.CONSUMER.AGE_RANGE_DESC,
  STAR.CONSUMER.GENDER_DESC
```

The query performance

Table 12-1 shows the results.

Table 12-1 Query performance result

	Time to refresh report 1: timeron
Without MQT	148,969
With MQT	3,986

The data access

Without MQTs, the data access used is described in Figure 12-34.

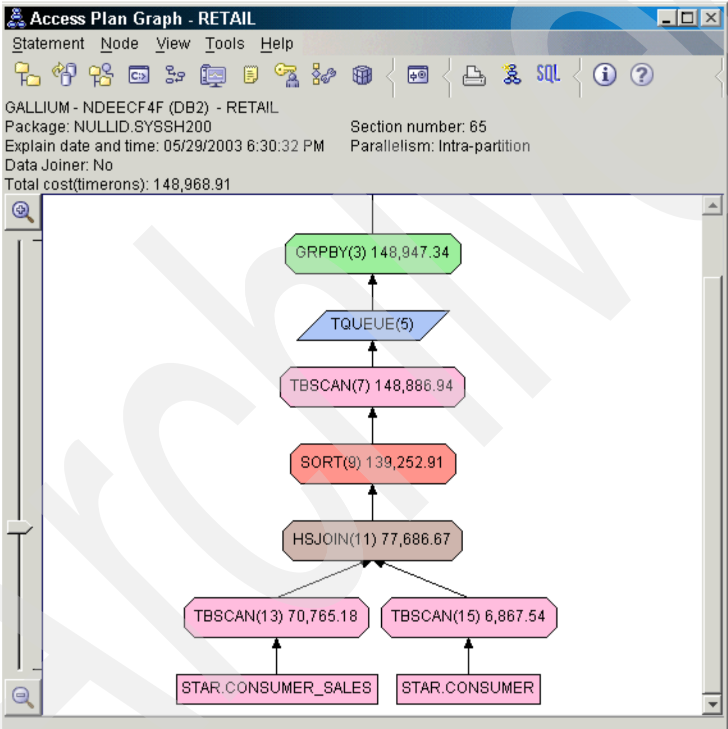


Figure 12-34 Without MQTS: DB2 explain result

The Access Plan Graph of the query shows that tablespace scans have been used because no MQTs can be used for query rewrites.

The measured response time for the refresh of the report is also long: 12 seconds.

With MQTs built by the Optimization Advisor, the data access used is described in Figure 12-35.

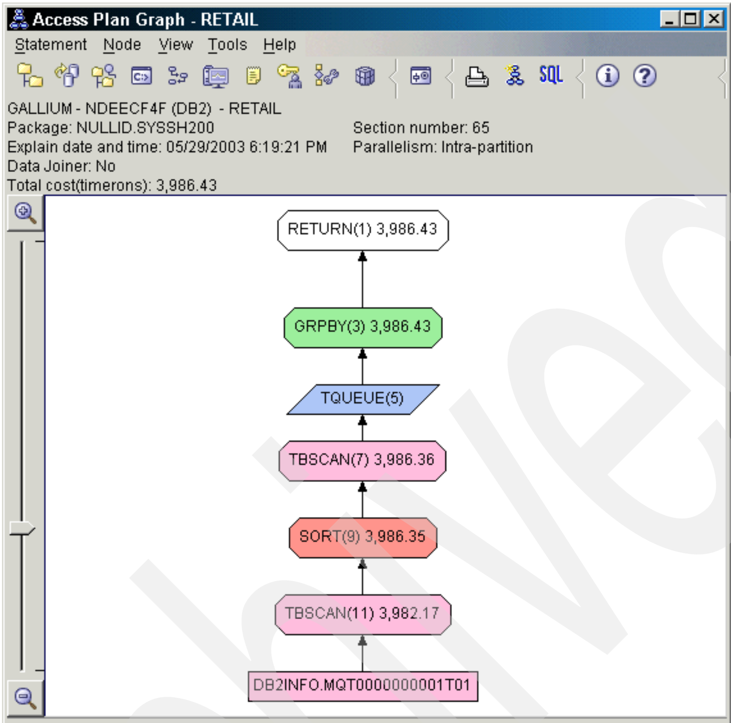


Figure 12-35 With MQTS: DB2 explain result

The Access Plan Graph of the query is simple; The MQT MQT0000000001T01 has been used to retrieve all the information.

The response time of “Refresh report” action is short in this case, thanks to the MQT. In this case less than 4 seconds were needed for refreshing this report, of which most can be attributed to system and network latency.

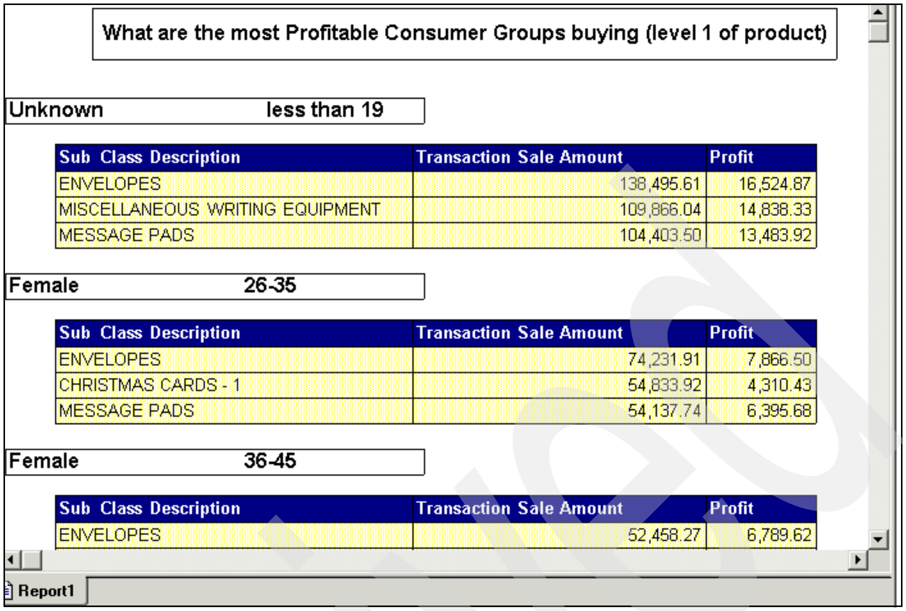
12.4.2 Query 2

Query 2 addresses the business question:

What are the most Profitable Consumer Groups buying (Level 1 of product)?

The report

The report is shown in Figure 12-36.



What are the most Profitable Consumer Groups buying (level 1 of product)

Unknown less than 19

Sub Class Description	Transaction Sale Amount	Profit
ENVELOPES	138,495.61	16,524.87
MISCELLANEOUS WRITING EQUIPMENT	109,866.04	14,838.33
MESSAGE PADS	104,403.50	13,483.92

Female 26-35

Sub Class Description	Transaction Sale Amount	Profit
ENVELOPES	74,231.91	7,866.50
CHRISTMAS CARDS - 1	54,833.92	4,310.43
MESSAGE PADS	54,137.74	6,395.68

Female 36-45

Sub Class Description	Transaction Sale Amount	Profit
ENVELOPES	52,458.27	6,789.62

Report1

Figure 12-36 Report 2

The SQL

The SQL is shown in Example 12-2.

Example 12-2 SQL 2

```
SQL
SELECT
  STAR.CONSUMER.AGE_RANGE_DESC,
  STAR.CONSUMER.GENDER_DESC,
  STAR.PRODUCT.SUB_CLASS_DESC,
  SUM(STAR.CONSUMER_SALES.TRXN_SALE_AMT - STAR.CONSUMER_SALES.TRXN_COST_AMT),
  SUM(STAR.CONSUMER_SALES.TRXN_SALE_AMT)
FROM
  STAR.CONSUMER,
  STAR.PRODUCT,
  STAR.CONSUMER_SALES
WHERE
  ( STAR.CONSUMER_SALES.CONSUMER_KEY= STAR.CONSUMER.IDENT_KEY  )
  AND  ( STAR.CONSUMER_SALES.ITEM_KEY= STAR.PRODUCT.IDENT_KEY  )
GROUP BY
  STAR.CONSUMER.AGE_RANGE_DESC,
  STAR.CONSUMER.GENDER_DESC,
  STAR.PRODUCT.SUB_CLASS_DESC
```

The query performance

Here we did not benefit from a query rewrite since the query went too low in the product dimension hierarchy, forcing DB2 to go to the base tables for the data.

This is acceptable for queries that only are run occasionally. Moreover since the cost of creating MQTs for low levels in the hierarchies are high in terms of space while yielding only few performance benefits we generally would avoid these types of MQTs.

12.4.3 Query 3

Query 3 addresses the business question:

> What are the top three most profitable departments per year by region?

The report

The report is shown in Figure 12-37.

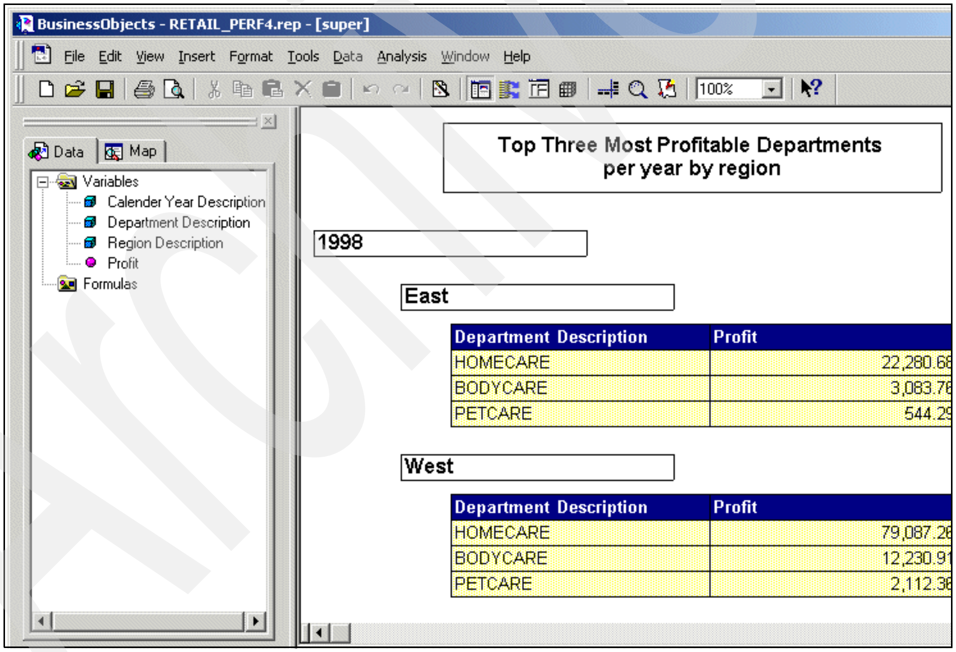


Figure 12-37 Report 3

The SQL

The SQL generated is shown in Example 12-3.

Example 12-3 SQL 3

```
SELECT
  SUM(STAR.CONSUMER_SALES.TRXN_SALE_AMT - STAR.CONSUMER_SALES.TRXN_COST_AMT),
  STAR.DATE.CAL_YEAR_DESC,
  STAR.STORE.REGION_DESC,
  STAR.PRODUCT.DEPARTMENT_DESC
FROM
  STAR.CONSUMER_SALES,
  STAR.DATE,
  STAR.STORE,
  STAR.PRODUCT
WHERE
  ( STAR.CONSUMER_SALES.DATE_KEY= STAR.DATE.IDENT_KEY  )
  AND  ( STAR.CONSUMER_SALES.ITEM_KEY= STAR.PRODUCT.IDENT_KEY  )
  AND  ( STAR.CONSUMER_SALES.STORE_ID= STAR.STORE.IDENT_KEY  )
GROUP BY
STAR.DATE.CAL_YEAR_DESC,
STAR.STORE.REGION_DESC,
STAR.PRODUCT.DEPARTMENT_DESC
```

The query performance

Table 12-2 shows the results.

Table 12-2 Query performance result

	Time to refresh report 3: timeron
Without MQT	762,632.19
With MQT	13,822.72

The data access

Without MQTs, the data access used is described in Figure 12-34.

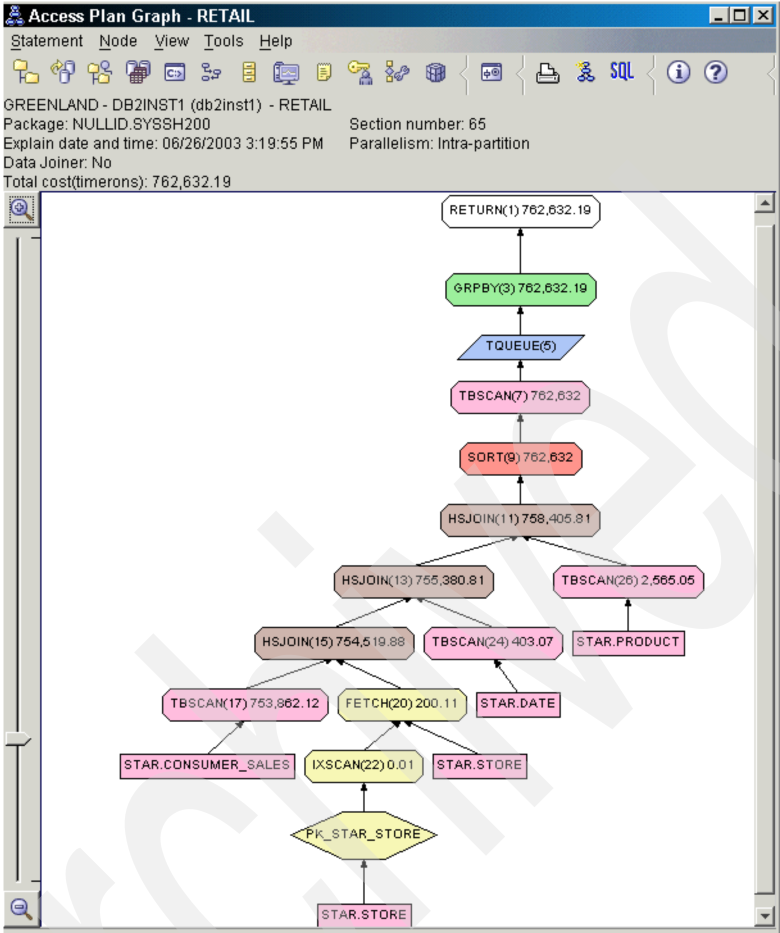


Figure 12-38 Without MQTS: DB2 explain result

The Access Plan Graph of the query shows that tablespace scans have been used because no MQTs can be used for query rewrites.

With MQTs built by the Optimization Advisor, the data access used is described in Figure 12-39.

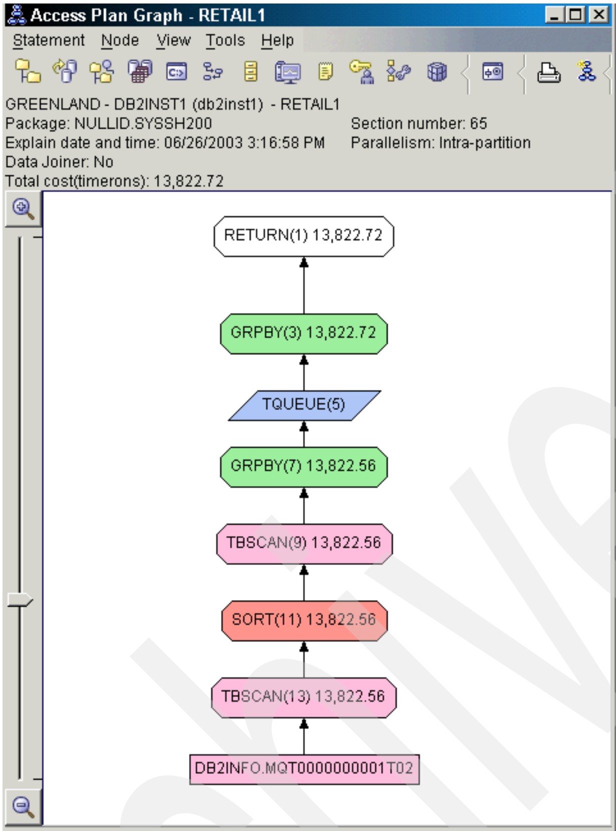


Figure 12-39 With MQTS: DB2 explain result

The Access Plan Graph of the query is simple; the MQT MQT0000000001T02 has been used to retrieve all the information. The response time of **Refresh report** action is shorter in this case, thanks to the MQT.

12.5 Deployment

The design of DB2 Cubes and MQTs is an iterative process. The review of the BusinessObjects Universe allows you to improve your Cube/Cube Model and the BusinessObjects query response time gives you indications on how you should use DB2 Optimization Advisor to create the required MQTs.

These are the five steps as shown in Figure 12-40 when deploying DB2 Cube Views with BusinessObjects:

1. Run BusinessObjects Universal Metadata Bridge to create Universes from DB2 Cubes/Cube Models.
2. Redesign DB2 Cube/Cube Model after universe review.
3. Run DB2 Optimization Advisor to take care of BusinessObjects report types.
4. Create new MQTs.
5. BusinessObjects reports leverage the query optimization by MQTs.

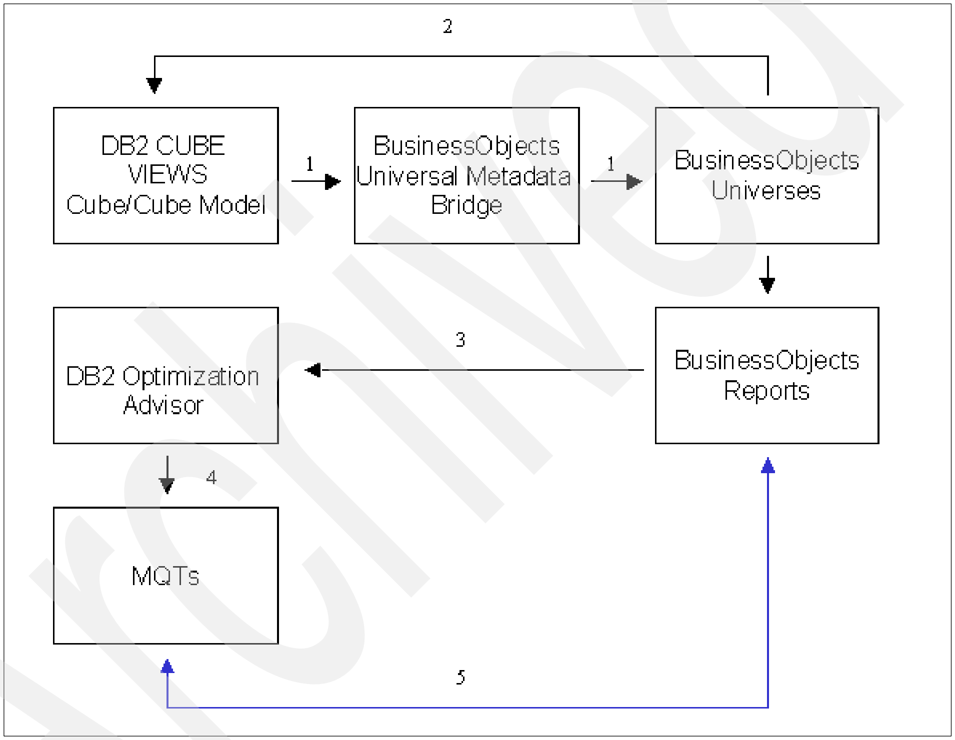


Figure 12-40 Iterative process

12.5.1 Optimization tips

To get full advantage of the DB2 Cube Views optimization, good practices could be as follows:

- **Step 1:** Identify which universes and which reports need to be optimized.

  BusinessObjects Auditor allows you to monitor and optimize your Business Intelligence deployment. With Auditor, you can answer questions such as:

- What universes are the most popular
- What objects are most commonly used
- What reports have the highest refresh frequency

Identify common universes, objects and reports to determine where you should start optimizing.

- **Step 2:** Optimize universes.

  Ad hoc users are often unpredictable and likely to build large queries so they are the ones who should benefit the most from performance optimization.

  Design ad hoc universes that are ideal for DB2 Cube Views. Star schemas are a good place to start, so think fact tables and dimension hierarchies.

  Canned reports that are based on optimized universes will also benefit and will refresh more quickly.

12.6 Conclusion: benefits

The two main benefits from using DB2 Cube Views with BusinessObjects are:

- Less administrative effort to create the universe
- Speeds up the query performance with DB2 Optimization Advisor

Business Objects is working on enhancing the bridge and is planning to add a reverse bridge capability that will allow customers to leverage some of their existing universes automatically to DB2 Cube Views.

12.6.1 Universe creation

With DB2 Cube Views and the BusinessObjects Universal Metadata Bridge, you can now leverage existing multidimensional data to automatically create universes. This means that you can create multidimensional metadata in a single place and use it in multiple tools. Less administration will result in lower Total Cost of Ownership (TCO).

12.6.2 Improving response time with MQTs

Using MQTs built through DB2 Cube Views Optimization Advisor, BusinessObjects users will experience a dramatic increase in query performance. In some cases, measures show that query response time can be divided by 10. Business Objects is committed to enhancing user experience by improving response times, and Business Objects tools combined with DB2 Cube Views can clearly bring this kind of performance benefits to users.

13

Accessing DB2 dimensional data using MicroStrategy

This chapter describes some scenarios for deployment and discusses benefits before showing how to implement and use the metadata bridge. The objective is to talk in a little more depth about the way the bridge carries out the mapping so the reader can understand where and why they can benefit from the bridge and know where the limits are.

Familiarity with DB2 Cube Views and the MicroStrategy product suite is assumed in the following sections of this chapter.

13.1 MicroStrategy product introduction

At the heart of a MicroStrategy BI system is MicroStrategy Intelligence Server, an analytical server that works in tandem with the back end DB2 database to fulfill end users' requests originating from the many user interfaces that MicroStrategy offers. MicroStrategy Intelligence Server leverages the full processing power of DB2 by issuing highly optimized SQL, keeping the bulk of the processing within the database and thus eliminating the need to replicate large amounts of data. When appropriate, MicroStrategy Intelligence Server will supplement the database capabilities and assume part of the processing. This tight integration results in a Business Intelligence system with unparalleled analytical sophistication, robustness, and performance.

MicroStrategy Intelligence Server also provides the required functionality for a true enterprise class system, including clustering and failover, centralized administration, pervasive security, mid-tier caching, and load governing. This set of functionality ensures the scalability and fault tolerance required for sophisticated analysis of terabyte databases and deployments to millions of users. Built on this strong foundation, several user interfaces are available, each aimed at a distinct user population or set of activities. Figure 13-1 summarizes the different user interfaces and their primary purpose.

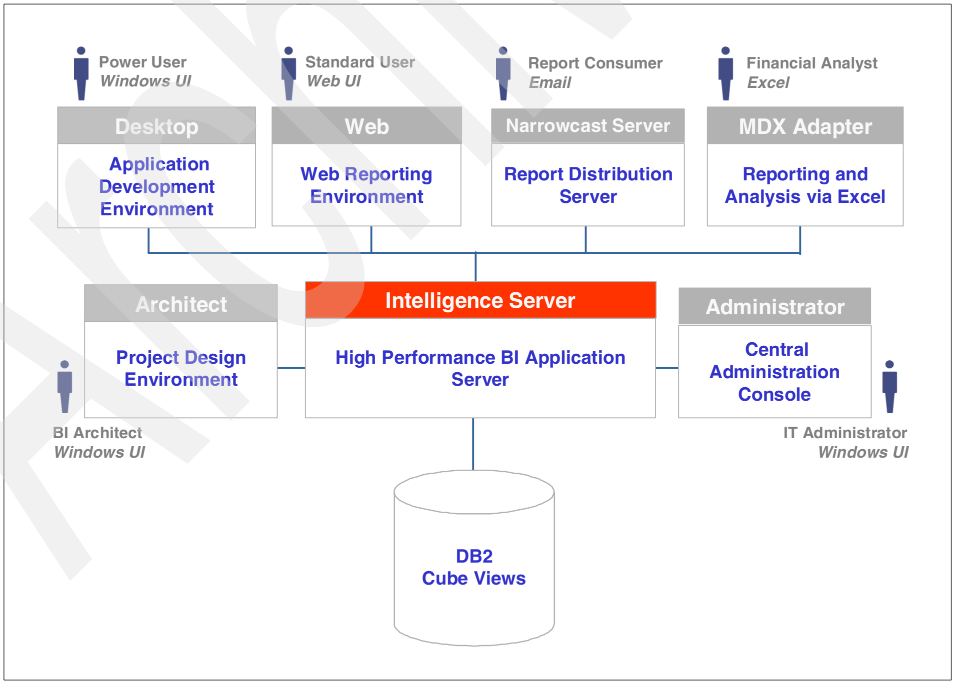


Figure 13-1 MicroStrategy product suite overview

Reporting is typically done using MicroStrategy Web, MicroStrategy Desktop, and MicroStrategy MDX Adapter, while report delivery is carried out with MicroStrategy Narrowcast Server. Administration is performed via MicroStrategy Administrator, and application development is completed with MicroStrategy Desktop and MicroStrategy Architect.

In order to facilitate deployment, MicroStrategy provides a metadata bridge between the DB2 Cube Views metadata and the MicroStrategy metadata. This metadata bridge takes the form of a tool, the MicroStrategy IBM DB2 Cube Views Import Assistant (henceforth referred to as Import Assistant). The Import Assistant allows a system architect to leverage a model developed in DB2 Cube Views to populate the ad hoc query model used by MicroStrategy, thus significantly reducing development times.

After the DB2 Cube Views metadata is defined, the Import Assistant can be used to convert this multidimensional information into its MicroStrategy equivalent that will serve as the basis for additional development or immediate reporting activities. The Import Assistant analyzes the DB2 Cube Views metadata — translating each component, including Attributes, Hierarchies, and Measures — and produces a MicroStrategy project ready for use. Within the MicroStrategy environment, one can then run queries and create reports right away or enhance the project further to take advantage of modeling facilities specific to MicroStrategy.

13.2 Architecture and components involved

This section describes at a high level the architecture of the MicroStrategy Import Assistant and its different components.

The MicroStrategy Import Assistant is made of several logical components, each responsible for a separate step in the import process:

- **Warehouse Catalog:** The Import Assistant connects to the DB2 Cube Views database through ODBC and reads the DB2 system catalog to obtain the relational metadata. It collects all relevant tables, columns, and data-types information.
- **Cube Reader:** Based on the published DB2 Cube Views object model and its XML schema definition, the Import Assistant reads the DB2 Cube Views multidimensional metadata, not directly from the database, but from the DB2 Cube Views XML representation, which should be generated beforehand.
- **Cube Translator:** The Import Assistant interprets the DB2 Cube Views multidimensional metadata and maps it to its equivalent MicroStrategy representation. Refer to 13.4, “Mapping considerations and metadata refresh” on page 592 for further detail.

- **Project Creator:** The Import Assistant connects to a MicroStrategy project source and creates all the MicroStrategy objects resulting from the mapping directly into the selected MicroStrategy object repository (also known as the MicroStrategy metadata) hosted in a relational database.

The flow of information during the import process is represented in Figure 13-2.

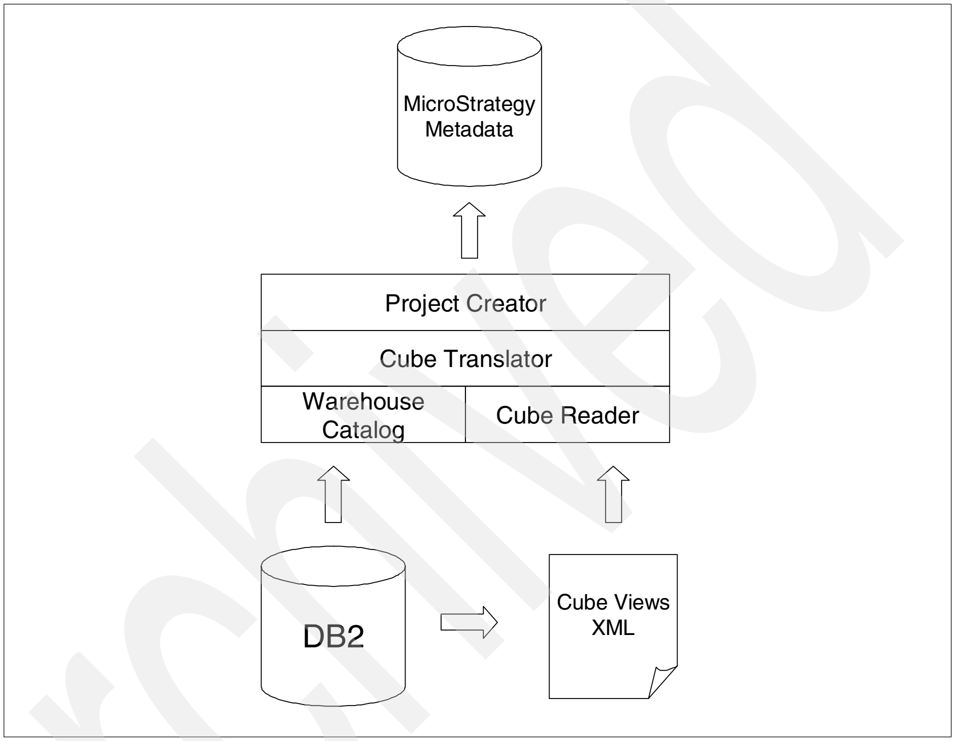


Figure 13-2 Import process information flow

13.3 Implementation steps

This section details the various steps necessary to use the Import Assistant, which can be downloaded, if authorized, from:

`https://solutions.microstrategy.com/Support_Search/open_file.asp?Doc_ID=tn1200-000-0038.asp`

13.3.1 Installation

The Import Assistant is not part of the main MicroStrategy product suite installation at this time. It is available as an independent tool to customers, partners and prospects via the MicroStrategy Knowledge Base at:

MicroStrategy DB2 Cube Views Import Assistant Installation

The ZIP file contains a stand-alone installation for the Import Assistant and its on-line help. The Import Assistant must be installed on a machine with MicroStrategy Architect V7.2.3.

13.3.2 Prerequisites

Prior to using the Import Assistant, the following prerequisites should be completed:

- The MicroStrategy product suite is installed on a machine and an ODBC DSN to the database is set up.
- A database is created for the purpose of hosting the MicroStrategy metadata and an ODBC DSN is established for it.
- The MicroStrategy metadata is configured using the MicroStrategy Configuration Wizard.
- The DB2 Cube Views metadata is defined and its XML representation is generated.

13.3.3 Import

To begin using the Import Assistant: Double-click **MstrDb2Import.exe**. The Import Assistant dialog is shown in Figure 13-3. You must enter the following input parameters:

- The location of the schema definition file that is the DB2 Cube Views XML file.
- The project source in which to create the project based on the imported metadata.
- The database instance that points to the DB2 Cube Views database.
- The location of the log file that the import process generates.

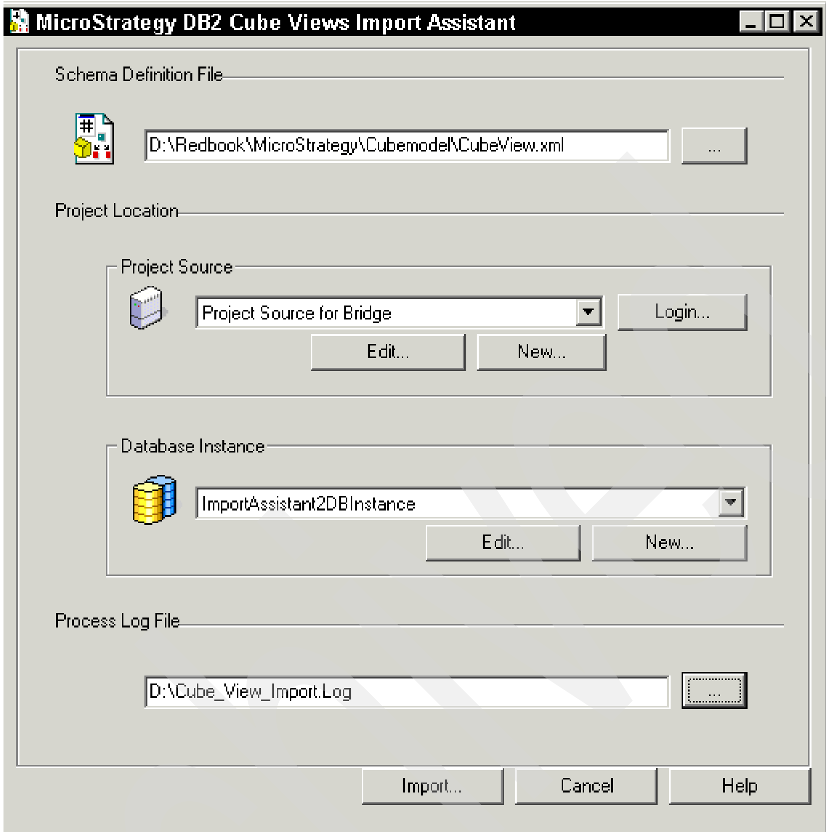


Figure 13-3 Import Assistant Dialog

The following section details the various input parameters.

Schema definition file

The Import Assistant reads the DB2 Cube Views metadata from its XML representation. You need to direct the Import Assistant to the proper cube XML file generated beforehand.

> **Note:** The DB2 Cube Views XML file must contain a single cube (not a cube model); otherwise the Import Assistant will not function properly.

To define the location of the schema definition file:

- Click the **...** button.

- Select the cube XML file that corresponds to the metadata in DB2 Cube Views. Alternately, if you know the path and file name of the cube XML file, you may enter it in the box.

Project source

The project source specifies the MicroStrategy metadata where to import the DB2 Cube Views metadata. You may determine the project source in one of the following ways:

- Select the appropriate project source from the drop-down menu.
- Click **New** to create a new project source. This opens the Project Source Manager. You need to choose a name for the new project source and enter the ODBC DSN for the MicroStrategy metadata and its corresponding database login and password.

When you have selected your project source, click Login. Enter the username and password, and click OK. You must have administrator privileges to log in

Database instance

The database instance specifies connection information to the DB2 Cube Views database. You may do this in one of the following ways:

- Select the appropriate database instance from the drop-down menu.
- Click **New** to create a new database instance. This opens the Database Instances dialog box. You need to choose a name for the new database instance and enter the ODBC DSN for the DB2 Cube Views database and its corresponding database login and password.

Process log file

The Import Assistant displays status information about the different steps of the import process in its feedback window and also logs this information into the process log file. To create a log file for the Import Assistant:

- Click the **...** button.
- Select the log file that you want to use. Alternately, if you know the path and file name of an existing log file, enter it in the box.

Import

When you have finished determining the schema definition file, project location and process log file, click **Import**. The metadata from IBM DB2 Cube Views begins to transfer to MicroStrategy 7i. The Import Assistant displays status information about the different steps of the import process in its feedback window. The feedback window is shown in Figure 13-4.

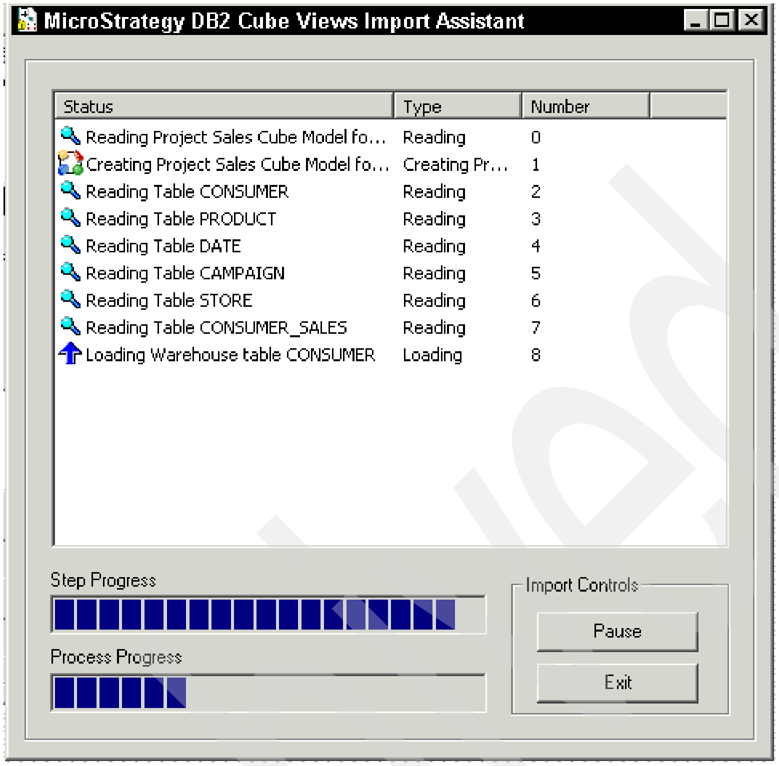


Figure 13-4 Import Assistant Feedback window

When the transfer is complete, open MicroStrategy Desktop and log in to the project source you selected to view your imported project.

13.4 Mapping considerations and metadata refresh

This section attempts to shed more light on the mapping performed by the Import Assistant.

13.4.1 Mapping fundamentals

The Import Assistant creates a MicroStrategy project by reading the DB2 Cube Views XML. The following objects from the DB2 Cube Views XML are used to derive the resulting output:

- Attributes
- Joins
- Attribute relationships

- Dimensions
- Hierarchies
- Measures

At the present time, cube models, cubes, facts, cube facts, cube dimensions and cube hierarchies are not used to extract information since they are either container objects or subset objects.

The Import Assistant creates a MicroStrategy project containing the following objects specific to the imported DB2 Cube Views metadata:

- Logical tables
- Attributes
- Hierarchies
- System Dimension
- Facts
- Base Formulae
- Metrics

The user can thus start creating new reports based on this infrastructure.

The DB2 Cube Views object model and the MicroStrategy object model do not coincide exactly. Table 13-1 summarizes the mapping between the two object models.

Table 13-1 Mapping summary

MicroStrategy objects	DB2 Cube Views objects
Attributes	Attributes Descriptive Relationships Joins
Hierarchies	Dimensions Hierarchies
System Dimension	Joins Associated Relationships
Facts	Measures
Base Formulae	Measures
Metrics	Measures

As is apparent in Table 13-1, some MicroStrategy objects span several DB2 Cube Views objects and vice-versa. This situation requires the Import Assistant to make some assumptions as well as some approximations.

13.4.2 Assumptions and best practices

The following discussion aims at providing some background information on assumptions made by the Import Assistant as well as significant known limitations. An attempt is also made to recommend best practices whenever possible. Please refer to the release notes distributed with the Import Assistant for additional up-to-date information.

Attributes

The Import Assistant supports all attribute definitions specified in the DB2 Cube Views XML including attributes that use IBM DB2 functions and attributes that are based on other attributes. One exception to note is the case of attributes defined across multiple tables.

Joins

MicroStrategy does not explicitly have the concept of a join. Join information is used in part to infer MicroStrategy attribute definitions and the MicroStrategy system dimension.

While there are no restrictions in defining the schema, it is recommended that designers use a snowflake schema. A snowflake schema has a lookup table for every attribute within a dimension. These lookup tables may be normalized or denormalized. The lowest level attributes within the various dimensions in the schema are joined by specifying appropriate columns in the lookup table and the fact table. In such cases, a combination of join information and descriptive relationships should suffice to completely describe the schema. Wherever associated relationships are specified, it is recommended that within a particular dimension every DB2 Cube Views attribute should have at least one link (join, associated or descriptive relationship) specified to another DB2 Cube Views attribute. This reduces ambiguity while creating MicroStrategy schema objects and leads to a more precise schema definition.

The Import Assistant does not support joins other than equi-joins nor joins on more than one column. These joins are simply ignored during the import process.

Currently, the Import Assistant does not handle joins between columns with different names. It is recommended to use MicroStrategy Architect after the import to properly map the columns. An alternative is to rename the columns directly in the database to render the naming convention consistent.

Attribute relationships

Two kinds of attribute relationships are defined in DB2 Cube Views cube model:

- Descriptive relationships
- Associated relationships

Descriptive relationships

The Import Assistant uses descriptive relationships to merge the relevant DB2 Cube Views attributes into a single MicroStrategy attribute with multiple attribute forms. The DB2 Cube Views designer should define as many descriptive relationships as possible to ensure the most accurate representation of the model while still avoiding redundant relationships.

Associated relationships

Associated relationships are used by the Import Assistant to further refine the model and link logically-connected parts of the system dimension. Whenever few joins exist within a dimension, it is strongly recommended to define as many associated relationships as the model logically requires. An alternative is to link the various resulting independent attributes with MicroStrategy Architect after the import.

Dimension

The Import Assistant supports all dimension definitions.

Hierarchies

The Import Assistant creates a hierarchy for every DB2 Cube Views hierarchy. It is recommended that hierarchies be defined using attributes that are ID forms for MicroStrategy attributes.

The Import Assistant does not support recursive hierarchies.

Measures

The Import Assistant converts each measure into a set of objects: facts, base formulae and metrics.

Symmetric measures that use standard aggregation functions are handled properly. However, in the event that the expression has multiple references, the user should edit the expression in MicroStrategy Architect. For instance, the measure Profit is defined as in Example 13-1.

Example 13-1 Measure Profit edited

