

Leveraging DB2 Data Warehouse Edition for Business Intelligence

Building complete and robust business intelligence solutions

Integrated tools for fast and easy deployment

Standards based for flexibility and change



Chuck Ballard
Angus Beaton
David Chiou
Janardhan Chodagam
Meridee Lowry
Andy Perkins
Richard Phillips
John Rollins



International Technical Support Organization

**Leveraging DB2 Data Warehouse Edition for
Business Intelligence**

November 2006

Archived

Note: Before using this information and the product it supports, read the information in “Notices” on page xi.

First Edition (November 2006)

This edition applies to Version 9.1 of DB2 Data Warehouse Edition.

© Copyright International Business Machines Corporation 2006. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

| | |
|--|------|
| Notices | xi |
| Trademarks | xii |
| Preface | xiii |
| The team that wrote this redbook | xiv |
| Become a published author | xvii |
| Comments welcome | xvii |
| Chapter 1. Information warehousing for business insight | 1 |
| 1.1 Current business challenges | 2 |
| 1.2 Information as a service | 2 |
| 1.3 Embedding analytics into business processes | 3 |
| 1.4 From data warehouse to information warehouse | 6 |
| 1.5 An integrated business intelligence framework | 8 |
| 1.6 The DB2 Business Intelligence solution framework | 9 |
| Chapter 2. DB2 DWE - A technical overview | 11 |
| 2.1 DWE architecture | 12 |
| 2.1.1 DWE Design Studio | 14 |
| 2.1.2 InLine Analytics using DWE Alphablox | 21 |
| 2.1.3 DWE production environment | 23 |
| 2.1.4 DB2 Query Patroller | 25 |
| 2.2 DWE topology | 25 |
| Chapter 3. The DB2 DWE Design Studio | 29 |
| 3.1 The Eclipse platform | 32 |
| 3.2 Introduction to DB2 DWE Design Studio | 33 |
| 3.2.1 The workspace | 33 |
| 3.2.2 Projects and the local file system | 34 |
| 3.2.3 The Welcome page | 36 |
| 3.3 The DB2 DWE Design Studio Workbench | 37 |
| 3.3.1 Perspectives | 37 |
| 3.3.2 Editors | 39 |
| 3.3.3 Views | 41 |
| 3.3.4 Common tasks | 48 |
| 3.3.5 Team component | 55 |
| Chapter 4. Developing the physical data model | 57 |
| 4.1 Physical data model | 59 |

| | | |
|-------------------|---|-----------|
| 4.2 | Creating the physical data model | 60 |
| 4.2.1 | From a template | 60 |
| 4.2.2 | From an existing database | 61 |
| 4.2.3 | From DDL | 64 |
| 4.2.4 | From the Database Explorer | 64 |
| 4.3 | Working with diagrams | 65 |
| 4.3.1 | Creating a diagram | 66 |
| 4.3.2 | Using the Diagram Editor | 67 |
| 4.4 | Editing physical data models | 69 |
| 4.4.1 | Using the Data Project Explorer | 69 |
| 4.5 | Model analysis | 69 |
| 4.6 | Deploying the data model | 72 |
| 4.6.1 | Using Design Studio to deploy a physical data model | 72 |
| 4.6.2 | Using the Admin Console to deploy a physical model | 74 |
| 4.7 | Maintaining model accuracy | 75 |
| 4.7.1 | Comparing objects within the physical data model | 75 |
| 4.7.2 | Visualizing differences between objects | 75 |
| 4.7.3 | Synchronization of differences | 76 |
| 4.7.4 | Impact analysis | 77 |
| 4.8 | Summary | 78 |
| Chapter 5. | Enabling OLAP in DB2 UDB | 79 |
| 5.1 | Implementing OLAP | 80 |
| 5.1.1 | OLAP architectures | 80 |
| 5.1.2 | DB2 UDB and OLAP | 81 |
| 5.2 | DWE OLAP | 82 |
| 5.2.1 | DWE OLAP metadata | 84 |
| 5.2.2 | Materialized Query Tables (MQTs) | 85 |
| 5.2.3 | Optimization Advisor | 87 |
| 5.3 | Modeling OLAP with OLAP Center | 89 |
| 5.3.1 | Migrating from OLAP Center to Design Studio | 90 |
| 5.4 | Modeling OLAP with DWE Design Studio | 90 |
| 5.4.1 | Database Explorer | 92 |
| 5.4.2 | Properties view | 92 |
| 5.4.3 | Data Project Explorer view | 94 |
| 5.4.4 | Creating the CVSAMPLE sample database | 95 |
| 5.4.5 | Preparing a DB2 UDB database for DWE OLAP | 95 |
| 5.4.6 | Creating a new data project | 99 |
| 5.4.7 | Creating a new physical data model | 99 |
| 5.4.8 | Quick Start Wizard | 103 |
| 5.4.9 | Defining cube models | 103 |
| 5.4.10 | Defining the fact table and measures | 104 |
| 5.4.11 | Defining dimensions | 106 |

| | | |
|-------------------|--|------------|
| 5.4.12 | Defining joins | 108 |
| 5.4.13 | Defining attributes | 109 |
| 5.4.14 | Defining levels | 110 |
| 5.4.15 | Defining hierarchies | 112 |
| 5.4.16 | Defining cubes | 113 |
| 5.4.17 | Analyze OLAP objects for validity | 117 |
| 5.5 | Deploying objects to DB2 UDB databases | 118 |
| 5.5.1 | Generate DDL | 118 |
| 5.5.2 | Compare and Sync Editor and Delta DDL | 119 |
| 5.5.3 | DWE OLAP Optimization Advisor | 121 |
| 5.6 | DWE OLAP optimization cycle | 129 |
| 5.7 | Metadata exchange | 132 |
| 5.7.1 | Import | 133 |
| 5.7.2 | Export | 134 |
| Chapter 6. | Data movement and transformation | 137 |
| 6.1 | DWE SQL Warehousing Tool (SQW) | 138 |
| 6.1.1 | SQW overview | 138 |
| 6.1.2 | Architecture | 139 |
| 6.1.3 | SQW warehouse application life cycle | 141 |
| 6.1.4 | Source, target, and execution databases | 145 |
| 6.1.5 | Setting up a data warehouse project | 147 |
| 6.2 | Developing data flows | 148 |
| 6.2.1 | Defining a data flow | 148 |
| 6.2.2 | Data flow editor | 151 |
| 6.2.3 | Data flow operators | 152 |
| 6.2.4 | Subflows | 195 |
| 6.2.5 | Validation and code generation | 197 |
| 6.2.6 | Testing and debugging a data flow | 200 |
| 6.2.7 | A complete data flow | 203 |
| 6.3 | Developing control flows | 205 |
| 6.3.1 | Defining a control flow | 205 |
| 6.3.2 | Control flow editor | 209 |
| 6.3.3 | Control flow operators | 209 |
| 6.3.4 | Validation and code generation | 219 |
| 6.3.5 | Testing and debugging a control flow | 221 |
| 6.4 | Variables in data flows and control flows | 222 |
| 6.5 | Preparing for deployment | 226 |
| 6.5.1 | Defining warehouse applications | 227 |
| 6.5.2 | Defining application profiles | 227 |
| 6.5.3 | Generating code and packaging for deployment | 229 |
| 6.6 | Integrating with IBM WebSphere DataStage | 229 |
| 6.6.1 | Overview of IBM WebSphere DataStage | 230 |

| | | |
|-------------------|---|------------|
| 6.6.2 | Key differences between SQW and DataStage | 232 |
| 6.6.3 | Integrating DataStage and SQW | 235 |
| 6.6.4 | Conclusion | 236 |
| Chapter 7. | DWE and data mining | 237 |
| 7.1 | Data mining overview | 238 |
| 7.1.1 | Discovery data mining | 238 |
| 7.1.2 | Predictive data mining | 239 |
| 7.2 | The data-mining process | 239 |
| 7.2.1 | Understanding the business problem | 240 |
| 7.2.2 | Understanding the data model | 241 |
| 7.2.3 | Identifying the data mining approach | 242 |
| 7.2.4 | Extracting and preparing the data | 242 |
| 7.2.5 | Building the data mining model | 243 |
| 7.2.6 | Interpreting and evaluating the data mining results | 244 |
| 7.2.7 | Deploying the data mining results | 245 |
| 7.3 | Embedded data mining | 246 |
| 7.4 | DWE data-mining components | 248 |
| 7.4.1 | Modeling | 248 |
| 7.4.2 | Visualization | 250 |
| 7.4.3 | Scoring | 250 |
| 7.5 | Data mining in the DWE Design Studio | 252 |
| 7.5.1 | Mining flows | 252 |
| 7.5.2 | Database enablement | 253 |
| 7.5.3 | Data exploration | 255 |
| 7.5.4 | Data mining operators | 262 |
| 7.5.5 | Building a mining flow | 275 |
| 7.5.6 | Validating and executing a mining flow | 287 |
| 7.5.7 | Visualizing models and test results | 291 |
| 7.5.8 | Scoring with other PMML models | 310 |
| 7.5.9 | Managing data mining models | 312 |
| Chapter 8. | InLine Analytic Applications | 315 |
| 8.1 | DB2 Alphablox | 317 |
| 8.1.1 | The architecture | 317 |
| 8.1.2 | Enabled applications | 319 |
| 8.1.3 | Deploying DB2 Alphablox | 328 |
| 8.1.4 | Services | 332 |
| 8.1.5 | Blox server/client structure | 336 |
| 8.2 | Integration with DWE OLAP | 338 |
| 8.2.1 | Metadata bridge | 339 |
| 8.2.2 | DB2 Alphablox Relational Cubing Engine | 340 |
| 8.2.3 | Application performance | 342 |

| | | |
|--------------------|---|------------|
| 8.3 | Developing an application | 343 |
| 8.3.1 | Creating a data source | 344 |
| 8.3.2 | DB2 Alphablox Cube. | 346 |
| 8.3.3 | Developing JSP code for the DB2 Alphablox application | 348 |
| 8.4 | Integration with DWE Mining | 358 |
| 8.5 | Integration with portal applications | 361 |
| Chapter 9. | Deploying and managing DWE solutions | 365 |
| 9.1 | The DWE Administration Console. | 367 |
| 9.1.1 | Functionality provided | 368 |
| 9.1.2 | Architecture | 368 |
| 9.1.3 | Deploying in a runtime environment | 370 |
| 9.1.4 | Security considerations | 373 |
| 9.1.5 | General administration tasks. | 378 |
| 9.1.6 | Locating and using diagnostics | 381 |
| 9.2 | Deploying the physical data model | 383 |
| 9.2.1 | Deployment using the DWE Design Studio. | 384 |
| 9.2.2 | Deployment using the DWE Administration Console | 385 |
| 9.2.3 | Deployment using native DB2 functionality. | 387 |
| 9.3 | DWE SQL Warehousing | 387 |
| 9.3.1 | An overview of the SQL Warehousing components | 389 |
| 9.3.2 | Runtime architecture of DWE SQL Warehousing | 391 |
| 9.3.3 | Managing warehouse resources | 405 |
| 9.3.4 | Deploying and managing warehouse applications | 409 |
| 9.3.5 | Warehouse processes | 415 |
| 9.3.6 | Warehousing diagnostics | 431 |
| 9.4 | Managing the OLAP functionality | 438 |
| 9.4.1 | Creating the OLAP environment | 439 |
| 9.4.2 | Optimizing the OLAP environment | 439 |
| 9.4.3 | Viewing the OLAP environment | 444 |
| 9.5 | Data mining | 447 |
| 9.5.1 | Data mining modules | 447 |
| 9.5.2 | Deploying and managing the data mining models | 450 |
| 9.5.3 | Caching the mining model for performance | 454 |
| 9.6 | Managing DB2 Alphablox within DWE | 455 |
| 9.6.1 | Defining DB2 Alphablox | 456 |
| 9.6.2 | DB2 Alphablox Administration Console | 459 |
| Chapter 10. | DB2 UDB Data Warehouse features | 465 |
| 10.1 | DB2 UDB ESE. | 466 |
| 10.2 | Architecture | 466 |
| 10.3 | Balanced Configuration Unit | 468 |
| 10.4 | Partitioning | 469 |

| | |
|--|------------|
| 10.4.1 Hash partitioning | 470 |
| 10.4.2 Multidimensional data clustering (MDC) | 476 |
| 10.4.3 Range partitioning | 481 |
| 10.4.4 Using all partitioning schemes together | 484 |
| 10.4.5 DB2 partitioning options summary | 485 |
| 10.5 DB2 performance features | 489 |
| 10.5.1 Materialized query tables (MQT) | 489 |
| 10.5.2 Replicated tables | 493 |
| 10.6 Compression | 495 |
| 10.6.1 Row compression | 495 |
| 10.6.2 Other forms of compression in DB2 | 499 |
| 10.7 Self-tuning memory | 500 |
| 10.8 DB2 Design Advisor | 503 |
| 10.9 High availability | 508 |
| Chapter 11. Managing SQL queries with DWE | 511 |
| 11.1 DB2 Query Patroller | 513 |
| 11.1.1 Query interception and management | 513 |
| 11.1.2 Query Patroller thresholds | 515 |
| 11.1.3 Using a query class | 516 |
| 11.1.4 Historical analysis | 517 |
| 11.2 Architecture and components | 517 |
| 11.2.1 The server | 518 |
| 11.2.2 Query Patroller Center | 519 |
| 11.2.3 Command-line support | 523 |
| 11.2.4 Components in a DWE environment | 523 |
| 11.3 Installing and configuring DB2 Query Patroller | 524 |
| 11.3.1 Installing the Query Patroller server | 524 |
| 11.3.2 Configuring the Query Patroller server after installation | 525 |
| 11.3.3 Installing the Query Patroller client | 527 |
| 11.4 Best practices for DB2 Query Patroller | 527 |
| 11.4.1 Initial configuration to capture the query workload | 527 |
| 11.4.2 Using the historical analysis data | 528 |
| 11.4.3 Additional best practices | 530 |
| 11.5 DB2 Query Patroller and the DB2 Governor | 531 |
| 11.5.1 DB2 Governor | 532 |
| 11.5.2 Differences between the Governor and Query Patroller | 534 |
| 11.5.3 Using the Governor and Query Patroller together | 534 |
| Appendix A. IBM data warehousing - complementary software | 537 |
| Data Modeling | 538 |
| Application development | 538 |
| Enterprise Extract Transform Load (ETL) | 538 |

| | |
|---|-----|
| Enterprise Information Integration (EII) | 539 |
| Enterprise Application Integration (EAI) | 539 |
| Data quality | 539 |
| Database Tools | 540 |
| Storage management | 540 |
| Portal and application server | 540 |
| Search tools | 541 |
| Master data management solutions | 541 |
| Appendix B. DWE Admin - problem determination example. | 543 |
| Glossary | 547 |
| Abbreviations and acronyms | 553 |
| Related publications | 557 |
| IBM Redbooks | 557 |
| Other publications | 557 |
| Online resources | 558 |
| How to get IBM Redbooks | 559 |
| Help from IBM | 559 |
| Index | 561 |

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:

Redbooks (logo) ™
developerWorks®
ibm.com®
pSeries®
z/OS®
AIX®
ClearCase®
Cloudscape™
Cube Views™

Database 2™
Distributed Relational Database
Architecture™
DB2®
DRDA®
HACMP™
Informix®
Intelligent Miner™
IBM®

IMS™
NUMA-Q®
OmniFind™
Rational®
Red Brick™
Redbooks™
Tivoli®
WebSphere®

The following terms are trademarks of other companies:

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Enterprise JavaBeans, EJB, Java, JavaBeans, JavaScript, JavaServer, JavaServer Pages, JDBC, JDK, JRE, JSP, JVM, J2EE, Solaris, Sun, Sun Enterprise, Sun Microsystems, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Preface

This IBM® Redbook is primarily intended for use by IBM Clients and IBM Business Partners. In it we describe and demonstrate Version 9 of DB2® Data Warehouse Edition (DWE).

DWE is a comprehensive platform with all the functionality required for developing business intelligence solutions. It enables a robust infrastructure for developing data warehouse-based analytics and Web-based applications, along with best practices for deployment. DB2 DWE integrates core components for data warehouse construction and administration, data mining, OLAP, and InLine Analytics and reporting.

A component-based architecture, DWE has client and server functions, both leveraging emerging IBM Software Group frameworks. It extends the DB2 data warehouse with design-side tooling and runtime infrastructure for OLAP, data mining, InLine Analytics and intra-warehouse data movement and transformation, in a common platform based on DB2 and WebSphere®. The platform pillars are based on the technology of DB2, Rational® Data Architect (for physical data modeling only), the SQL Warehousing Tool, Intelligent Miner™, DB2 Cube Views™, and Alphablox.

DWE is comprised of a suite of products that combines the strength of DB2 with the powerful business intelligence infrastructure from IBM. An Eclipse-based design environment, DWE Design Studio, integrates the DWE products (except Alphablox and Query Patroller) with a common framework and user interface. In DWE Design Studio, physical data modeling including OLAP metadata, cube modeling, data mining modeling, and SQL data flow/control modeling are unified in one common design environment. Also, the DWE Admin Console is a Web-based product that can be used for model deployment, management, and administration. A new SQL Warehousing Tool is used for visual design of intra-warehouse, table-to-table data flows and control flows using generated SQL. DB2 Alphablox is the tool for developing custom applications with embedded analytics-based visual components. DWE enables faster time-to-value for enterprise analytics, while limiting the number of vendors, tools, skill sets, and licenses required.

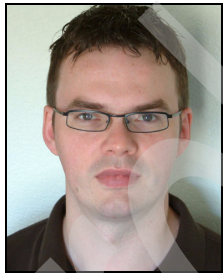
The team that wrote this redbook

This IBM Redbook was produced by a team of business and technical specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

The team members are depicted below, along with a short biographical sketch of each:



Chuck Ballard is a Project Manager at the International Technical Support organization in San Jose, California. He has over 35 years of experience, holding positions in the areas of Product Engineering, Sales, Marketing, Technical Support, and Management. His expertise is in the areas of database, data management, data warehousing, business intelligence, and process re-engineering. He has written extensively on these subjects, taught classes, and presented at conferences and seminars worldwide. Chuck has both a Bachelors degree and a Masters degree in Industrial Engineering from Purdue University.



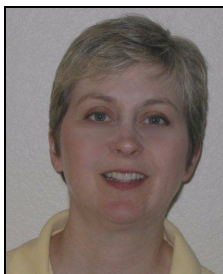
Angus Beaton is a Business Intelligence (BI) specialist for the Information Management SWG Services team in the EMEA North region. He has over seven years of experience working in the IBM Software Group as a pre-sales specialist, database migration specialist, and is currently in SWG services. Angus has worked with a number of IBM Information Management products, recently specializing in the implementation of DB2 UDB with DPF. He holds an LLB in Law from the University of Strathclyde in Glasgow, UK, and an MSc in Information Technology from the University of Glasgow, UK.



David Chiou is a Certified Consulting Software IT Specialist from Chicago, Illinois. He has over 16 years of experience working in technical presales and assisting IBM clients with DB2 UDB, data warehouse, OLAP, and data mining implementations. He holds a Bachelor's degree in Electrical Engineering and a Masters degree in Business Administration from the University of Illinois. His areas of expertise include data management, data warehousing, and business intelligence.



Janardhan Chodagam is a Business Intelligence (BI) Specialist in the Americas TechWorks Organization based in San Francisco, CA. He is responsible for enablement of DB2 Business Intelligence products, such as Alphablox, DB2 Cube Views, and DWE in the Americas region. He has over six years of experience in application integration and data warehousing. Janardhan holds a Master's degree in Petroleum Engineering from the University of Texas at Austin.



Meridee Lowry is a Senior IT Specialist for the IBM Information Management team located in Pittsburgh, Pennsylvania, providing presales consulting and implementation support. She has over 20 years of experience in the Information Technology field, with expertise in data modeling, business intelligence, and data integration solutions. Meridee holds a Bachelor's degree in Music from DePauw University and a Masters degree in Information Science from the University of Pittsburgh.



Andrew Perkins is a Business Intelligence (BI) Specialist for the IBM TechWorks team in Dallas, Texas. He has 23 years of experience with client database and BI projects, providing support in consulting, architecture design, data modeling, and implementation of data warehouses and analytical solutions. Over the years Andy has worked with the majority of the IBM and Information Management products portfolio on platforms ranging from the mainframe to the PC.



Richard T. Phillips is IBM Certified Consulting Senior IT Specialist on the IBM New York Metro Technical Team. He has a background in Electrical Engineering, and 20 years of experience in the IT industry, plus three years as a Manager of Information Systems for a Laser Communication Company. Rich has worked on many Business Intelligence projects, providing support in architecture design, data modeling, consulting, and implementation of data warehouses and analytical solutions. He has held technical management positions and has expertise in both DB2 and Informix® database solutions. Rich has worked for IBM for over five years.



John Rollins, Ph.D., P.E., is the Technical Leader for data mining in the IBM Americas Software Group, where he has been involved in technical sales support, field enablement, intellectual capital development, and consulting for the past 10 years. Prior to joining IBM in 1996, he was an engineering consultant, professor, and researcher. He has authored many technical papers, a textbook, and six patent disclosures. John holds doctoral degrees in economics and engineering from Texas A&M University and is a registered professional engineer in Texas.

Other contributors

Thanks to the following people for their contributions to this project.

From IBM locations worldwide

- ▶ Rav Ahuja - DB2 Program Manager, DB2 Technical Marketing, Markham, Ontario, Canada
- ▶ Marion Behnen - Information Management, BI Development, Austin, Texas
- ▶ Paul McInerney - User-Centered Design Specialist, DB2 Development, Markham, Ontario, Canada
- ▶ Melissa Montoya - DB2 Information Management Skills Segment Manager, Menlo Park, California
- ▶ Steven Siu - Information Management, DB2 Query Patroller Development, Markham, Ontario, Canada
- ▶ Dwaine Snow - Information Management, Senior DB2 Product Manager, Competitive Technologies, Silicon Valley Lab, San Jose, California
- ▶ Patrick Titzler - Information Management, DWE Development, Silicon Valley Lab, San Jose, California
- ▶ Bill Wilkins - Partner Enablement, Information Content Manager, Markham, Ontario, Canada
- ▶ Cheung-Yuk Wu - Information Management, DB2 DWE Development, Silicon Valley Lab, San Jose, California
- ▶ Lin Xu - Information Management, DB2 DWE Development, Silicon Valley Lab, San Jose, California
- ▶ Danny Zilio - Development Analyst, DB2 Optimizer Development, Markham, Ontario, Canada

From the International Technical Support Organization, San Jose Center
Mary Comianos - Operations and Communications
Deanna Polm - Residency Administration
Emma Jacobs - Graphics
Julie Czubik - Editor

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 clients.

Your efforts will help increase product acceptance and client 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 e-mail to:

redbook@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400



Information warehousing for business insight

In this chapter we explore the idea of building a business intelligence architectural framework to provide analytic services to users as part of their normal daily business processes. These service should be easily accessible to users in a format with which they are familiar, and should require little to no training.

1.1 Current business challenges

Over the last decade or so, there have been some interesting trends in the business world. There was a time when the focus was on unrestricted growth of the business. This, of course, led to an extremely painful corrective period where expenses had to be brought back in line and the business back to profitability. There now seems to be a movement where companies are returning to the fundamentals of sound business models.

There is also a renewed emphasis on growth and increasing the value of their business, but with a more balanced approach. That is, companies are looking for growth opportunities, but keeping a careful eye on costs. CEOs must have the ability to be responsive and flexible to meet changing business requirements by maximizing the efficiency of the entire operation in terms of people and processes.

The key to this is being able to make more informed decisions more quickly, from the CEO on down to the line-of-business employees. To obtain the required responsiveness and flexibility, decision making has to be pushed much lower in the organization. Access to the right information at the right time by the right people is the challenge currently faced by the CIO.

Expanding decision making to the larger numbers and roles of employees requires a different approach of access to information than is currently used in most companies. In addition to working with the typical operational type of data found in the daily business processes, they need an expanded reach, which would require access to the type of data found in analytic tools more typically used by knowledge workers and data analysts. This is direction-of-business intelligence, and one in which all companies should be heading.

1.2 Information as a service

Companies are literally drowning in data that has been captured in various kinds of data repositories over the years. The goal has been to accurately capture data in a safe, reliable, and secure fashion. However, the struggle has always been to find a simple but powerful way to organize and filter the data for use to discover the hidden business value. There are many technical and business reasons for the difficulty in organizing, accessing, and understanding the company data. As examples, consider the sheer volume of data, structured and unstructured data, and data in particular silos kept for the exclusive use of a particular organization in the company. These can all be deterrents to providing the right data at the right time to the right people. There are solutions, and companies must begin to employ them.

For example, there is great promise in the concept of delivering information as a service, or *Information On Demand*. Getting there requires that information be freed from the complexities of individual data stores or formats in a process called virtualization, which enables you to adapt business processes more rapidly while being able to optimize the underlying technology. You must also provide a unified view of the information, ensuring consistent formats and increased quality of data, which is accomplished through information integration. Also, you must accelerate the use of information by providing new, innovative ways to organize, maintain, and analyze it.

As seen in Figure 1-1, Information On Demand is all about delivering information to any tool, process, or person that can benefit from it. In this book we discuss how to deliver advanced, custom analytics as an embeddable service.

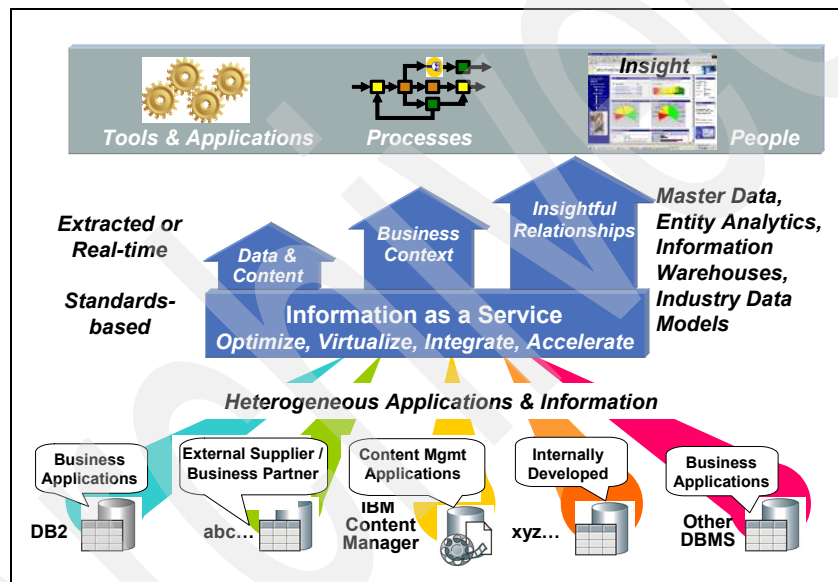


Figure 1-1 Information On Demand

1.3 Embedding analytics into business processes

The IT community is well acquainted with the value proposition of the data warehouse as a vital foundation for any strategic approach to leveraging information assets, by establishing consistent, predictable levels of data quality, breadth, and depth within a managed environment. However, the scope of the data warehouse has expanded beyond the traditional use as a reporting base. Advanced analytics in the form of Online Analytical Processing (OLAP) and data mining has greatly increased the value of the data warehouse to the corporation.

This advanced type of analytics has typically been performed by knowledge workers in the corporate office using specialized tools. While this has provided much value to the corporation, as more decision making is pushed to the line of business workers, the requirement for delivering advanced analytical applications outside of the corporate office, and sometimes outside of the corporation, has become the competitive edge.

This means delivering advanced analytic capabilities to hundreds, or even thousands, of people who are not trained in, or perhaps even interested in, the use of these specialized OLAP and data mining tools. In addition to the expertise needed, these tools are expensive at a per-seat cost and typically require some type of local client application to be installed. Paying for and managing this for hundreds or thousands of users is prohibitive.

These new users are already familiar with using applications over the Web, typically via Web browsers. Corporations have found delivering operational applications over the Web to be very effective and these line-of-business workers are very well versed in using Web browsers. The goal is to be able to deliver advanced analytic capabilities without the user really even knowing that they are performing advanced functions. They should not really even be cognizant that they are doing something different than they normally do everyday in their job. When a worker logs into the corporate portal, there should just be objects on the screen that represent tasks that they need to do, whether it be running payroll, ordering inventory, scheduling personnel, interacting with an OLAP database using interactive charts, or even performing data mining via, as an example, a Market Basket Analysis application.

This concept of integrating advanced analytics into everyday business processes is commonly referred to as Operational Business Intelligence. It is critical to have the ability to embed analytic functionality into the daily routine of a line-of-business employee with a minimum amount of effort and interruption.

Figure 1-2 on page 5 depicts some screens from a Web-based workbench for a grocery store manager. It is not necessary to read or understand all the information in that figure; it is simply a depiction of the types of information that might be made available. For example, this workbench contains access to all of the applications that the store manager needs to accomplish his daily job in one integrated workbench. Some of the functionality is delivered by traditional applications and some is delivered by analytical applications, but the store manager does not know or care, as these are seamlessly embedded and integrated into the workbench. In the example, the store manager is using the workbench to analyze some out of stock (OOS) situations that were highlighted by the key performance indicator (KPI) component on the home page. Clicking the OOS alert directs the store manager through a series of analytics to determine where the problem lies and how best to handle it, and results in a

stock reorder being generated. All of this is done seamlessly without the store manager noticing the difference between analytics and operational systems. He is just interacting with functional objects on his workbench.