```
<measure name="Profit">
  <sqlExpression template="{$$1} - {$$2}">
<column name="TRXN_SALE_AMT"/>
    <column name="TRXN_COST_AMT"/>
  </sqlExpression>
  <aggregation function="SUM" />
</measure>
```

The expression for the imported fact Profit is defined as:

```
ApplySimple("#0 -{$$2}", TRXN_SALE_AMT)
```

however it should be redefined as:

```
TRXN_SALE_AMT - TRXN_COST_AMT
```

Asymmetric measures are currently not handled by the Import Assistant. They should ideally result in nested aggregation metrics in MicroStrategy. For example, the measure *PROMO_SAVINGS_PTS* defined below should be aggregated with MAX along the Campaign dimension, AVG along the Time dimension and SUM along other dimensions as specified in Example 13-2.

Example 13-2 Asymmetric measure example

```
<measure name="PROMO_SAVINGS_PTS">
  <sqlExpression template="{$$1}">
    <column name="PROMO_SAVINGS_PTS"/>
  </sqlExpression>
  <aggregation function="MAX">
    <dimensionRef name="CAMPAIGN"/>
  </aggregation>
  <aggregation function="AVG">
    <dimensionRef name="DATE"/>
  </aggregation>
  <aggregation function="SUM"/>
</measure>
```

The correct definition to use in MicroStrategy is:

```
SUM(AVG(MAX(PROMO_SAVINGS_PTS){~, Store, Consumer, Item, Date}){~, Store,
Consumer, Item}){~}
```

For unsupported aggregation function cases, the Import Assistant creates a metric with a **SUM** aggregation function. Designers must alter the definitions of these metrics in MicroStrategy Desktop to suit their needs.

The Import Assistant does not handle correctly measures defined from other measures. The corresponding metrics need to be created manually in MicroStrategy Desktop. For example, the measure Profit% should be created as:

```
Profit / TRXN_SALE_AMT where Profit and TRXN_SALE_AMT are other metrics.
```

The Import Assistant does not support measures defined across multiple tables.

13.4.3 Metadata refresh

The Import Assistant does not provide a synchronization mechanism with the original source. This implies that changes performed in DB2 Cube Views or in MicroStrategy after the import was completed have to be managed manually.

13.5 Reports and query examples

The following section demonstrates the performance results from running MicroStrategy reports against a DB2 database with Cube Views enabled making use of MQTs versus a database without Cube Views and with no MQTs available.

> **Note:** The performance results are expressed in DB2 access path timerons cost measure.

> **Note:** In order to ensure that MicroStrategy will make use of DB2 MQTs, a good practice is to include ID columns in the DB2 Cube Views cube model design since the SQL built through MicroStrategy include ID columns.

The following business case is sued to demonstrate all performance results.

13.5.1 The business case and the business questions

Let us assume you are the Manager of the East Region. You would like to assess how each of the departments contributed to the rest of the region in terms of Sales. In addition, your company has recently launched a new series of campaigns targeted at certain age range groups with high growth opportunity. In your region, this consists of the age groups of 26-35 and 46-55. In order to answer your business questions you will make use of MicroStrategy technology to obtain results.

As a regional manager you would like to answer the following business questions:

- **Question 1:** How did each of the departments in your region contribute to the rest of the region in terms of sales?
- **Question 2:** How did each of the campaigns in your region contribute to the rest of the region in terms of sales?
- **Question 3:** How are each of the campaigns in your region ranked with the rest of the regions and within your own region?
- **Question 4:** What are the Top 5 campaigns ranked over all regions?
- **Question 5:** How has each of the campaigns in your region impacted sales for the age groups of 26-35 and 46-55?

> **Note:** For the following examples, the Attribute names in MicroStrategy have been modified to enhance the readability of the reports. For example, REGION_ID has been renamed Region.

13.5.2 Question 1: department contributions to sales

The business issue to solve is to assess how each of the departments in your region contributed to the rest of the region in terms of sales.

The business question mentioned above has been resolved making use of MicroStrategy *Metric Level* functionality, which allows the user to create metrics with a specific dimensionality. In this case, the user has created a Transaction Sales Amount metric (Trnx Sale Amt) at the report level and a second Transaction Sales Amount metric at the Region level (see Figure 13-5 for the result).

Note 1: For more information on how to create MicroStrategy reports, please refer to the MicroStrategy product manuals.

Note 2: The Regional Transaction Sales Amount Contribution metric has been created using the *Derived Metrics* functionality in MicroStrategy. Derived metrics allow users to create metrics *on the fly* after report results have been returned.

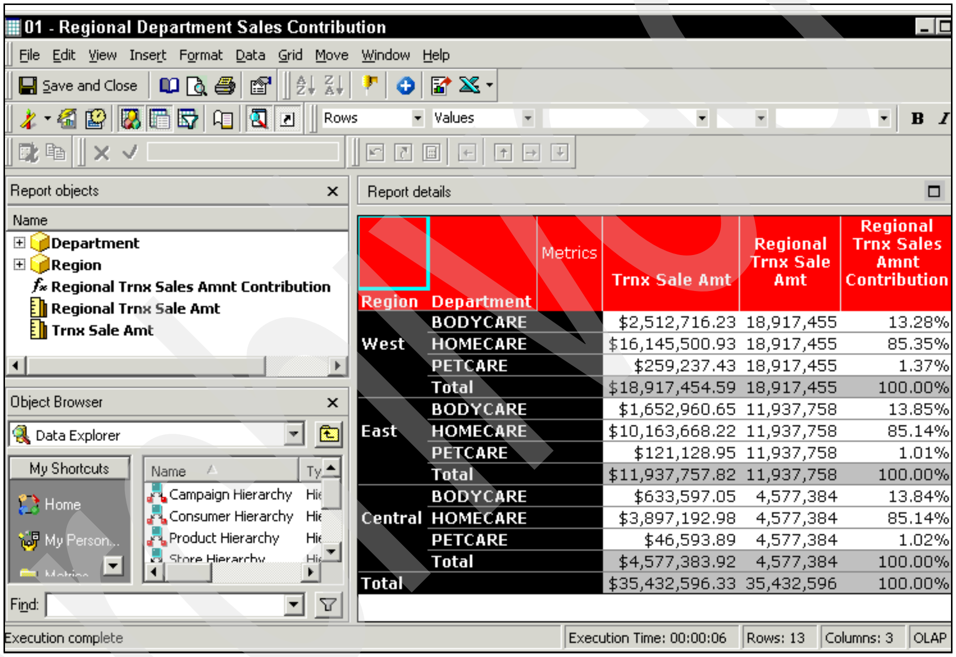


Figure 13-5 Question 1: report grid

How to View SQL in MicroStrategy

Once the report results have been returned the user can obtain the SQL from the SQL View option in the View menu in the Report Editor as shown in Figure 13-6.

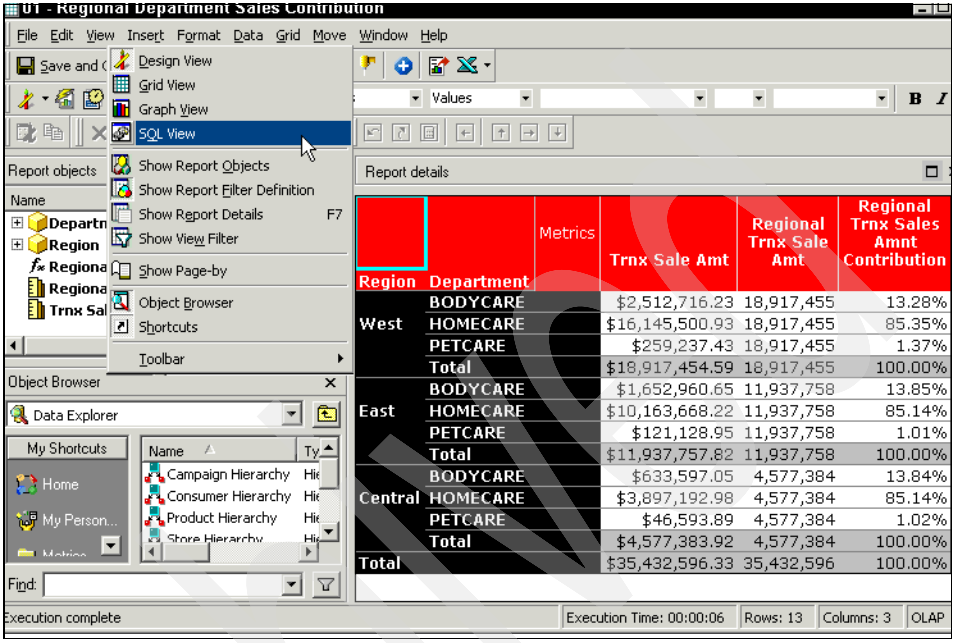


Figure 13-6 SQL View option

Once in SQL View mode, the user may copy the SQL by scrolling down to the SQL Statements section and highlighting the text as shown in Figure 13-7.

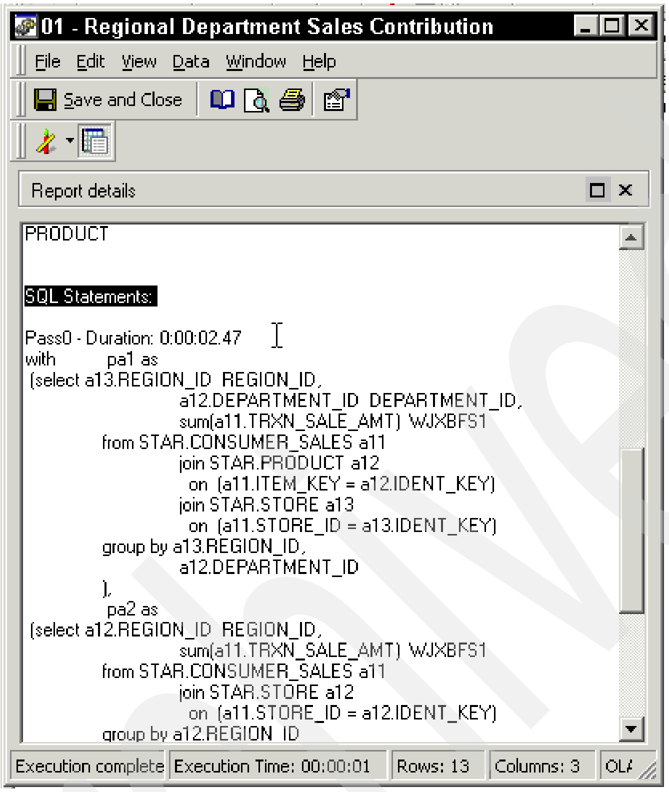


Figure 13-7 Scroll down to SQL Statements section

> **Note:** Make sure you remove all tabs from the SQL by using a text editor before pasting the SQL in the DB2 Explain SQL Dialog.

Query performance results

After submitting the SQL of the Regional Department Sales Contribution report directly into the DB2 Explain SQL Dialog, we were able to obtain the database cost, which was 735,263.62 timerons with a very complex construction generation, as shown in Figure 13-8.

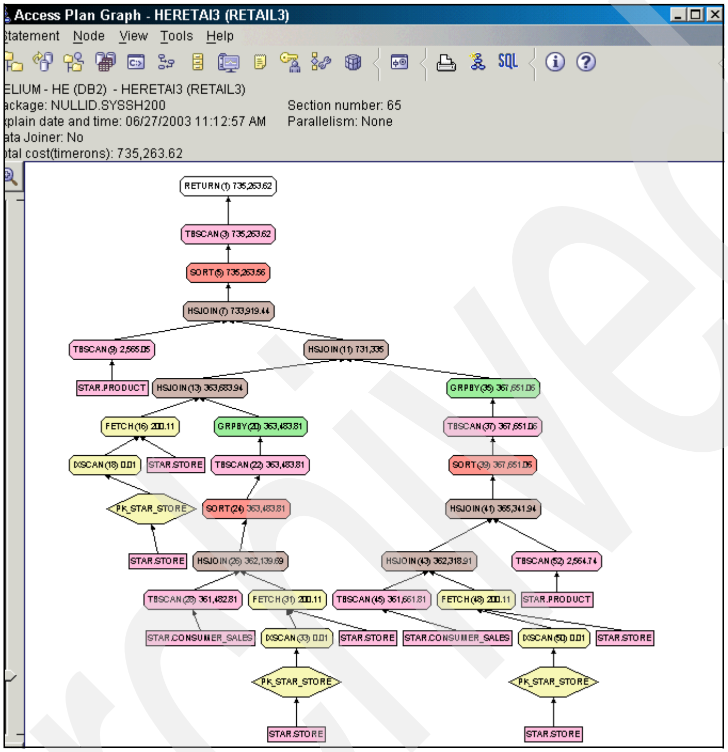


Figure 13-8 DB2 explain for Question 1: without MQT

When submitting the same SQL from Regional Department Sales Contribution report into a DB2 Database with Cube Views and MQTs enabled the Database was able to determine the usage of MQT tables dynamically and used them as fact tables for the resolution of the results.

The database cost for solving this SQL was 3,779.44 timerons with a more simplified construction complexity. The results are shown in Figure 13-9.

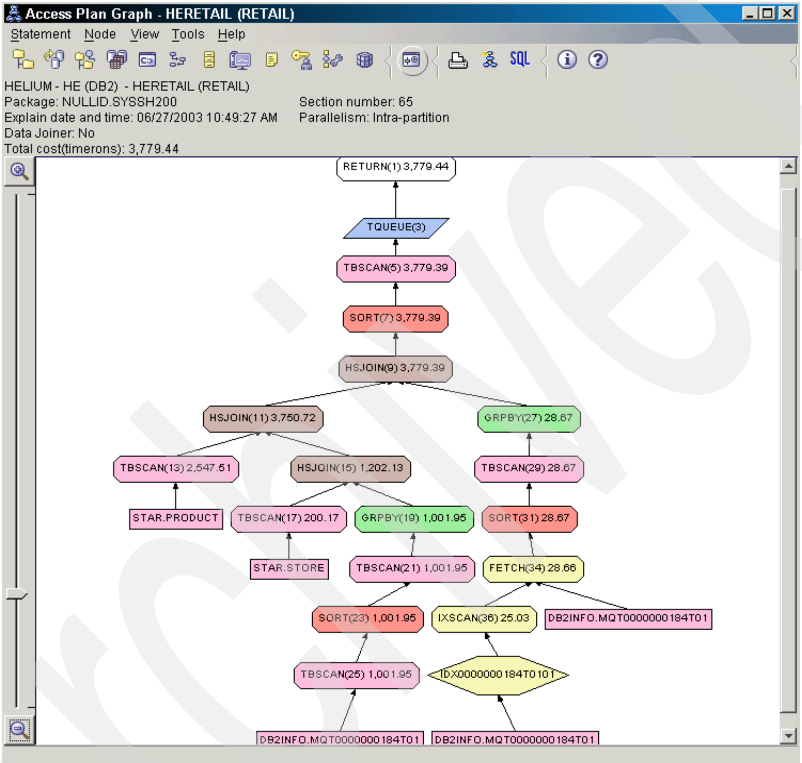


Figure 13-9 DB2 explain for question 1: with MQT

Table 13-2 summarizes the data access path costs issued from DB2 explain when using DB2 Cube Views MQTs.

Table 13-2 Query performance result

	Timeron
Without MQT	735,263.62
With MQT	3,779.44

13.5.3 Question 2: campaign contributions to sales

The business issue to solve is to assess how each of the campaigns in your region contributed to the rest of the region in terms of sales.

The business issue mentioned above has been resolved using the *Drill Anywhere* functionality in MicroStrategy which allows the user to drill anywhere in the project's browsing hierarchies. In this case, the user has drilled to the Campaign attribute from Region in the 01 – Regional Department Sales Contribution report, as shown in Figure 13-10.

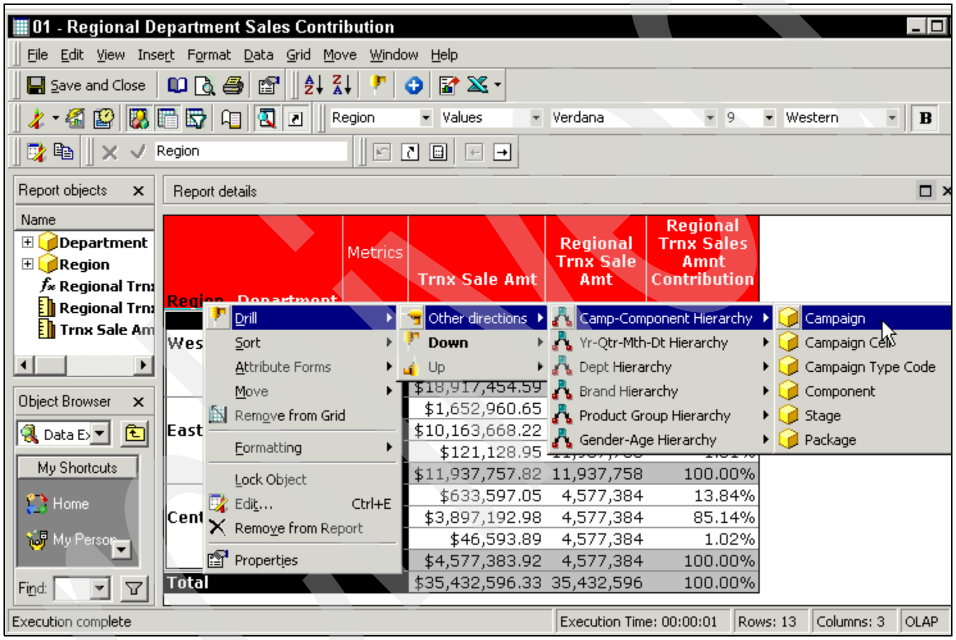
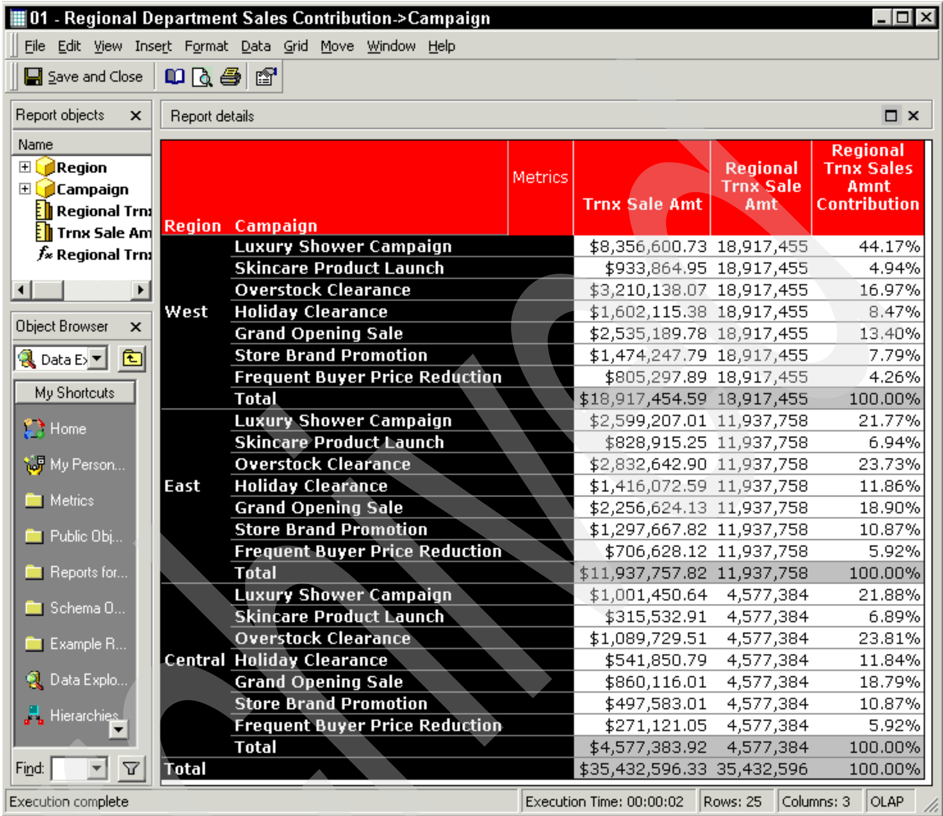


Figure 13-10 Drilling to campaign

Once the user has drilled to Campaign attribute, the MicroStrategy Intelligence Server will generate the corresponding SQL to bring back to the user a report with data at the Campaign level, as shown in Figure 13-11.



Region	Campaign	Trnx Sale Amt	Regional Trnx Sale Amt	Regional Trnx Sales Amnt Contribution
West	Luxury Shower Campaign	$8,356,600.73	18,917,455	44.17%
	Skincare Product Launch	$933,864.95	18,917,455	4.94%
	Overstock Clearance	$3,210,138.07	18,917,455	16.97%
	Holiday Clearance	$1,602,115.38	18,917,455	8.47%
	Grand Opening Sale	$2,535,189.78	18,917,455	13.40%
	Store Brand Promotion	$1,474,247.79	18,917,455	7.79%
	Frequent Buyer Price Reduction	$805,297.89	18,917,455	4.26%
	Total	$18,917,454.59	18,917,455	100.00%
East	Luxury Shower Campaign	$2,599,207.01	11,937,758	21.77%
	Skincare Product Launch	$828,915.25	11,937,758	6.94%
	Overstock Clearance	$2,832,642.90	11,937,758	23.73%
	Holiday Clearance	$1,416,072.59	11,937,758	11.86%
	Grand Opening Sale	$2,256,624.13	11,937,758	18.90%
	Store Brand Promotion	$1,297,667.82	11,937,758	10.87%
	Frequent Buyer Price Reduction	$706,628.12	11,937,758	5.92%
	Total	$11,937,757.82	11,937,758	100.00%
Central	Luxury Shower Campaign	$1,001,450.64	4,577,384	21.88%
	Skincare Product Launch	$315,532.91	4,577,384	6.89%
	Overstock Clearance	$1,089,729.51	4,577,384	23.81%
	Holiday Clearance	$541,850.79	4,577,384	11.84%
	Grand Opening Sale	$860,116.01	4,577,384	18.79%
	Store Brand Promotion	$497,583.01	4,577,384	10.87%
	Frequent Buyer Price Reduction	$271,121.05	4,577,384	5.92%
	Total	$4,577,383.92	4,577,384	100.00%
Total		$35,432,596.33	35,432,596	100.00%

Figure 13-11 Question 2: report grid

Query performance results

After submitting the SQL to a DB2 database without DB2 Cube Views MQTs enabled, the database access timerons cost for generating results was of 735,798.81, as shown in Figure 13-12.

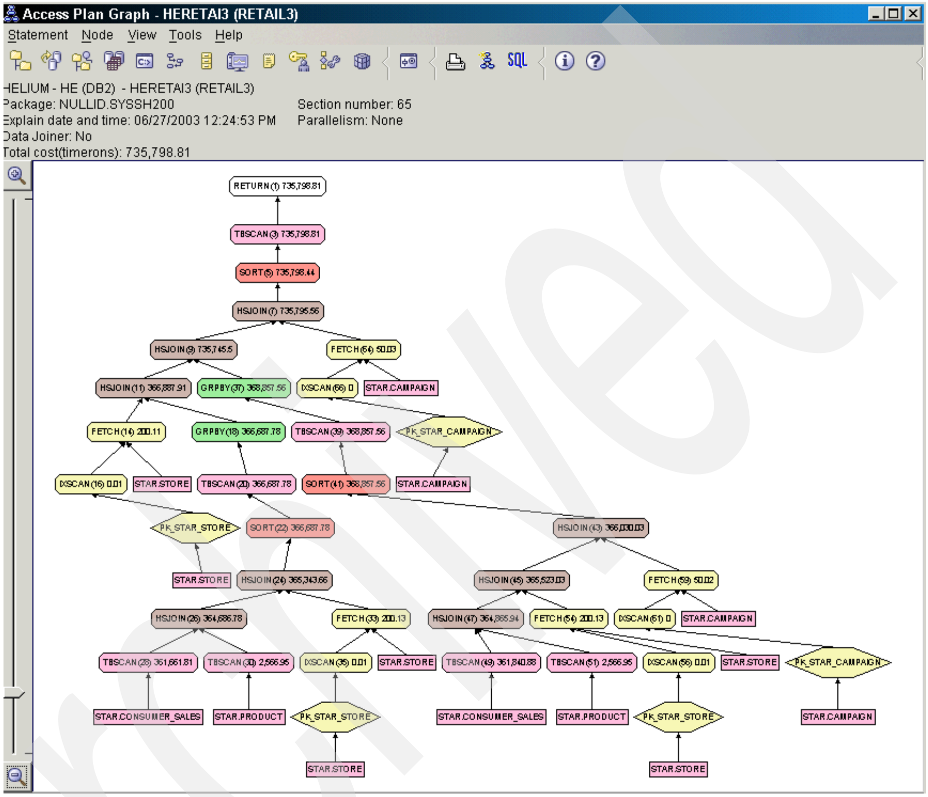


Figure 13-12 DB2 explain for question 2: without MQT

After submitting the report's SQL to a database with DB2 Cube Views and MQTs available, the total database timerons cost for generating results is 2279.19, as shown in Figure 13-13.

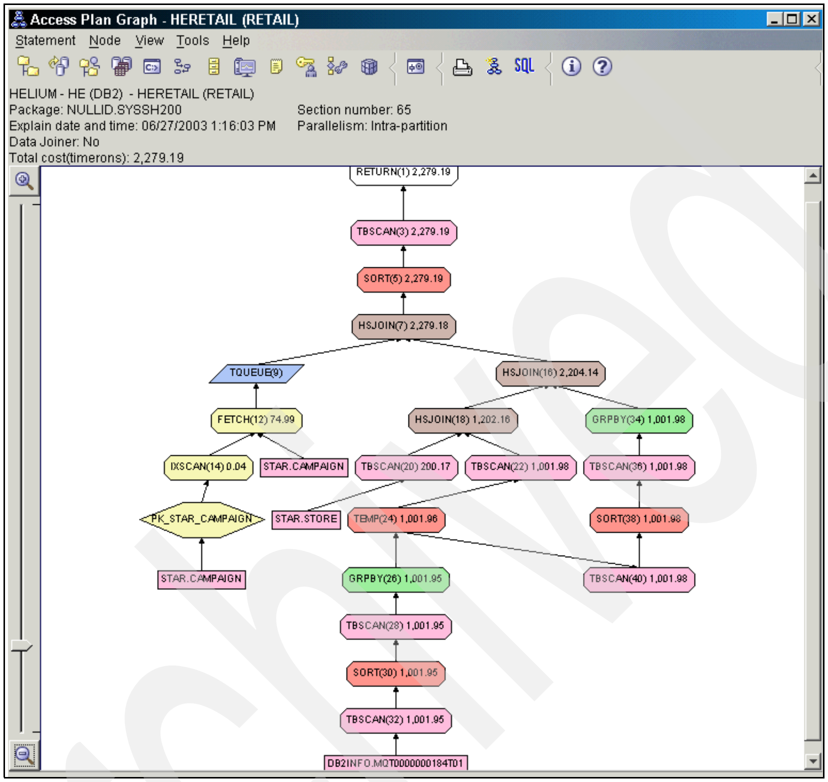


Figure 13-13 DB2 explain for question 2: with MQT

Table 13-3 summarizes the data access paths costs issued from DB2 explain when using DB2 Cube Views MQTs.

Table 13-3 Query performance result

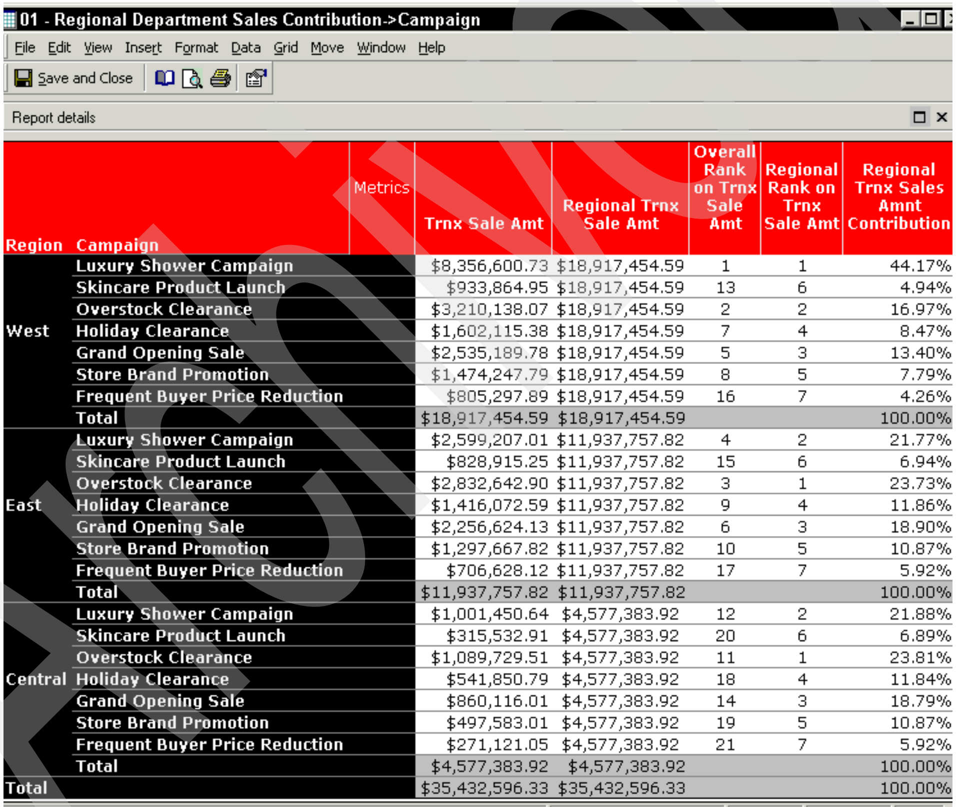
	timeron
Without MQT	735,798.81
With MQT	2,279.19

13.5.4 Question 3: ranking the campaigns by region

The business issue to solve is to assess how each one of the campaigns in your regions are ranked with the rest of the regions and within your own region.

The business issue mentioned above has been resolved by making use of the Rank function in MicroStrategy. Two metrics have been developed using this functionality: one that ranks the Transaction Sales Amount over all Regions, and a second one that makes use of the Break By function parameter set at the Region level. This second metric will provide the user with a Rank on Campaign per Region while the first one will provide the user with a Rank over all Regions.

The results of this report are shown in Figure 13-14.



Region	Campaign	Trnx Sale Amt	Regional Trnx Sale Amt	Overall Rank on Trnx Sale Amt	Regional Rank on Trnx Sale Amt	Regional Trnx Sales Amnt Contribution
West	Luxury Shower Campaign	$8,356,600.73	$18,917,454.59	1	1	44.17%
	Skincare Product Launch	$933,864.95	$18,917,454.59	13	6	4.94%
	Overstock Clearance	$3,210,138.07	$18,917,454.59	2	2	16.97%
	Holiday Clearance	$1,602,115.38	$18,917,454.59	7	4	8.47%
	Grand Opening Sale	$2,535,189.78	$18,917,454.59	5	3	13.40%
	Store Brand Promotion	$1,474,247.79	$18,917,454.59	8	5	7.79%
	Frequent Buyer Price Reduction	$805,297.89	$18,917,454.59	16	7	4.26%
	Total	$18,917,454.59	$18,917,454.59			100.00%
East	Luxury Shower Campaign	$2,599,207.01	$11,937,757.82	4	2	21.77%
	Skincare Product Launch	$828,915.25	$11,937,757.82	15	6	6.94%
	Overstock Clearance	$2,832,642.90	$11,937,757.82	3	1	23.73%
	Holiday Clearance	$1,416,072.59	$11,937,757.82	9	4	11.86%
	Grand Opening Sale	$2,256,624.13	$11,937,757.82	6	3	18.90%
	Store Brand Promotion	$1,297,667.82	$11,937,757.82	10	5	10.87%
	Frequent Buyer Price Reduction	$706,628.12	$11,937,757.82	17	7	5.92%
	Total	$11,937,757.82	$11,937,757.82			100.00%
Central	Luxury Shower Campaign	$1,001,450.64	$4,577,383.92	12	2	21.88%
	Skincare Product Launch	$315,532.91	$4,577,383.92	20	6	6.89%
	Overstock Clearance	$1,089,729.51	$4,577,383.92	11	1	23.81%
	Holiday Clearance	$541,850.79	$4,577,383.92	18	4	11.84%
	Grand Opening Sale	$860,116.01	$4,577,383.92	14	3	18.79%
	Store Brand Promotion	$497,583.01	$4,577,383.92	19	5	10.87%
	Frequent Buyer Price Reduction	$271,121.05	$4,577,383.92	21	7	5.92%
	Total	$4,577,383.92	$4,577,383.92			100.00%
Total		$35,432,596.33	$35,432,596.33			100.00%

Figure 13-14 Question 3: report grid

Query performance results

After submitting the SQL from this report to a DB2 database without DB2 Cube Views MQTs available, we were able to estimate the database total cost of 735,798.94 timerons.

After submitting the report's SQL to a database with DB2 Cube Views and MQTs available, the total database cost for generating results is 3779.44 timerons.

Table 13-4 summarizes the data access paths costs issued from DB2 explain when using DB2 Cube Views MQTs.

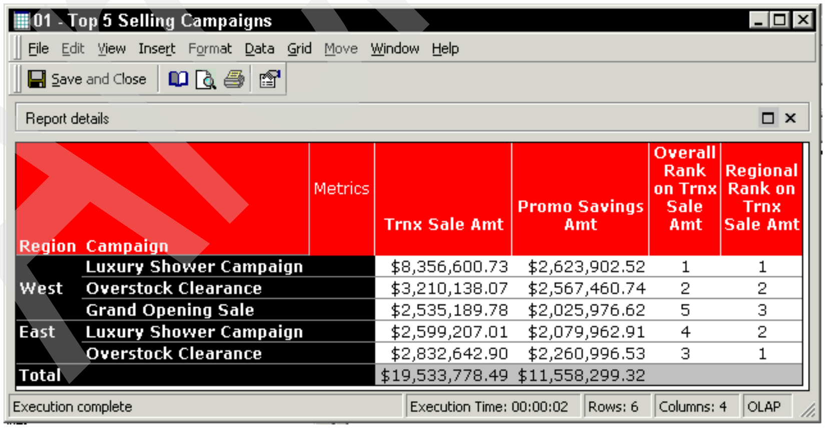
Table 13-4 Query performance result

	timeron
Without MQT	735,798.94
With MQT	3,779.44

13.5.5 Question 4: obtaining the Top 5 campaigns

The business issue to solve is to assess what are the Top 5 campaigns ranked with the rest of the regions and within your own region.

The business issue mentioned above has been resolved by making use of the *Report Limit* functionality in the *Report Data* options in MicroStrategy. The user has created a Top 5 filter based on the Rank of Trnx Sales Amt and added the filter to the Report Limit properties in the Report Data options. The report is displayed in Figure 13-15.



Region	Campaign	Trnx Sale Amt	Promo Savings Amt	Overall Rank on Trnx Sale Amt	Regional Rank on Trnx Sale Amt
West	Luxury Shower Campaign	$8,356,600.73	$2,623,902.52	1	1
	Overstock Clearance	$3,210,138.07	$2,567,460.74	2	2
	Grand Opening Sale	$2,535,189.78	$2,025,976.62	5	3
East	Luxury Shower Campaign	$2,599,207.01	$2,079,962.91	4	2
	Overstock Clearance	$2,832,642.90	$2,260,996.53	3	1
Total		$19,533,778.49	$11,558,299.32		

Figure 13-15 Question 4: report grid

Query performance results

After submitting the SQL from this report to a DB2 database without DB2 Cube Views MQTs available, we were able to estimate the database total cost of 372,805.06 timerons.

After submitting the report's SQL to a database with DB2 Cube Views and MQTs available, the total database cost for generating results is 1002.01 timerons.

Table 13-5 summarizes the data access path costs issued from DB2 explain when using DB2 Cube Views MQTs.

Table 13-5 Query performance result

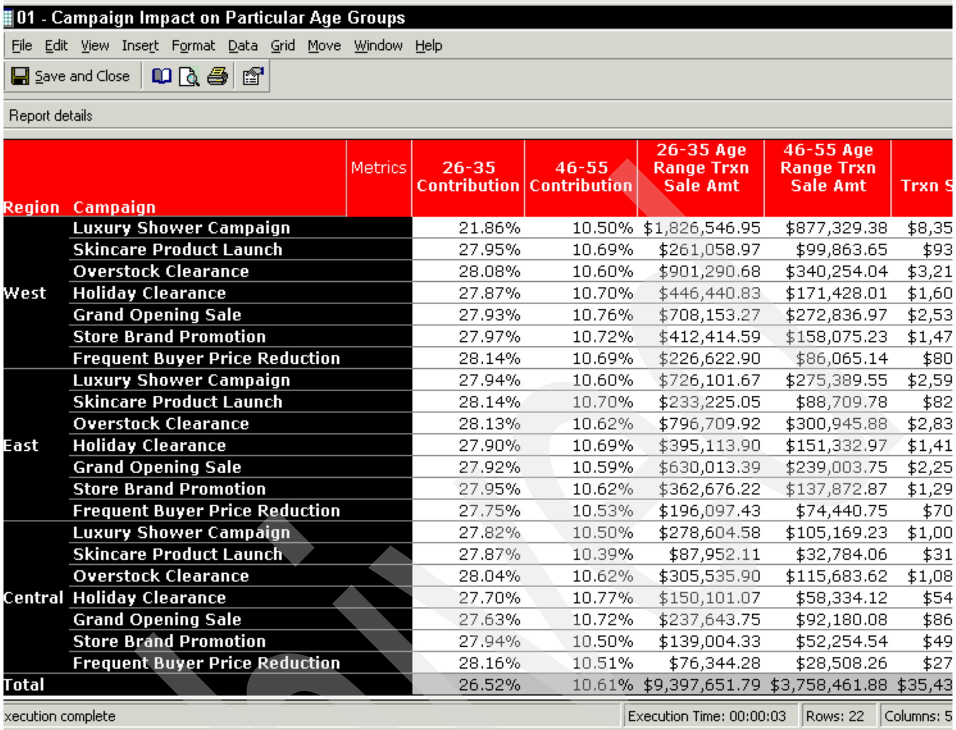
	timeron
Without MQT	372,805.06
With MQT	1,002.01

13.5.6 Question 5: campaign impact by age range

The business issue to solve is to assess how each of the campaigns in your region impact the age groups of 26-35 and 46-55.

The business issue mentioned above has been resolved by making use of the *Conditional Metrics* functionality in MicroStrategy. The user has created two metrics based on the sum of Transaction Sale Amount and adding to each one of them a filter on the desired age groups.

The results of this report are shown in Figure 13-16.



01 - Campaign Impact on Particular Age Groups

File Edit View Insert Format Data Grid Move Window Help

Save and Close

Report details

Region	Campaign	26-35 Contribution	46-55 Contribution	26-35 Age Range Trxn Sale Amt	46-55 Age Range Trxn Sale Amt	Trxn S
West	Luxury Shower Campaign	21.86%	10.50%	$1,826,546.95	$877,329.38	$8,35
	Skincare Product Launch	27.95%	10.69%	$261,058.97	$99,863.65	$93
	Overstock Clearance	28.08%	10.60%	$901,290.68	$340,254.04	$3,21
	Holiday Clearance	27.87%	10.70%	$446,440.83	$171,428.01	$1,60
	Grand Opening Sale	27.93%	10.76%	$708,153.27	$272,836.97	$2,53
	Store Brand Promotion	27.97%	10.72%	$412,414.59	$158,075.23	$1,47
	Frequent Buyer Price Reduction	28.14%	10.69%	$226,622.90	$86,065.14	$80
East	Luxury Shower Campaign	27.94%	10.60%	$726,101.67	$275,389.55	$2,59
	Skincare Product Launch	28.14%	10.70%	$233,225.05	$88,709.78	$82
	Overstock Clearance	28.13%	10.62%	$796,709.92	$300,945.88	$2,83
	Holiday Clearance	27.90%	10.69%	$395,113.90	$151,332.97	$1,41
	Grand Opening Sale	27.92%	10.59%	$630,013.39	$239,003.75	$2,25
	Store Brand Promotion	27.95%	10.62%	$362,676.22	$137,872.87	$1,29
	Frequent Buyer Price Reduction	27.75%	10.53%	$196,097.43	$74,440.75	$70
Central	Luxury Shower Campaign	27.82%	10.50%	$278,604.58	$105,169.23	$1,00
	Skincare Product Launch	27.87%	10.39%	$87,952.11	$32,784.06	$31
	Overstock Clearance	28.04%	10.62%	$305,535.90	$115,683.62	$1,08
	Holiday Clearance	27.70%	10.77%	$150,101.07	$58,334.12	$54
	Grand Opening Sale	27.63%	10.72%	$237,643.75	$92,180.08	$86
	Store Brand Promotion	27.94%	10.50%	$139,004.33	$52,254.54	$49
	Frequent Buyer Price Reduction	28.16%	10.51%	$76,344.28	$28,508.26	$27
Total		26.52%	10.61%	$9,397,651.79	$3,758,461.88	$35,43

xecution complete | Execution Time: 00:00:03 | Rows: 22 | Columns: 5

Figure 13-16 Question 5: report grid

> **Note:** The metrics "26-25 Contribution" and "46-55 Contribution" were created using the *Derived Metrics* functionality in MicroStrategy.

Query performance results

After submitting the SQL from this report to a DB2 database without DB2 Cube Views MQTs available, we were able to estimate the database total cost of 1,105,888.88 timerons.

After submitting the report's SQL to a database with DB2 Cube Views and MQTs available, the total database cost for generating results is 3,281.01 timerons.

Table 13-6 summarizes the data access path costs issued from DB2 explain when using DB2 Cube Views MQTs.

Table 13-6 Query performance result

	timeron
Without MQT	1,105,888.88
With MQT	3,281.08

13.6 Conclusion: benefits

MicroStrategy has always placed considerable emphasis on the critical role of the database in a Business Intelligence (BI) system and thus continues its integration efforts with the DB2 product line. Integration with DB2 Cube Views is yet another way MicroStrategy strives at bringing our joint customers the best BI platform on top of DB2. In this chapter, we showed how customers stand to benefit from this exciting new technology in the following two ways:

- Accelerated deployment using the metadata bridge
- Increased query performance with MQTs

Recognizing the value of existing development investments, MicroStrategy allows system architects to import multidimensional metadata from DB2 Cube Views into its metadata via a metadata bridge. The bridge substantially reduces application development times and enables a straightforward migration of applications to the MicroStrategy platform.

Given MicroStrategy's ROLAP architecture leveraging the database as a query engine, a MicroStrategy BI system is best positioned to fully benefit from performance enhancements brought into DB2 by MQTs. Illustrative results presented in this chapter showed a high correlation between the presence of adequate MQTs and improved query times. While exact figures will vary by environments, one can still take away from this data a significant performance amelioration. Combined with highly optimized SQL of the MicroStrategy SQL Engine, MQTs therefore form a cohesive technology.

14

Web services for DB2 Cube Views

In this chapter, we describe how to access DB2 Cube Views cube metadata and the measures data using the Web services for DB2 Cube Views.

The following topics are addressed:

- Advantages of using Web services for DB2 Cube Views
- Overview of the technologies used
- Architecture of Web services for DB2 Cube Views
- Web services for DB2 Cube Views: Describe, Members, and Execute

14.1 Web services for DB2 Cube Views: advantages

Exposing OLAP functionality as Web services benefits other applications that require access to OLAP data. OLAP Web services deliver cubes, slices, or cells from a multidimensional model to be used in a client analytical application.

Web services are unlikely to become the new slice, dice, and drill interface for dedicated OLAP tools. These tools require the high-speed service they get from existing native interfaces. But Web services-based analytic applications will need access to multidimensional information. These new applications will be cross organizational and business boundaries, assembling information from a variety of sources and using it to inform and drive business processes.

Web services for DB2 Cube Views provides the following simple, high-level Web services using XPath as the query language.

- **Describe:** To query and navigate through OLAP metadata
- **Members:** To retrieve dimension member data
- **Execute:** To execute slice and dice queries on a cube

The following are the primary advantages of using Web services for DB2 Cube Views:

- Allows application developers to provide analytic capabilities to any client on any device or platform using any programming language. These Web services are based on open standards like XML, HTTP and SOAP, so the clients can have an independent implementation using any tool, technology or hardware platform. For example, client applications that run on pervasive devices like PDA can access OLAP data. Refer to 14.2, “Overview of the technologies used” on page 615 for more understanding of XML and SOAP.
- Allows client applications to easily and securely access remote analytical data hosted by partners, customers or suppliers over the Web. This helps in building analytical applications from diverse sources of data.
- Transition from a tightly-coupled client-server paradigm to loosely-coupled Web-based analytical systems. Prior to Web services, the client component to access OLAP server had to be installed on each client.
- Input to these Web services need to be specified as an XPath expression and the output is an XML document. So, the application developers can leverage on their existing knowledge on XML and XPath without requiring to learn OLAP interface and query languages. Refer to 14.2, “Overview of the technologies used” on page 615 for more understanding of XPath.

14.2 Overview of the technologies used

Web services are self-contained, software components that can be described, published, located, and invoked over the Web. It provides an universal program-to-program communication model based on open standards and common infrastructure for description, discovery, and invocation.

This section introduces the major components used in a Web services technology:

- XML to provide the interoperable content model
- SOAP to structure messages (Web services uses a message based communication) into requests and responses
- WSDL and XML Schema to describe a service, its bindings, and its location
- UDDI to register and find services
- XPath to query and select the information elements needed

14.2.1 Web services technology

Web services technology is essentially a programming model to aid development and deployment of loosely coupled applications within a company or across industries. For example, you can use Web services to connect the back-end of an enterprise system to markets and other industrial partners. The success of the World Wide Web is rapidly expanding the use of Web services in response to the need for application-to-application communication and interoperability. These services provide a standard means of communication among different software applications involved in presenting dynamic context-driven information to users.

Web services provide an easy-to-understand interface between the provider and consumer of the application resources using a Web Service Description Language (WSDL). Web services adopt the layered architecture to simplify the implementation of distributed applications, as illustrated in Figure 14-1.

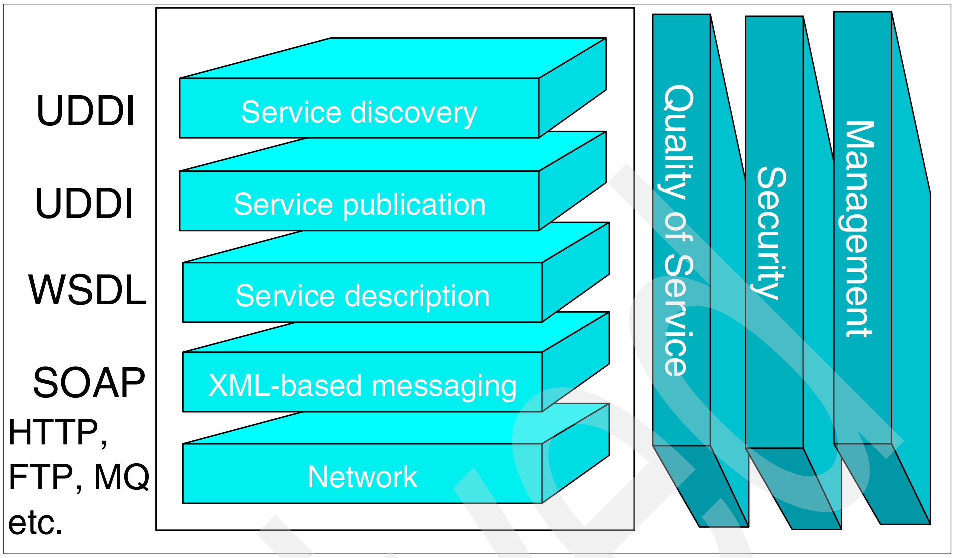


Figure 14-1 Web services layered architecture

The Web services layered architecture provides the following features:

- Application interface discovery using Universal Description, Discovery, and Integration (UDDI).
- Application interface description using Web Services Description Language (WSDL) and again UDDI.
- A standard message format using Simple Object Access Protocol (SOAP).
- A standard transport protocol using HyperText Transport Protocol (HTTP).
- A standard network protocol using TCP/IP.

14.2.2 XML

eXtensible Markup Language is an extensible tag language that can describe complicated structures in ways that are easy for programs to understand. Web services depend heavily on XML. XML is language- and platform-independent. it is XML that enables the conversation between business programs.

XML is a meta-markup language and is used for creating your own markup languages. Using XML, you can define the tags for your markup language. XML tags are used to describe the contents of the document. This means that any type of data can be defined easily using XML. XML is universal not only by its range of applications but also by its ease of use: Its text-based nature makes it easy to create tools, and it is also an open, license-free, cross-platform standard, which means anyone can create, develop, and use tools for XML. What also makes XML universal is its power. XML is a structured data format, which allows it to store complex data, whether it is originally textual, binary, or object-oriented.

14.2.3 SOAP

The current industry standard for XML messaging is Simple Object Access Protocol (SOAP). SOAP is the basis for the W3C XML Protocol Working Group.

SOAP has the following characteristics:

- SOAP is designed to be simple and extensible.
- All SOAP messages are encoded using XML.
- SOAP is transport protocol independent. HTTP is one of the supported transport. Hence, SOAP runs on top of the existing Internet infrastructure.
- SOAP does not support distributed garbage collection. Therefore, call by reference is not supported by SOAP; a SOAP client does not hold any stateful references to remote objects.
- SOAP is operating system independent and not tied to any programming language or component technology. It is object model neutral.

SOAP clients can be implemented independent of technology as long as the clients can issue service request through XML messages. Similarly, the service can be implemented in any language, platform as long as it can process XML messages and package XML messages as responses.

Originally, SOAP was created to be a network and transport neutral protocol to carry XML messages around. SOAP over HTTP became the premier way of implementing this protocol, to the point that the latest SOAP specification mandates HTTP support. Conceptually, there is no limitation for the network protocol that can be utilized.

SOAP Remote Procedure Call (RPC) is the latest stage in the evolution of SOAP; the body of a SOAP message contains a call to a remote procedure and the parameters to pass in. Both, the call and the parameters are expressed in XML.

14.2.4 WSDL

If we want to find services automatically, we require a way to formally describe both their invocation interface and their location. The Web Services Description Language (WSDL) V 1.1. provides a notation serving these purposes.

WSDL allows a service provider to specify the following characteristics of a Web service:

- Name of the Web service and addressing information
- Protocol and encoding style to be used when accessing the public operations of the Web service
- Type information: operations, parameters, and data types comprising the interface of the Web service, plus a name for this interface

A WSDL specification uses XML syntax, therefore, there is an XML Schema that defines the WSDL document.

14.2.5 UDDI

UDDI stands for universal description, discovery, and integration. UDDI is a technical discovery layer. It can be seen as the Yellow Pages in the Web services world. It defines:

- The structure for a registry of service providers and services
- The API that can be used to access registries with this structure
- The organization and project defining this registry structure and its API

UDDI is a search engine for application clients.

14.2.6 XPath

XML Path Language (XPath) provides a notation for selecting elements within an XML document. That is, XPath is a language for addressing and matching parts of an XML document when considered as a tree of nodes. It uses a compact and non-XML syntax. XPath operates on the logical structure underlying XML. Xpath models an XML document as a tree of nodes (root nodes, element nodes, attribute nodes, text nodes, namespace nodes, processing instruction nodes, and comment nodes).

The basic syntactic construct in XPath is the expression. An expression is the full XPath syntax. An object is obtained by evaluating an expression, which has one of the following four basic types:

- Node-set (an unordered collection of nodes without duplicates)
- Boolean
- Number
- String

XPath uses path notation to define locations within a document. The paths starting with a “/” signifies an absolute path. A simple example of this follows.

Let us consider an XML document that describes a Library System:

```
<LIBRARY>
   <BOOK ID=”B1.1”>
      <TITLE>xml</TITLE>
      <COPIES>5</COPIES>
   </BOOK>
   <BOOK ID=”B2.1”>
      <TITLE>WebSphere</TITLE>
      <COPIES>10</COPIES>
   </BOOK>
   <BOOK ID=”B3.2”>
      <TITLE>great novel</TITLE>
      <COPIES>10</COPIES>
   </BOOK>
   <BOOK ID=”B5.5”>
      <TITLE>good story</TITLE>
      <COPIES>10</COPIES>
   </BOOK>
</LIBRARY>
```

The path /child::book/child::copies selects all copies element children of book which are defined under the document’s root. The above path can also be written as /library/book/copies.

The XPath location step makes the selection of document part based on the basis and the predicate. The basis performs a selection based on Axis Name and Node Test. Then the predicate performs additional selections based on the outcome of the selection from the basis. A simple example of this is as follows:

The path /library/book[1] selects the first book element under library.

14.3 Architecture of Web services for DB2 Cube Views

Figure 14-2 sketches the architecture of Web services for DB2 Cube Views.

The OLAP service provider may be registered in a UDDI registry for service requestors or clients to find and discover Web services to retrieve meta-data, to execute slice and dice queries, and to retrieve member data.

OLAP Web services client can discover OLAP providers in UDDI registries, access the provider through the Web services to retrieve XML descriptions of cubes and execute slice and dice queries on the cubes.

A client application composes a SOAP request envelop containing the input parameter values and sends it through SOAP and HTTP to the OLAP provider.

To respond to the request, the OLAP provider queries the OLAP metadata or OLAP data depending on the request, computes a result and sends a SOAP response envelope back to the client application. SOAP essentially defines an RPC (remote procedure call)-like XML protocol over HTTP between the client and provider.

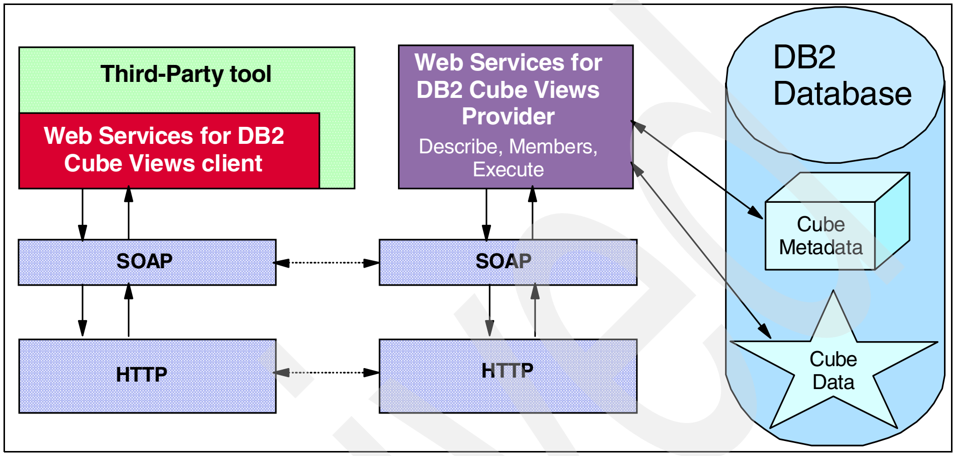


Figure 14-2 Web services for DB2 Cube Views Architecture

Service requestors or clients might reside in small devices such as cellular phones, in thin Web clients that deploy a browser interface, or in thick clients that perform some data analysis and visualization.

14.4 Web services for DB2 Cube Views

Web services for DB2 Cube Views offers the following high-level Web services:

- **Describe:** To query the cube metadata defined in DB2 Cube Views
- **Members**: To get the members of a cube dimension defined in DB2 Cube Views
- **Execute**: To execute slice and dice queries on the DB2 Cube Views cube

These Web services provide a means to query a cube defined in DB2 Cube Views for its metadata, members and measures data, in XML format.

The Describe Web service accesses the DB2 Cube Views metadata catalog tables using DB2 Cube Views API to retrieve the information. The Members and Execute Web services use cube metadata and its input parameter values to construct the SQL to query the base tables of the cube (star schema tables). Figure 14-3 shows the input and output for the Web services provided by IBM.

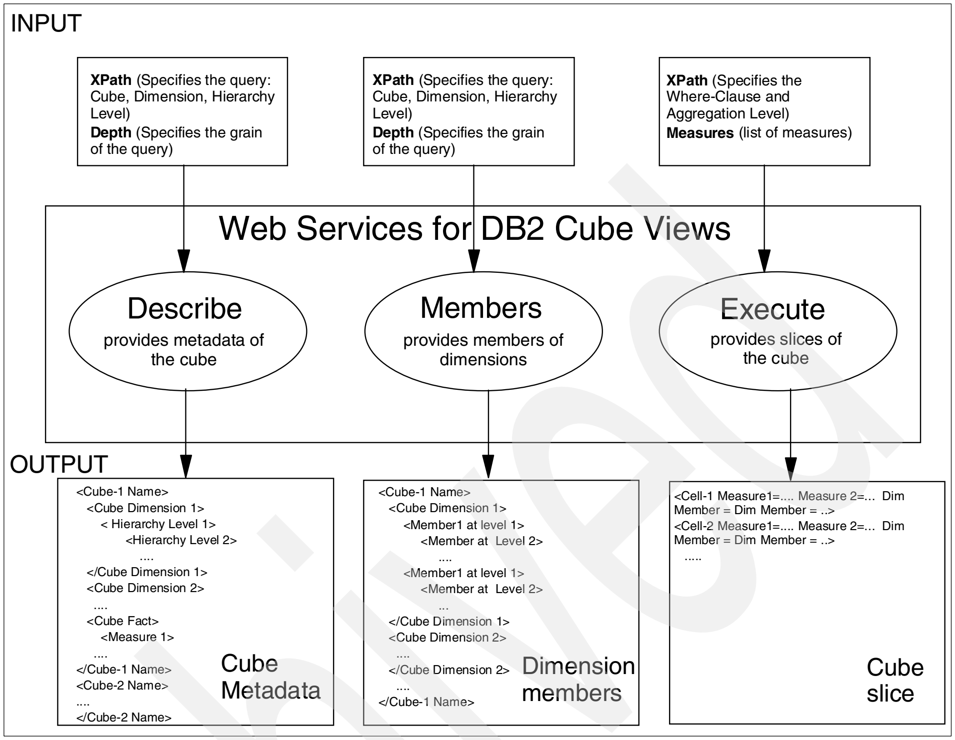


Figure 14-3 Web services for DB2 Cube Views

> **Note:** In the actual implementation, a client application composes a SOAP request envelope containing the input parameter values and sends it through SOAP and HTTP to the Web service provider. To respond to the request, the Web service queries the OLAP Metadata or OLAP data, computes a result and sends a SOAP response envelope back to the client application.

To describe each of the Web services for DB2 Cube Views, let us consider the representation of the *Sales* cube in Figure 14-4.

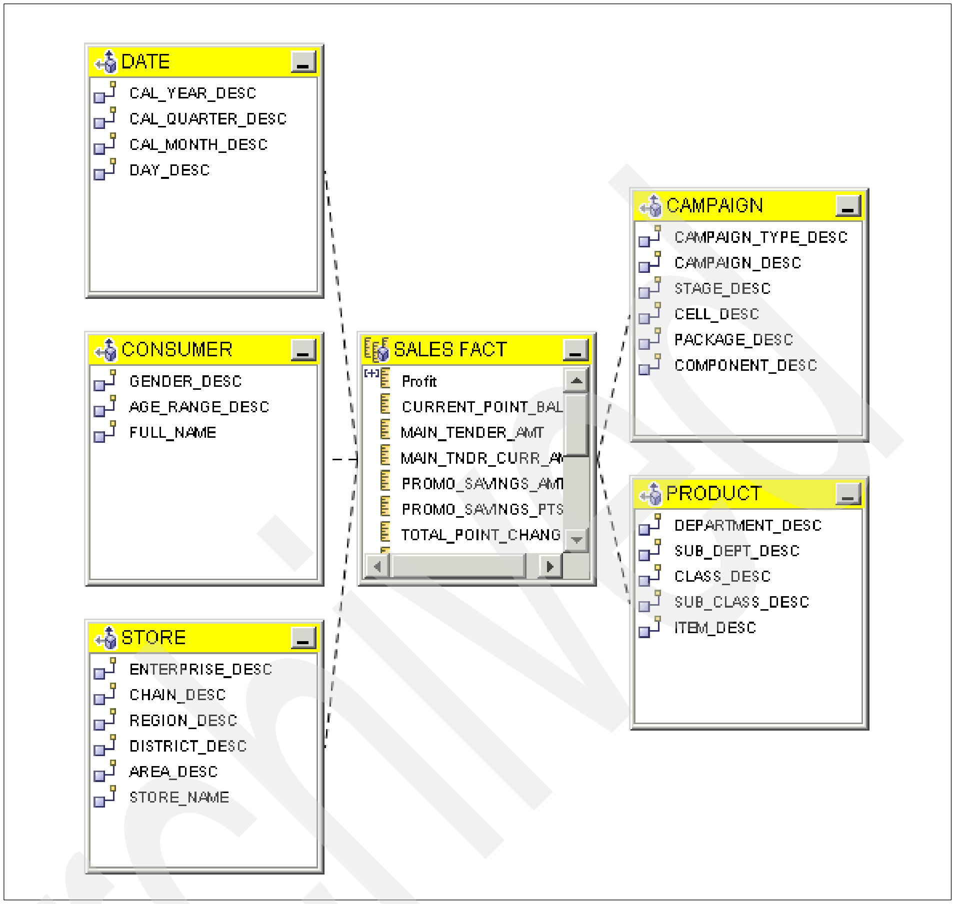


Figure 14-4 Sales cube

14.4.1 Describe

Client applications can retrieve the cube metadata defined in DB2 Cube Views using the Describe Web service. The metadata for a cube defined in DB2 Cube Views includes:

- Cube dimensions
- Cube dimensions hierarchy
- Cube fact (cube measures)

The metadata also includes the business names for the cube, each of its dimensions, the levels in the dimension hierarchy and the measures.

The metadata does not include the actual member and fact data.

Consider the representation of the Sales cube in Figure 14-4 as an XML document in Figure 14-5 and Figure 14-6. The Describe Web service works on the XML representation of the Star schema to generate the metadata output.

```
<Sales_Cube businessName="Sales Cube">                                Sales Cube
    <DATE businessName="DATE" kind="cubeDimension">                     DATE Dimension
        <CAL_YEAR_DESC businessName="Calender Year Description">
            <CAL_QUARTER_DESC businessName="Calender Quarter
            Description">
                CAL_MONTH_DESC businessName="Calender Month Name">
                    <DAY_DESC businessName="Day Description"/>          DATE Hierarchy
                </CAL_MONTH_DESC>
            </CAL_QUARTER_DESC>
        </CAL_YEAR_DESC>
    </DATE>
    <CAMPAIGN businessName="CAMPAIGN" kind="cubeDimension">             CAMPAIGN Dimension
        <CAMPAIGN_TYPE_DESC businessName="Campaign Type Description">
            <CAMPAIGN_DESC businessName="Campaign Description">
                <STAGE_DESC businessName="Stage Description">
                    <CELL_DESC businessName="Cell Description">
                        <PACKAGE_DESC businessName="Package Description">
                            <COMPONENT_DESC
            businessName="ComponentDescription"/>                       CAMPAIGN Hierarchy
                        </PACKAGE_DESC>
                    </CELL_DESC>
                </STAGE_DESC>
            </CAMPAIGN_DESC>
        </CAMPAIGN_TYPE_DESC>
    </CAMPAIGN>
    <CONSUMER businessName="CONSUMER" kind="cubeDimension">             CONSUMER Dimension
        <GENDER_DESC businessName="Gender Description">
            <AGE_RANGE_DESC businessName="Age Range Description">
                <FULL_NAME businessName="Full Name"/>                   CONSUMER Hierarchy
            </AGE_RANGE_DESC>
        </GENDER_DESC>
    </CONSUMER>
    <PRODUCT businessName="PRODUCT" kind="cubeDimension">               CONSUMER Dimension
        <DEPARTMENT_DESC businessName="Department Description">
            <SUB_DEPT_DESC businessName="Sub Department Description">
                <CLASS_DESC businessName="Class Description">
                    <SUB_CLASS_DESC businessName="Sub Class Description">
                        <ITEM_DESC businessName="Item Description"/>    PRODUCT Hierarchy
                    </SUB_CLASS_DESC>
                </CLASS_DESC>
            </SUB_DEPT_DESC>
        </DEPARTMENT_DESC>
    </PRODUCT>
```

Figure 14-5 XML Representation of Sales Cube (Part 1 of 2)

```
   <STORE businessName="STORE" kind="cubeDimension">
      <ENTERPRISE_DESC businessName="Enterprise Description">
         <CHAIN_DESC businessName="Chain Description">
            <REGION_DESC businessName="Region Description">
               <DISTRICT_DESC businessName="District Description">
                  <AREA_DESC businessName="Area Description">
                     <STORE_NAME businessName="Store Name"/>
                  </AREA_DESC>
               </DISTRICT_DESC>
            </REGION_DESC>
         </CHAIN_DESC>
      </ENTERPRISE_DESC>
   </STORE>
   <SALES_FACT businessName="SALES FACT" kind="cubeFacts">
      <Profit businessName="Profit" />
      <CURRENT_POINT_BAL businessName="Consumer Point Balance" />
      <MAIN_TENDER_AMT businessName="Main Tender Amount"/>
      <MAIN_TNDR_CURR_AMT businessName="Main Tender Current Amount" />
      <PROMO_SAVINGS_AMT businessName="Promotion Savings Amount" />
      <PROMO_SAVINGS_PTS businessName="Promotion Savings Points" />
      <TOTAL_POINT_CHANGE businessName="Total Point Change" />
      <TRXN_COST_AMT businessName="Transaction Cost Amount" />
      <TRXN_SALE_AMT businessName="Transaction Sale Amount" />
      <TRXN_SALE_QTY businessName="Transaction Sale Quantity" />
      <TRXN_SAVINGS_AMT businessName="Transaction Savings Amount" />
      <TRXN_SAVINGS_PTS businessName="Transaction Savings in Points"
      />
      <CONSUMER_QTY businessName="Consumer Quantity" />
      <ITEM_QTY businessName="Item Quantity" />
      <TRXN_QTY businessName="Transaction Quantity" />
   </SALES_FACT>
</Sales_Cube>
```

STORE Dimension

STORE Hierarchy

Fact

Measures

Figure 14-6 XML Representation of Sales Cube (Part 2 of 2)

The XML document contains the following:

- Cube: Sales_Cube XML element
- Dimensions: XML elements for *DATE*, *CAMPAIGN*, *PRODUCT*, *STORE* and *CONSUMER* dimensions.
- Dimension hierarchy: XML elements for the levels in the dimension hierarchy.

  For example, the CONSUMER Hierarchy has 3 levels: GENDER_DESC, AGE_RANGE_DESC and FULL_NAME.

- Measures: XML elements for each of the measures are represented under a cube facts element, all at the same level (without nesting).

  For example, In Sales cube, the measures under *SALES* FACT include Profit, *TRXN_SALE_AMT*, *TRXN_COST_AMT*, *CURRENT_POINT_BAL*, *MAIN_TENDER_AMT*, and so on, all at the same level, even though there are measures which are derived from other measures. Example, *Profit* is derived from *TRXN_SALE_AMT* and *TRXN_COST_AMT.*

There are also other attributes associated with each of the elements. For example, Business Name is an attribute for each of the elements in the XML document.

Any element in the XML document can be referenced by specifying the XPath. Refer to section Section 14.2.6, "XPath" on page 618 to understand XPath. For example, DATE within the XML document in Figure 14-5 can be referenced as Sales_Cube/DATE.

Depth is used to filter out children-nodes below a certain level from the nodes selected by the XPath. Depth -1 indicates no filter.

A client application can use Describe Web service to query specific metadata information by specifying an XPath query expression and depth.

Table 14-1 explains the input and output parameters of the Describe Web service.

Table 14-1 Parameters for Describe

Type of Parameter	Parameter Name	Description
Input	XPath	XPath defines the query. It specifies the location of the element (Cube Name, Dimension Names, Hierarchy levels, Cube Fact) in the XML representation of the cube.
Input	Depth	Depth defines the grain of the query. This is used to filter the children-nodes below a certain level from the selection made by the XPath query.
Output	XML document	XML document containing the metadata

If a client application wants to query the *STORE* dimension in Sales cube and restrict selection to only 3 levels deep, the Describe Web service will be invoked with the following input.

- **XPath:** Sales_Cube/STORE

- **Depth:** 3

The output shown in Figure 14-7 from the Describe Web service will be the metadata for *STORE* dimension with information on the top 3 levels in the hierarchy.

```
<STORE businessName="STORE" kind="cubeDimension">
  <ENTERPRISE_DESC businessName="Enterprise Description" >
    <CHAIN_DESC businessName="Chain Description">
      <REGION_DESC businessName="Region Description" />
    </CHAIN_DESC>
  </ENTERPRISE_DESC>
</STORE>
```

STORE Dimension

STORE Hierarchy (top 3 levels)

Figure 14-7 Metadata for STORE dimension

A client application can query the Sales_Cube for all its metadata by invoking the Describe Web service with the following input.

- **XPath:** Sales_Cube
- **Depth:** -1

The output will be the same as the complete XML representation of the Sales Cube as in Figure 14-2.

A client application can query the Sales_Cube for only its high level metadata by invoking Describe Web service with the following input.

- **XPath:** Sales_Cube
- **Depth:** 1

The output will be as in Figure 14-8.

```
<Sales_Cube businessName="Sales Cube">
  <DATE businessName="DATE" / kind="cubeDimension"/>
  <CAMPAIGN businessName="CAMPAIGN" kind="cubeDimension"/>
  <STORE businessName="STORE" kind="cubeDimension"/>
  <CONSUMER businessName="CONSUMER" kind="cubeDimension"/>
  <PRODUCT businessName="PRODUCT" kind="cubeDimension"/>
  <SALES_FACT businessName="SALES FACT" kind="cubeFacts"/>
</Sales_Cube>
```

Sales Cube

Dimensions & Fact (Toop Level)

Figure 14-8 Dimensions in Sales cube

As you can see, the metadata retrieved by the XPath is controlled by altering the value of the Depth parameter.

14.4.2 Members

The Members Web service returns the members of the cube dimensions.

Consider the representation of the Sales cube dimension members as an XML document in Figure 14-9 as per the dimension hierarchy defined for the cube in DB2 Cube Views. The Members Web service works on the XML representation of the dimension members to generate the dimension members output.

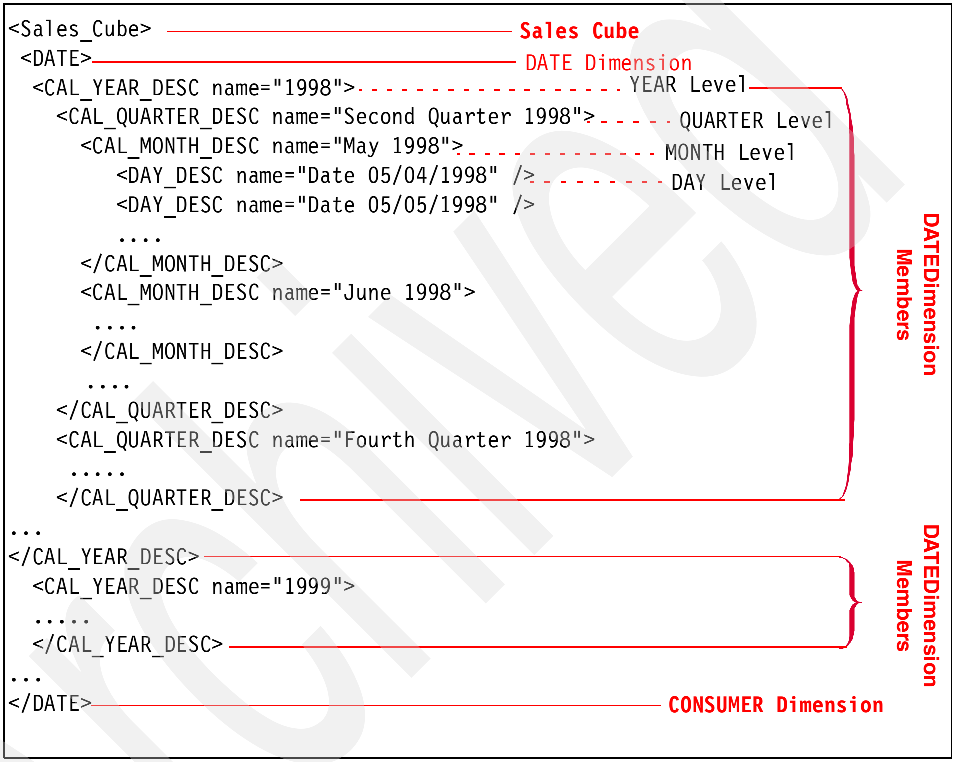


Figure 14-9 XML Representation of Dimension members in Sales Cube (1 of 2)

```
<CONSUMER>                                              ┐
  <GENDER_DESC name="Female">                           │
    <AGE_RANGE_DESC name="Less than 19">                │
      <FULL_NAME name="Ana Brady" />                    │  Consumer
      <FULL_NAME name="Elvira Ricks" />                 │  Dimension
      ....                                              │  Members
    </AGE_RANGE_DESC>                                   │
    <AGE_RANGE_DESC name="19-25">                       │
     .....                                              │
    </AGE_RANGE_DESC>                                   │
    ...                                                 ┘
</GENDER_DESC>
  <GENDER_DESC name="Male">
  .....
  </GENDER_DESC>
 </CONSUMER>  ----------------------------------- STORE Dimension
 <STORE>
 ....
 </STORE>  ---------------------------------- CAMPAIGN Dimension
 <CAMPAIGN>
 ...
</CAMPAIGN>  ------------------------------------ PRODUCT Dimension
 <PRODUCT>
 ...
 </PRODUCT>
</Sales_Cube>
```

Figure 14-10 XML Representation of Dimension members in Sales Cube (2 of 2)

The XML document contains the following:

- Cube: Sales_Cube XML element
- Dimensions: XML elements for DATE, CAMPAIGN, PRODUCT, STORE and CONSUMER dimensions.
- Dimension members: XML elements for the members in each level of the dimension hierarchy.

  For example, the DATE Hierarchy has 4 levels: CAL_YEAR_DESC, CAL_QUARTER_DESC, CAL_MONTH_DESC and DAY_DESC. The first level lists the CAL_YEAR_DESC members, contained within that are CAL_QUARTER_DESC members and within that CAL_QUARTER_DESC members and within that DAY_DESC members.

The cube, the dimensions or the levels in the dimension hierarchy can be referenced by specifying the XPath. Refer to section Section 14.2.6, “XPath” on page 618 to understand XPath. For example, the dimension DATE can be referenced as Sales_Cube/DATE.

Depth is used to filter out children-nodes below a certain level from the nodes selected by the XPath. Depth -1 indicates no filter.

A client application can use Members Web service to query dimension members by specifying an XPath query expression and depth.

Table 14-2 explains the input and output parameters of the Members Web service.

Table 14-2 Parameters for Members

Type of Parameter	Parameter Name	Description
Input	XPath	XPath defines the query. It specifies the location of the element (Cube Name, Dimension Names, Hierarchy levels) in the XML representation of the cube.
Input	Depth	Depth defines the grain of the query. This is used to filter the children-nodes below a certain level from the selection made by the XPath query.
Output	XML document	XML document containing the dimension members. The XML document does not return the fact data.

For example, a client application can query the STORE dimension in Sales cube for all its members by invoking the Members Web service with the following input.

- **XPath:** Sales_Cube/STORE
- **Depth:** -1

The output shown in Figure 14-11 from the Members Web service will be all the members in the STORE dimension for all levels in the hierarchy.

```
<STORE> ------------------------------------------ STORE Dimension
   <ENTERPRISE_DESC name="Enterprise">      - - - - - - ENTERPRISE Level
      <CHAIN_DESC name="Chain Retail Market">  - - - - - - - - -CHAIN Level
         <REGION_DESC name="Central">  - - - - - - - - - - - - - - -REGION Level
            <DISTRICT_DESC name="Ohio">  - - - - - - - - - - - - - -DISTRICT Level
               <AREA_DESC name="Kent">   - - - - - - - - - - - - - -AREA Level
                  <STORE_NAME name="Store #71" />   - - - -STORE Level
                  <STORE_NAME name="Store #72" />
                  <STORE_NAME name="Store #73" />
                  <STORE_NAME name="Store #74" />
               </AREA_DESC>
               <AREA_DESC name="Lancaster">
                  <STORE_NAME name="Store #75" />
                  <STORE_NAME name="Store #76" />
                  ....
               </AREA_DESC>
               ....
            </DISTRICT_DESC>
            <DISTRICT_DESC name="Texas">
            ....
            </DISTRICT_DESC>
            ...
         </REGION_DESC>
         <REGION_DESC name="East">
         ....
         ....
         </REGION_DESC>
         <REGION_DESC name="West">
         ....
         ....
         </REGION_DESC>
         ...
      </CHAIN_DESC>
      ...
   </ENTERPRISE_DESC>
   ...
</STORE>
```

STORE Dimension Members

Figure 14-11 Dimension Members - STORE dimension

If a client application wants to list the top level members in the *DATE* dimension, the Members Web service will be invoked with the following input. The output will be as in Figure 14-12.

- **XPath:** Sales_Cube/DATE
- **Depth:** 1

```
<DATE>                                                DATE Dimension
   <CAL_YEAR_DESC name="1998" />
   <CAL_YEAR_DESC name="1999" />
   <CAL_YEAR_DESC name="2000" />
   <CAL_YEAR_DESC name="2001" />
</DATE>
```

Top Level Members (Years)

Figure 14-12 Top level members in DATE dimension

As you can see, the Members Web service lists only the top level for example, *CAL_YEAR_DESC* members of the DATE dimension as defined by the Depth parameter.

14.4.3 Execute

The Execute Web service retrieves an XML representation of the cube. An XML cube contains members and measures data.

Consider a slice of the Sales Cube in Figure 14-13. The slice contains data for the cross-section of 2 dimensions *DATE* and *PRODUCT* up to the Month/Sub-Department level. The NULL values in the table denote the highest level of aggregation for the specific column.

CAL_YEAR_DESC	CAL_QUARTER_DESC	CAL_MONTH_DESC	DEPARTMENT_DESC	SUB_DEPT_DESC	TRXN_SALE_AMT
-	-	-	-	-	9000
1999	-	-	-	-	9000
1999	First Quarter 1999	-	-	-	3000
1999	Fourth Quarter 1999	-	-	-	3000
1999	Second Quarter 1999	-	-	-	3000
1999	First Quarter 1999	Jan-99	-	-	3000
1999	Fourth Quarter 1999	Nov-99	-	`	3000
1999	Second Quarter 1999	Jun-99	-	-	3000
-	-	-	HOMECARE	-	9000
-	-	-	HOMECARE	GARDEN	480
-	-	-	HOMECARE	STATIONERY	8520
1999	-	-	HOMECARE		9000
1999	First Quarter 1999	-	HOMECARE		3000
1999	Fourth Quarter 1999	-	HOMECARE		3000
1999	Second Quarter 1999	-	HOMECARE		3000
1999	First Quarter 1999	Jan-99	HOMECARE		3000
1999	Fourth Quarter 1999	Nov-99	HOMECARE		3000
1999	Second Quarter 1999	Jun-99	HOMECARE		3000
1999	-	-	HOMECARE	GARDEN	480
1999	First Quarter 1999	-	HOMECARE	GARDEN	160
1999	Fourth Quarter 1999	-	HOMECARE	GARDEN	160
1999	Second Quarter 1999	-	HOMECARE	GARDEN	160
1999	First Quarter 1999	Jan-99	HOMECARE	GARDEN	160
1999	Fourth Quarter 1999	Nov-99	HOMECARE	GARDEN	160
1999	Second Quarter 1999	Jun-99	HOMECARE	GARDEN	160
1999	-	-	HOMECARE	STATIONERY	8520
1999	First Quarter 1999	-	HOMECARE	STATIONERY	2840
1999	Fourth Quarter 1999	-	HOMECARE	STATIONERY	2840
1999	Second Quarter 1999	-	HOMECARE	STATIONERY	2840
1999	First Quarter 1999	Jan-99	HOMECARE	STATIONERY	2840
1999	Fourth Quarter 1999	Nov-99	HOMECARE	STATIONERY	2840
1999	Second Quarter 1999	Jun-99	HOMECARE	STATIONERY	2840

Figure 14-13 Slice of Sales Cube

This slice can be represented in XML as in Example 14-1.

Example 14-1 XML Representation of the Sales Cube slice

```
<cell TRXN_SALE_AMT="9000" />
<cell CAL_YEAR_DESC="1999" TRXN_SALE_AMT="9000" />
<cell CAL_QUARTER_DESC="First Quarter 1999" CAL_YEAR_DESC="1999"
TRXN_SALE_AMT="3000" />
<cell CAL_QUARTER_DESC="Fourth Quarter 1999" CAL_YEAR_DESC="1999"
TRXN_SALE_AMT="3000" />
<cell CAL_QUARTER_DESC="Second Quarter 1999" CAL_YEAR_DESC="1999"
TRXN_SALE_AMT="3000" />
<cell CAL_MONTH_DESC=”Jan-99” CAL_QUARTER_DESC="First Quarter 1999"
CAL_YEAR_DESC="1999" TRXN_SALE_AMT="3000" />
<cell CAL_MONTH_DESC=”Nov-99” CAL_QUARTER_DESC="Fourth Quarter 1999"
CAL_YEAR_DESC="1999" TRXN_SALE_AMT="3000" />
<cell CAL_MONTH_DESC=”Jun-99” CAL_QUARTER_DESC="Second Quarter 1999"
CAL_YEAR_DESC="1999" TRXN_SALE_AMT="3000" />
<cell DEPARTMENT_DESC="HOMECARE" TRXN_SALE_AMT="9000" />
```

```
<cell DEPARTMENT_DESC="HOMECARE" SUB_DEPT_DESC="GARDEN" TRXN_SALE_AMT="480" />
<cell DEPARTMENT_DESC="HOMECARE" SUB_DEPT_DESC="STATIONERY"
TRXN_SALE_AMT="8520"/>
<cell CAL_YEAR_DESC="1999" DEPARTMENT_DESC="HOMECARE" TRXN_SALE_AMT="9000" />
<cell CAL_QUARTER_DESC="First Quarter 1999" CAL_YEAR_DESC="1999"
DEPARTMENT_DESC="HOMECARE" TRXN_SALE_AMT="3000" />
<cell CAL_QUARTER_DESC="Fourth Quarter 1999" CAL_YEAR_DESC="1999"
DEPARTMENT_DESC="HOMECARE" TRXN_SALE_AMT="3000" />
<cell CAL_QUARTER_DESC="Second Quarter 1999" CAL_YEAR_DESC="1999"
DEPARTMENT_DESC="HOMECARE" TRXN_SALE_AMT="3000" />
<cell CAL_MONTH_DESC=”Jan-99” CAL_QUARTER_DESC="First Quarter 1999"
CAL_YEAR_DESC="1999" DEPARTMENT_DESC="HOMECARE" TRXN_SALE_AMT="3000" />
<cell CAL_MONTH_DESC=”Nov-99” CAL_QUARTER_DESC="Fourth Quarter 1999"
CAL_YEAR_DESC="1999" DEPARTMENT_DESC="HOMECARE" TRXN_SALE_AMT="3000" />
<cell CAL_MONTH_DESC=”Jun-99” CAL_QUARTER_DESC="Second Quarter 1999"
CAL_YEAR_DESC="1999" DEPARTMENT_DESC="HOMECARE" TRXN_SALE_AMT="3000" />
<cell CAL_YEAR_DESC="1999" DEPARTMENT_DESC="HOMECARE" SUB_DEPT_DESC="GARDEN"
TRXN_SALE_AMT="480" />
<cell CAL_QUARTER_DESC="First Quarter 1999" CAL_YEAR_DESC="1999"
DEPARTMENT_DESC="HOMECARE" SUB_DEPT_DESC="GARDEN" TRXN_SALE_AMT="160" />
<cell CAL_QUARTER_DESC="Fourth Quarter 1999" CAL_YEAR_DESC="1999"
DEPARTMENT_DESC="HOMECARE" SUB_DEPT_DESC="GARDEN" TRXN_SALE_AMT="160" />
<cell CAL_QUARTER_DESC="Second Quarter 1999" CAL_YEAR_DESC="1999"
DEPARTMENT_DESC="HOMECARE" SUB_DEPT_DESC="GARDEN" TRXN_SALE_AMT="160" />
<cell CAL_MONTH_DESC=”Jan-99” CAL_QUARTER_DESC="First Quarter 1999"
CAL_YEAR_DESC="1999" DEPARTMENT_DESC="HOMECARE" SUB_DEPT_DESC="GARDEN"
TRXN_SALE_AMT="160" />
<cell CAL_MONTH_DESC=”Nov-99” CAL_QUARTER_DESC="Fourth Quarter 1999"
CAL_YEAR_DESC="1999" DEPARTMENT_DESC="HOMECARE" SUB_DEPT_DESC="GARDEN"
TRXN_SALE_AMT="160" />
<cell CAL_MONTH_DESC=”Jun-99” CAL_QUARTER_DESC="Second Quarter 1999"
CAL_YEAR_DESC="1999" DEPARTMENT_DESC="HOMECARE" SUB_DEPT_DESC="GARDEN"
TRXN_SALE_AMT="160" />
<cell CAL_YEAR_DESC="1999" DEPARTMENT_DESC="HOMECARE"
SUB_DEPT_DESC="STATIONERY" TRXN_SALE_AMT="8520" />
<cell CAL_QUARTER_DESC="First Quarter 1999" CAL_YEAR_DESC="1999"
DEPARTMENT_DESC="HOMECARE" SUB_DEPT_DESC="STATIONERY" TRXN_SALE_AMT="2840" />
<cell CAL_QUARTER_DESC="Fourth Quarter 1999" CAL_YEAR_DESC="1999"
DEPARTMENT_DESC="HOMECARE" SUB_DEPT_DESC="STATIONERY" TRXN_SALE_AMT="2840" />
<cell CAL_QUARTER_DESC="Second Quarter 1999" CAL_YEAR_DESC="1999"
DEPARTMENT_DESC="HOMECARE" SUB_DEPT_DESC="STATIONERY" TRXN_SALE_AMT="2840" />
<cell CAL_MONTH_DESC=”Jan-99” CAL_QUARTER_DESC="First Quarter 1999"
CAL_YEAR_DESC="1999" DEPARTMENT_DESC="HOMECARE" SUB_DEPT_DESC="STATIONERY"
TRXN_SALE_AMT="2840" />
<cell CAL_MONTH_DESC=”Nov-99” CAL_QUARTER_DESC="Fourth Quarter 1999"
CAL_YEAR_DESC="1999" DEPARTMENT_DESC="HOMECARE" SUB_DEPT_DESC="STATIONERY"
TRXN_SALE_AMT="2840" />
```

```
<cell CAL_MONTH_DESC=”Jun-99” CAL_QUARTER_DESC="Second Quarter 1999"
CAL_YEAR_DESC="1999" DEPARTMENT_DESC="HOMECARE" SUB_DEPT_DESC="STATIONERY"
TRXN_SALE_AMT="2840" />
```

Each cell represents a row of data in the slice represented in Figure 14-13. The column values are represented as attributes of the cell element. Attributes for NULL values are absent. Member values identify a cell in a cube.

Table 14-3 explains the input and output parameters of the Execute Web service:

Table 14-3 Parameters for Execute

Type of Parameter	Parameter Name	Description
Input	XPath	XPath defines the query. It specifies the cube name, where-clause and the aggregation level. ► cube name is the name of the DB2 Cube Views cube on which the slice and dice query will be executed. (Only one cube name can be specified) ► where-clause filters rows in the slice, and consequently remove cell XML elements in the XML cube ► aggregation level defines the level of aggregation of data returned and allows identification of the dimensions and levels that should be retrieved in an XML Cube
Input	Measures	List of measures.
Output	XML document	XML document containing the slice of the cube.

For example, a client application can query the measure *TRXN_SALE_AMT* in Sales cube for the Month “January 1999” for all sub-departments under the department “HOMECARE” by invoking the Execute Web service with the following input.

- **Cube Name:** Sales_Cube
- **XPath for where-clause**: PRODUCT/DEPARTMENT_DESC/@name=”HOMECARE” and DATE/CAL_YEAR_DESC/CAL_QUARTER_DESC/CAL_MONTH_DESC/@name="Jan-99"
- **XPath for aggregation level**: PRODUCT/DEPARTMENT/SUB_DEPT_DESC
- **Measures:** TRXN_SALE_AMT

The output from the Execute Web service will have the following attributes for each cell as defined by the aggregation level in addition to the measure:

- *DEPARTMENT_DESC*
- *SUB_DEPT_DESC*

The cells are filtered from Figure 14-13 based on the where-clause. As a result, the output will only have 3 cells as in Example 14-2.

Example 14-2 XML ouput for Execute