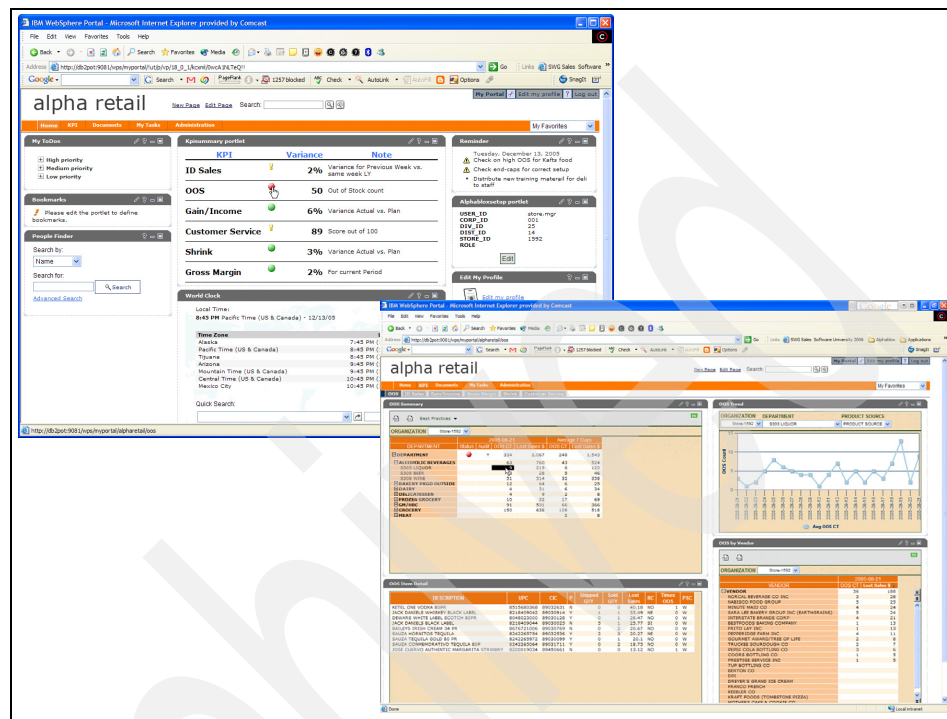


Figure 1-2 Grocery store manager workbench with embedded analytics

Embedding analytics into a company intranet is not limited to the OLAP type of analytics. For example, store managers must understand what products their clients are buying together in a specific trip to the store by analyzing what is in the basket. This can be accomplished with an association's analysis provided by data mining tools, which have traditionally been developed by some type of statistician at corporate headquarters.

However, in the business environment today, store managers need to be much more responsive. To do so, they need a capability such as a Web-based application on their workbench with which to perform their own market basket analysis. Figure 1-3 shows a data mining application that enables the grocery store manager to do data mining with a simple Market Basket Analysis application. Without even realizing that what is being performed is advanced analysis using data mining. It is not necessary to read or understand all the data on the screens, but simply to have a depiction of the types of information that can be displayed to enable the analysis.

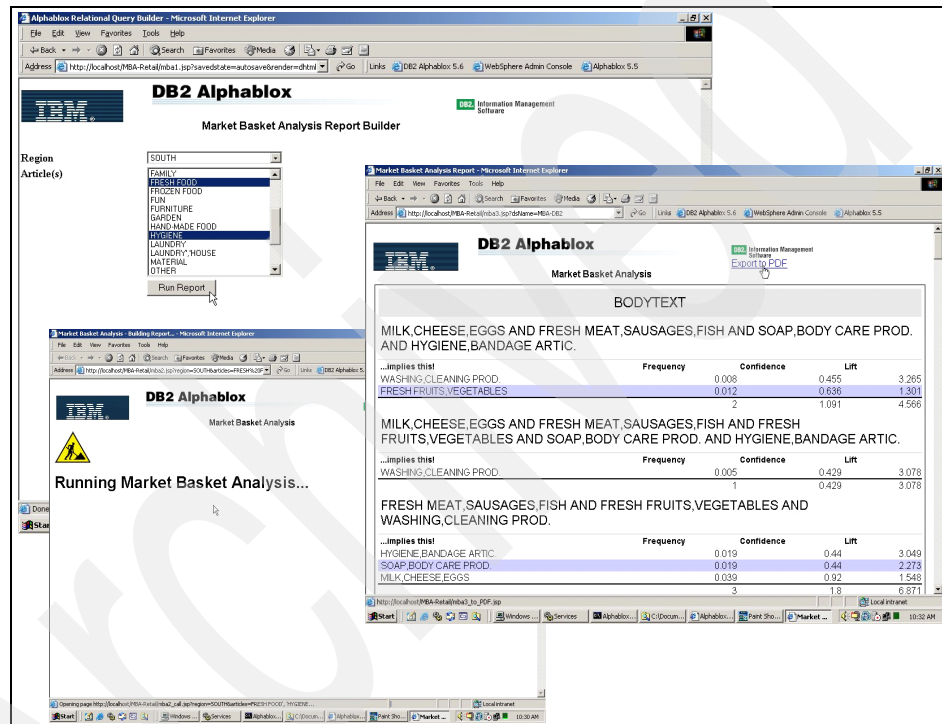


Figure 1-3 An embedded data mining application

1.4 From data warehouse to information warehouse

Now take a different view and consider how to deliver information as a service from the perspective of the enterprise data warehouse. Information as a service extends the notion of the traditional data warehouse by providing layers that apply business context to the data as well as better insights into that data. The result is the higher value notion of the information warehouse, which enables more efficient and effective decision making, and faster action to be taken to

avoid problems that could impact business performance. This is depicted in Figure 1-4.

The base of the information warehouse is an enterprise data warehouse that provides a single version, or view, of the business. It is implemented as a single, scalable, consolidated data warehouse.

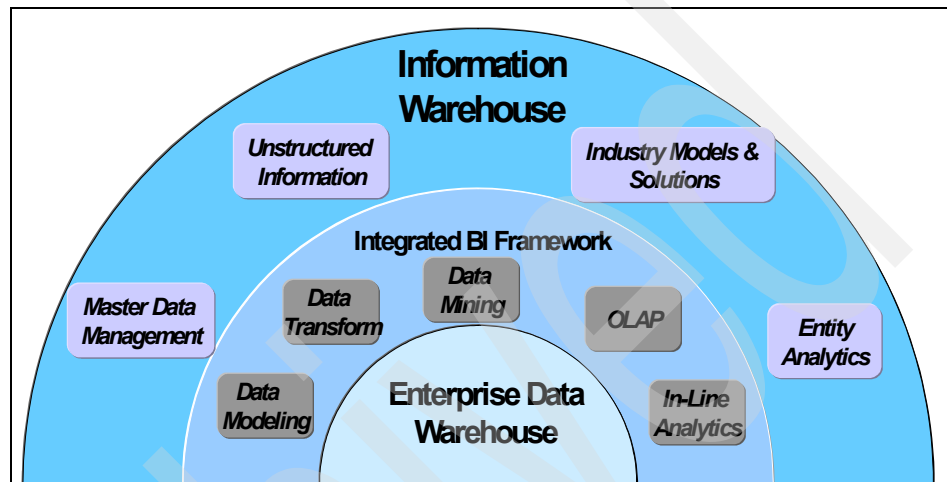


Figure 1-4 From data warehouse to information warehouse

With the foundation of the consolidated enterprise data warehouse in place, the next step is to begin enhancing the value of the data warehouse by incorporating capabilities once relegated to independent data marts. Capabilities as intra-warehouse data movement and transformation, data mining, OLAP modeling, and embedded analytics. These critical horizontal capabilities typically have been developed alongside the data warehouse, but often in different areas of the larger organization. Their lack of integration with the data warehouse impacted their value and increased the overall cost of ownership.

Data mining is a great example of this. Data mining has long had tremendous potential for increasing the value of the warehouse, but it is a complex technology that has been very loosely integrated with the data warehouse, and is not very accessible by administrators and users. Consequently, it has not been widely accepted and has not realized the real potential of data mining.

The integrated Business Intelligence framework is all about integrating capabilities such as mining, providing support to improve the quality of data it is processing and support to make better and easier use of the results. In addition, it provides integrated tooling that makes it easier to incorporate this kind of function into the larger data warehouse/application environment and makes it more consumable.

The final layer of the information warehouse takes us from the traditional data warehouse to providing a higher level of value. This is achieved by incorporating richer sources of information (such as unstructured data), applying additional business context to data (such as with master data management), extracting additional insight from data (Entity Analytics), and tying it all together in consumable solutions that solve specific business problems.

1.5 An integrated business intelligence framework

To deliver analytic functionality in a service-oriented environment requires building BI functionality into the data warehouse and making it accessible through open, standard interfaces as embeddable components in an integrated BI platform. To be able to quickly respond to the needs of the business requires that you have a business intelligence framework, or architecture, in place with an integrated data warehouse.

Components of a BI framework

A BI framework should consist of components and processes that allow you to quickly respond to the analysis needs of the company in a timely fashion. If it takes too long to provide a new analytic function, the window of opportunity may close.

Such a framework must provide for the development and delivery of advanced analytic applications containing OLAP and data mining capabilities and a standard Web-based delivery mechanism. To accomplish this some base infrastructure capabilities are needed, such as:

- ▶ Moving and transforming data within the data warehouse for populating the data structures needed for OLAP and data mining
- ▶ Modeling and implementing OLAP, or cube, structures
- ▶ Creating data mining models and applying, or scoring, the models without having to move data out of, and back into, the data warehouse
- ▶ Delivering OLAP and data mining applications via the Web in an embedded and customized approach so that these analytical applications seamlessly integrate into the existing Web-based application infrastructure

1.6 The DB2 Business Intelligence solution framework

In this section we discuss an integrated BI solution framework. This enables you to extend the value of your enterprise data warehouse, and is based on the DB2 Data Warehouse Edition, which is depicted in Figure 1-5. The primary purpose of this framework is to provide the components necessary to build and support the Business Intelligence solutions. There are components for moving and transforming data in the data warehouse and for modeling data including creating OLAP models, creating and scoring data mining models within the database engine, and the ability to develop and deliver embedded analytic components into the existing corporate Web infrastructure.

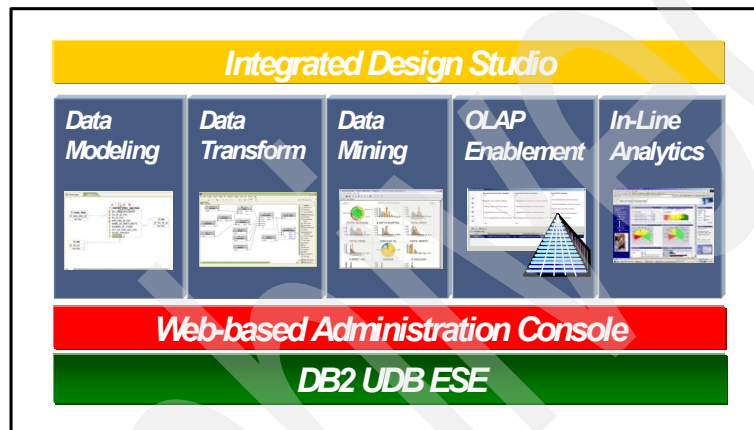


Figure 1-5 DB2 Data Warehouse Edition

DB2 Data Warehouse Edition V9.1 provides the components for building and delivering embedded Web-based analytic solutions. At the core is the DB2 UDB ESE relational database engine, which provides a robust, scalable platform for the data warehouse on which the BI framework is built.

The integrated Design Studio provides the core components to graphically model data structures, move and transform data within the data warehouse, implement OLAP, build and score data mining models, and finally the ability to develop embedded analytic application components that expose the BI services available for embedding into the company Web-based infrastructure.

The data modeling component enables the creation of physical data models and provides the base metadata to the other components. Physical data models may be developed or reverse-engineered from existing databases. This is a subset of functionality from the IBM Rational Data Architect data modeling product.

The data transformation component, called SQL Warehousing (SQW), provides the ability to develop SQL-based data movement and transformation flows within the data warehouse. It leverages the SQL functionality of the DB2 relational database engine. Data flows and control flows are developed using graphical editors within the Design Studio and leverage the physical data model metadata.

The data mining component is an extension to the SQW component and adds the ability to graphically develop data mining models using mining flows to do data preparation, model creation, model visualization, model extraction, and model scoring. Model scoring can also be embedded into data flows to allow batch scoring within the data warehouse. There are also SQL and Java™ bean APIs for embedding data mining into applications.

To support slice and dice analytics, the OLAP component enables modeling and optimization of OLAP, or cube, structures. The modeling of OLAP structures, facts, dimension, and hierarchies, as examples, is performed as an extension to the physical data model and is typically used with star-schema like structures. The OLAP component also provides wizards to optimize OLAP processing.

The In-Line Analytics component allows the development of visual analytic components, or Blox, that are embedded into, and delivered by, Web-based applications. These components can access and visualize data from OLAP structures, data mining, and relational tables.

Runtime environments are administered using a Web-based administration console. The Administration Console allows the deployment of DWE applications into the runtime environment, allows the scheduled execution of data movement flows, administration of OLAP, and data mining functions. The Web-based analytic applications can also be deployed via the provided J2EE™-compliant WebSphere Application Server.

The final component of DB2 Data Warehouse Edition proactively manages the query workload allowing a balanced use of the data warehouse data assets. This is provided by the included DB2 Query Patroller.

That is a very brief and high-level overview of the BI framework, along with the products and components to enable implementation. Additional details and examples are provided in the remainder of this book to help get you started towards your development and implementation of business intelligence solutions.

DB2 DWE - A technical overview

In this chapter we provide a brief overview of DB2 Data Warehouse Edition (DWE), from a technical perspective. This includes a brief description of the architecture, the development components, and the runtime components. The remainder of this book provides a detailed look at each of the various components of DB2 DWE.

DB2 DWE represents the IBM offering for implementing integrated Business Intelligence solutions. The BI framework enables the transformation of the data warehouse from a static repository primarily used for batch reporting into an active end-to-end platform for Business Intelligence solutions. DWE integrates design-time tooling and runtime infrastructure for OLAP and data mining, and the delivery of InLine, embedded analytics on a common platform based on the IBM DB2 Relational Database Server and the WebSphere Application Server.

The goal of DWE is to remove cost and complexity as barriers, and enable you to deliver powerful analytics to the enterprise. The integrated tooling enables a low total cost of ownership (TCO) and provides improved time-to-value for developing and delivering analytics enterprise-wide. DWE also avoids the requirement for multiple purchase justifications for BI tools and enables you to have the tools available when you need them.

2.1 DWE architecture

At the core of DWE is the DB2 relational database engine to support the development and delivery of the advanced analytics. There is design time capability and a runtime infrastructure that supports the five major functional components of DWE: Data Modeling, Data Transformation, data mining, OLAP Enablement, and Inline Analytics. There is also additional tooling and components for such tasks as managing the query workload and performance as part of the complete DWE offering. This is depicted in Figure 2-1.

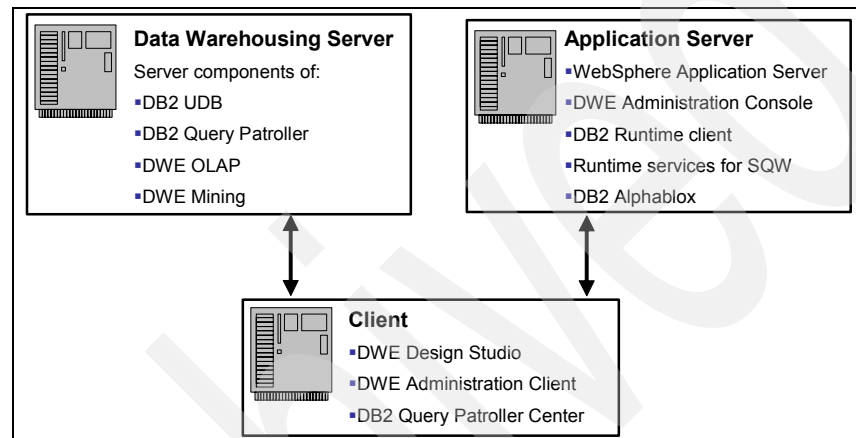


Figure 2-1 DB2 DWE Functional Component Architecture

The robust and scalable DB2 for Linux®, UNIX®, and Windows® relational database is the heart of the data warehouse. DB2 would naturally be the repository for the data in the data warehouse as well as any data structures needed for developing and delivering analytics. DB2 capabilities are leveraged to perform data movement and transformations, optimize OLAP queries, and execute data mining algorithms.

The DWE Design Studio is the primary development tool for implementing physical data models, developing data movement and transformation flows, creating and scoring data mining models, and creating and optimizing OLAP models. The DWE Admin client can post http requests and get http responses to and from the application server. Inline Analytics are developed using standard J2EE development tooling such as WebSphere Application Developer.

The runtime environment is a WebSphere Application Server application that provides Web-based administration services across the five functional components, for administering the DWE runtime system and for managing the execution of data movement and transformation flows.

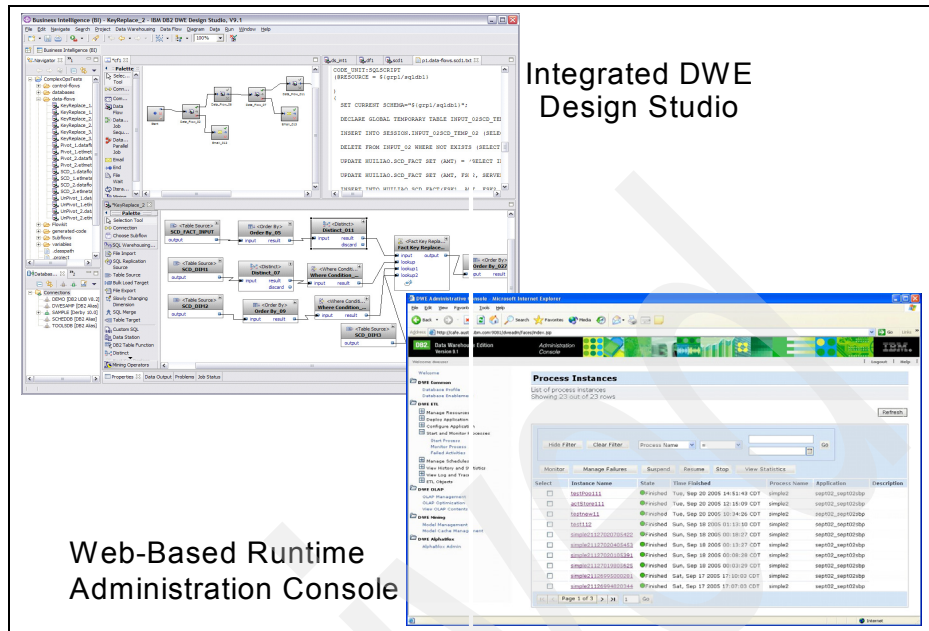


Figure 2-2 DWE Design Studio and Administration Console

The five functional components are:

- ▶ **Data Modeling** allows you to work with the physical metadata of the source and target databases, the physical data model. The physical data model is used by the other functional components.
- ▶ **Data Transforms** enables development of SQL-based, intra-warehouse data movement and transformation flows to build and maintain analytic and performance data structures.
- ▶ **Data mining** provides functions to support the creation and application (scoring) of data mining models directly in the DB2 database.
- ▶ **OLAP Enablement** enables the data model to be extended to include information pertinent to OLAP analytics, so you can define cubes over star schema type of data models, and enables optimization of query access to the star schemas.
- ▶ **Inline Analytics** enables creation of visual components and applications that utilize the analytic structures created by the other components. The analytic components can be embedded into any Java-based Web application or portal.

The DB2 Query Patroller product is included in DWE to proactively manage the query workload against the data warehouse.

2.1.1 DWE Design Studio

The DWE Design Studio is the Integrated Development Environment (IDE) for developing the components of the BI framework. It consists of a set of integrated, graphical tools for developing Business Intelligence solutions. You can use these tools to build, populate, and maintain tables and other structures for data mining and analysis of a DB2 data warehouse.

The Design Studio includes the following tools and features:

- ▶ Integrated physical data modeling, based on Rational Data Architect
- ▶ SQL Warehousing Tool for data flow and control flow design including integration points with WebSphere and DataStage ETL systems
- ▶ Data mining, data exploration, and data visualization tools
- ▶ Tools for designing OLAP metadata, MQTs, and cube models

Each tool within the DWE Design Studio has the same look, feel, and mode of operation, and there are standard explorers and views used by all of the component tools. This greatly decreases the learning curve as users move from tool to tool. Figure 2-3 shows the DWE Design Studio with the Project Explorer, the Database Explorer, standard view pages, and a graphical editor.

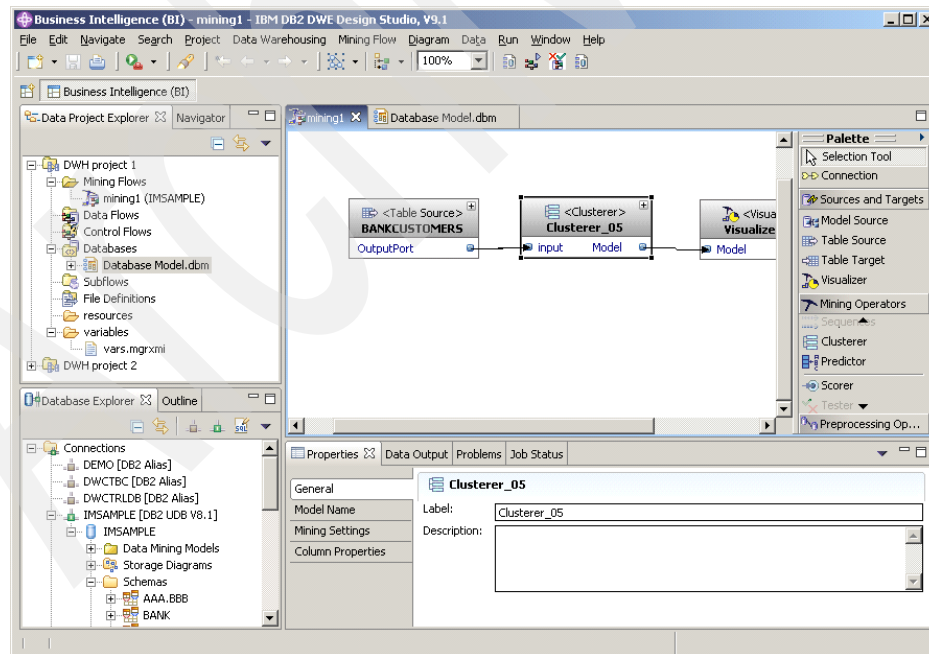


Figure 2-3 DWE Design Studio

The DWE Design Studio is based on the popular Eclipse Workbench. Eclipse is a platform of frameworks and tools that make it easy and cost effective to build and deploy software. Eclipse provides an extensible architecture based on the concept of plug-ins and extension points. This allows the DWE Design Studio to take advantage of code reuse, integration, and many other development functions that are provided by Eclipse.

Physical Data Modeling

The Physical modeling component is the subset of the IBM Rational Data Architect tool that is used to develop and maintain physical data models. Application developers and data warehouse architects use this component to work with physical data models for source and target databases and staging tables.

A physical data model is essentially the metadata representing the physical characteristics of a database. The data model can be newly developed using the data model graphical editor or can be reverse-engineered from an existing database. From the physical data model, you can create the physical objects in a database, compare a data model to the database, generate just the delta changes, and analyze the model for such things as naming violations and perform impact analysis.

The physical model is also used to provide metadata information to the other functional components of DWE, such as the SQL Warehousing Tool. Figure 2-4 shows a physical data model open in the Data Project Explorer, a graphical editor view of the physical data model, and the properties view of a selected table. It is not necessary to read or understand that data model. The figure is simply to provide a depiction of it for familiarity.

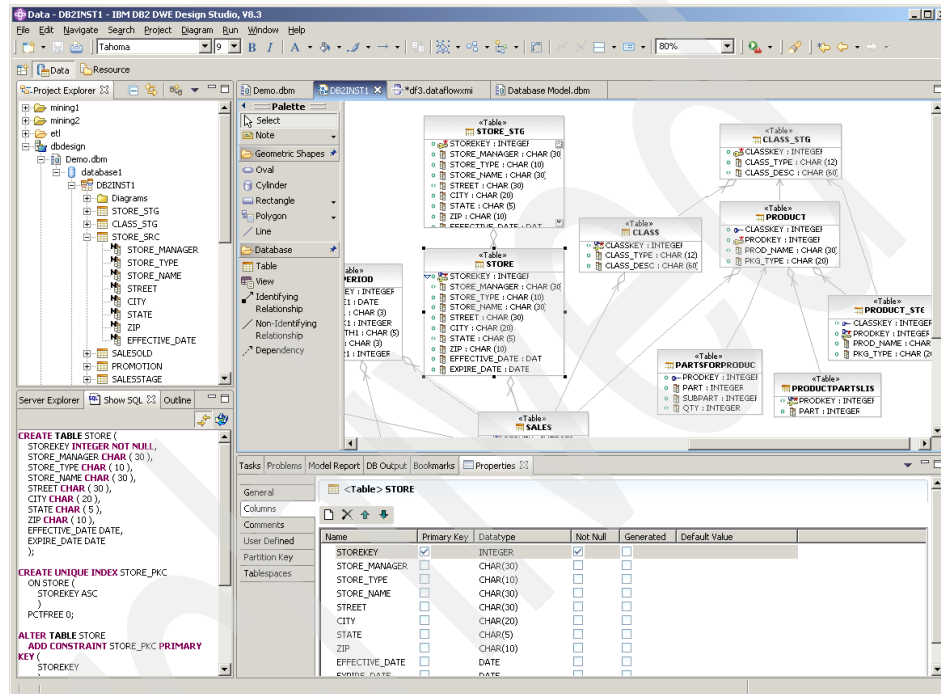


Figure 2-4 DWE Physical Data Model

SQL Warehousing Tool for data movement and transformation

The SQL Warehousing Tool (SQW) is a graphical tool that is used to develop flows for moving and transforming data within the data warehouse. The SQW tool automatically generates DB2-specific SQL based on visual operator flows that you model in the DWE Design Studio. The library of SQL operators covers the in-database data operations that are typically needed to move data between DB2 tables and to populate analytical structures, such as data mining models and multidimensional cubes. SQW complements and works well with ETL products from IBM and third parties with specific integration points for IBM WebSphere DataStage.

There are two basic types of flows in SQW. A data flow moves and transforms data using SQL-based operators to accomplish tasks such as joins, filtering,

Figure 2-5 shows a data flow open in a data flow editor, a control flow open in a control flow editor, and the generated sql of the data flow. It is not necessary to read or understand the data on these flows, but simply understand the types of information that are there. Data flows and control flows are organized under a data warehouse project.



Data mining is the process of using algorithms to analyze data and discover new insights. Historically, data mining has been performed by statisticians using specialized (and expensive) tools. These tools usually require data to be extracted from the data warehouse into flat files for processing when creating the models as well as applying, or scoring, the data models. In some companies the storage used to keep data files for mining greatly exceeds the size of the entire data warehouse.

Instead of taking the data to the mining tool, DWE brings the mining tool to the data. The creation of data mining models and the application of mining models happens within the DB2 database, acting directly upon the data without removing it from the data warehouse.

The data mining component of the DWE Design Studio has functions to explore and visualize the data to increase understanding prior to mining it. There is a graphical mining flow editor with operators for data preparation, execution of data mining algorithms, visualization of the results, and extraction of information from the model to store in tables for use by analytic applications.

Once a model is built to satisfaction, it can be applied to new data for the purpose of scoring or prediction. This can be integrated into a mining flow or a data flow for batch scoring directly within the database. Mining flows can also be integrated into SQW control flows.

Modeling and scoring functions can be called via standard SQL functions, which enable data mining modeling and scoring functions to be embedded into applications for real-time data mining. For example, a market basket analysis application can easily be developed such that a store manager, just by making a few product selections, can invoke it and see the results of data mining.

Figure 2-6 shows a mining flow open in the graphical mining flow editor, with data exploration visualizations and the data mining model visualizer. Data mining flows are also part of a data warehouse project along with data flows and mining flows. Again, with this figure, it is not necessary to read or understand the data in the figures, but simply to be aware of the types of data and their relationships.

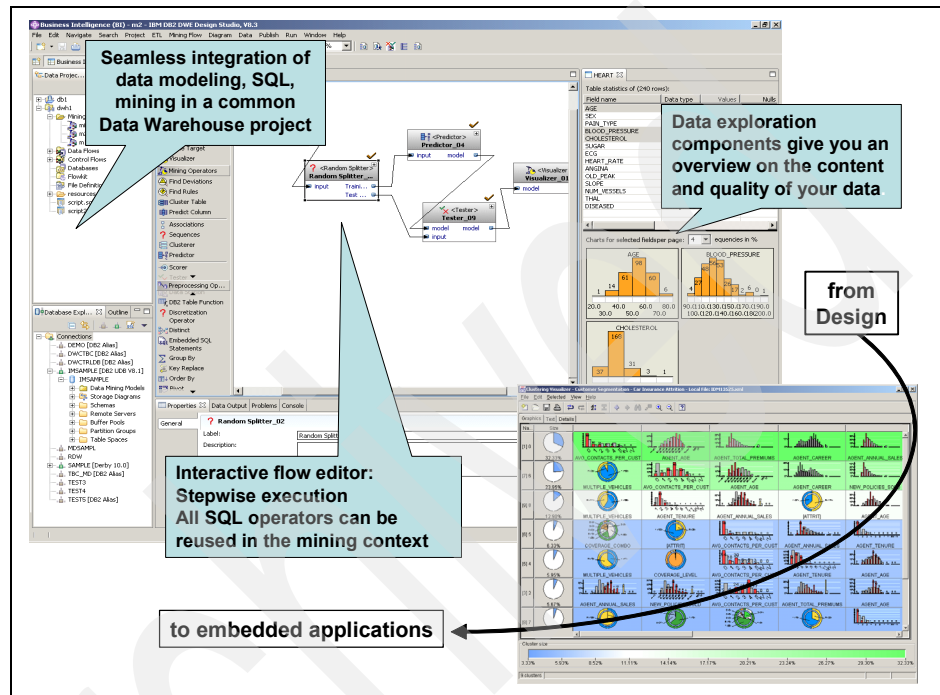


Figure 2-6 DWE Data Mining

DWE OLAP

A star schema (or dimensional model) structure is typically used to store data for analysis needs. There is a fact table that contains all of the measurable information and a set of dimension tables that specify the various ways in which the facts can be viewed, including various hierarchical relationships within a dimension. However, to the relational database engine, these are just a set of tables in the database with no special meaning.