```
<cell DEPARTMENT_DESC="HOMECARE" TRXN_SALE_AMT="160" SUB_DEPT_DESC="GARDEN" />
<cell DEPARTMENT_DESC="HOMECARE" TRXN_SALE_AMT="2840"
SUB_DEPT_DESC="STATIONERY" />
<cell DEPARTMENT_DESC="HOMECARE" TRXN_SALE_AMT="3000" />
```

14.5 Conclusion

Web services for DB2 Cube Views presents a new opportunity for Web services-based analytical applications running on any device or platform using any programming language to access OLAP metadata and data.

Part 4

Appendixes

A

DataStage: operational process metadata configuration and DataStage job example

This appendix describes how to configure an Ascential DataStage project to produce event metadata that will be used for data lineage analysis of the DataStage design and operation as described in 8.2.4, “Performing data lineage and process analysis in MetaStage” on page 308.

It also describes in detail how to design and run a DataStage job that will be used to populate our sales model datamart.

Configure the operational metadata components

The example environment consists of two Windows machines acting as client and server. The functions that each machine will perform are summarized in Table A-1. Installation of each component will not be covered. It is assumed that installation of each component was performed successfully by following respective product installation documentation.

Table A-1 Client and server summary

Windows server	Windows client
Ascential DataStage Server Ascential Process MetaBroker IBM DB2 v8.1	Ascential MetaStage and clients (Administrator and Explorer) Ascential DataStage clients (Administrator, Manager and Designer) Ascential MetaStage Listener Ascential MetaStage RunImport

Configure the server machine

Firstly we will configure the server machine. The steps involved in configuring the server are:

1. Configure the DataStage project to emit process metadata
2. Configure the Process MetaBroker parameters

The first requirement for capturing process metadata is to configure your DataStage project to emit process metadata. You can selectively choose whether or not to produce process metadata on a project basis.

1. Start the DataStage Administrator and click the **Projects** tab. You will see a list of DataStage projects as in Figure A-1.

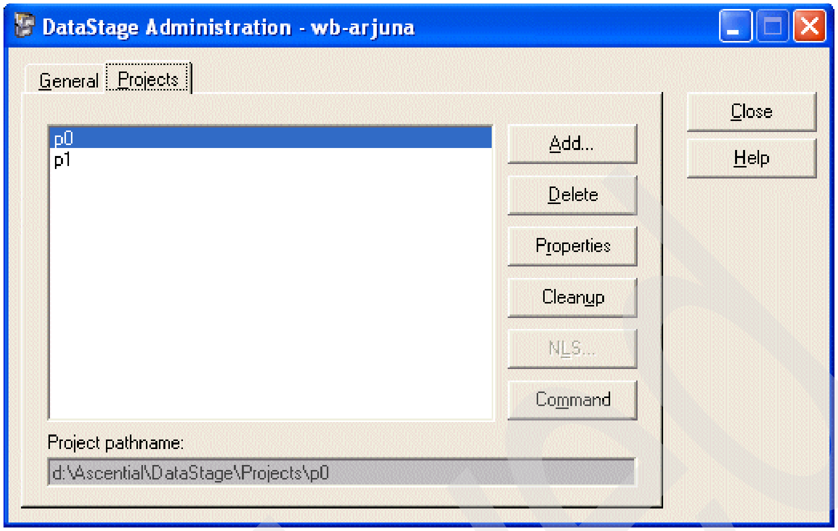


Figure A-1 DataStage Administration

2. Choose your project and click **Properties**.

   You will see the project properties dialog shown in Figure A-2. Check the Operate in MetaStage proxy mode check box. By selecting this option all jobs contained in this project will now emit process metadata when they run. You must now stop and start the DataStage server for this environment change to be effective.

> **Note:** The Operate in MetaStage proxy mode option in the DataStage Administrator will only be displayed after the Process MetaBroker is installed on the DataStage server machine.

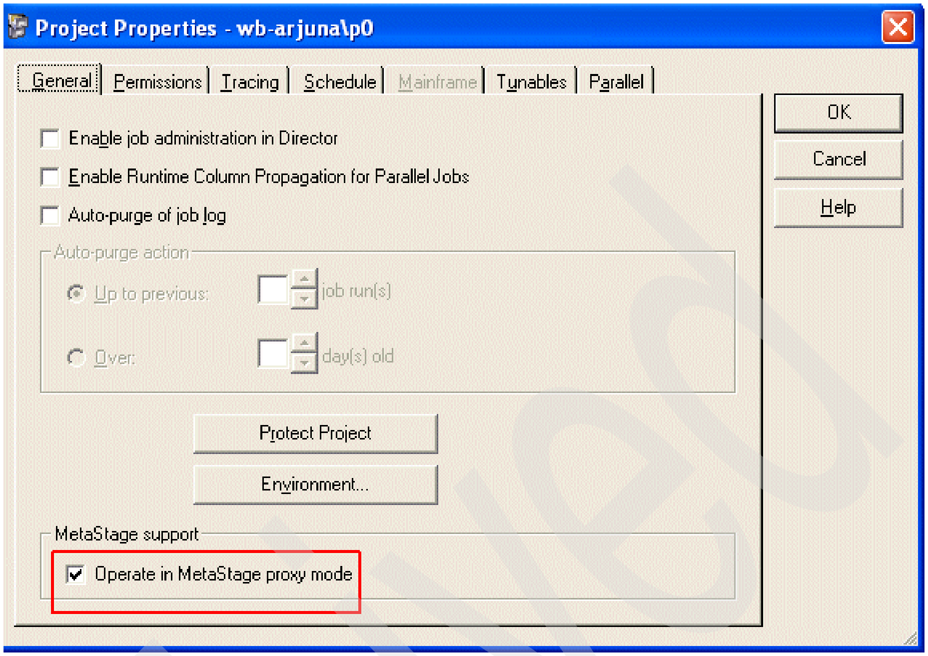


Figure A-2 Project properties dialog

3. Next, we must ensure that the Process MetaBroker configuration file has the correct startup parameters for our environment. To do this, navigate to the installation directory of the Process MetaBroker on the server machine and open the file processmb.cfg. By default on Windows, the directory is:

```
C:\Program Files\Ascential\MetaStage\Process MetaBroker\processmb.cfg
```

All default settings for the Process MetaBroker are acceptable, however, you may choose to reconfigure the variables shown in Table A-2.

Table A-2 Default Process MetaBroker variables

Variable	Default Value	Description
LogDirectory	C:\Program Files\Ascential\MetaStage\Process MetaBroker\Logs	LogDirectory is the file system subdirectory where Process MetaBroker log file(s) are located
Port	4379	Port is the TCP/IP port number on which the Process MetaBroker listens for the Activity Command Interface.

ListenerServer	<Entered During Installation>	ListenerServer is the host name or IP address of the computer where the Listener is installed.
ListenerPort	2379	ListenerPort is the TCP/IP port number of the Listener.
EventsDirectory	C:\Program Files\Ascential\MetaStage\Process MetaBroker\Events	EventsDirectory is the subdirectory in which to store incoming events.

If you change any variable in the Process MetaBroker configuration file you must stop and start the Process MetaBroker. To do this in the example Windows environment, open the Services Manager via the Windows Control Panel as shown in Figure A-3 to stop and start the Process MetaBroker so that changed variables will take effect.

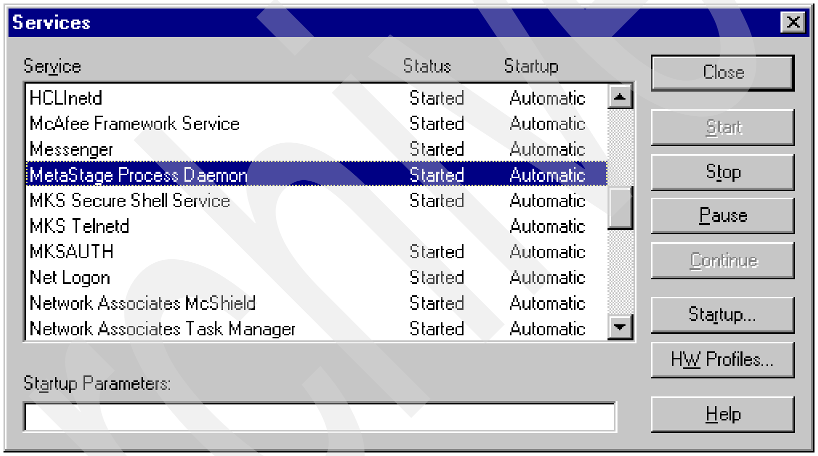


Figure A-3 Services manager under Windows

The DataStage Server and Process MetaBroker have now been configured to produce process metadata when jobs contained by the project you selected in Figure 38 run.

Now that the server machine has been configured to produce process metadata, the client must be configured to accept the process metadata that the Process MetaBroker receives from the running DataStage job.

Configure the client

There are three steps involved in configuring the client to accept process metadata from the Process MetaBroker:

1. Configure the Listener to accept DataStage job run XML files
2. Create a MetaStage Directory
3. Configure RunImport to import the DataStage job runs

The detailed steps are:

1. First, we must ensure that the Listener configuration file has the correct startup parameters for our environment on the client machine. To do this, navigate to the installation directory of the Listener on the client machine and open the file listener.cfg. By default on Windows, the directory is:

   ```
   C:\Program Files\Ascential\MetaStage\Listener\listener.cfg
   ```

   All default settings for the Listener are acceptable however you may choose to reconfigure the following variables shown in Table A-3.:

   Table A-3 Listener configuration variables

Variable	Default value	Description
LogDirectory	C:\Program Files\Ascential\MetaStage\Listener\Logs	LogDirectory is the file system subdirectory where Listener log file(s) are located
Port	2379	Port is the TCP/IP port on which the Listener listens for the Process MetaBroker.
RunsDirectory	C:\Program Files\Ascential\MetaStage\Listener\Runs	RunsDirectory is the name and path of the subdirectory where XML run files will be written.

   If you change any variable in the Listener configuration file you must stop and start the MetaStage Listener using Services Manager under Windows Control Panel so that changed variables will take effect.

2. Secondly, if you have not done so already, you must create a MetaStage Directory (Directory) so that the RunImport configuration file can reference the Directory name.

   a. Run the MetaStage Directory Administrator. You will see the Directory Administrator dialog shown in Figure A-4.

Figure A-4 Directory Administrator

b. Click **New**. You will now be asked to select a data source in which to create the MetaStage Directory shown in Figure A-5.

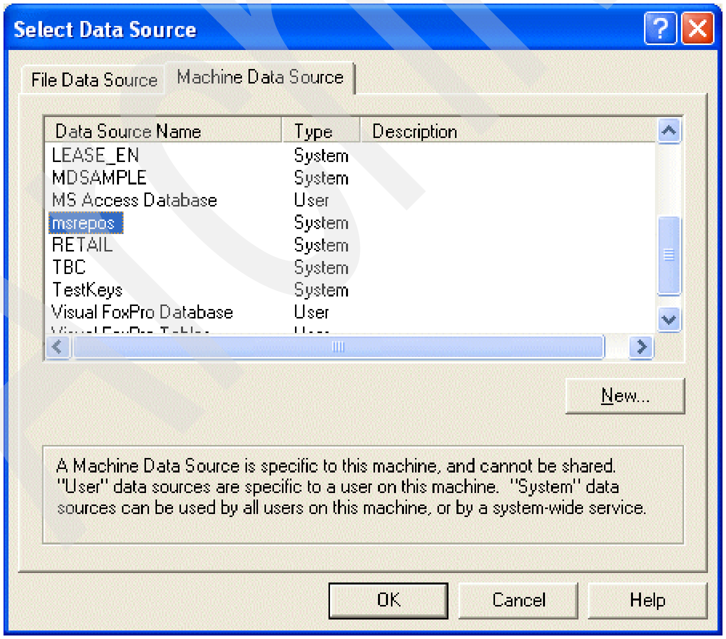


Figure A-5 Select data source

Choose the **Machine Data Source** tab and either create a new data source name (DSN) or create a new DSN. Before you click **OK**, make note of the DSN you chose or created. This name will become the name of your MetaStage directory. You will need to use this value later when configuring the RunImport. You will be asked to enter any login details and then click OK. When the Directory Administrator completes you will have an empty MetaStage Directory to work with.

3. Finally, before the client is ready to accept process metadata, we must ensure that the RunImport configuration file has the correct startup parameters on our client machine. To do this, navigate to the installation directory of the RunImport on the client machine and open the file runimport.cfg. By default on Windows, the directory is:

```
C:\Program Files\Ascential\MetaStage\Listener\runimport.cfg
```

All default settings for the RunImport are acceptable however you may choose to reconfigure the variables shown in Table A-4.

Table A-4 RunImport configuration parameters

Variable	Default value	Description
LogDirectory	C:\Program Files\Ascential\MetaStage\Listener\Logs	LogDirectory is the file system subdirectory where RunImport log file(s) are located
MetaStageDirectory	<Enter the Directory name selected or created here>	MetaStageDirectory is the name of the MetaStage Directory to import metadata into. The Run Importer imports to only one directory at a time. Change this entry whenever you need to import process metadata to a different directory.
User		User is the user name required to access the MetaStage directory as specified in Figure 43.
EncryptedPassword		Password is the password required to access the MetaStage directory. When the Run Importer runs, the password is encrypted in the configuration file.

Schema	<enter schema/owner name here>	Schema is the name of the Schema/Owner required to access the MetaStage directory. If the owner/schema of the MetaStage Directory tables is different to the value entered for User, the Schema value must specify the correct owner/schema.

At a minimum, you will change the variable MetaStageDirectory in runimport.cfg. You must enter here the value you chose or created in Figure A-5.

Now we have configured both the client and the server in our environment to be able to produce and consume process metadata. We can move on to creating the DataStage Server jobs that will produce our process metadata.

Creating DataStage Server jobs

The different steps are:

1. Configure the PMB, Listener and RunImport
   a. Install the ActivityCommand and Process MetaBroker
   b. Stop and start DataStage Server
2. Import ERwin MetaData (use lowercase server name to match DataStage server)
3. Export metadata to DataStage
4. Create DataStage jobs
5. Import DataStage jobs
6. Create the Locator table
7. Insert the locator record
8. Run the jobs
9. RunImport
10. Do data lineage (Show transformer derivation and rows processed)
11. Do process analysis

In this example we will discuss a sample DataStage job that will be used to load our consumer sales model data warehouse. To build our DataStage job we will need source and target metadata. We will get the source and target metadata from the ERwin consumer sales model shown in Figure 8-6 on page 282 and Figure 8-22 on page 296.

We will now obtain the source and target metadata. To do this we will use MetaStage as the metadata integration hub and Directory. We will first import the ERwin consumer sales model into MetaStage and then export the metadata for sources and targets to DataStage so that we can used the metadata definitions to build a DataStage job to load our data warehouse:

1. Start MetaStage Explorer.

   In the Attach dialog shown in Figure A-6, make sure that you choose the Directory name created or selected Figure A-5. In the case of the example it is *msrepos*. Click Current to open the current version of the Directory. MetaStage allows you to connect to different versions of the Directory, but since this is a new directory and no imports have been done, simply choose Current.

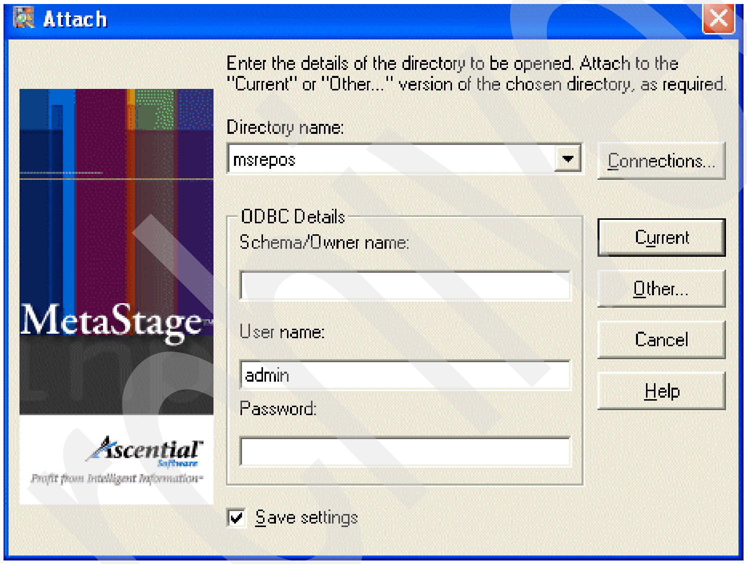


Figure A-6 MetaStage attach

2. When you open MetaStage you will see the screen shown in Figure A-7, but you will not have an ERwin Import Category yet.

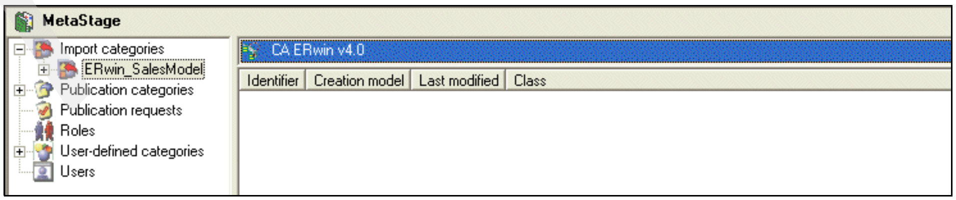


Figure A-7 ERwin import category

To create a new ERwin Import Category into which the source and target metadata will be imported, right-click **Import Categories**. A context menu will be shown. Click **New Import Category** as shown in Figure A-8 and enter a name for the Category. In our example, the category is named ERwin_SalesModel.

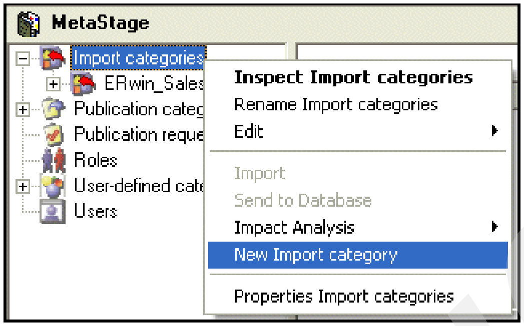


Figure A-8 New import category

3. Now we have a container for our ERwin metadata, we can import the metadata objects into MetaStage. Right-click the Import Category ERwin_SalesModel and choose **Import->New** as shown in Figure A-9.

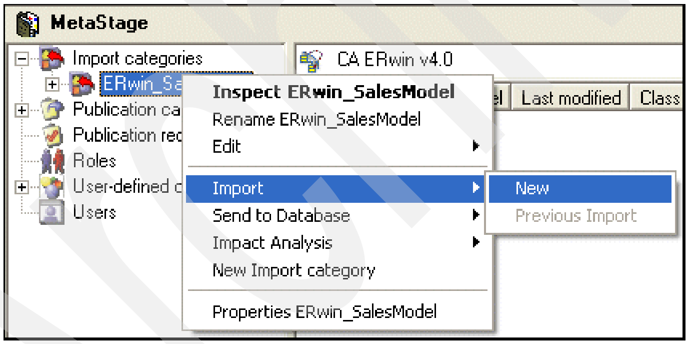


Figure A-9 New import

4. After choosing to perform a new import, you will be asked to make an **Import Selection** as shown in Figure A-10. Choose **CA ERwin v4.0** from the Source MetaBroker drop down list.

> **Note:** The CA ERwin v4.0 MetaBroker is forward compatible with ERwin v4.1.

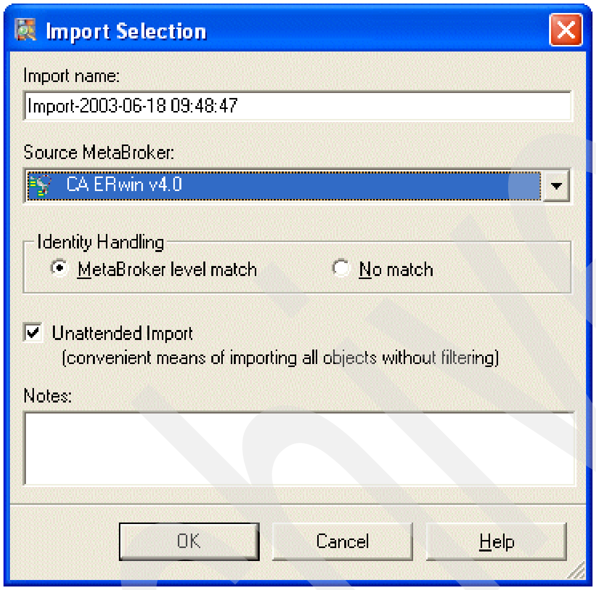


Figure A-10 Import selection

5. Click **OK**. The **ERwin MetaBroker Parameters** dialog will be shown as in Figure A-11.

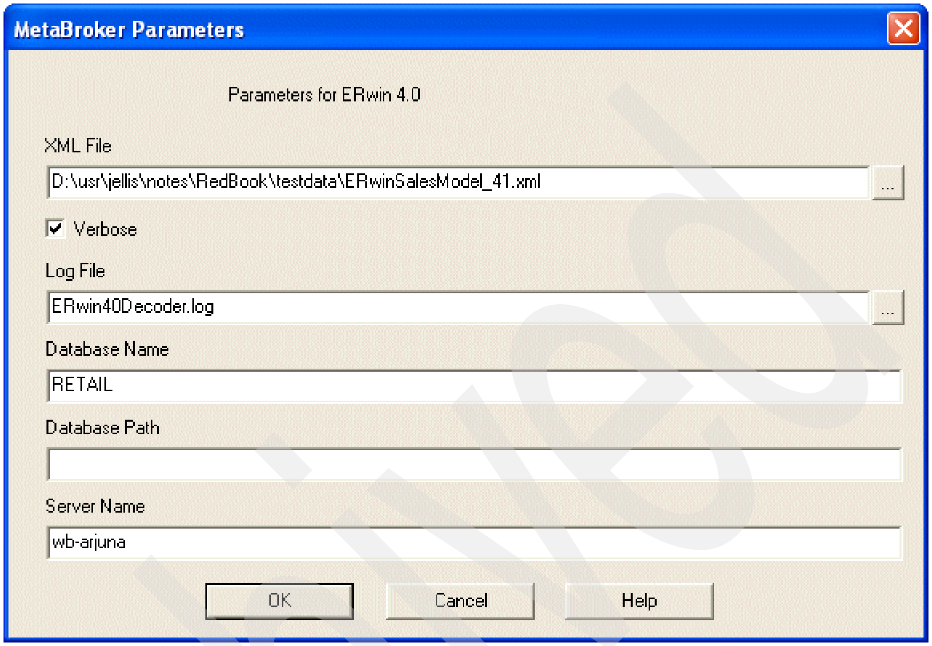


Figure A-11 ERwin import parameters

For our example it is important to make note of three parameters in Figure A-11.

– **XML File**: The input XML file is the consumer sales model shown in Figure 8-6 on page 282 and Figure 8-22 on page 296. To produce the ERwin XML file we need to export the consumer sales model using ERwin Data Modeler shown in Figure A-12.

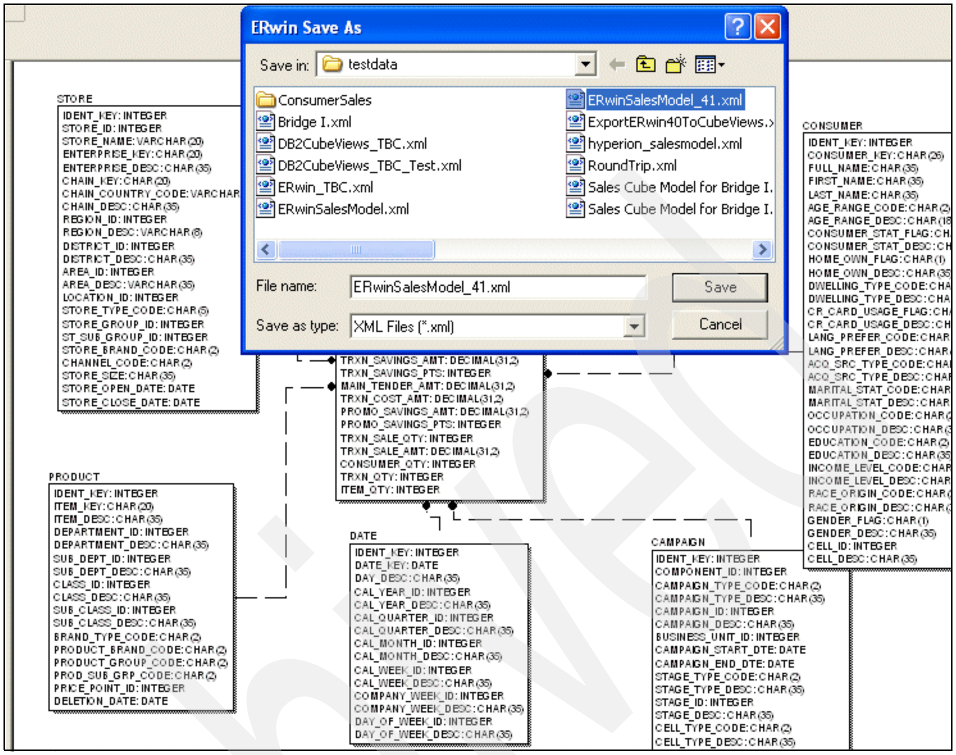


Figure A-12 ERwin saved as XML

– **Database Name:** This parameter is case sensitive. This parameter must match the name of the DB2 client connection used by the DB2 API plug-in that our DataStage job will use to connect to the DB2 database storing our star schema. In our example, a DB2 connection using the Configuration Assistant has been specified as shown in Figure A-13.

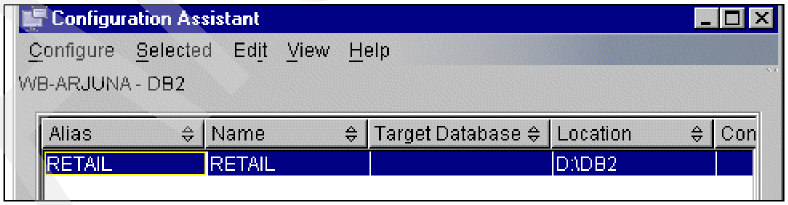


Figure A-13 DB2 Configuration Assistant

– **Server Name**: This parameter is case sensitive. This parameter must be in lower case and will be the value of the host name of the DataStage Server shown in Figure A-1 on page 641. In our example it is known that the host name is: *wb-arjuna.*

It is important to enter the host name as lower case in the **ERwin MetaBroker Parameters** dialog shown in Figure A-11 on page 651. This is because later on when we import the DataStage job design metadata, the host name will be imported as lower case. For the Locator lookup mechanism to work correctly, the case of identifiers must match.

Both parameters form part of the Locator for process metadata. The metadata Locator is a path specification used to look up design metadata when process metadata is imported into the MetaStage Directory. The path specification is used to find design metadata so that a relationship may be set to the process metadata. This relationship will then be used to perform data lineage and process analysis queries. For example, the locator for TableDefinition metadata object might be:

```
wb-arjuna->RETAIL->STAR->CONSUMER_SALES
```

Locators will be described in more detail in “Locators in MetaStage” on page 316.

6. Click **OK** on **ERwin MetaBroker Parameters** dialog shown in Figure A-11 on page 651. The ERwin MetaBroker will run and import the ERwin consumer sales XML.
7. When the ERwin MetaBroker completes the import, the ERwin data model for our consumer sales star schema will be in MetaStage. To build DataStage jobs to populate the star schema with data we now need to export the data model to DataStage. To export the ERwin design metadata to DataStage we must first place the design objects into a MetaStage User Defined category.

   Figure A-14 shows the context menu displayed after right-clicking User-defined categories in MetaStage.

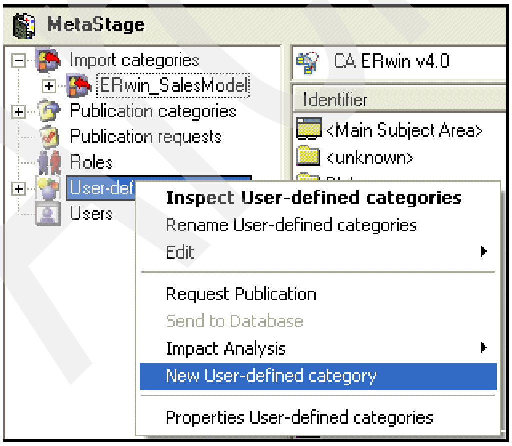


Figure A-14 New user-defined category

8. We will create a new **User-defined category** called ERwin_SalesModel to match our Import category as shown in Figure A-15.

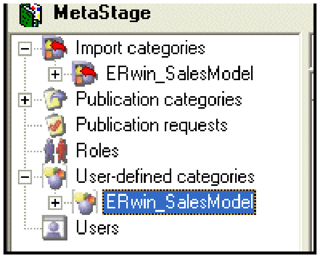


Figure A-15 ERwin Sales model User Category

9. To export the ERwin design metadata to DataStage we must copy the relevant ERwin design metadata to the User-defined category and Publish the design metadata using MetaStage. Publishing the metadata means that it is now available for users to Subscribe to that metadata. In our case we will Subscribe to the Published metadata using the DataStage MetaBroker.

   a. First, highlight all the ERwin objects contained in the ERwin_SalesModel Import category. Right-click and choose **Add Selection to Category** as shown in Figure A-16.

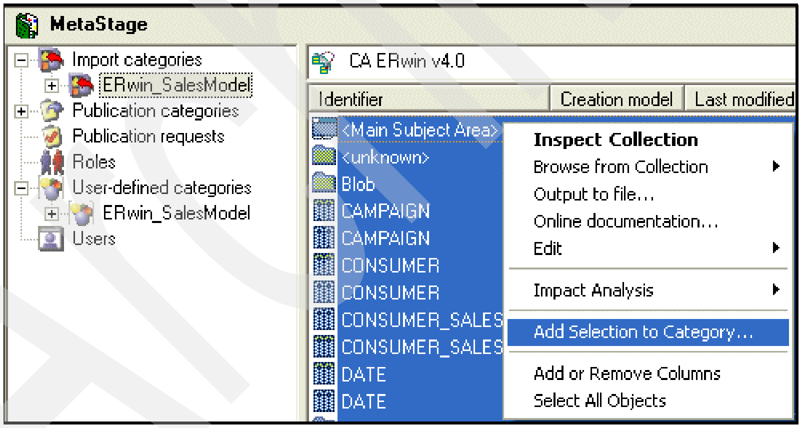


Figure A-16 Add Selection to Category

   b. We now see the **Select Category** dialog shown in Figure A-17. Select ERwin_SalesModel and click **OK**. The ERwin design metadata will now be inserted into our user-defined category.

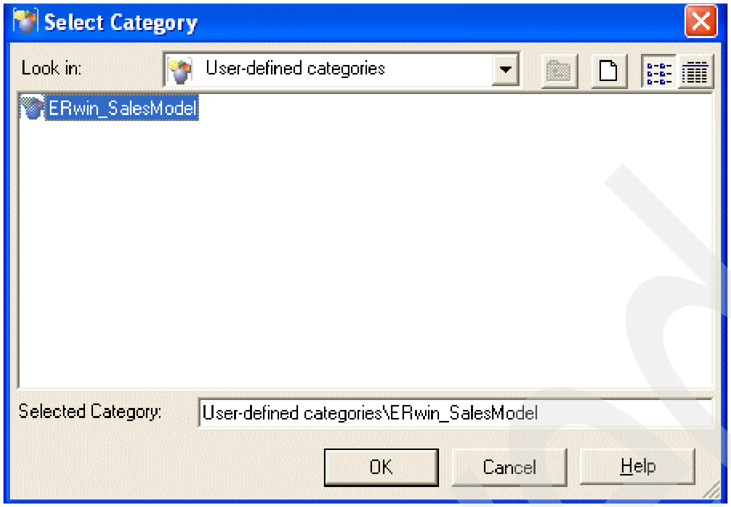


Figure A-17 Select Category

c. Now publish the ERwin design metadata. Right-click the ERwin_SalesModel User-defined category and choose **Request Publication** from the context menu shown in Figure A-18. You will be asked to enter a name for the publication. Click **Publish** to continue. The objects will now be published and read for Subscription (export) to DataStage.

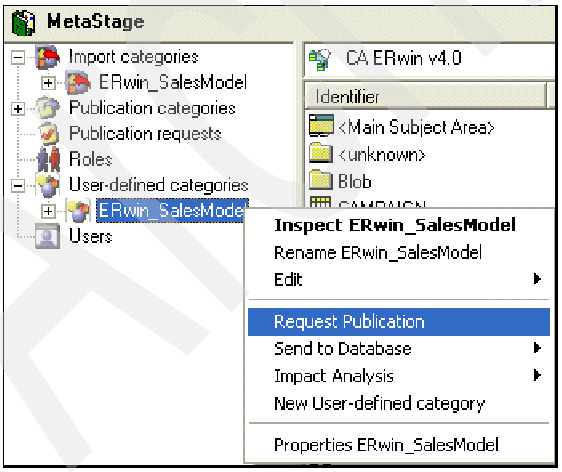


Figure A-18 Request Publication

d. To export the ERwin metadata to DataStage we will subscribe to the objects. Right-click the ERwin_SalesModel Publication category and choose **Subscribe to** on the context menu shown in Figure A-19.

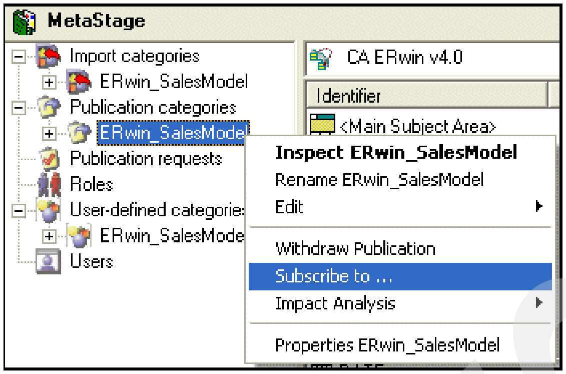


Figure A-19   MetaStage Subscribe

e. The Subscription wizard will be shown. Click **Next** and the **New Subscription** dialog will be shown as in Figure A-20. Choose **Ascential DataStage v7** and click **Next**.

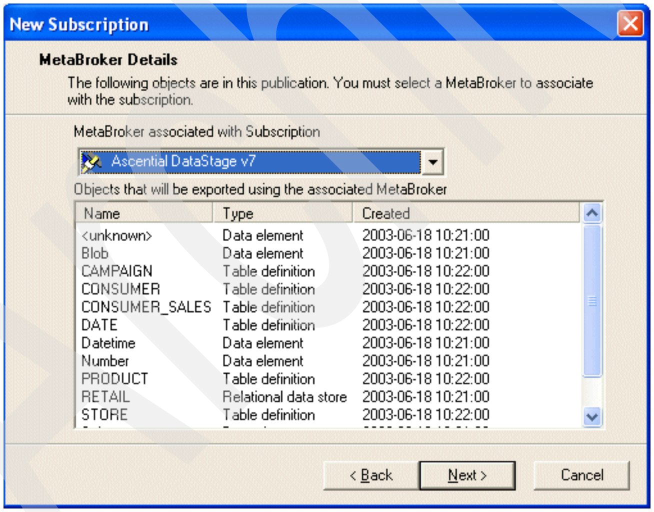


Figure A-20   New subscription

f. In the **Subscription Options** shown in Figure A-21 screen simply choose **Just run the Export**. Click **Next** and then **Finish**.

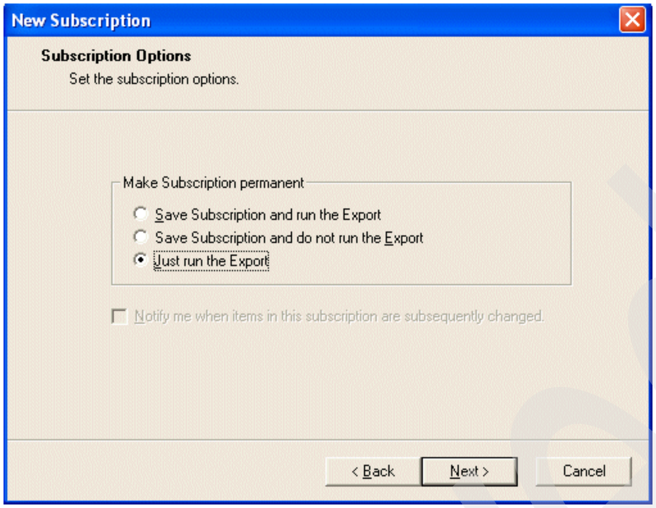


Figure A-21 Subscription options

g. We now see the DataStage MetaBroker parameters dialog as shown in Figure A-22. We will leave the default settings and simply click **OK**.

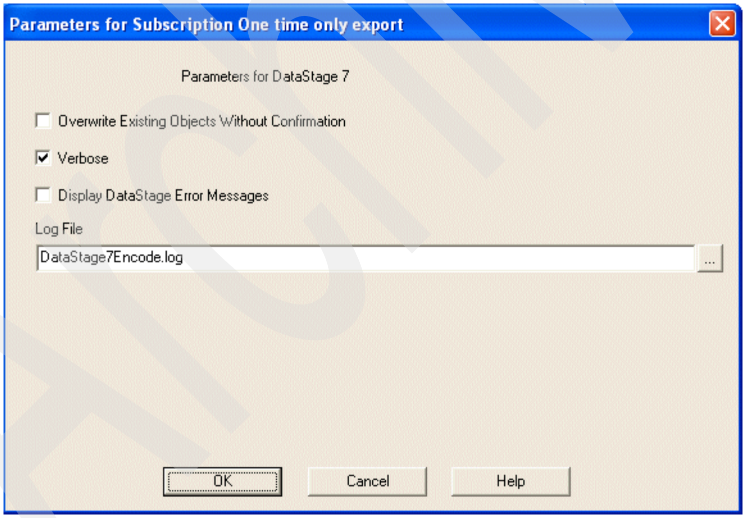


Figure A-22 DataStage MetaBroker subscription parameters

h. Next we see the DataStage client login screen shown in Figure A-23. Here is where we will select the destination DataStage project that will receive the ERwin design metadata and which host the server resides. As can be seen from Figure A-23, we chose the host *wb-arjuna* which was the

ERwin import parameter we chose in Figure A-11 on page 651. For our example we will export all the metadata definitions to the Project p0. In this example we are not required to enter any login information to DataStage. Clicking **OK** will run the export to DataStage.

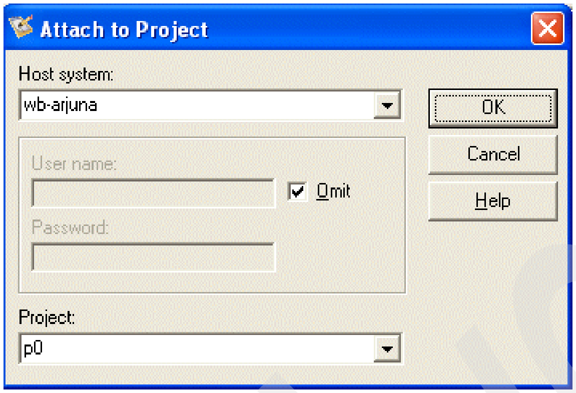


Figure A-23 DataStage client login

When the export completes we can open the DataStage Manager as shown in Figure A-24. If we navigate to the **Table Definitions** Folder and open **DataStage7_MetaBroker->STAR** we will see that our ERwin tables for the consumer sales model are now in DataStage.

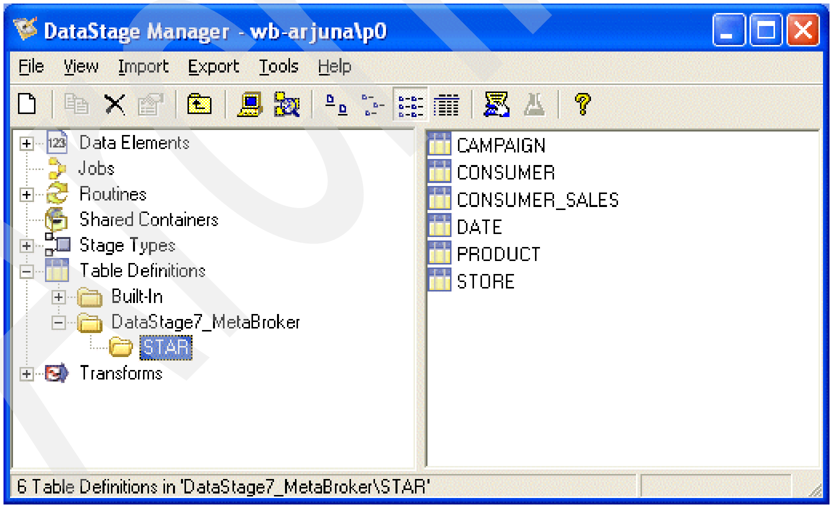


Figure A-24 DataStage Manager

10. We can now create a DataStage job to load our sales model data warehouse. For our example we have made the assumption that some other system will provide the data in a sequential delimited flat file format. It is also assumed that each respective file will format will match the ColumnDefinitions of respective TableDefinitions.

   a. Our first DataStage job will load the star schema dimension tables.

      Figure A-25 shows the DataStage job we will used to load the dimension tables. The figure shows that each dimension has a source file that will be transformed and inserted into each respective dimension table.

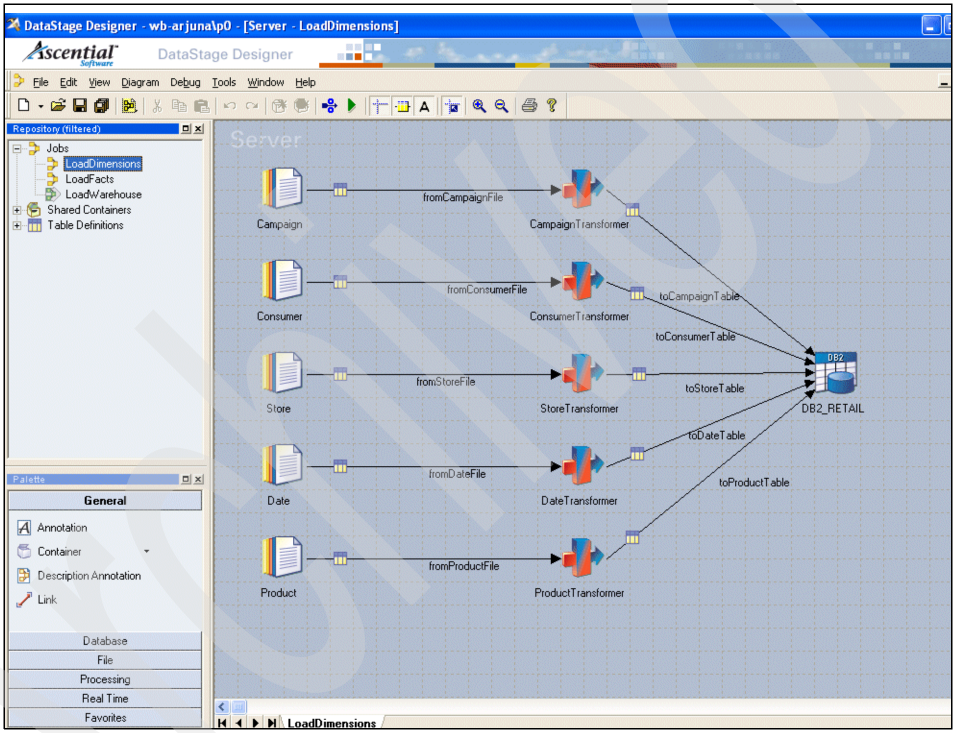


Figure A-25   DataStage Designer: load dimensions

Similarly, Figure A-26 shows the DataStage job that we will use to load the consumer sales fact table. Both DataStage jobs shown in Figure A-25 and Figure A-26 were built using the ERwin design metadata imported into DataStage.

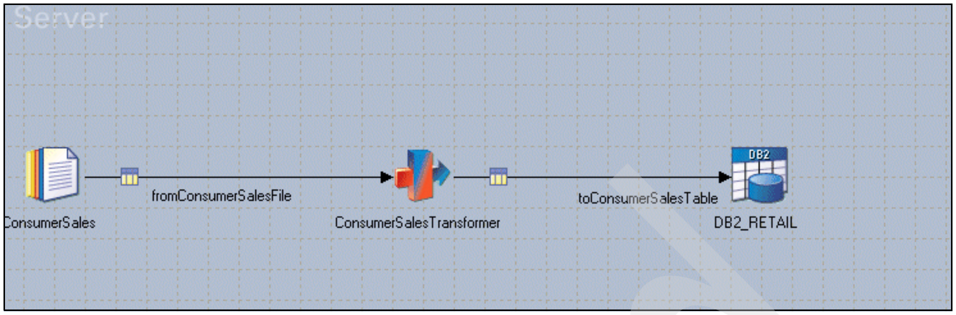


Figure A-26 DataStage Designer: load fact

Figure A-27 shows that we used the ERwin metadata exported by the DataStage MetaBroker to load the source columns for the file access. A similar operation was performed for the target side of each DataStage job.

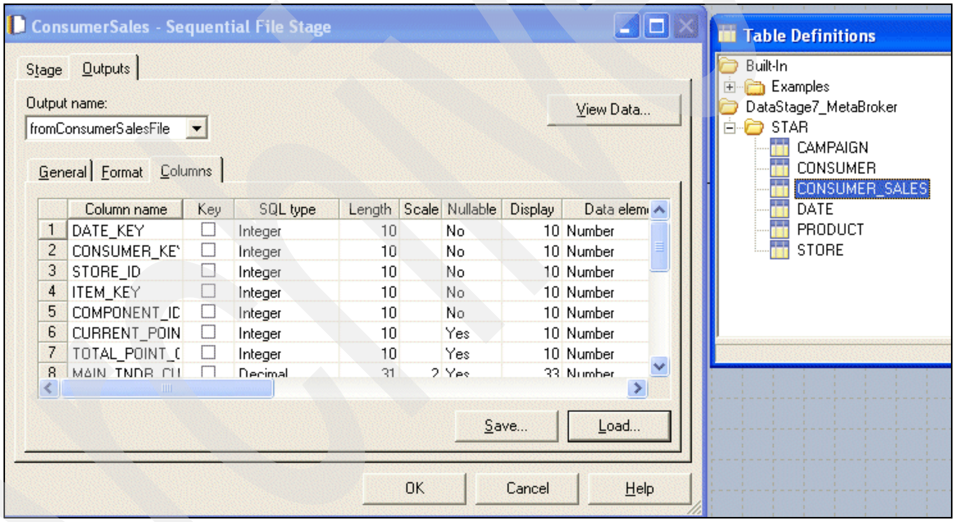


Figure A-27 DataStage Designer: load columns

We have now accomplished two major parts of capturing operational metadata:

1. We have configured the operational metadata components.
2. We have created DataStage jobs that will produce process metadata.

We will now perform the steps required to import the process metadata so it is ready for data lineage and process analysis queries as described in 8.2.4, “Performing data lineage and process analysis in MetaStage” on page 308.

B

Hybrid Analysis query performance results

Table B-1 shows the query results of each of the individual queries that were generated as a result of the DB2 OLAP Server Hybrid Analysis workloads. A single query in a Hybrid Analysis environment will cause two or more SQL queries to be generated. It is interesting to see the number of queries generated for each of the Hybrid Analysis queries and to see where the MQT comes into play.

Table B-1 lists all of the queries without the use of an MQT and when there was a drill through query type MQT available.

Table B-1 Detailed query performance results

Query workload	Query ID	Elapsed time (without MQT)	Elapsed time (with MQT)	Re-routed to MQT?
H1 Query 1	35801	0.242	0.322	N
	35802	16.699	2.344	Y
H1 Query 2	35803	0.128	0.144	N
	35804	0.137	0.139	N
	35805	0.137	0.131	N

Query workload	Query ID	Elapsed time (without MQT)	Elapsed time (with MQT)	Re-routed to MQT?
	35806	0.139	0.166	N
	35807	0.140	0.133	N
	35808	0.133	0.151	N
	35809	2.913	4.166	Y
H2 Query 1a	35821	0.116	0.13	N
	35822	161.839	5.139	Y
H2 Query 1b	35823	0.131	0.12	N
	35824	11.811	0.365	Y
	35825	0.474	0.342	Y
H2_Query 2a	35826	0.127	0.138	N
	35827	0.139	0.141	N
	35828	13.971	1.36	Y
H2_Query 2b	35829	0.119	0.127	N
	35830	0.147	0.131	N
	35831	0.131	0.131	N
	35832	0.124	0.133	N
	35833	0.132	0.132	N
	35834	0.126	0.128	N
	35835	0.314	1.466	Y
H3_Query 1a	35836	0.107	0.13	N
	35837	0.404	0.344	Y
H3_Query 1b	35838	0.125	0.116	N
	35839	0.423	0.206	Y
	35840	0.353	0.204	Y
H3_Query 2	35841	0.089	0.098	N
	35842	0.114	0.107	N

Query workload	Query ID	Elapsed time (without MQT)	Elapsed time (with MQT)	Re-routed to MQT?
	35843	0.125	0.128	N
	35844	0.094	0.119	N
	35845	0.105	0.137	N
	35846	0.090	0.106	N
	35847	0.121	0.142	N
	35848	0.081	0.109	N
	35849	0.132	0.143	N
	35850	0.098	0.101	N
	35851	0.125	0.154	N
	35852	0.095	0.114	N
	35853	0.106	0.132	N
	35854	0.097	0.125	N
	35855	0.106	0.085	N
	35856	0.091	0.095	N
	35857	0.110	0.112	N
	35858	0.313	1.442	Y
H4_Query 1a	35859	0.140	0.126	N
	35860	161.016	1.017	Y
H4_Query 1b	35861	0.122	0.147	N
	35862	11.23	0.342	Y
	35863	0.418	0.378	Y
H4_Query 1c	35864	0.088	0.111	N
	35865	0.369	0.251	Y
	35866	0.358	0.238	Y
H4_Query 2a	35867	0.102	0.106	N
	35868	0.096	0.109	N

Query workload	Query ID	Elapsed time (without MQT)	Elapsed time (with MQT)	Re-routed to MQT?
	35869	0.149	0.161	N
	35870	0.090	0.103	N
	35871	0.141	0.156	N
	35872	0.086	0.107	N
	35873	0.098	0.124	N
	35874	0.098	0.102	N
	35875	0.116	0.095	N
	35876	13.783	1.44	Y
H4_Query 2b	35877	0.086	0.098	N
	35878	0.092	0.111	N
	35879	0.137	0.154	N
	35880	0.115	0.113	N
	35881	0.135	0.156	N
	35882	0.101	0.115	N
	35883	0.138	0.15	N
	35884	0.142	0.108	N
	35885	0.152	0.13	N
	35886	0.111	0.122	N
	35887	0.156	0.131	N
	35888	0.113	0.092	N
	35889	0.143	0.141	N
	35890	0.138	0.094	N
	35891	0.116	0.096	N
	35892	0.112	0.114	N
	35893	0.103	0.108	N
	35894	0.334	1.429	Y

Query workload	Query ID	Elapsed time (without MQT)	Elapsed time (with MQT)	Re-routed to MQT?
H5_Query 1a	35895	0.150	0.138	N
	35896	162.485	1.885	Y
H5_Query 1b	35897	0.143	0.134	N
	35898	11.428	0.379	Y
	35899	0.452	0.426	Y
H5_Query 1c	35900	0.116	0.107	N
	35901	0.371	0.262	Y
	35902	0.407	0.241	Y
H5_Query 1d	35903	0.112	0.114	N
	35904	0.361	0.208	Y
	35905	0.368	0.198	Y
H5_Query 2a	35906	0.116	0.125	N
	35907	0.112	0.109	N
	35908	0.164	0.185	N
	35909	0.107	0.117	N
	35910	0.171	0.17	N
	35911	0.097	0.121	N
	35912	0.148	0.15	N
	35913	0.167	0.161	N
	35914	0.120	0.111	N
	35915	0.155	0.145	N
	35916	0.114	0.117	N
	35917	0.168	0.152	N
	35918	0.120	0.112	N
	35919	0.155	0.148	N
	35920	0.115	0.126	N

Query workload	Query ID	Elapsed time (without MQT)	Elapsed time (with MQT)	Re-routed to MQT?
	35921	0.119	0.112	N
	35922	2.427	1.453	Y
H5_Query 1b	35923	0.113	0.147	N
	35924	0.126	0.126	N
	35925	0.153	0.163	N
	35926	0.114	0.114	N
	35927	0.149	0.167	N
	35928	0.152	0.117	N
	35929	0.103	0.129	N
	35930	0.104	0.14	N
	35931	0.115	0.133	N
	35932	12.468	1.416	Y
H5_Query 1c	35933	0.110	0.119	N
	35934	0.114	0.131	N
	35935	0.148	0.16	N
	35936	0.121	0.119	N
	35937	0.165	0.163	N
	35938	0.152	0.141	N
	35939	0.144	0.152	N
	35940	0.106	0.118	N
	35941	0.149	0.163	N
	35942	0.116	0.116	N
	35943	0.169	0.168	N
	35944	0.105	0.118	N
	35945	0.148	0.145	N
	35946	0.329	1.536	Y

Query workload	Query ID	Elapsed time (without MQT)	Elapsed time (with MQT)	Re-routed to MQT?
H5_Query 1d	35947	0.117	0.115	N
	35948	0.112	0.092	N
	35949	0.166	0.172	N
	35950	0.117	0.117	N
	35951	0.162	0.168	N
	35952	0.451	1.433	Y

C

FAQs, diagnostics, and tracing

This appendix discusses FAQs, diagnostics, and tracing.

Setup questions

- **Q:** What version of DB2?

  A: DB2UDB v8.1 FP2+

- **Q:** What edition of DB2?

  A: ESE, DB2 Warehouse Edition.

  DB2 Warehouse Enterprise Edition includes DB2 UDB V8.1 Enterprise Server Edition (ESE).

  DB2 Warehouse Standard Edition includes DB2 UDB V8.1 Workgroup Server Unlimited Edition

Metadata questions

- **Q:** How do I model a dimension that resides in the fact table?

  A: Usually, this scenario may occur in a non-star schema design. For example, Product may have only one column and data for product is merged with the fact table instead of having a separate table for product and referencing it as a foreign key in the fact table. In this case, you can model a dimension based on just the columns(s) that you need from the fact table.

  Note that this does not mean that you can build a DB2 Cube Views model with any non star schema design. It is strongly recommended to start with a star schema for which the product is better optimized and designed.

- **Q:** How do I control the order of members in a level?

  A: The metadata does not capture member ordering information. But front-end tools can provide the ability to order members. The tool can then implement the ordering by adding an ORDER BY clause to SQL it generates, or sort the results before displaying cube data to end-users.

- **Q:** What joins should be included in the facts object?

  A: Inner joins are strongly recommended for Optmization Advisor. Other types of joins (left outer, right outer, and so on) will include NULL for missing values, which does not augur well with the Optimization Advisor.

OLAP Center

- **Q:** OLAP Center won't start, or can't connect to DB2.

  A: This may happen, though very rare. Increase the value of the application heap size (APPLHEAPSZ) parameter in the database configuration file.

- How do I … ?

  – **Q:** Delete a set of objects?

    A: You cannot perform this action at present. You will have to delete objects one by one.

  – **Q:** Delete all objects?

    A: Use db2mdapiclient, which provides the ability to delete all the objects at a time.

  – **Q:** Create complex measures?

    A: Use the Aggregration Script Builder.

- **Q:** Is there a Tutorial?

  A: No, but there is online help and info pops.

Tracing

- *Server side tracing*

  A configuration file called db2md_config.xml (found in the \sqllib directory) is used to set error logging and runtime tracing. By modifying the contents of the configuration file, an administrator can specify the level of tracing, the severity of errors to log, the buffer size (in bytes) to use when logging, and the filenames of logs. This type of tracing is explained in detail in Appendix D, (see Error logging and tracing)

Example: C-1 db2md_config.xml

```
<olap:config xmlns:olap="http://www.ibm.com/olap"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xsi:schemaLocation="http://www.ibm.com/olap db2md_config.xsd">
      <log>
         <trace level="none" logFile="mdtrace.log" bufferSize="0"/>
         <error level="medium" logFile="mderror.log" bufferSize="0"/>
      </log>
</olap:config>
```

- *Client side tracing*

  Turn on OLAP Center trace by running it from the command line with the -logfile option: “db2mdoc -logfile <path\filename> “

Example: C-2 Olap center trace command

1. db2mdoc -logfile mylog.txt =>This puts the trace file in sqllib\tools by default
2. db2mdoc -logfile C:\logs\mylog.txt => tracefile location: C:\logs

D

DB2 Cube Views stored procedure API

DB2 Cube Views offers Multidimensional Services, which is an application programming Interface (API) that provides programmatic access to metadata.

This appendix contains a brief introduction to the following topics in the DB2 Cube Views API:

- API architecture overview
- Purposes and functionality of the API
- Stored procedure interface
- API operations
- Error handling and tracing
- db2mdapiclient utility

API architecture overview

The API is composed of a single stored procedure registered to a DB2 database.

Figure D-1 shows a diagrammatic representation of the DB2 Cube Views API and how metadata is exchanged through the API.

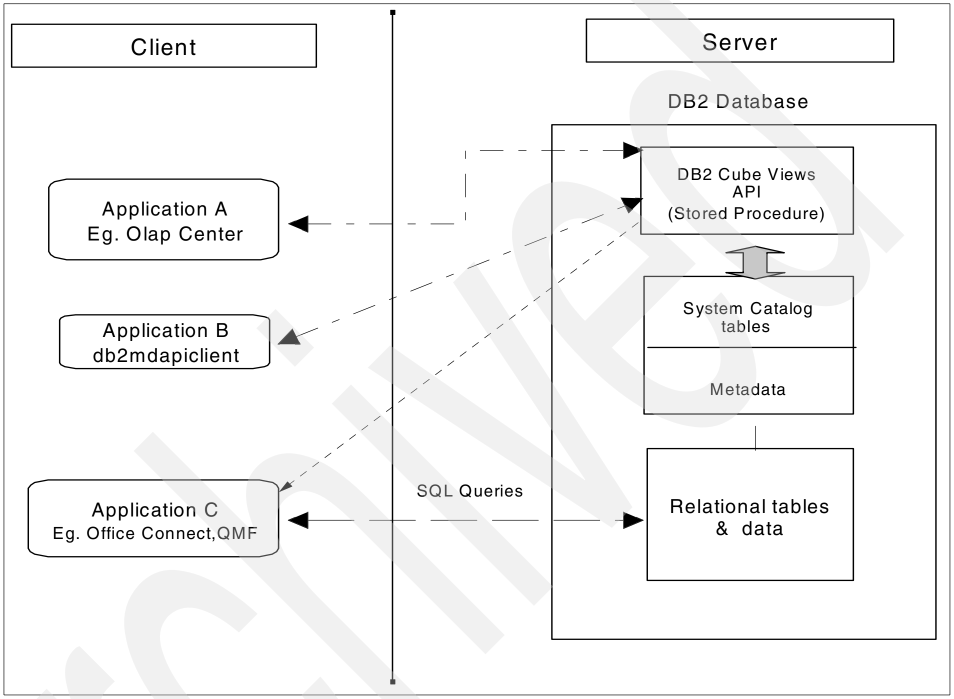


Figure D-1 API architecture

As described by the diagram, the API is an interface that allows access to metadata.

This stored procedure accepts input and output parameters in which you can express complex metadata and metadata operations. Applications A & B push metadata (by creating and manipulating) to the DB2 catalog and also pull metadata from the DB2 Catalog. Application C just pulls metadata from DB2 Cube Views. Flow of metadata in all these cases is through the stored procedure interface.

Purposes and functionality of the API

- The DB2 Cube Views API offers three types of metadata operations:
  - Retrieve the DB2 Cube Views metadata (only) or describe operation. It does not provide data access.

    The API for DB2 Cube Views provides access to the metadata stored in the system catalog tables of a DB2 database. The type of metadata operation in this case is '**retrieval**'
  - Metadata management.

    Using the API, applications can interact with metadata using DB2 Cube Views metadata objects without having to interact with relational tables and joins. Applications using the API can create and modify metadata objects that model multidimensional and OLAP constructs in a data warehouse. The type of metadata operation in this case is '**modification**' (which includes create, alter, import, rename and drop)
  - Metadata rules enforcement.

    The API possesses a 'validate' functionality which checks a metadata objects conformance to DB2 Cube Views object rules In this case, the metadata operation of type *administration*.

    The following are examples of the types of check performed as part of the validate operation:

    - Completeness of metadata object information
    - Referential integrity between metadata objects
    - Existence of referenced relational table columns and views
    - Correctness of SQL Expression stored in metadata objects (i.e., attributes and measures)
    - Specialization/subset relationships between various objects (i.e., cube model and cube, dimension and cube dimension, facts and cube facts, hierarchy and cube hierarchy)
- Each metadata operation has input and output parameters.
  - There are two kinds of input parameters: request and metadata
  - There are two kinds of output parameters: response and metadata
- The API delivers information that can be used to form SQL queries.
- The API can be invoked using any of DB2's programming interfaces - CLI, JDBC, ODBC, Embedded SQL and makes extensive use of XML.

See Figure D-2 for a diagrammatic representation of the metadata operations and its parameters

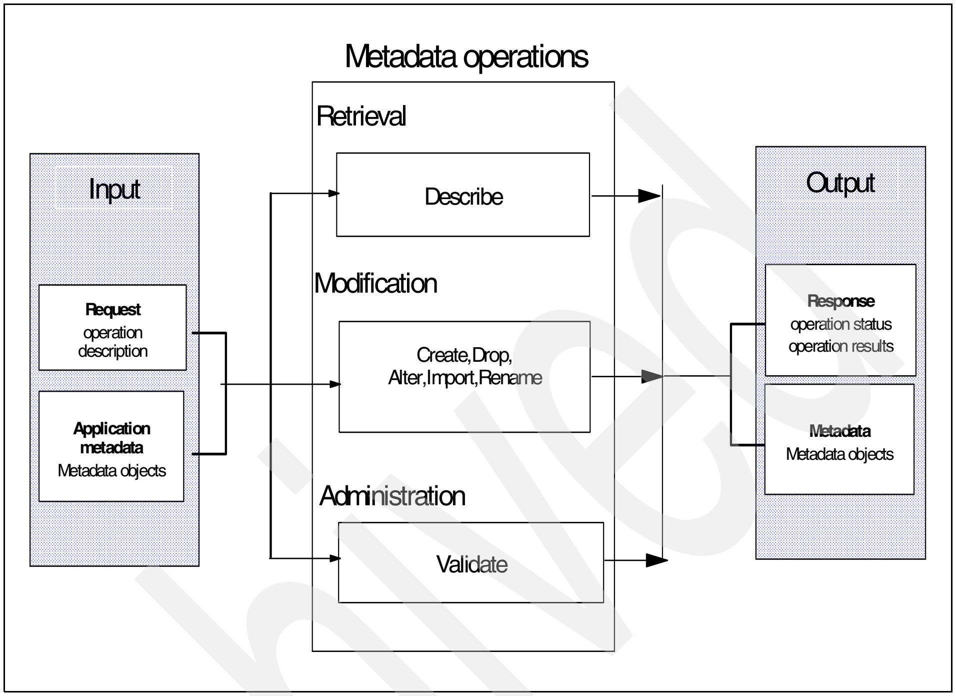


Figure D-2 Metadata operations and its parameters

The stored procedure interface

- The DB2 Cube Views stored procedure is called **md_message** and it processes parameters expressed in the DB2 Cube Views parameter format. The procedure extracts operation and metadata information from the input parameters, and then it performs the requested metadata operations. The procedure generates output parameters that contain the execution status (success or failure) of requested operations, in addition to containing metadata information, depending on the operation.
- The DB2 Cube Views stored procedure is implemented as a DB2 stored procedure. It can be used by any application that makes use of any of DB2's programming interfaces. The name of the stored procedure is case insensitive, while the name and contents of the stored procedure's parameters are case sensitive.
- The parameter format defines the standard by which metadata operations and objects are represented and exchanged between BI applications and DB2 Cube Views. The parameter format uses XML to represent DB2 Cube Views metadata operations and objects. This XML format directly maps to the metadata object model by capturing associations between objects. It also delivers relational database information in an OLAP context such that SQL statements for OLAP data can be formed

The syntax of md_message and a prototype are shown in Figure D-3.