OLAP tools are usually built around the knowledge and relationships that are special in a dimensional model. Since the relational database itself has had no special knowledge of the relationships inherent in the dimensional mode, OLAP tools have had to provide mechanisms in the tool to supply the additional meaning.

The DWE OLAP tooling extends the relational database model to include the dimensional model relationships. This metadata is stored once in the relational database and can be used by DB2 Alphablox and other IBM partner BI tools.

The DWE Design Studio OLAP component extends the physical data model to include the dimensional model and is stored in cube definitions. Cube definitions include information about the facts, the dimensions, and the hierarchies in the dimensional model. This dimensional, or OLAP, metadata is available for use by BI tools and is used by DWE Inline Analytics (DB 2 Alphablox). Because the shared common metadata includes aggregation formulas and calculations, you benefit from greater consistency of analytical results across the enterprise.

Once a BI tool has the dimensional knowledge about the underlying star schema structure, it can use that information to form SQL queries to retrieve the data. Depending on the granularity of the data and the granularity of the SQL query the SQL may cause quite a bit of work to be done by DB2 to join, sort, and aggregate data to the requested level. To help optimize the performance of OLAP type queries, the DWE OLAP component also provides an optimization wizard to make model-based recommendations to build performance structures called Materialized Query Tables (MQTs). These tables pre-aggregate the most commonly requested data slices to be used by the DB2 Optimizer and transparently reroute queries to the pre-aggregated data, which greatly accelerates access to the data.

Figure 2-7 shows a star schema data model, as viewed in DWE OLAP. It depicts the OLAP (cube) metadata extension to the physical data model and the OLAP cube as deployed to a physical database. It is to give you an understanding of the type of data presented and its placement.

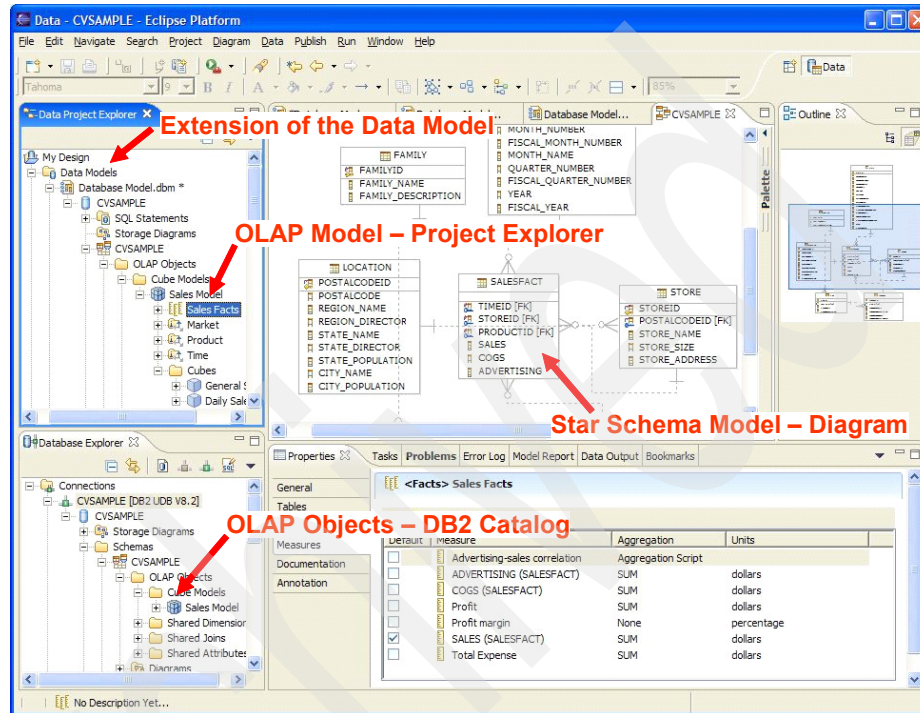


Figure 2-7 DWE OLAP

2.1.2 InLine Analytics using DWE Alphablox

DWE Alphablox provides the ability to rapidly create custom, Web-based analytic applications. These applications can fit into the corporate infrastructure and have the ability to reach users both inside and outside of the corporation. Applications built with DB2 Alphablox run in standard Web browsers, enabling the execution of real-time, highly customizable multidimensional analysis in a Web browser.

DWE Alphablox accomplishes this by providing a set of pre-built components, or Blox, that can be incorporated via a tag language into JSP™ Web pages or portlets. These Blox are essentially calls to an application hosted by a J2EE-compliant application server, such as the WebSphere Application Server, which will query the DB2 OLAP cube via the Alphablox Cube Server. Figure 2-8 shows Alphablox Blox embedded into a JSP page, and as portlets.

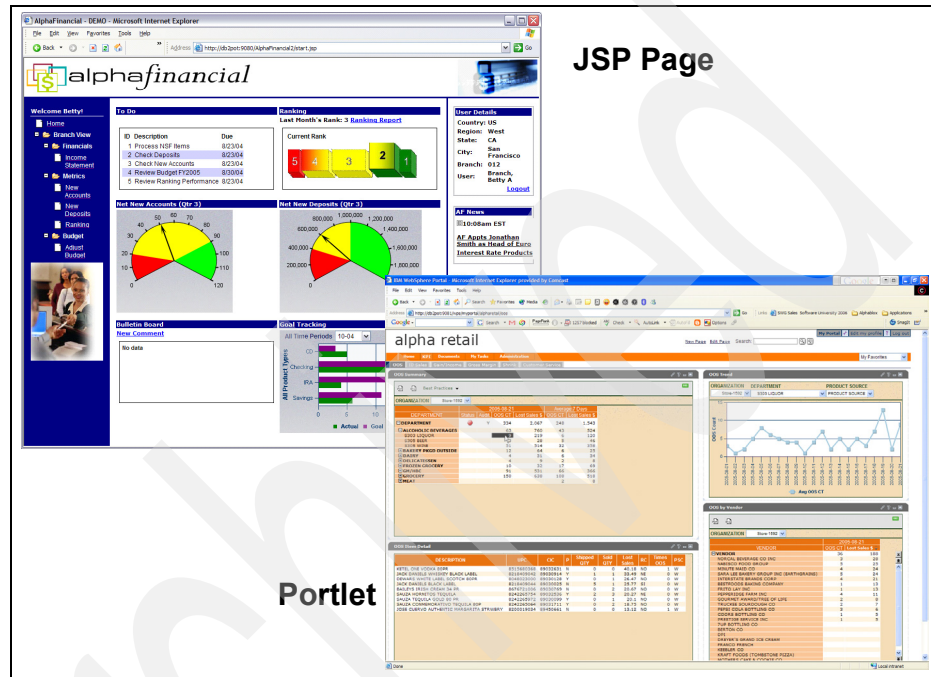


Figure 2-8 Embedded Alphablox blox

There are two components of the DWE Alphablox tooling. The Alphablox Cubing Engine (or cube server) uses the metadata created by the DWE OLAP component, and stored in the database, to instantiate a cube that can be queried by the embedded Blox. Blox tags are created to make the call to the Alphablox Cubing Engine to request data. Alphablox Blox can be embedded into Web pages,

or portlets, using a Java Server Page or Portal development tool such as the Rational Application Developer. Figure 2-9 shows how the Alphablox components use the OLAP definitions to instantiate the cube and how the Alphablox Cubing Engine translates MDX queries from the Blox into SQL queries for accessing the data.

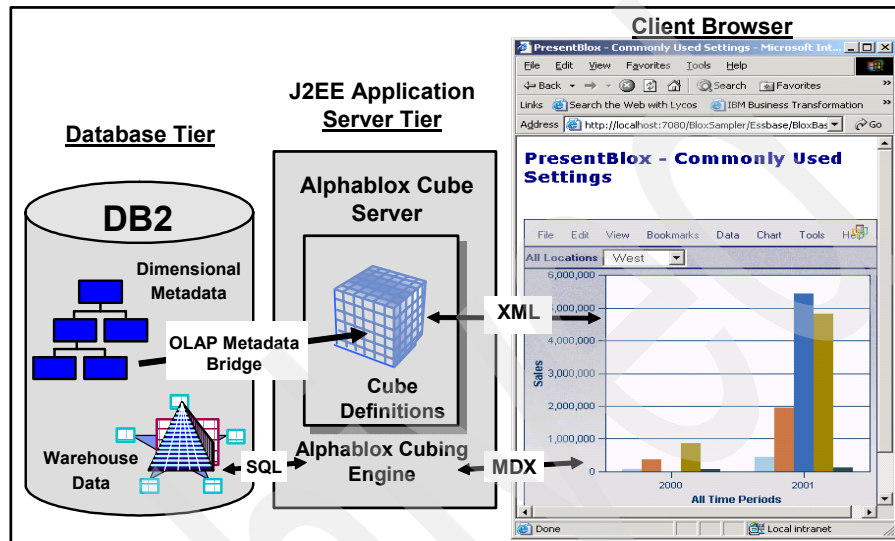


Figure 2-9 Alphablox architecture

2.1.3 DWE production environment

The DWE runtime infrastructure components provide the tools needed to manage test and production environments. It consists of a Web-based Administration Console, a runtime engine, and a metadata database. Refer to Figure 2-10 on page 24 while reading this section, which depicts this environment.

The DWE Administration Console is the Web-based user interface to configure and manage a DWE runtime environment. It is a WebSphere application that provides a single tool for administration functions across the functional components:

- ▶ The DWE Common functions are used to manage database connection profiles for OLAP and data mining and to manage the enablement of OLAP and data mining features in those databases.
- ▶ The DWE SQL Warehousing functions are the largest set of functions in the DWE Administration Console. There are functions to create and manage data sources and system resources used in warehouse applications; functions to

deploy and manage warehouse applications; and functions to manage, run, schedule, monitor, troubleshoot, and view execution statistics of processes.

- ▶ The DWE OLAP functions are used to import, export, and view OLAP metadata; run the Optimization Advisor; deploy the recommended MQTs; and analyze MQT usage.
- ▶ The DWE Data Mining functions are used to import, export, manage, and view mining models and manage the mining model cache used to keep mining models in memory for real-time scoring applications.
- ▶ The DWE Alphablox function provides a bridge to the Alphablox Administration pages.

The SQW runtime server, called Data Integration Service (DIS), manages the execution of SQW processes and remote database connections. Processes may be executed on demand or scheduled via the DWE Administration Console. DIS will gather the required information from the metadata database and run the task utilizing the DB2 execution database, a DataStage server, or perhaps the local operating system as appropriate. DIS monitors the execution of the task and gathers runtime and completion information and stores it in the metadata database.

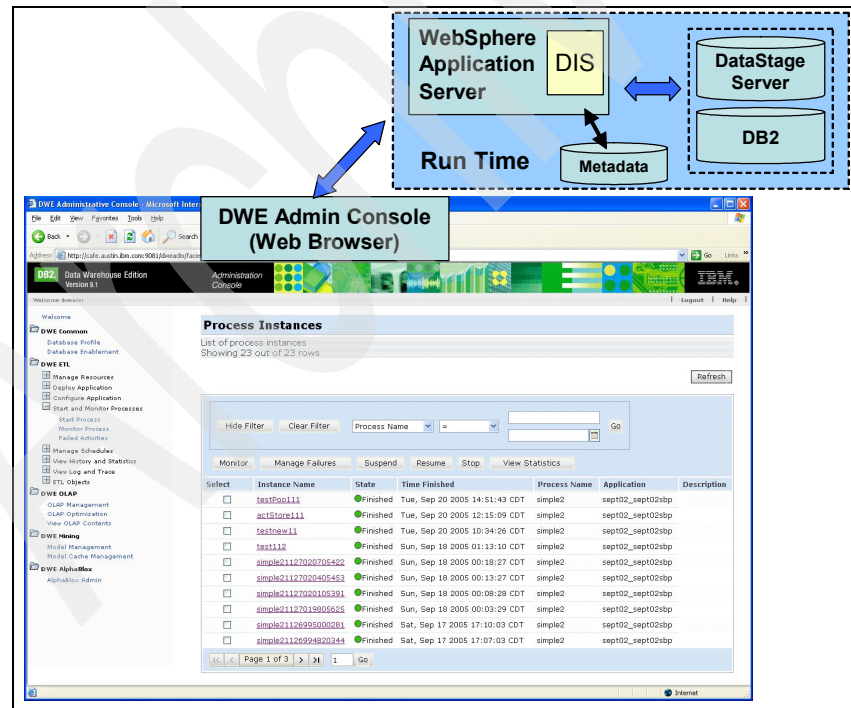


Figure 2-10 DWE runtime architecture

2.1.4 DB2 Query Patroller

DB2 Query Patroller uses the predictive query costing of the DB2 Optimizer to help manage the data warehouse workload. DB2 Query Patroller matches projected resource costs to predefined user profiles, system thresholds, and queue criteria, and dynamically routes queries accordingly. DB2 Query Patroller can be used to dynamically control the flow of queries to the DB2 UDB database.

With DB2 Query Patroller, the DB2 query workload can be regulated so that small queries and high-priority queries can run promptly, and ensures that the system resources are used efficiently. In addition, information regarding the completed queries can be collected and analyzed to determine trends across queries, and identify heavy users and frequently used tables and indexes.

2.2 DWE topology

DWE components are grouped into four categories for the purpose of installation:

- ▶ **Clients:** The client category includes the DWE Administration Client, the DWE Design Studio, the optional V8 OLAP Center, the Query Patroller Center, and the data mining visualization programs. Clients are supported on Windows 32-bit systems.
- ▶ **Database servers:** This category includes the DB2 database server with DPF support plus the database functions to support DWE Intelligent Miner, DB2 Cube Views, and DB2 Query Patroller. The database server is supported on AIX®, various flavors of Linux, and Windows Server® 2003.
- ▶ **Application servers:** The application server category includes the WebSphere Application Server, DB2 Alphablox server components, and DWE Administration Console server components.
- ▶ **Documentation:** This category includes the PDF and online versions of the manuals and can be installed with any of the other categories.

These categories are depicted in Figure 2-11. For more details and current information about DWE Installation see the *DWE Installation Guide*, GC18-9800.

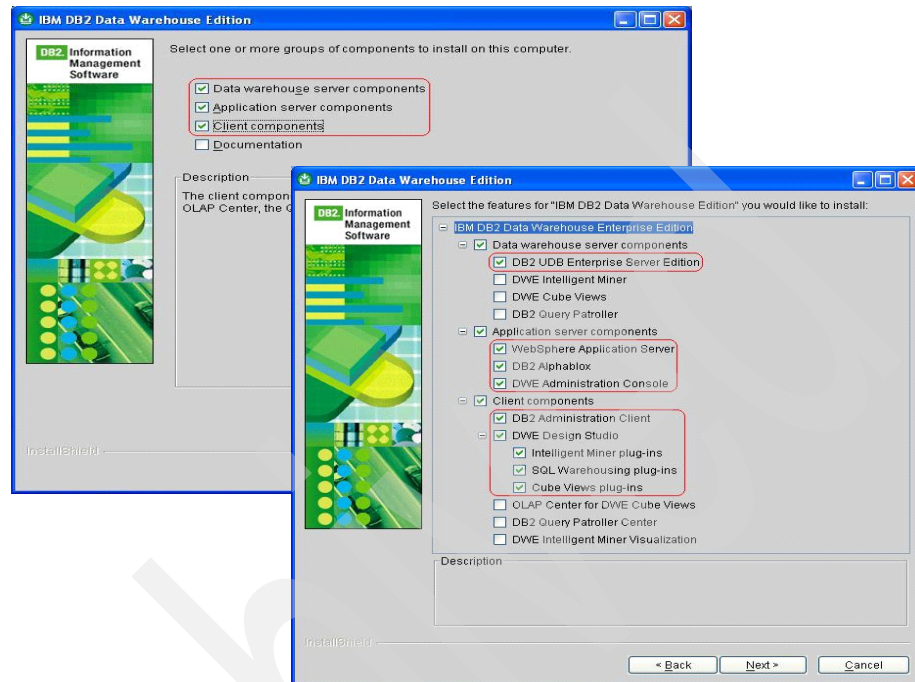


Figure 2-11 DWE installation categories

The DWE components can be installed on multiple machines in a number of topologies. There are three common topologies that are typical, as depicted in Figure 2-12.

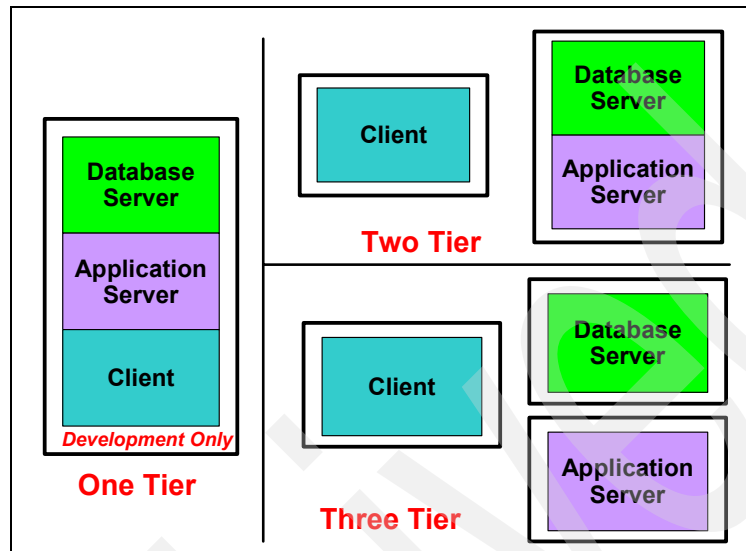


Figure 2-12 Common DWE installation topologies

The three common topologies are:

- ▶ One tier: Here the DWE Client, DWE Database Server, and the DWE Application Server are all on one system. This would only be used for development and testing purposes and only on a Windows platform.
- ▶ Two tier: The DWE Database Server and the DWE Application Server are on one system with DWE Clients on separate systems. This could suffice for a test system or for smaller installations. The Database and Application Servers could be any of the supported Windows, Linux, or AIX platforms.
- ▶ Three tier: In any large installation, the DWE Database Server, the DWE Application Server, and the DWE Clients should all be installed on separate servers. A DB2 client, at a minimum, is required to connect to database servers. We recommend installing a DB2 server for local access to the runtime metadata databases. The application server is supported on AIX, Linux, and Windows Server 2003.

The DB2 DWE Design Studio

The DB2 DWE Design Studio provides a platform and a set of integrated tools for developing Business Intelligence (BI) solutions. You can use these tools to build, populate, and maintain tables and other structures for data mining and OLAP analysis in the context of a DB2 data warehouse. The Design Studio includes the following tools and features:

- ▶ Integrated physical data modeling, based on Rational Data Architect
- ▶ SQL Warehousing Tool for data flow and control flow design
- ▶ Data mining, exploration, and visualization tools
- ▶ Tools for designing OLAP metadata, MQTs, and cube models
- ▶ Integration points with WebSphere DataStage ETL systems

By integrating these tools, the Design Studio offers time to value and managed cost for warehouse-based analytics. In this chapter we introduce the DB2 DWE Design Studio, including such topics as:

- ▶ The Eclipse Platform
- ▶ The Design Studio Workbench
- ▶ Exploring data
- ▶ Designing physical database models
- ▶ Designing OLAP objects
- ▶ Designing and deploying SQL Warehousing data and control flows
- ▶ Designing and deploying data mining flows

The DWE Design Studio supports the Business Intelligence solutions development life cycle.

The BI solutions development life cycle

When developing BI solutions, one typically follows a common sequence of steps or functions to design, build, and populate the data structures that are required. These functions are typically performed by people in specific roles. For example:

- ▶ Data architect: The data architect models the database schemas that are needed to support the analytical solution.
- ▶ Warehouse administrator: The warehouse administrator performs tasks such as creating the tables and ETL or data movement processes or flows to populate the data structures.
- ▶ OLAP developer: The OLAP developer models and creates the OLAP metadata.
- ▶ Application developer: The application developer builds and delivers front-end analytical applications to the business owners within the corporation, utilizing analysis tools such as IBM DB2 AlphaBlox.
- ▶ Data steward: The data steward administers the entire warehousing environment.

We call this entire sequence the BI solutions development life cycle, which is shown in Figure 3-1. This life cycle is an iterative process, further enhancing the analytic capabilities of the data warehouse by extending the models to incorporate additional business data to be analyzed.

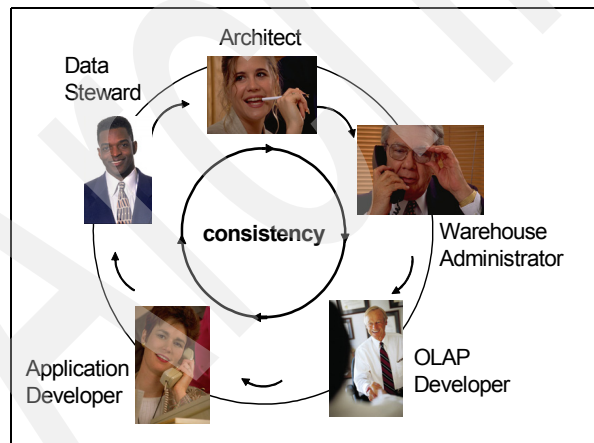


Figure 3-1 The Business Intelligence solutions development life cycle

Historically, these steps are often executed by different people using different tools, which may even be from different software vendors. The use of un-integrated tools makes transitioning between each phase in the life cycle

difficult, and requires a great deal of coordination and communication. These factors have a negative impact by prolonging the development cycle.

The DWE Design Studio delivers a great deal of consistency to the BI solution development process by integrating the tooling that supports this life cycle. This simplifies the transitions between tasks and helps BI teams deliver solutions to their users more quickly and easily.

By integrating your information assets and applying real-time analytics to turn your information into intelligence, your DB2 DWE data warehousing environment can enable your company to deliver information on demand — selective, transparent access to distributed data sources.

In the remainder of this chapter we discuss the DWE Design Studio in detail. Design Studio is based on the Eclipse platform, which is a powerful development environment. We provide background on Eclipse and then delve into the Design Studio user interface and common tasks.

3.1 The Eclipse platform

The DB2 DWE Design Studio is based upon the open source Eclipse platform. Eclipse is an open source community of companies that focus on providing a universal framework for tools integration. The Eclipse consortium was founded in late 2001, and has now grown to over 80 members, including many industry-leading software vendors. The Eclipse platform provides a powerful framework and the common GUI and infrastructure required to integrate tools together. The platform is extended by installing plug-ins that are developed by tools providers to provide specific features.

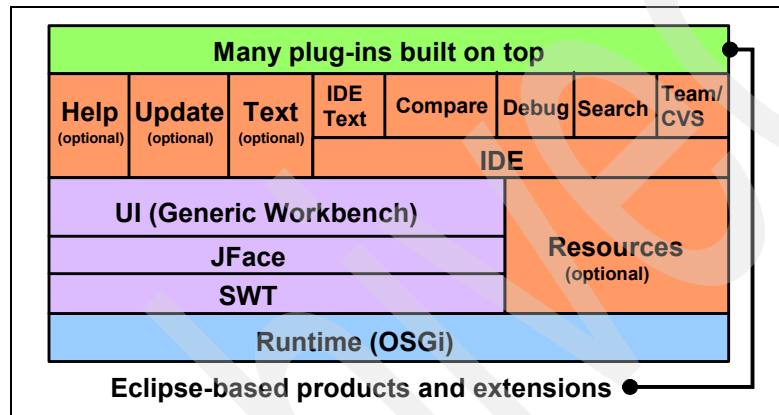


Figure 3-2 Eclipse basic platform

The basic architecture of Eclipse provides numerous services that tools developers would have to write if they did not use the Eclipse platform. Eclipse has a rich infrastructure, shown in Figure 3-2, including such components as a runtime environment, a generic user interface, and a help system. Tools vendors that utilize Eclipse are able to develop their products quickly. They are able to focus on their core competency and only need to build the features that comprise their specialty. The additional capabilities that the tools vendors provide are delivered as a plug-in to Eclipse. The plug-in is installed into an existing Eclipse environment. The DB2 DWE Design Studio demonstrates this point. Each of the capabilities that Design Studio provides are packaged together and are installed on top of the basic Eclipse platform.

Note: You do not need to download and install Eclipse before installing DWE Design Studio. The Eclipse base product has been packaged along with Design Studio so that the Eclipse platform is installed as part of the Design Studio installation process.

Users of Eclipse-based tools enjoy many benefits, including:

- ▶ A rich user experience that is common across all Eclipse-based products such as DWE Design Studio, WebSphere development tools including WebSphere Business Modeler, and the suite of Rational tools
- ▶ A wide array of instructional resources on the Internet that explain how to extend the Eclipse platform or write tools for it
- ▶ A broad selection of third-party tools that have already been developed and are available to be installed into DWE Design Studio

For more information about Eclipse and its community, go to:

<http://www.eclipse.org>

3.2 Introduction to DB2 DWE Design Studio

In this section we provide information to help you familiarize yourself with the DWE Design Studio interface. The Workbench is the main interface to Design Studio. But before you can begin using the Workbench, there are some basic concepts that you must know and understand.

3.2.1 The workspace

Every time the DB2 DWE Design Studio is launched, you will be prompted to provide a path to the workspace, as shown in Figure 3-3. A workspace is a collection of resources and is the central repository for your data files.

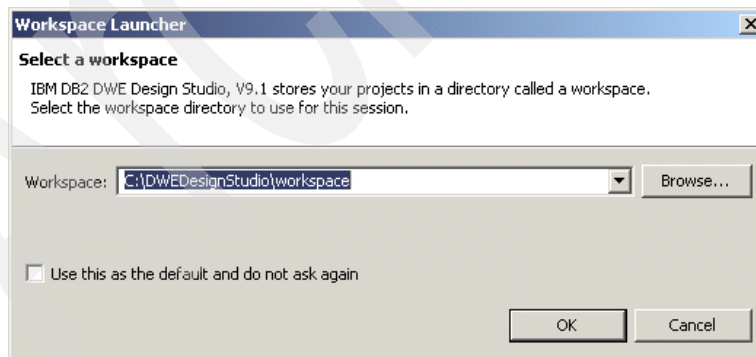


Figure 3-3 The prompt for the workspace location

DWE Design Studio, like other Eclipse-based tools, helps you manage various resources, which take the form of projects, folders, and files.

A project is a container used to organize resources pertaining to a specific subject area. The workspace resources are displayed in a tree structure with projects containing folders and files being at the highest level. Projects may not contain other projects.

You may specify different workspaces for different projects, but only one workspace is active per running instance of the DB2 DWE Design Studio. To change workspaces in order to gain access to other projects, choose **File → Switch Workspace**. A workspace may hold multiple projects.

If you specify a local directory for your workspace, we recommend that this directory be backed up on a regular basis. Another option is to utilize teaming software such as Concurrent Versions System or Rational ClearCase. Refer to 3.3.5, “Team component” on page 55, for more information.

3.2.2 Projects and the local file system

When you create a new project, you will find a new subdirectory on disk, located under the workspace directory that was specified at start-up. Within the project directory there is a special file called `.project`. The `.project` file holds metadata about the project, including information that can be viewed by selecting the Properties view within Design Studio. Inside the project subdirectory you will also see all of the files and folders that have been created as part of the project. The file names and content are the same, whether accessed from the file system or via the Design Studio. You will also see a folder called `.metadata`, located under the workspace directory, at the same level as the projects that are part of that workspace. The `.metadata` directory holds platform-specific information, including workspace structure information. The contents of this directory should never be altered or manipulated outside of the Design Studio API.

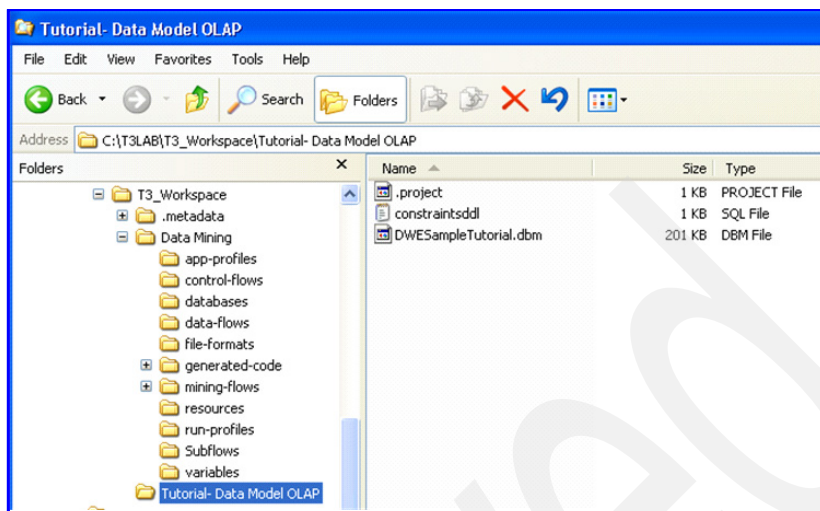


Figure 3-4 Project structure on local file system

The project type controls or determines the kinds of objects that are available for you to work with. In the DB2 DWE Design Studio, the data design project (OLAP) project type allows you to work with physical data models and OLAP objects. The Data Warehousing Project provides objects such as SQL Warehousing objects, including data flows, control flows, and mining flows.