```
Syntax:
call md_message (request, metadata, response)

Prototype:
md_message (request IN CLOB(1M),
            metadata INOUT CLOB(1M),
            response OUT CLOB(1M))

Remarks:
- Request and response parameters are mandatory
- Metadata parameter is optional
- XML parameters exchanged using Character Large Object (CLOB) structures
- "CALL" SQL statement invokes the stored procedure
```

Figure D-3 Syntax of md_message stored procedure

Example

The following example of a retrieval operation (called describe) shows how it is structured. In this example, portions of the XML structures are excluded, but are represented with an ellipsis (...).

Example: D-1   Structure of a retrieval operation (describe)

```
EXEC SQL CALL DB2INFO.MD_MESSAGE(:request,:metadata,:response);     Embedded
                                                                    SQL
<olap:request xmlns:olap="http://www.ibm.com/olap" ... >            statement
<describe objectType="cube" recurse="no">
<restriction>                                                       Request
<predicate property="schema" operator="=" value ="myschema"/>
</restriction>
</describe>
</olap:request>

<olap:metadata xmlns:olap="http://www.ibm.com/olap" ... />          Metadata


<olap:response xmlns:olap="http://www.ibm.com/olap" ... >
<describe>
<status id="0" text="Operation completed successfully.              Response
"type="informational"/>
</describe>
</olap:response>


<olap:metadata xmlns:olap="http://www.ibm.com/olap" ... >           Metadata
<cube name="cube1" schema="myschema" ... > ... </cube>
...
<cube name="cubeN" schema="myschema" ... > ... </cube>"
</olap:metadata>
```

Here, the metadata parameter is empty on input, but populated on output.

Error logging and tracing

Tracing and error logging is set by modifying the contents of a configuration file (called **db2md_config.xml**) on the server where DB2 resides. By modifying the contents of this configuration file, an administrator can specify the level of tracing, the severity of errors to log, the buffer size (in bytes) to use when logging, and the filenames of logs. The content structure of db2md_config.xml is defined by the db2md_config.xsd XML schema file. Example D-2 provides an example of the contents of the configuration file.

Example: D-2 db2md_config.xml

```
<olap:config xmlns:olap="http://www.ibm.com/olap"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xsi:schemaLocation="http://www.ibm.com/olap db2md_config.xsd">
<log>
<trace level="none" logFile="db2mdtrace.log" bufferSize="0"/>
<error level="medium" logFile="db2mderror.log" bufferSize="0"/>
</log>
</olap:config>
```

Tracing

The API supports three priorities of tracing (low, medium and high). Using the configuration file, an administrator can set the level of tracing to log to file. Runtime tracing is turned off by default, and the trace file name is **db2mdtrace.log**.

When tracing is turned on, with the level set to a value other than none, errors that occur in the API might be recorded in both the error log and the trace log, depending on the level and severity setting for these logs.

Error logging

The API distinguishes between three severities of errors (low, medium and high). The default severity setting is medium, and the error log file name is **db2mderror.log**. When an error occurs while reading the configuration file, this error is logged in a file named **db2mdapi.log**.

When the API is configured to high or medium error logging, and a high or medium error occurs, the API generates a callstack beginning at the point where the error occurs in the API. This callstack is similar to a medium-level trace, but the data is sent to the error log instead of the trace log.

Note: The default location of the trace and error logs is ..\sqllib\db2 directory on Windows (../sqllib/db2dump directory on AIX).

db2mdapiclient utility

This utility is a thin wrapper to the DB2 Cube Views stored procedure interface. The utility is provided as sample source code to show how to code an application against the API.

Location

The source code, db2mdapiclient.cpp is located in \SQLLIB\samples\olap\client\ directory on Windows (/home/db2inst1/sqllib/samples/olap/client on AIX).

Tasks

You can use the **`db2mdapiclient`** utility to perform any of the operations that are supported by the DB2 Cube Views stored procedure, md_message() as described in Table D-1

Table D-1 db2mdapiclient utility tasks

Task	Operation
Export metadata (to an XML file)	describe
Import metadata (from an XML file)	create, import
Change metadata	alter, rename
Delete metadata	drop
Verify validity of existing metadata	validate

Usage

The db2mdapiclient utility uses files to hold the XML that is passed to and received from the md_message() stored procedure (see Figure D-4).

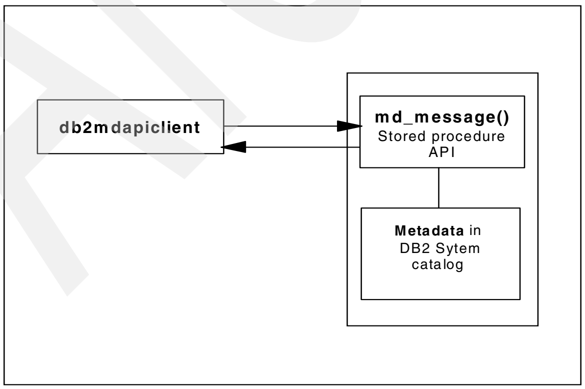


Figure D-4 How the db2mdapiclient utility works

For example, while importing metadata in to the DB2 Cube Views metadata catalog, the db2mdapiclient utility typically uses an XML file that was produced by a DB2 Cube Views bridge or an XML file that was exported from the OLAP Center. For exporting, the db2mdapiclient utility produces an XML file that a DB2 Cube Views bridge utility can use to add metadata to a database or OLAP tool.

To see a list of parameters for the db2mdapiclient command, you can enter 'db2mdapiclient' at a command line (on both Windows and AIX) as shown in Figure D-5.

```
USAGE:
        db2mdapiclient [OPTIONS]
        Options can be specified in any order
REQUIRED OPTIONS:
        -d or --database              database name
        -i or --inputoperation        input operation file name
        -o or --outputoperation      output operation file name
OTHER OPTIONS:
        -u or --userid                userid for database
        -p or --password              password for database
        -m or --inputmetadata         input metadata file name. Required for
                                     operations such as "create" & "import"
        -n or --outputmetadata        output metadata file name. If output
                                     metadata file is not specified & there
                                     is metadata returned by stored procedure
                                     then output metadata will be written to
                                     outputmetadata.xml
        -a or --parameterbuffersize   parameter buffer size, defaults to
                                     1000000 bytes
        -b or --metadatabuffersize    metadata buffer size, defaults to
                                     1000000 bytes
        -v or --verbose               print extra information while
                                     processing
        -h or --help                  this usage text
```

Figure D-5 Usage of the db2mdapiclient utility

The typical syntax for the db2mdapiclient command is:

```
db2mdapiclient -d dbname -u user -p password -i request.xml -o response.xml
-m inputmetadata.xml -n outputmetadata.xml
```

Examples

To further illustrate the usage, we will look an import /export and validate scenarios, using the db2mdapiclient utility.

1. **Import**

   To import DB2 Cube Views metadata for a database (say SAMPLE), change to the ..\SQLLIB\samples\olap\xml\input directory (on Windows) and enter the command shown in Example D-3

   Example: D-3 Using db2mdapiclient utility to import metadata

   ```
   db2mdapiclient -d SAMPLE -u db2admin -p mypasswrd -i create.xml -o
   myresponse.xml -m MDSampleMetadata.xml
   ```

   Here, create.xml, MDSampleMetadata.xml and myresponse.xml are the values of the arguments of the md_message(request, metadata, response) stored procedure. That is, create.xml provides the request, MDSampleMetadata.xml is the metadata(input) and myresponse.xml is the response(status).

2. **Export**

   To export DB2 Cube Views metadata for a database (say SAMPLE), change to the ..\SQLLIB\samples\olap\xml\input directory (on Windows) and enter the command shown in Example D-4

   Example: D-4 Using db2mdapiclient utility to export metadata

   ```
   db2mdapiclient -d SAMPLE -u db2admin -p mypasswrd -i describe.xml -o
   MyOutput.xml -n SampleOut.xml
   ```

   Here, describe.xml is the request, SampleOut.xml is the exported metadata (output) and MyOutput.xml contains the response (status).

3. **Validate**

   To validate DB2 Cube Views metadata for a database (say SAMPLE), change to the ..\SQLLIB\samples\olap\xml\input directory (on Windows) and enter the command shown in Example D-5

   Example: D-5 Using db2mdapiclient utility to validate metadata

   ```
   db2mdapiclient -d SAMPLE -u db2admin -p mypasswrd -i validate.xml -o
   validateout.xml -v
   ```

   The default structure of the validate.xml allows validation all metadata objects in the DB2 catalog for optimization (which is, checking for conformance to base rules, cube completeness rules and optimization rules).

If you wish to check validity of a particular cube model for completeness, then the structure of the validate.xml will have to be changed (shown in Example D-6) to accommodate the restrictions.

Example: D-6 Validate.xml

```
<olap:request xmlns:olap="http://www.ibm.com/olap"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" version="8.1.2.1.0">
<validate objectType="cubeModel" mode="cubeModel completeness">
<restriction>
<predicate property="name" operator="=" value="Sales Cube Model"/>
</restriction>
</validate>
</olap:request>
```

E

The case study: retail datamart

This appendix provides a high-level overview of the retail star schema datamart used in this redbook.

We used one AIX machine (GREENLAND) and two Windows 2000 machines (HELIUM and GALLIUM) but the results provided in this redbook are the results of our testing on non-optimized configurations.

The cube model

The cube model used is depicted in Figure E-1.

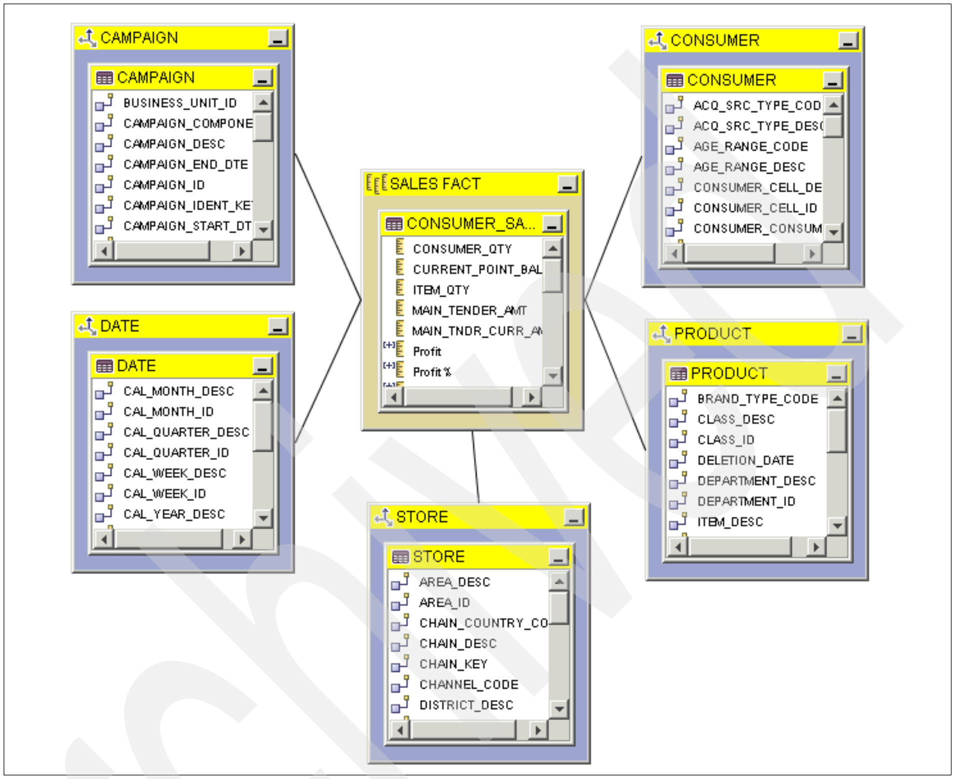


Figure E-1 The cube model

The number of rows in the fact table CONSUMER_SALES is: 4415575

The number of rows in the dimension tables is:

- CAMPAIGN : 17
- CONSUMER : 8749
- PRODUCT : 10357
- STORE: 100
- DATE : 1366

The cube

The cube used was a subset of the cube model and is depicted in Figure E-2.

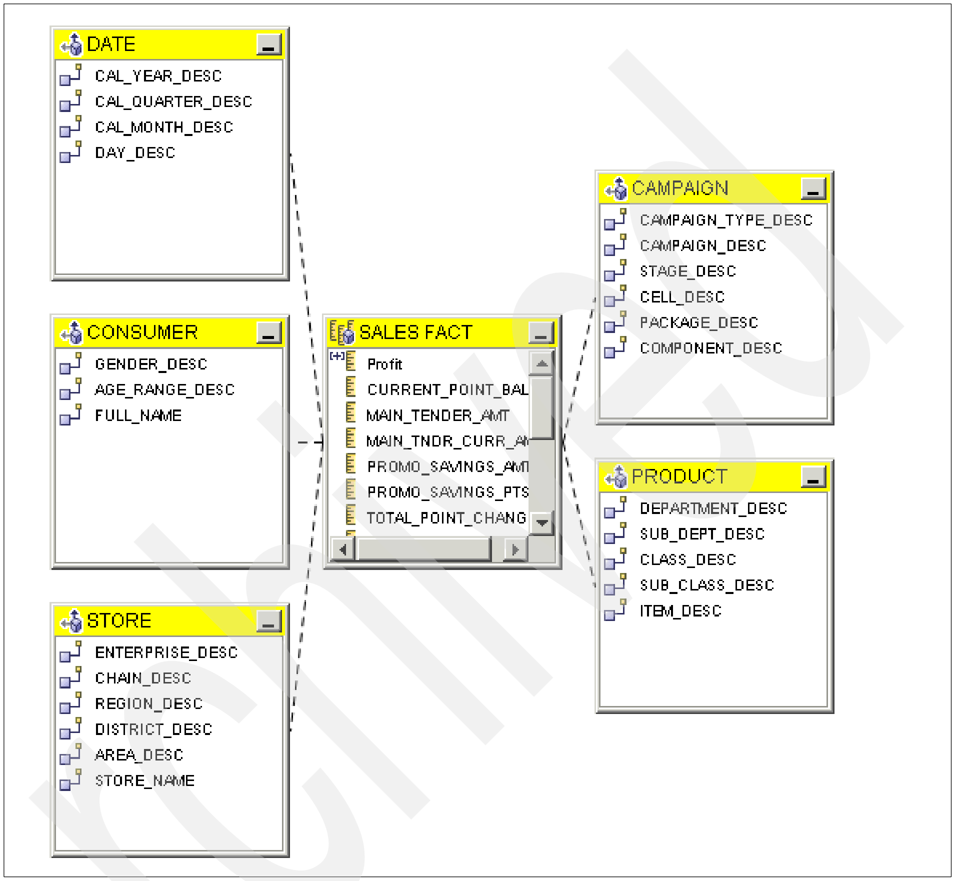


Figure E-2 The cube

Tables in the star schema

The dimension tables are:

- Dimension name: Consumer as described in

  Table name: STAR.CONSUMER

  The fields are described in .Table E-1.

Table E-1 Consumer dimension

Attribute Name	Business Name
ACQ_SRC_TYPE_CODE	Acquire Source Type Code
ACQ_SRC_TYPE_DESC	Acquire Source Type Description
AGE_RANGE_CODE	Age Range Code
AGE_RANGE_DESC	Age Range Description
CONSUMER_CELL_DESC	Consumer Cell Description
CONSUMER_CELL_ID	Consumer Cell Identifier
CONSUMER_CONSUMER_KEY	Consumer Key
CONSUMER_IDENT_KEY	Consumer Identifier Key
CONSUMER_STAT_DESC	Consumer Status Description
CONSUMER_STAT_FLAG	Consumer Status Flag
CR_CARD_USAGE_DESC	Credit Card Usage Description
CR_CARD_USAGE_FLAG	Credit Card Usage Flag
DWELLING_TYPE_CODE	Dwelling Type Code
DWELLING_TYPE_DESC	Dwelling Type Description
EDUCATION_CODE	Education Code
EDUCATION_DESC	Education Description
FIRST_NAME	First Name
FULL_NAME	Full Name
GENDER_FLAG	Gender Flag
GENDER_DESC	Gender Description
HOME_OWN_DESC	Own Home Description
HOME_OWN_FLAG	Own Home Flag
INCOME_LEVEL_CODE	Income Level Code
INCOME_LEVEL_DESC	Income Level Description
LANG_PREFER_CODE	Language Preference Code
LANG_PREFER_DESC	Language Preference Description

Attribute Name	Business Name
LAST_NAME	Last Name
MARITAL_STAT_CODE	Marital Status Code
MARITAL_STAT_DESC	Marital Status Description
OCCUPATION_CODE	Occupation Code
OCCUPATION_DESC	Occupation Description
RACE_ORIGIN_CODE	Race Origin Code
RACE_ORIGIN_DESC	Race Origin Description

- Dimension Name: Campaign

  Table Name: STAR.CAMPAIGN

  The fields are described in Table E-2.

Table E-2 Campaign dimension

Attribute Name	Business Name
BUSINESS_UNIT_ID	Business Unit Identifier
CAMPAIGN_COMPONENT_ID	Component Identifier
CAMPAIGN_DESC	Campaign Description
CAMPAIGN_END_DATE	Campaign End Date
CAMPAIGN_ID	Campaign Identifier
CAMPAIGN_IDENT_KEY	Campaign Identifier Key
CAMPAIGN_START_DATE	Campaign Start Date
CAMPAIGN_TYPE_CODE	Campaign Type Code
CAMPAIGN_TYPE_DESC	Campaign Type Description
CELL_DESC	Cell Description
CELL_ID	Cell Identifier
CELL_TYPE_CODE	Cell Type Code
CELL_TYPE_DESC	Cell Type Description
CMPNT_TYPE_CODE	Component Type Code
CMPNT_TYPE_DESC	Component Type Description

Attribute Name	Business Name
COMM_CHANNEL_CODE	Communication Channel Code
COMPONENT_DESC	Component Description
FACE_VALUE_AMOUNT	Face Value Amount
FACE_VALUE_POINTS	Face Value Points
PACKAGE_CMPNT_KEY	Package Component Key
PACKAGE_DESC	Package Description
PACKAGE_ID	Package Identifier
STAGE_DESC	Stage Description
STAGE_ID	Stage Identifier
STAGE_TYPE_CODE	Stage Type Code
STAGE_TYPE_DESC	Stage Type Description

- Dimension name: Product

  Table name: STAR.PRODUCT

  The fields are described in Table E-3.

Table E-3 Product dimension

Attribute Name	Business Name
BRAND_TYPE_CODE	Brand Type Code
CLASS_DESC	Class Description
CLASS_ID	Class Identifier
DELETION_DATE	Deletion Date
DEPARTMENT_DESC	Department Description
DEPARTMENT_ID	Department Identifier
ITEM_DESC	Item Description
PRICE_POINT_ID	Price Point Identifier
PROD_SUB_GRP_CODE	Product Sub group Code
PRODUCT_BRAND_CODE	Product Brand Code
PRODUCT_GROUP_CODE	Product Group Code

Attribute Name	Business Name
PRODUCT_IDENT_KEY	Product Identifier Key
PRODUCT_ITEM_KEY	Product Item Key
SUB_CLASS_DESC	Sub class Description
SUB_CLASS_ID	Sub Class Identifier
SUB_DEPT_DESC	Sub Department Description
SUB_DEPT_ID	Sub Department Identifier

- Dimension name: Date

  Table name: STAR.DATE

  The fields are described in Table E-4.

Table E-4 Date dimension

Attribute Name	Business Name
CAL_MONTH_DESC	Calender Month Description
CAL_MONTH_ID	Calender Month Identifier
CAL_QUARTER_DESC	Calender Quarter Description
CAL_QUARTER_ID	Calender Quarter Identifier
CAL_WEEK_DESC	Calender Week Description
CAL_WEEK_ID	Calender Week Identifier
CAL_YEAR_DESC	Calender Year Description
CAL_YEAR_ID	Calender Year Identifier
COMPANY_WEEK_DESC	Company Week Description
COMPANY_WEEK_ID	Company Week Identifier
DATE_DATE_KEY	Date Key
DAY_OF_WEEK_DESC	Day of Week Description
DAY_OF_WEEK_ID	Day of Week Identifier
IDENT_KEY	Identifier Key

- Dimension name: Store

  Table name: STAR.STORE

  The fields are described in Table E-5.

Table E-5 Store Dimension

Attribute Name	Business Name
AREA_DESC	Area Description
AREA_ID	Area Identifier
CHAIN_COUNTRY_CODE	Channel Country Code
CHAIN_DESC	Chain Description
CHAIN_KEY	Chain Key
CHANNEL_CODE	Channel Code
DISTRICT_DESC	District Description
DISTRICT_ID	District Identifier
ENTERPRISE_DESC	Enterprise Description
ENTERPRISE_KEY	Enterprise Key
LOCATION_ID	Location Identifier
REGION_DESC	Region Description
ST_SUB_GROUP_ID	Store sub group Identifier
STORE_BRAND_CODE	Store Brand Code
STORE_CLOSE_DATE	Store Close Date
STORE_GROUP_ID	Store Group Identifier
STORE_IDENT_KEY	Store Identifier Key
STORE_NAME	Store Name
STORE_OPEN_DATE	Store Open Date
STORE_SIZE	Store Size
STORE_STORE_ID	Store Identifier
STORE_TYPE_CODE	Store Type Code

The Fact table is:

Table name: STAR.CONSUMER_SALES

The fields are described in Table E-6.

Table E-6 Fact Table

Attribute Name	Business Name
COMPONENT_ID	Component Identifier
CONSUMER_KEY	Consumer Key
DATE_KEY	Date Key
ITEM_KEY	Item Key
STORE_ID	Store Identifier
CONSUMER_QTY	Consumer Quantity
CURRENT_POINT_BAL	Current Point Balance
ITEM_QTY	Item Quantity
MAIN_TENDER_AMT	Main Tender Amount
MAIN_TNDR_CURR_AMT	Main Tender Current Amount
Profit	Profit
Profit%	Profit Percentage
Promo%	Promotion Percentage
PROMO_SAVINGS_AMT	Promotion Savings Amount
PROMO_SAVINGS_PTS	Promotion Saving in Points
TOTAL_POINT_CHANGE	Total Point Change
TRXN_COST_AMT	Transaction Cost Amount
TRXN_QTY	Transaction Quantity
TRXN_SALE_AMT	Transaction Sale Amount
TRXN_SALE_QTY	Transaction Sale Quantity
TRXN_SAVINGS_AMT	Transaction Savings Amount
TRXN_SAVINGS_PTS	Transaction Savings in Points

MQT

Example E-1 provides the script to create and refresh an MQT that has been built for report queries.

Example: E-1 Script to create/refresh summary tables

```
--* Cube model schema: STAR
--* Cube model name:   STAR Base  Model
--* Diskspace limit:   0
--* Time limit:        0
--* Sampling:          Yes
--* Drill down:        No
-- * Report:            Yes
-- * Drill through:     No
-- * Extract:           No
-- * Refresh type:      Refresh deferred
- * Tablespace name:   USERSPACE1
-- * Indexspace name:   USERSPACE1
==========================================================================
DROP TABLE DB2INFO.MQT0000000001T01;
UPDATE COMMAND OPTIONS USING c OFF;
CREATE TABLE DB2INFO.MQT0000000001T01 AS
(SELECT
SUM(T1."CONSUMER_QTY") AS "CONSUMER_QTY",
SUM(T1."CURRENT_POINT_BAL") AS "CURRENT_POINT_BAL",
SUM(T1."ITEM_QTY") AS "ITEM_QTY",
SUM(T1."MAIN_TENDER_AMT") AS "MAIN_TENDER_AMT",
SUM(T1."MAIN_TNDR_CURR_AMT") AS "MAIN_TNDR_CURR_AMT",
SUM(T1."TRXN_SALE_AMT" - T1."TRXN_COST_AMT") AS "Profit",
SUM(T1."PROMO_SAVINGS_AMT") AS "PROMO_SAVINGS_AMT",
SUM(T1."TOTAL_POINT_CHANGE") AS "TOTAL_POINT_CHANGE",
SUM(T1."TRXN_COST_AMT") AS "TRXN_COST_AMT",
SUM(T1."TRXN_QTY") AS "TRXN_QTY",
SUM(T1."TRXN_SALE_AMT") AS "TRXN_SALE_AMT",
SUM(T1."TRXN_SALE_QTY") AS "TRXN_SALE_QTY",
SUM(T1."TRXN_SAVINGS_AMT") AS "TRXN_SAVINGS_AMT",
SUM(T1."TRXN_SAVINGS_PTS") AS "TRXN_SAVINGS_PTS",
(SUM(T1."TRXN_SALE_AMT" - T1."TRXN_COST_AMT")* 100.0)/ SUM(T1."TRXN_SALE_AMT")
AS "Profit%",
( SUM(T1."PROMO_SAVINGS_AMT") * 100.0)/ SUM(T1."TRXN_SALE_AMT") AS "Promo%",
T2."CAMPAIGN_TYPE_DESC" AS "CAMPAIGN_TYPE_DESC",
T2."CAMPAIGN_DESC" AS "CAMPAIGN_DESC",
T2."STAGE_DESC" AS "STAGE_DESC",
T2."CELL_DESC" AS "CELL_DESC",
T2."PACKAGE_DESC" AS "PACKAGE_DESC",
T2."COMPONENT_DESC" AS "COMPONENT_DESC",
T3."GENDER_DESC" AS "GENDER_DESC",
GROUPING(T3."GENDER_DESC") AS "GRP_GENDER_DESC",
```

```
T3."AGE_RANGE_DESC" AS "AGE_RANGE_DESC",
GROUPING(T3."AGE_RANGE_DESC") AS "GRP_AGE_RANGE_DESC",
T4."CAL_YEAR_DESC" AS "CAL_YEAR_DESC",
T5."DEPARTMENT_DESC" AS "DEPARTMENT_DESC",
T5."SUB_DEPT_DESC" AS "SUB_DEPT_DESC",
T6."ENTERPRISE_DESC" AS "ENTERPRISE_DESC",
T6."CHAIN_DESC" AS "CHAIN_DESC",
T6."REGION_DESC" AS "REGION_DESC",
GROUPING(T6."REGION_DESC") AS "GRP_REGION_DESC",
T6."DISTRICT_DESC" AS "DISTRICT_DESC",
T6."AREA_DESC" AS "AREA_DESC",
GROUPING(T6."AREA_DESC") AS "GRP_AREA_DESC"

FROM
"STAR"."CONSUMER_SALES" AS T1,
"STAR"."CAMPAIGN" AS T2,
"STAR"."CONSUMER" AS T3,
"STAR"."DATE" AS T4,
"STAR"."PRODUCT" AS T5,
"STAR"."STORE" AS T6

WHERE
T1."COMPONENT_ID"=T2."IDENT_KEY" AND
T1."CONSUMER_KEY"=T3."IDENT_KEY" AND
T1."DATE_KEY"=T4."IDENT_KEY" AND
T1."ITEM_KEY"=T5."IDENT_KEY" AND
T1."STORE_ID"=T6."IDENT_KEY"

GROUP BY GROUPING SETS (
(
T2."CAMPAIGN_TYPE_DESC",
T2."CAMPAIGN_DESC",
T2."STAGE_DESC",
T2."CELL_DESC",
T2."PACKAGE_DESC",
T2."COMPONENT_DESC",
T3."GENDER_DESC",
T3."AGE_RANGE_DESC",
T4."CAL_YEAR_DESC",
T5."DEPARTMENT_DESC",
T5."SUB_DEPT_DESC",
T6."ENTERPRISE_DESC",
T6."CHAIN_DESC",
T6."REGION_DESC"
),
(
T3."GENDER_DESC",
T5."DEPARTMENT_DESC",
T5."SUB_DEPT_DESC",
```

```
T6."ENTERPRISE_DESC",
T6."CHAIN_DESC",
T6."REGION_DESC",
T6."DISTRICT_DESC",
T6."AREA_DESC"
)))