3.2.3 The Welcome page

The very first time that the Design Studio is started in a new workspace, the Welcome page will be displayed (Figure 3-5). The Welcome page contains links to general information, the help system, recorded demos, and tutorials about DB2 DWE and the Design Studio. DWE also provides sample projects that you can work through to become familiar with the workbench and the steps involved in creating data models, data and control flows, and data mining flows.



Figure 3-5 The Welcome page for DWE Design Studio

Click the Go to the Workbench link (an arrow displayed in the top right corner) or close the Welcome page to launch the workbench. Once you have launched the workbench, you can re-display the Welcome page at any time by selecting **Help → Welcome**.

3.3 The DB2 DWE Design Studio Workbench

The term *workbench* refers to the desktop development environment. The workbench delivers the mechanism for navigating the functions provided by the various Design Studio plug-ins. The workbench offers one or more windows, which contain one or more perspectives, views, and editors, allowing you to manipulate the resources within your project. The default workbench for DB2 DWE Design Studio is shown in Figure 3-6. Important aspects of the interface are highlighted in the figure. What you will see within your own workbench environment may vary, based upon which element has focus.

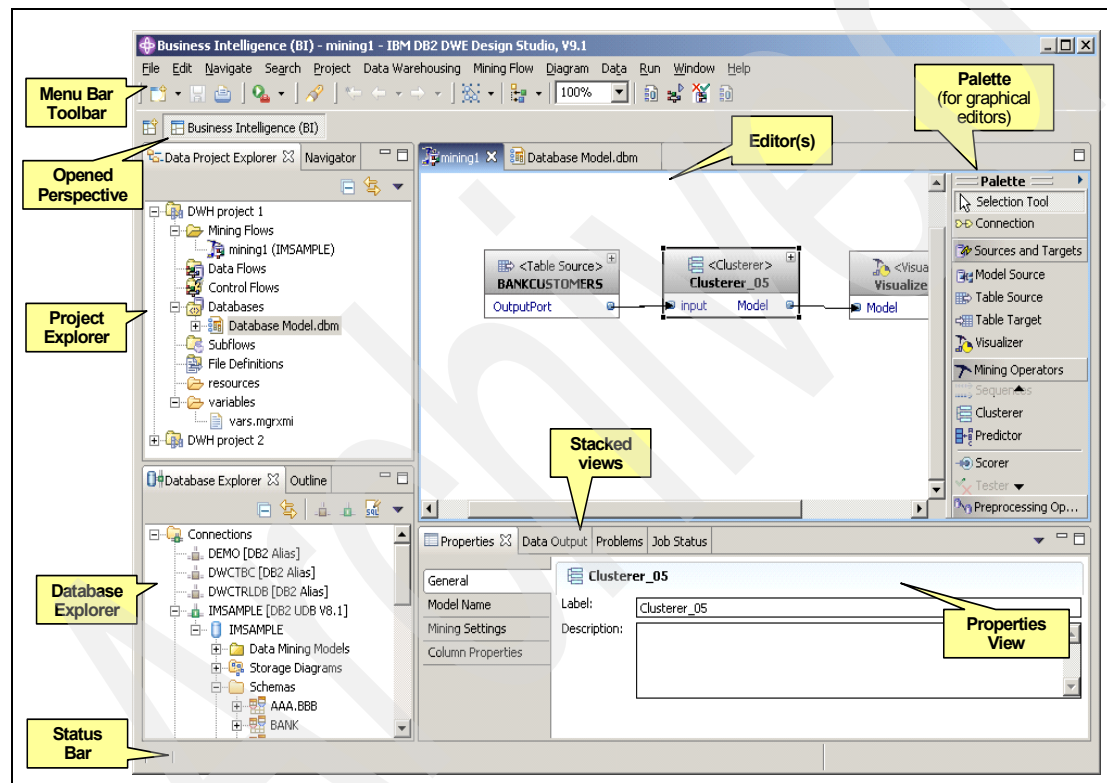


Figure 3-6 The DWE Design Studio workbench

3.3.1 Perspectives

Perspectives define an initial layout of views and editors within a workbench window. Perspectives provide the functionality required to accomplish a particular task or work with a particular resource. Perspectives also control what options may appear in menus and task bars. Perspectives can be customized or

modified and then saved for reuse by selecting **Window** → **Save Perspective As**. If you have rearranged views or closed views, you may easily reset the perspective by choosing **Window** → **Reset Perspective**. There are a number of perspectives within the DWE Design Studio, but primary perspectives with which you will work include:

- ▶ **Business Intelligence (BI):** The BI perspective, which is the default perspective for the DWE Design Studio, includes functions that are tailored for building information warehouses and enabling warehouse-based analytics such as OLAP and data mining.
- ▶ **Data:** The data perspective provides physical data modeling functions, such as the ability to reverse engineer from existing data structures, compare data objects, and analyze models against a set of enterprise rules and standards.
- ▶ **Team/CVS:** These perspectives are used for source repository management functions such as synchronization and version control.

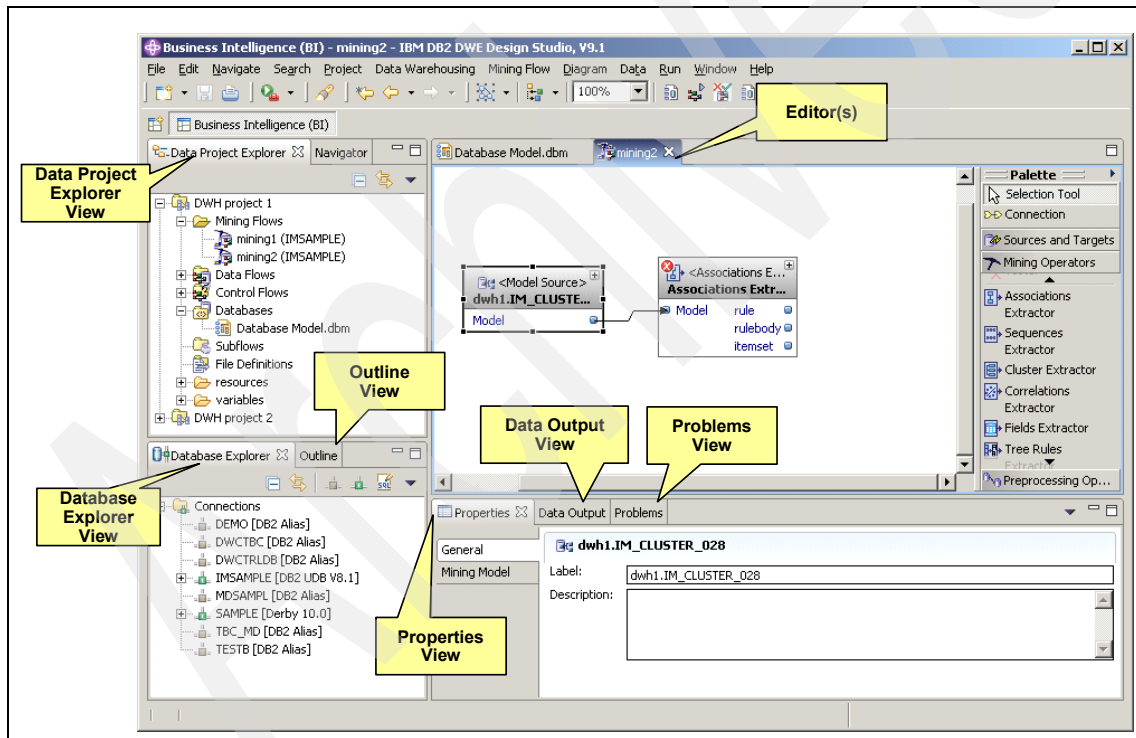


Figure 3-7 The Business Intelligence perspective

To change to a new perspective, select **Window** → **Open Perspective** or click the Open Perspective button on the shortcut bar on the left side of the Workbench window, as shown in Figure 3-8. The perspective that is currently opened is always reflected within the title bar of the Design Studio workbench window. An icon is also added to the shortcut bar to enable quick switching between the perspectives that you have opened.

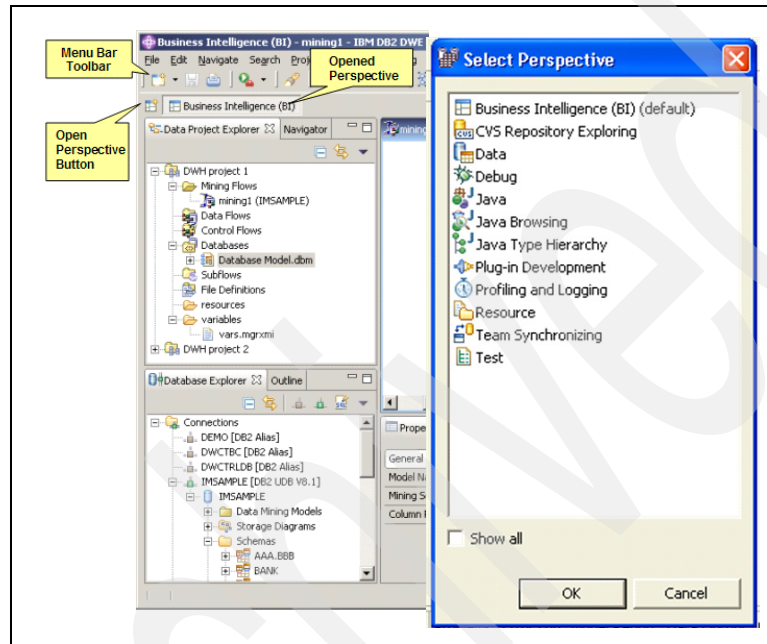


Figure 3-8 Switching between perspectives

3.3.2 Editors

Within the DB2 Design Studio, there are different editors available for different types of files. Text and SQL editors are provided for resources such as SQL scripts, and diagram editors are available for resources such as data models. An editor is a visual component that you typically use to edit or browse a resource. Modifications that you make in an editor are not always automatically saved. It is a good practice to explicitly save your changes. Tabs in the editor area reflect the names of the resources that are open for editing. An asterisk (*) by the name of

the resource in the tab indicates that there are unsaved changes to that resource. The border area on the left margin of the editing window may contain icons that indicate errors and warnings, as can be seen in Figure 3-9.

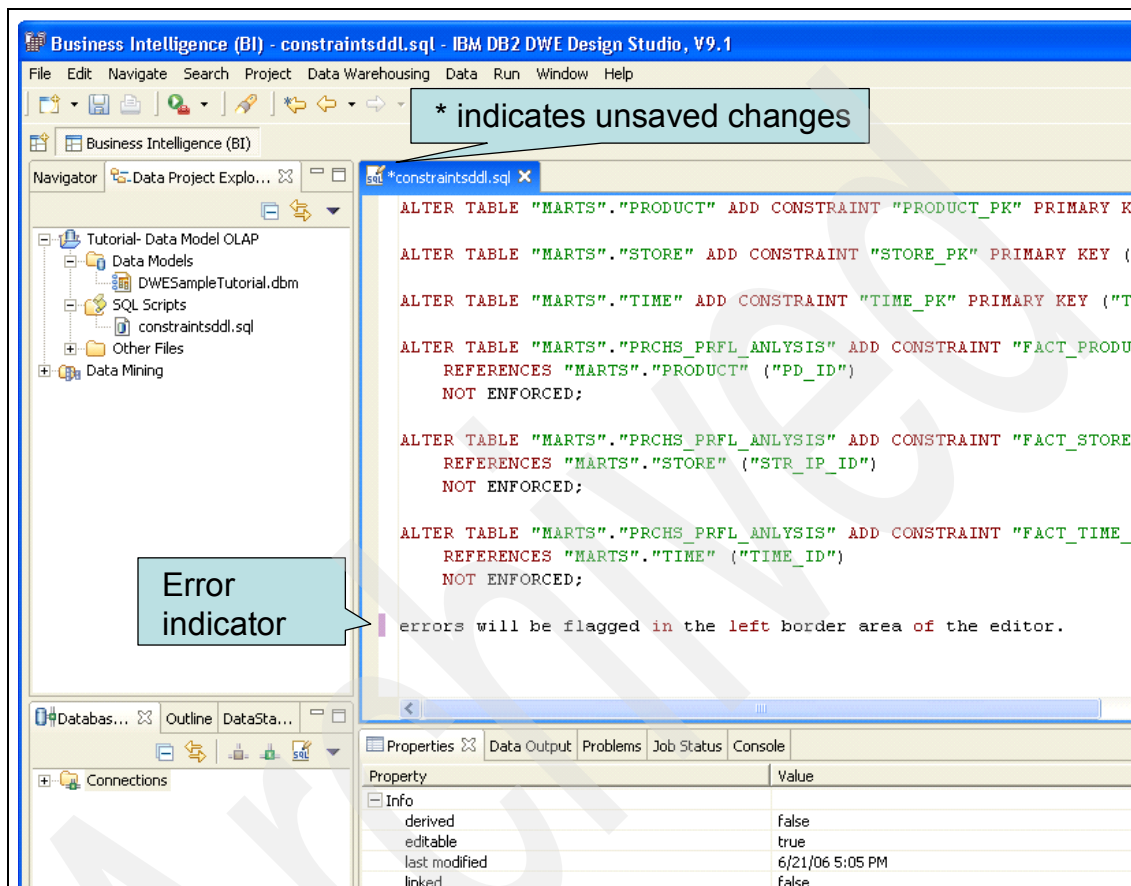


Figure 3-9 SQL Editor within DB2 DWE Design Studio

The editors that are available in the BI perspective depend on the type of object that you are working with. The editors usually include a customized palette located to the right of the canvas.

The editors that are available within Design Studio to work with data warehouse objects include:

- ▶ Physical data model editor
- ▶ SQL Scripts editor
- ▶ Data flow editor
- ▶ Control flow editor
- ▶ Data mining flow editor

3.3.3 Views

A view is a component that you typically use to navigate a hierarchy of information, open an editor, or display properties for the active editor. Modifications that you make in a view are saved immediately.

As mentioned earlier, perspectives, which are combinations of views and editors, may be arranged on the screen to your liking. Views may be docked and stacked by grabbing the view's title bar and dragging from one area of the UI to another. As the view is dragged around within the workbench window, you will notice a drop cursor that reflects where the view will be docked, as shown in Figure 3-10.







| | |
|---|--|
|  | Dock above: If the mouse button is released when a dock above cursor is displayed, the view will appear above the view underneath the cursor. |
|  | Dock below: If the mouse button is released when a dock below cursor is displayed, the view will appear below the view underneath the cursor. |
|  | Dock to the right: If the mouse button is released when a dock to the right cursor is displayed, the view will appear to the right of the view underneath the cursor. |
|  | Dock to the left: If the mouse button is released when a dock to the left cursor is displayed, the view will appear to the left of the view underneath the cursor. |
|  | Stack: If the mouse button is released when a stack cursor is displayed, the view will appear in the same pane as the view underneath the cursor. |
|  | Restricted: If the mouse button is released when a restricted cursor is displayed, the view will not dock there. For example, a view cannot be docked in the editor area. |

Figure 3-10 Drop cursor behavior

Views may also be closed by clicking the **X** located on the right side of its title bar. Once closed, a view may be re-displayed by clicking the menu option **Window** → **Show View** and selecting the view you wish to re-display. In order to maximize a view or editor, double-click its title bar. Double-clicking the title bar of a maximized view or editor will return it to its original size.