DATA INITIALLY DEFERRED
REFRESH DEFERRED
ENABLE QUERY OPTIMIZATION
MAINTAINED BY SYSTEM
IN "ITSOTSFACT"
INDEX IN "ITSOTSFACT"
NOT LOGGED INITIALLY;
```

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see "How to get IBM Redbooks" on page 699. Note that some of the documents referenced here may be available in softcopy only.

- *Getting Started on Integrating Your Information*, SG24-6892
- *Integrating XML with DB2 XML Extender and DB2 Text Extender*, SG24-6130
- *DB2 UDB's High Function Business Intelligence in e-business*, SG24-6546
- *DB2 OLAP Server, Theory and Practices,* SG24-6138-00
- *DB2 OLAP Server V8.1: Using Advanced Functions*, SG24-6599
- *Data Modeling Techniques for Data Warehousing*, SG24-2238-00
- *Up and Running with DB2 UDB ESE Partitioning for Performance in an e-Business Intelligence World,* SG24-6917-00

Other publications

These publications are also relevant as further information sources:

- *IBM DB2 Cube Views Setup and User's Guide,* SC18-7298
- *Bridge for Integration Server User's Guide*, SC18-7300
- *IBM DB2 Cube Views Business Modeling Scenarios Manual,* SC18-7803
- *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling (Second Edition) by Ralph Kimball, April 2002*, ISBN 0471-200247

Online resources

These Web sites and URLs are also relevant as further information sources:

- IBM DB2 Cube Views Homepage:

  `http://www.ibm.com/software/data/db2/db2md/`

- IBM Software Homepage:

  `http://www.software.ibm.com/`

- IBM Information Management Homepage:

  `http://www.software.ibm.com/data/`

- "The OLAP Aware Database". An article written by Michael L. Gonzales and Gary Robinson in the quarter 2, 2003 edition of the DB2 Magazine:

  `http://www.db2mag.com/db_area/archives/2003/q2/gonzales.shtml`

- "How to Build a Metadata Bridge for DB2 UDB Cube Views". An article written by John Poelman and made available on the DB2 Developer Domain Library:

  `http://www7b.software.ibm.com/dmdd/library/techarticle/0305poelman/0305poelman.html`

- "Relational extensions for OLAP". An article written by N.Colossi, W.Malloy and B.Reinwald in IBM Systems Journal - Vol. 41, No. 4, 2002:

  `http://researchweb.watson.ibm.com/journal/sj/414/colossi.pdf`

- DB2 Cubes Views Web services: available from the alphaWorks IBM Web site:

  `http://www.alphaworks.ibm.com`

- DB2 OLAP Server Homepage:

  `http://www-3.ibm.com/software/data/db2/db2olap/`

- QMF for Windows Homepage:

  `http://www.ibm.com/qmf`

- BusinessObjects Homepage:

  `http://www.businessobjects.com`

- Ascential Homepage:

  `www.ascential.com`

- Cognos Homepage:

  `http://www.cognos.com`

- Meta Integration Model Bridge Homepage:

  `www.metaintegration.net/Products/MIMB`

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Index

Numerics

3NF 65

A

activity
  class 310
ActivityCommand 313, 647
actual parameter set 311
ad-hoc reports 26, 86
administrative effort 33, 44, 49, 52, 56
administrative tasks 81
advanced measures 119
aggregate-awareness 49
aggregates 10–11, 35
  pre-built 43
  size 26
  space 12
  time 12
aggregation 241, 450
  rules 38
aggregation functions 69, 129, 438
aggregation scripts 99, 120, 424, 438
aislog.xml 431
AllFusion ERwin v4.x Data Modeler 339
alphaWorks 56
alternate hierarchies 513
analytics 84–85
API xxxi, 32, 54, 222, 280, 673
architecture 81
Ascential xxxi
  DB2 Cube Views MetaBroker 271
  Process MetaBroker 640
Ascential DataStage 315
  clients 640
  project 639
  Server 640
Ascential MetaStage 640
  Administrator 640
  Directory 279, 283, 296
  Explorer 640
  Integration Hub 279
  Listener 640
  RunImport 640
atomic 67
atomic semantic 279
attribute 70, 75
  type 38
    associated 38
    descriptive 38
attribute relationship 76–77
  associated 76, 436
  creating 122
  descriptive 76, 436
  MQTs 151
attributes 22, 36, 63, 77
  associated 479
  calculated 38
automation 314
AVG 38, 69, 129

B

back-end tools 29, 31–32, 63
balanced hierarchy 73
base rules 140
base tables 127
  LOAD (ALLOW READ ACCESS) 133
  LOAD INSERT 133
batch window 476
BI 278
bind
  cubes 227
  data 233
bridge 9, 13, 83, 223
bridgeless ROLAP tools 54
business
  analysts 7
  area 4
  metrics 4–5
  subject 80
  subject area 85
Business Intelligence xxxi, 4, 8
  tools 9
Business Intelligence Portal 545
Business Intelligence Tools 82
business names 86, 254, 344, 362, 364, 378, 389, 397, 435, 453, 495

Business Objects xxxi
business requirements 85
BusinessObjects 281
bridge
API mode 546, 559, 567
application mode 546, 559
batch file arguments 568
batch file sequences 568
batch mode 546, 560, 568
benefits 583
business names 553
data types conversion 558
DB2 database connection 560
deployment 581
import 562
mapping 548
complex measures 553–554
descriptive attribute relationship 552
measure description 554
metadata exchange modes 559
optimization tips 582
SQL expression 554
universe creation 547, 583
universe mapping 550
warning messages 569
DB2 Explain 572
examples 573
MQTs 572, 583
queries
access plan graph 572
query
query rewrite 575
timeron 579
query performance 571
SQL statement 571
BusinessObjects Broadcast Agent 545
BusinessObjects Designer 546, 562
BusinessObjects Developer Suite 546
BusinessObjects Enterprise 544
BusinessObjects InfoView 545
BusinessObjects Universal Metadata Bridge 543, 546

C

C++ 280
calculated attributes 103, 120
calculated measures 98, 436
calculation
time 476
cardinality 53, 76, 105, 448, 459
CAST 120
category 277
CDM 365
challenges 11
changes 413, 449
chart 239
Chart Wizard 239
child 65, 73–75
child objects 301
children-nodes 629
classes 310, 393
Cognos xxxi, 486
bridge 487
alternate hierarchies 515
attribute 499
attribute relationship 499
calculated measures 513
cube 498
cube model 498
default template 498
dimension 499
hierarchy 498
implementation 492
join 499
mappings 498
measure 498
measure formatting 501
multiple hierarchies 498
ragged hierarchy 498
scenarios 524
DB2 Cube Views
connect 493
DB2 Dimensional Metadata Wizard 491, 541
drill through optimization 502
drill through report 497–498
enhancements
category counts 497
dimensions renaming 497
hierarchies 497
levels renaming 497
measures 497
time-based partitions 497
Transformer 497
event detection 486
filtered dimensions 505
Impromptu 488–489
catalogs 489

query definitions 495
reports 489, 510
Impromptu Catalog 495
log 497
model enhancements 497
MQTs
performance results 539
naming conventions 498
performance 541
portal 486
PowerCube building 496
PowerCubes 495
PowerPlay 485, 490
Transformation Server 490
Transformer models 495
PowerPlay Cube Groups 501
Query 485
reporting and analysis 485
scorecard 486
security 501
timerons 540
Transformation Server 489
Transformation Server Model 487
Transformer model 490, 495, 503
Cognos Analytic Applications 487
Cognos Business Intelligence 484
Cognos Impromptu 487
Cognos Metrics Manager 486, 491
Cognos NoticeCast 486
Cognos Query 491
Cognos Reporting and Analysis 486
Cognos ReportNet 541
Cognos Visualizer 486, 491
column definitions 316
column names
uniqueness 435
columns 32–33
COM 280
Common Warehouse Metamodel 393
complex aggregations 130
Computer Associates 412
Computer Associates AllFusion ERwin Data Modeler 338
Computer Associates ERWin 338
configuration file 313–314, 321, 679
Connection Manager 227, 230
consistency 65
constraints 140
correlated subqueries 214
COUNT 38, 45, 129, 134, 151, 201–202
COUNT_BIG 134, 201–202
Created_Resource 317
creating
an empty cube model 93
dimension objects 100
the facts objects 94
cross-tool 33–34
impact analysis 295
cross-tool data 33
cross-tool data lineage 31
cross-tool impact analysis 31
CUBE 10
cube 41, 80
bottom 26
creating 114
derived 80
dimensions 80
export 429
hierarchies 80
modeling 63
multiple 434
top levels 26
cube dimensions 80
cube fact 80
Cube Import wizard 227, 230
cube model 9, 14, 36, 64, 67, 79, 81, 289
building 85
creation
import 85
methodology 85
enhancement 118, 122, 520
date 520
measure formatting 522
export 424
from scratch 85, 92
implementation 75
import 85
optimization 84
planning 84
preparing the relational database 85
Quick Start wizard 85
refresh 523
storage 79
updates 118
validation 118
cube model and cube
export 424
cube model completeness 140

Cube wizard 80, 114
cubes 23, 228, 240
  number 150
  slice and dice queries 619
  sorts 252
  XML descriptions 619
customized report 234, 244
CWM 393
  ObjectModel package 397
  RDB package 397
CWM file 398, 403
CWM XMI 412
CWM XMI file 396, 402

D
data
  change 295
  history 308
  models 281
  movement 331
  schemas 64
  study 85
data lineage 31–33, 278, 281, 309, 314, 316, 647
  path 309, 323
  query 309–310
    run 321
data lineage analysis 639
data load 420
  performance 426
  time 451
data model 4
  definitions 278
data model enrichment 33–34
data modelers 278, 308
data modeling 278
data modeling tool 23, 31, 33–34, 295
data retrieval 227
  performance 64
data sampling 16
data sources 87, 295, 310, 312, 322, 493
data transformation 399
data warehouse 65, 126, 276
  update 312
Data Warehouse Center
  save the model 402
database
  catalogs 63
  connect 86, 88, 160
  schema 65
  set up 86
datamart 65, 126, 309
DataStage 309
  Administrator 640
  DataStore 309
  design 314
  Designer 640, 660
  DSN 309
  File 309
  import category 314
  job 309–311, 643
    run 318, 639
    status 310
  job design 314
  job run 324
  login 316
  Manager 640, 658
  operation 639
  project 314, 640
  runs 324
  Server 313, 320
  TableDefinition 309
  view 311
DataStage Director 318
DataStage project
  client
    configuration 644
  configuration 640
  server
    configuration 640
DataStage Server
  jobs 647
DB2
  common classes 82
  create tables 343
  create the tables 370
DB2 access plan 456
DB2 access plan graph 480
DB2 catalog 63, 81
DB2 Configuration Assistant 226
DB2 connection 317
DB2 constraints 154
DB2 Control Center 82
DB2 Cube Views 414
  API 15
    error logging 679
    export 682
    Import 682

trace file 679
tracing 679
validate 682
benefits 21
create a cube model 396
cube 15, 421
cube model 14, 421
cube model export 349, 363
diagnostics 669
export 376, 391, 396, 561
high-level web services 620
import 87, 346, 360, 373, 388, 407, 411
XML file 88
import options 90
interfaces 17
MetaBroker 295–296
metadata model 14
model 64
Quick Start wizard 137
restore 122
stored procedure 85–86, 91
stored procedure API 673
tracing 669
unique names 423
web services
describe 620
execute 620
members 620
DB2 Cube Views with Sybase PowerDesigner 338
DB2 Data Warehouse Center 399
star schema 400
DB2 Dimensional Metadata Wizard 487, 493
DB2 Explain 455, 508, 513, 539, 602
DB2 GUI tools 82
DB2 interfaces 63
DB2 OLAP Server 10
Administration Services 421
analysis functions 439
data load 449
Enterprise Services 421
functions 418, 435, 439
load 10
OLAP Mining 421
Spreadsheet Services 421
DB2 optimizer 10, 24, 27, 40, 45, 128, 137–138, 184, 193, 450
DB2 parameters 171
DB2 QMF for TSO/CICS 248
DB2 QMF for WebSphere 248
DB2 QMF for Windows 248
DB2 QMF High Performance Option 248
DB2 Snapshot monitor 171
DB2 UDB V8.1 Data Warehouse Enterprise Edition 13
DB2 UDB V8.1 Data Warehouse Standard Edition 13
DB2 Visual Explain 178, 479
DB2 Warehouse Center 400
DB2 Warehouse Manager 399
db2advis 182
db2advis.in 182
db2exfmt 173
db2info.md_message() 222
db2md_config.xml 679
db2md_config.xsd 679
db2mdapi.log 679
db2mdapi.sql script 86
db2mdapiclient 88, 91, 448, 673, 680
db2mdapiclient.cpp 680
db2mderror.log 679
db2mdtrace.log 679
defining
SQL expression 71
delete objects from report 237
dense 434, 453
dependency analysis 295
deployment 4
recursive 75
standard 75
deployment mode 75
derivations 22, 309
descriptive data 64
design 282, 314
the multidimensional schema 87
design tools 19, 87, 413
Desktop OLAP 7
desktop PowerCubes 502
DFT_QUERYOPT 185
DFT_REFRESH_AGE 170
diagrams 393
dice 11, 22, 46
Dimension 282
dimension 71, 81
Accounts 434–435
attributes 38
dense 420
name 446
regular 38, 103

sparse 420
tables 4, 65
Time 103, 434–435
type 38, 103
dimension type 100
time 38
dimensional
schemas 64
dimensional model 32, 87
dimensional modeling 85
dimensionality 42, 45
dimensions 9, 22, 29, 32, 36, 63, 127
describe 102
hierarchies 15
higher level 9
identify specific attributes 102
lowest level 9
order 453
select the existing attributes 102
sequence 434–435
Directory 301, 648
directory
persistent 328
disk consumption 128
disk space limit 478
DOLAP 7–8, 11
drag and drop 235, 238, 256
drill down 22, 26, 46, 56, 84, 86, 160, 234, 236, 240, 244, 260, 464–468
time 246
drill through 11, 26, 53, 84, 86, 160, 263, 459, 477, 479, 502
drill through query 505
drill through report 450
cost timerons 509
SQL 506
drill up 237, 261
Drill-Down Path 498
dynamic calc 453

E

end-user
analytics requirements 119
Enterprise Information Portals 545
error code 327
error message 252
ERwin 278, 280–281, 304, 412
ERX file import 364
MetaBroker 281, 283
process 283
save the model 357
SQL DDL 356
star schema 353
XML export 282
XML export file 281
ERwin MetaBroker 653
parameters 651
ERwin v4.x
import 351
save the model 343
SQL DDL 342
star schema 340
ERX 412
ERX file 357, 363
Essbase 227
ETL 278, 330, 399
ETL tools 4, 8, 19, 23, 29, 31, 87, 413–414
event 311, 326
class 310
date 310
time 310
event metadata 639
Event's Message attribute 311
events 309–310, 313
Excel 226, 240
Excel worksheet 233
executables 311
Explain tables 173
EXPLAIN.DDL 183
export 9, 39, 233, 284, 294, 421, 429, 440, 452
eXtended Markup Language 83
eXtensible Markup Language 616
extract 26, 43, 84, 86, 160, 450
MOLAP 44

F

fact 29, 282
fact table 4, 65
granularity 41
facts 32, 70, 127
Facts wizard 99
failed runs 311
failure 310–311
filter 236, 257
Find Executables 311
Find Runs 310

foreign key 33, 65, 67, 85, 92
format report 240
formatting
    add 260
formatting tool bar 260
formulas 23
forward engineering 412–413
front end tool 9
front-end MOLAP tool 43
front-end tool 22, 34
front-end tools xxxi, 4, 7, 19, 39, 63
function templates 420
functional dependency 76

G
grain 70
granularity 9, 41, 45–46, 450
Graphical User Interface 81
GROUP BY 130
GROUPING 134
GROUPING SETS 10, 130
GUI 36, 81, 83, 225

H
hide 236
    dimension 237
    level. 237
    measure 237
hierarchies 9, 22, 63, 75, 77, 80, 85, 127, 130, 419
    alternate 447
    creating 107
    multiple 255, 447
    ragged 121
hierarchy 5, 73, 423
    level 38
        attributes 38
    names 446
hierarchy type 38
    balanced 38
    network 38
    ragged 38
    recursive 38
    standard 38
    unbalanced 38
Hierarchy wizard 38, 109
HOLAP 6, 8, 16, 26, 40, 50–51, 53, 86, 149, 418
    drill through 51
Hosts_Resource 317
HPO 248
HTML 278
Hybrid Analysis 11, 420, 426, 439, 449–450, 458
    performance 426
    performance results 472, 475
    query
        performance results 661
    scenario 460
Hybrid OLAP 7, 420
Hyperion 290
    MOLAP database model 290

I
IBM 412
IBM DB2 640
IBM DB2 Office Connect Analytic Edition 225
IBM DB2 OLAP Server 418
IBM DB2 Query Management Facility 248
IBM QMF for Windows 247
IBM Rational Rose 339
impact analysis 31–32, 34, 278, 281, 295
    query path viewer 307
    root 302
impexp.bat 448
import 33–34, 227, 231, 283–284, 293, 295
    options 90
    process 433
    results 433
import category 324
Import wizard 83, 89
Impromptu
    catalogs 495
    hierarchies 500
    queries 487
    Reports 487
    reports 489, 495
Impromptu catalog 487
    folders 497
Impromptu files 487
Impromptu Query Wizard 510
incremental changes 223
Index Advisor 183
indexes 15
Informatica 408, 412
    PowerCenter 408
    PowerMart 339, 408
    PowerMart Designer 408
information

integration 58
informational constraints 136, 138
inner join 141–142
inspect 311
integration 280, 327
API 280
file format 280
Integration Server
adding back objects 435, 438
bridge
log file 441
calculated measures 435–436
data load
SQL 455
drill through report 449, 477–478
template SQL 479
duplicate names 423
dynamic calc 439
mapping 424, 431
measure hierarchy 437
measures 444
metadata 418
metaoutline 421, 424–425, 429, 437–438, 440
model 419, 421, 424–425, 429, 433, 438, 440
naming 435
related attributes 435–436
rename 435
results review 434
scenario 477
storage settings 439
transformation rule 437
two-pass calculation 439
Integration Server Bridge 417, 421
Integration Server bridge
alternate hierarchies 444, 447
attribute relationship 436
benefits 449
column names 436
formula 438
hidden columns 444, 447
launch 429
log 431
mapping 444
naming 444–445
time dimension 444, 447
window 430
interchange 281
isalog.xml 441, 447
ISBridge 448
ISBridge.bat 429

J

J.D. Edwards 312
Java 82, 280, 331
JMS 312
job 310
relationship 325
job design 309, 324
jobs 310, 318
compiled 325
run 647
joins 5, 22, 36, 63, 77, 97, 102, 450
attributes 141
auto-detection 92, 104
cardinality 141, 448
creation 105
inner 448
JSP-based 278

L

Layout Designer 255
Measures 256
online mode 256
Side Dimensions 256
Top Dimensions 256
left attribute 76
level of aggregation 450
levels 75
limit 269
link 324, 327
Link object 324
links 309
Listener 313, 318–319, 644, 647
Listener configuration file 644
listener.cfg 644
LOAD
ALLOW READ ACCESS 201
Locator 316
Locator path 317
Locator table 317, 647
log 202, 311, 321, 441
logical model 344
lookup table 594
lookups 309
lowest level 44

M

mainframe
database sources 312
flat files 312
mapping 9, 14, 32, 35, 39, 309, 436, 438, 441, 449, 458
relational source 9, 419
relational star chema to OLAP 29
mapping back 446
master copy 302
Materialized Query Tables 10, 125
MAX 38
MDL 412
MDL file 384, 391
MDSampleExplain.sql 173
measure 68
calculations 513
function 120
measures 9, 22–23, 36, 63, 67, 98, 234, 419
additive 119
aggregated 69
asymmetrical 119
calculated 38, 69, 119
complex 120, 420
derived 69, 119
distributive 119, 129, 151
non-distributive 119, 151
semi-additive 119
symmetrical 119
types 85
member selection 236
members
prefix 453
Meta Integration
bridges
refresh 412
data model
BI vendors 337
DB2 Warehouse Manager 337
ETL tools 337
forward engineering 334, 337
Informatica 337
metadata standards 337
OMG CWM XMI 337
reverse engineering 335, 338
forward engineering
from ERwin v3.x 353
from ERwinv4.x 339
from PowerDesigner 366
from Rational Rose 379
generic metadata flows 335
implementation steps 338
metadata movement
bi-directional 334
Model Bridges 330
Model Browser 330
Model Comparator 330
Model Integrator 330
Model Mapper 330
repository metamodel 332
reverse engineering
to ERwin v3.x 362
to ERwinv4.x 348
to PowerDesigner 375
to Rational Rose 390
Meta Integration Model Bridge 332, 413
Meta Integration Repository 332, 413
Meta Integration Works 413
Meta Object Facility 393
MetaBrokers 279
metadata 8
bridge xxxi, 19, 23, 53, 88
building 84
business 278
Business Name 119
catalog tables 85
comparator 413
configuration management 413
creation 83–84
data model 316
definitions 278
description 90
exchange xxxi, 9, 414
export 13, 83
flow 276, 281, 423
scenarios 281
generating 426
highly refined 38
import 13, 23, 83
integrator 413
interchange 11
job design 316
maintenance 19, 81
managing 276
manipulation 83
mapper 413
mapping 29
model 63

move 123
movement 330
operational 314
configuration 660
parse 495
publishing 278
pull 19
push 19
repository manager 413
sharing 32
store 9, 88
structured 8
subscribing 294
synchronizing 188, 328
technical 278
tools 276
update 36
versions 413
metadata catalog tables
create 86
metadata management tools 31
metadata model
import 79
metadata object
rules 118
metadata repository 413
MetaIntegration metadata bridge xxxi
MetaIntegration Technologies xxxi
meta-metamodel 393
metamodel 31, 393
metaoutline 420, 424, 435, 452
export 425
name 444
suffix 453
MetaStage xxxi, 271
Activity 313
classes 310
connected objects 301
contained objects 301
Data Collection 309
data collections 316
data items 316
Data Schema 309
Data Store 309
Directory 313, 646
Directory Administrator 644
Event 313
File 309
import 284
Import Category 302
Object Connector 300
objects 302
Operate 641
Path Viewer 310–311
Process
MetaBroker 309
query 278
repository 278, 301
Run 313
RunContext 313
RunImport 646
subscription 294
subscription wizard 287
MetaStage Class Browser 317
MetaStage Directory 321, 644, 647
MetaStage Explorer 317, 320–321, 324, 648
MicroStrategy xxxi
queries performance 597
query examples 597
query reports
business cases 597
MQTs 597
query performance results 602
SQL View option 600
timerons 597
MicroStrategy Administrator 587
MicroStrategy Desktop 587, 596
MicroStrategy Import Assistant 587
associated relationships 595
asymmetric measures 596
attribute relationships 595
attributes 594
best practices 594
Cube Reader 587
Cube Translator 587
database instance 591
descriptive relationships 595
dimension 595
hierarchies 595
implementation steps 588
import 591
input parameters 590
installation 589
join 594
log file 589
mapping 592
measure 595
process log file 591

Project Creator 588
project source 591
refresh 597
schema definition file 590
steps
status information 591
symmetric measures 595
Warehouse Catalo 587
MicroStrategy Intelligence Server 586
MicroStrategy MDX Adapter 587
MicroStrategy Narrowcast Server 587
MicroStrategy Web 587
MIMB 332–333, 414
convert 371, 376, 384, 391, 396, 403, 409
convert ERX file 358
convert XML file 344, 363
MIMB utility 400
MIMB, convert XML file 350
MIN 38
MIW 330
model 64, 278
conceptual 276
export 425
name 444
physical 276, 344
MOF 393
MOLAP 6, 8, 11, 16, 26, 29, 34, 40, 86, 145, 149, 418, 458
data load 426
load 12, 44–45
optimize 12
performance 12
multiple loads 45
refreshes 44
MOLAP database 450
calculation 476
size 476
MQT 241, 244, 454, 477
re-routed 471, 475, 481–482
response times 475
MQT routing 124
MQTs 10, 13, 15, 35, 44, 125, 241, 449, 541
access path 175
access plan graph 177
advisor 13
attribute relationship 151
centralized 50
CHECK PENDING 135, 185, 200
check pending state 170, 188
collecting statistics 134, 170
configuration 216
cost timeron 176, 190
create 132
CREATE TABLE 132
creating indexes 134
creation 170
data latency 201
DB2 Explain 170
DB2 parameters 181
DB2 privileges 159
ALTER 159
CONTROL 159
CREATEIN 159
DROPIN 159
SELECT 159
DB2 Snapshot Monitor 172
DB2 special registers 183
DEFERRED REFRESH 198–199
deployment
methodology 169
depth 181
design recommendation 136
disk space 131
drop 135
ENABLE FOR QUERY OPTIMIZATION 184
Explain SQL 171
foreign key 137
FULL REFRESH 133, 198
GROUPING SETS 178
hierarchies 179
inconsistencies 188
incremental refreshes 45
index 215
indexes 11, 181, 198
recommendations 161
maintenance 170, 198
maintenance options 128
memory 216
multiple 131
multiple slices 48, 53, 56
number 129–130, 135
performance 197
performance benefits 482
performance optimization 169
planning 135
point-in-time 199
pre-built 52
primary key 137

production 170
query rewrite 184
query type specification 186
query workload 181, 188
referential integrity 136
refresh 45, 204
implementation 208
schedule 45
REFRESH DEFERRED 132, 134, 164
REFRESH IMMEDIATE 132, 134, 164, 198, 200
requirements 201
REFRESH INCREMENTAL 133, 198
frequency 202
limitations 214
requirements 202
refreshing 134
reorganizing 134, 170
reroute 151, 181
RUNSTATS 183, 215
Set Refresh Age 199
sharing 45
SORTHEAP 216
space estimation 216
SQL script
create 160
SQL scripts
run 160
SQL statements run 172
staging table 215
STATEMENT HEAP 216
summary table update 134
synchronization 45, 132, 134, 200
System Maintained 132
tailored 51
TEMP space 217
tuning 215
User Maintained 132
MstrDb2Import.exe 589
multidimensional
data models 4
model 5
multidimensional data 11
describe 263
manipulate 263
visualize 263
multidimensional metadata management 63
Multidimensional OLAP 6
multidimensional structure 13
multidimensionality xxxi
multiple joins 67

N

names
changes 446
collisions 445
prefixed 446
suffix 446–447
uniqueness 445
navigate 64, 236, 304, 432
network hierarchy 75
non-DB2 data sources 426
non-nullable
foreign keys 140
normalization 66
normalized 64
NULLS 140
number of joins 64
number of rows 323, 450
read 310
written 310

O

Object Creation wizards 83
Object Explorer 254
Object Management Group 393
ODBC 226, 230, 240, 316
Oerational Data Store 295
Office Connect 9, 225
access plan 245
Add-in 229
DB2 Explain 241–242
SQL trace 242
trace 241
Office Connect tool bar 235
Office2000 226
OLAP 3, 5, 126
application developers 614
OLAP Center 15, 22, 29, 36, 75, 81, 157, 264, 289, 295, 297
architecture 83
export 428
GUI 84
Import wizard 442
results 443
SQL expression builder 98
OLAP cube 80

OLAP database 9
OLAP metadata 23
  sharing 13
OLAP model 9, 12, 14
OLAP provider 57, 619
OLAP queries
  invalidation 268
  performance 268
OLAP query 251
OLAP service provider 619
OLAP solutions
  deployment 63
OLAP structure 9
OLAP system xxxi
OLAP tools 19, 64, 283
OLAP web services 614
OLAP web services client 619
OLAP-related metadata objects 81
OMG 393, 412
OMG CWM 332, 339
OMG CWM XMI 399
OMG UML 332
Online Analytical Processing 5
operational
  analysis query 324
Operational Data Store 309
operational database 33
operational metadata 311–312, 314
  configure 640
Optimization Advisor 15, 25, 39, 44, 80, 129, 241, 246, 448–449, 454, 459, 477
  data sampling 26, 161
  design recommendation 149
  disk space 161
  MQTs
    changing workloads 169
    checklist 170
    creation script 168
    errors 167
    information 167
    recommendations 187
    REFRESH DEFERRED 167
    refresh method 164
    SQL scripts 166
    tablespaces 165
    validate 170
    warnings 167
  MQTs tablespaces 164
  query type
    combining 148
    drill down 143
    drill through 147
    extract 145
    report 144
  space 26
  summary table indexes 152
  time limit 162
  using 152
  wizard 152
    cube model metadata 154
    data sampling 153–154
    DB2 statistics 154
    disk space 157, 159
    disk space limitations 153
    file names 159
    limitations 159
    MQTs creation options 159
    parameters 159–160
    query type 153, 159
    run 157
    SQL files generated 154
    SQL scripts 159
    time 157, 159
    time limitations 153
Optimization Advisor wizard 84, 126
optimization rule
  cube model 141
  dimension 141
  enforce 142
optimization rules 140
  join 141
optimizer 127
optimizing
  query performance 80
Oracle Applications 312
outline 419, 434, 439
  size 476

P

packages 393
parent 73–75, 453
partitioned database 458
paths 311
PDM 365, 412
PDM file 371
PeopleSoft 312
performance 10, 49, 64, 84, 122, 127, 159, 507

performance enhancements 449
performance optimization 39
physical data 282
physical outline 419
physical storage 280
pivot 262, 268
    tables 235
Pivot Chart 239
Pivot Table Field List. 234
pivot table services 226–227
PivotTable layout wizard 238
PMB 647
populate
    the multidimensional schema 87
portals 59
PowerCenter
    save target datamart 409
PowerCube level security 501
PowerCubes 489, 498
PowerDesigner 412
    Conceptual Data Models 365
    import 377
    Physical Data Models 365
    save the model 370
    shortcut 371
    SQL DDL 369
    star schema PDM 366
primary key 33, 65, 67, 85, 92, 448
Process
    MetaBroker 310
process
    analysis 310
    design 278
    failure 309
    history 310
    import 314
    metadata 310, 312, 314
    runtime 278
    success 309
process analysis 309, 314, 316, 647
    paths 310
process flow 423
Process MetaBroker 316, 318, 640, 647
    configuration file 642
    EventsDirectory 643
    ListenerPort 643
    ListenerServer 643
    LogDirectory 642, 644
    Port 642, 644
    variables 642
process metadata 318, 640
process runs 309
processes 278
processmb.cfg 642
Project Manager 227, 230, 240
protocols 59
public data 277
publisher 279
pull 88, 222
push 423

Q

QMF
    OLAP Query 266
        Filter 257
        object 266
    OLAP Query wizard 251, 254
QMF for Windows xxxi, 9
    Administrator 251
    control tables 251
    drill down 257
    error message 252
    example 265
    filter options 257
    form 250
    formatting 260
    hierarchy levels 255
    job 250
    Layout Designer 254, 256
    Layout tree control 256
    list 250
    maintenance 268
    map 250
    Maximum Bytes to Fetch 269
    Maximum Rows to Fetch 269
    Object Explorer 254
    OLAP query 251
    OLAP Query wizard 251
    procedure 250
    Prompted Query 257
    query 250
    Query Results View 254, 256
    Resource Limits 269
    rollup 257
    sort
        model 252
        schema 252

SQL execution 257
SQL status 257
Quality Assurance 331
query
drill down 46
drill through 53
Natural Language 250
performance xxxi, 13
Prompted 250
report 46
response 47
rewrite 10
routed to MQT 241
SQL 250
types 460
workload 460, 472
query response time 56
query rewrite 124, 127, 133, 135, 137, 185, 200
query tools 15
query workload
type 129
Quick Start wizard 36, 83, 91, 124

R

ragged hierarchy 75
Rational Rose 278
RDBMS xxxi, 312
real time 56
recursive deployment 111
Redbooks Web site 699
Contact us xxxv
Referential Integrity 36, 85, 92, 124
referential integrity 136
creation 137
ENFORCED 137
NOT ENFORCED 137
referential integrity constraint 137
refresh data 240
REFRESH IMMEDIATE 201–202
REFRWESH INCREMENTAL 201–202
relational 462
extraction 450
relational database 420, 451, 460
Relational OLAP 7
relational tables 86
relationships 301, 393
dependency 311
remove dimensions 238
report 56, 84, 160
report drill paths 505
report layout 238
report manipulation 237
reporting 11, 26
reporting tools 19, 39, 263
reports
delete 240
save 240
response time 13, 24
retrieve
data 244
reverse engineer 426
reverse engineering 412–413
RI 36
right attribute 76
Rocket Software xxxi
ROLAP 6, 8, 11, 16, 26, 34, 40, 46, 86
drill-down 47
report 47
ROLLUP 130
rollup 10, 22, 56, 130, 151, 262
root object 302
Rose 412
create the model 379
import 391
MDL file 378
save the model 384
SQL DDL 383
rules 118
RunImport 313, 320, 644, 647
configuration file 644
EncryptedPassword 646
installation directory 320
LogDirectory 646
MetaStageDirectory 646
Schema 647
settings 646
User 646
runimport.cfg 647
RunImportStart.bat 320
runs 311

S

sampling 16
SAP R/3 312
scalability 7
scale economies 45, 50

scenarios 281
scheduler 320
schemas 63
scratch 36, 45, 423
script 38
semantic 64
semantic translation 279
service requestors 620
SET INTEGRITY 133, 185, 213
Siebel 312
slice 5–6, 9, 11, 15, 22, 40, 44, 46, 80, 459
slice and dice 262
snowflake 64, 66, 77
snowflake schema 64, 79, 138, 282, 594
  constraints 143
SOAP 615
SOAP request 619
Software Executable
  class 310
source system 309
source table 309
sparse 434, 453
sparsity 26
spreadsheet 225, 233, 460, 479
  add-in 226
SQL 280
SQL expression 71
SQL queries 40, 227, 240–241, 450
  aggregation 24
  joins 24
SQLDebug 241
staging database 295
standard 65, 395
star schema 22, 32, 64–65, 79, 81, 84–85, 92, 127, 138, 282, 498, 659
  subset 41
start-up 35
STDDEV 129
stddev 38
storage 12
storage requirement 11
stored procedure 222, 227, 252
subscribe
  data model definitions 278
subscriber 279
SUM 38, 45, 69, 129, 134, 151, 201–202
summaries 10
summary tables 12, 126
  model-based xxxi
  storage 127
super aggregates 27
super-aggregate operators 10, 16
swap rows and columns 236, 238
Sybase 412
synchronization 451

T

table definitions 316
table scan 64
TableDefinition objects 318
tables 32–33
TABLESAMPLE 16, 162
target 309
target table 309
tasks 84
TCO 583
Technology Preview 56
Time 444
tools 296
Total Cost of Ownership 583
transformations 309–310, 313
transformer derivation 647
Transformer files 487
Transformer Modeling environment 516
trends 22
two-way bridge 223, 421

U

UDDI 615
UDDI registries 619
UML 381, 393
UML class 278
UML XMI 339
unbalanced hierarchy 74
Unified Modeling Language 393
uniqueness 453
Universal metadata bridge xxxi
unusual occurrences 310
user-defined 277
user-defined functions 71
utf-8 397

V

VALIDATE 222
vendors 223
views 162

W

warnings 309
web interface 278
web servers 59
web service
  client application 621
  Describe
    input 626
    output 626
  Execute 631
    aggregation level 635
    input 635
    output 635
  Members 627
    input 629
    output 630
web service provider 621
Web Services
  access to remote information 59
Web services 56, 312
web services 613
  access remote analytical data 614
  cells 614
  cubes 614
  HTTP 616
  HyperText Transport Protocol 616
  interface 17
  Simple Object Access Protocol 616
  slices 614
  SOAP 616–617
  TCP/IP 616
  UDDI 616, 618
  Web Services Description Language 616
  WSDL 616–617
  XPath 614, 618
WebSphere MQ 312
wizard 81
worksheets 227, 240
wrapper 91, 680
WSDL 615

X

XMI 278, 393
XMI compliant 332
XML 19, 56, 83, 222, 250, 282, 312, 319, 412, 615, 623–624
XML element 628
XML file 32, 87, 222, 313, 344, 422, 424, 440, 442, 644, 651
  full path name 430
  import 432
  location 299
  process 429
XML files
  combined 425
XML Metadata Interchange 393
XML Path Language 618
XML Schema 615
XPath 18, 56, 614–615, 625
  depth 625
  metadata retrieved 626
  query expression 625
XSL
  transformations 278

IBM

Redbooks

DB2 Cube Views
A Primer

(1.0" spine)
0.875"<->1.498"
460 <-> 788 pages

DB2 Cube Views
A Primer

Introduce DB2 Cube Views as a key player in the OLAP world

Understand cube models, cubes and optimization

Improve your metadata flow and speed up queries

Business Intelligence and OLAP systems are no longer limited to the privileged few business analysts: they are being democratized by being shared with the rank and file employee demanding a Relational Database Management System (RDBMS) that is more OLAP-aware.

DB2 Cube Views and its cube model provide DB2 the ability to address multidimensional analysis and become an actor in the OLAP world.

This IBM Redbook focuses on the innovative technical functionalities of IBM DB2 Cube Views V8.1 to store multidimensional metadata in DB2 catalog; to build automatically model-based summary tables to speed up query performance; and to provide an advanced API to allow other Business Intelligence partners' tools to benefit from both metadata exchange and improved query performance.

This book positions the new functionalities and their benefits, so you can understand and evaluate their applicability in your own Business Intelligence and OLAP system environment. It provides information and examples to help you to get started planning and implementing the new functionalities.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7002-00 ISBN 0738499730