Several views are available in the BI perspective. You will work with the following views in the BI perspective most often.

Data Project Explorer view

By default, this view opens in the upper left area of the Design Studio, as shown in Figure 3-11. The Data Project Explorer shows a logical representation of the currently opened projects. The representation is logical, because the name of the folders and resources represented here need not match the real files and directories on the file system. For example, an SQLW data flow is physically made up of two files, but is represented in the Project Explorer as one single node.

With this view you can navigate your projects and the objects in your projects. You will work with this view most often to make changes to your objects. Many actions can be triggered from the Project Explorer by right-clicking an element in the tree.

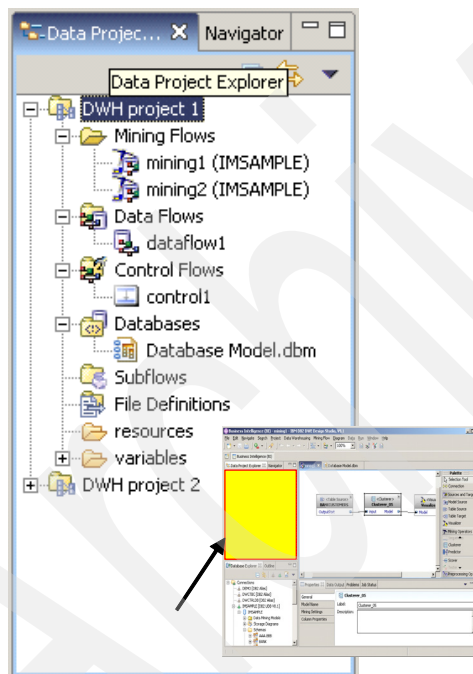


Figure 3-11 Data Project Explorer view

Resource Navigator view

The Resource Navigator shows a physical representation of the files and directories of the opened projects on the file system. While you will typically work with the logical Data Project Explorer, there are some functions that can only be performed within the Resource Navigator. For example, the ability to copy files or

project elements such as data flows from one project to another is enabled with this view.

Database Explorer view

By default, the Database Explorer opens in the lower left area of the Design Studio. This view enables you to connect to and explore a database, as seen in Figure 3-12 on page 44. You can make only the changes to the database that you have the proper authority and privileges to complete.

The Database Explorer view is used to:

- ▶ Create JDBC™ connections to databases.
- ▶ Explore database content including schemata, table relationships and content, and value distributions.
- ▶ Generate storage overview diagrams from databases, and overview diagrams of schemas using either Information Engineering (IE) notation or Unified Modeling Language (UML) notation. Diagrams are a helpful way to visualize data warehousing projects.
- ▶ Reverse engineer databases.
- ▶ Compare database objects.
- ▶ Analyze a database or a schema to ensure that it meets certain specifications. Model analysis helps to ensure model integrity and helps to improve model quality by providing design suggestions and best practices.
- ▶ Analyze the impact of changes to models. You can also use the Impact Analysis features to find dependencies. For example, if you want to copy a schema from the Database Explorer to the Data Project Explorer, you can find dependencies on the schema to ensure that all references are resolved. You can analyze data objects in the Database Explorer, the Data Project Explorer, or in the data diagram.
- ▶ Create new database objects.
- ▶ Drag and drop database objects to the Data Project Explorer.

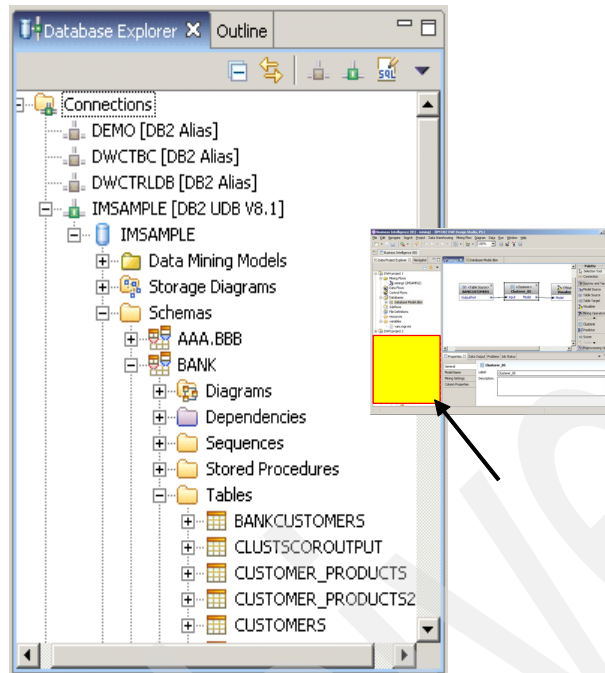


Figure 3-12 Database Explorer view

Outline view

The Outline view shows an overview of the structural elements of the file that is currently open for editing. The contents of this view depend upon the nature of the edited file. When flows or diagrams are edited, the outline offers two representations: a tree representation where objects composing the flow can easily be selected and a graphical representation showing the whole flow or the whole entity relationship diagram.

The Outline view is particularly helpful when you are working with large flows or diagrams. You can use the Outline view to quickly find your location in a large diagram. The Outline view shows the entire diagram with a small gray box that represents the viewing area. You can drag the gray box to display the portion of the diagram that you wish to work on. Figure 3-13 shows both the tree and the graphical representation. The Outline view is stacked with the Database Explorer view. Click its identifying tab to bring it to the forefront.

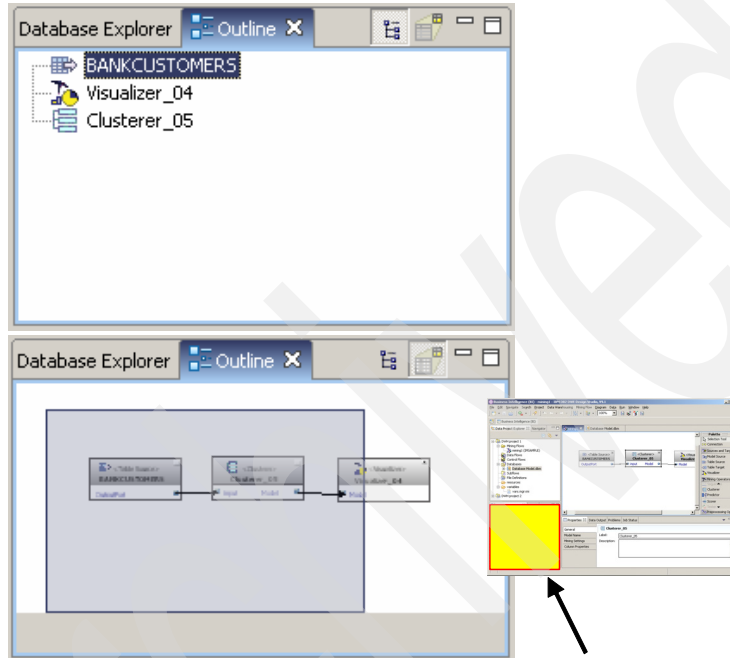


Figure 3-13 The Outline view

Properties view

The Properties view, shown in Figure 3-14, allows you to view and modify the properties of the current selected object. The edit capabilities of this view are dependent on the type of object that is currently selected. This is one of the most important views when designing flows or database objects. The properties of the newly created objects are primarily edited in this view.

Note: When objects are selected in the Database Explorer view, the Properties view is read-only. In order to edit the properties of an object you must select it from the Data Project Explorer view.

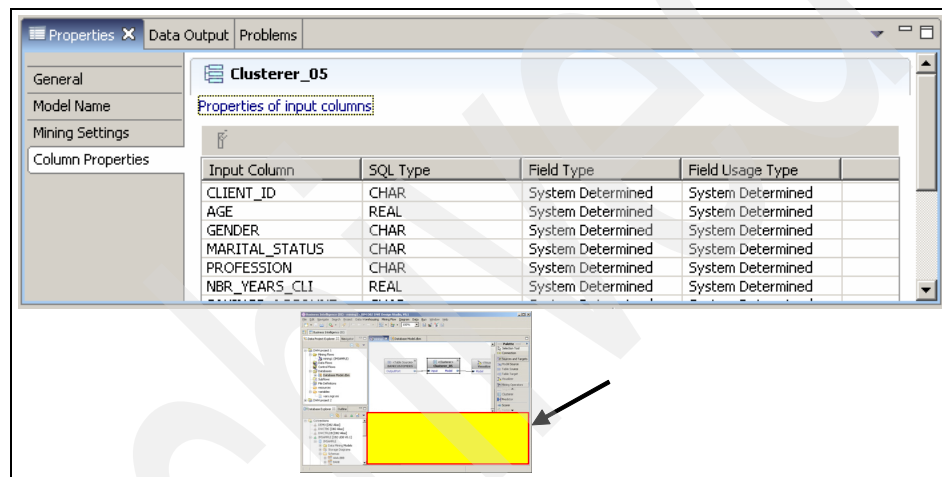


Figure 3-14 The Properties view within DB2 DWE Design Studio

Data Output view

The Data Output view is used to see messages, parameters, and results of the objects that you are working with. The Data Output view displays the results of various actions when they are executed on a database. You will use this view when you need to inspect the contents of a table via the Sample Contents feature, execute a data mining flow, or execute an SQL or DDL script or perform any operation on a database. The Data Output view is stacked with the Properties view.

The Data Output view is divided into two parts, as depicted in Figure 3-15 on page 47. The left part of the view contains a read-only table with three columns:

1. Status indicates the state of the associated action.
2. Action indicates what kind of action occurred.
3. Object name is name of the statement or tool that runs the statement.

The top row of the table contains the information for the most recent run.

The right part of the Data Output view contains three tabs:

- Messages - shows any messages that were generated while the statement was being run. If there are errors in the SQL statement, an error message appears on this page. The SQL source of the statement that is being run is shown in this view also. Refer to your database product's SQL documentation to check the validity of the structure of the SQL statement. Edit the statement by making the changes in the SQL builder or SQL editor as necessary, and then run the statement again.

For INSERT, UPDATE, and DELETE statements, a message is displayed on this page if the statement runs successfully. If an INSERT, UPDATE, or DELETE statement runs successfully, the database is modified.

- Parameters - shows any parameters that were passed to the statement. The name of each parameter and its input value is listed. This page contains values only if host variables are defined for the statement.
- Results - contains any results that were generated from running the statement (for example, a table of sales data). The Results page is selected by default.

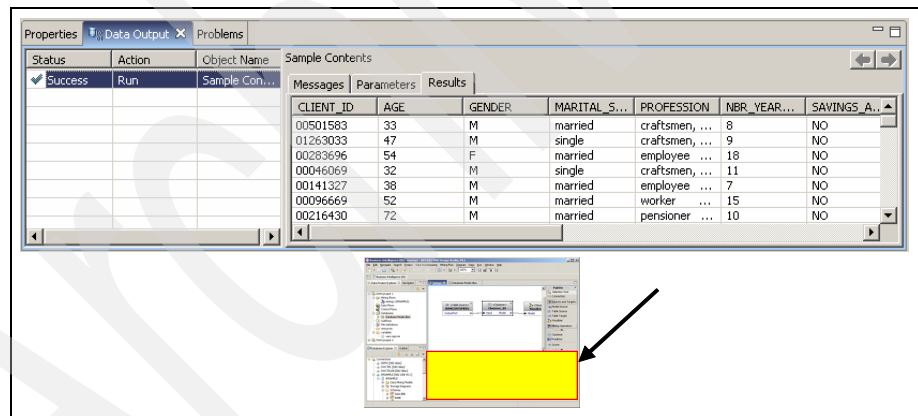


Figure 3-15 The Data Output view

Problems view

While working with the resources in your data warehousing projects, certain editors may log problems, errors, or warnings to the Problems view, as shown in Figure 3-16 on page 48. The Problems view is stacked with the Properties view and the Data Output view. The Problems view will show three levels of severity: errors, warnings, and information. These messages can be sorted by their severity, the resource name, or the problem description. Messages can also be

filtered by severity or resource. From the problem list, you can double-click an error, which will launch the editor for the resource containing that error, with the corresponding object highlighted.

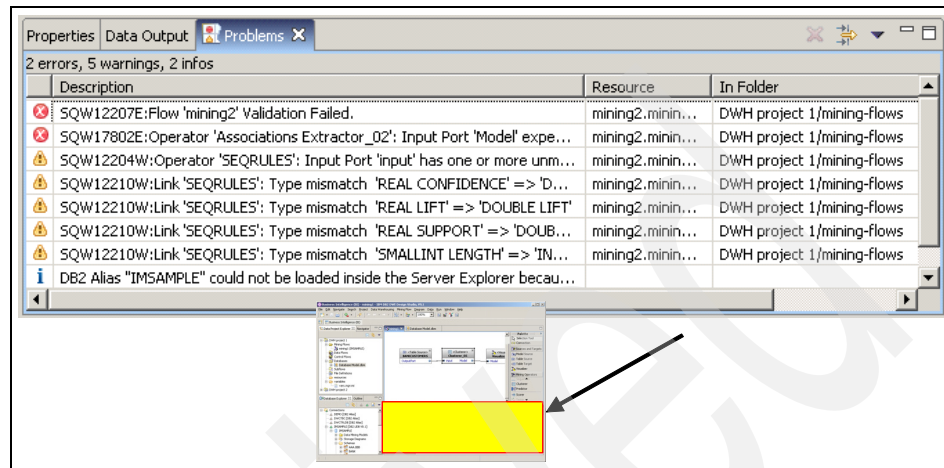


Figure 3-16 Problem view

3.3.4 Common tasks

Now that you are familiar with the Design Studio user interface, you are ready to learn how to use Design Studio to work on your data warehousing environment. The remainder of this chapter reviews the major tasks that you will perform within Design Studio.

Create projects

Data warehouse projects are containers for the development of data warehouse applications in the Design Studio. Before you can use the Design Studio to perform any of the processes within the BI Solutions development life cycle, you must create a project. The data warehouse project contains artifacts such as your data schemas, SQL control flows, and mining flows. To create a project, from the main menu select **File** → **New** → **Project** → **Data Warehousing**. There are two types of projects provided by the DWE Design Studio:

- ▶ *Data design projects (OLAP)* - Design database physical models, including OLAP models. You can forward or reverse engineer database models.
- ▶ *Data Warehouse Projects* - Design SQL Warehouse and data mining flows. This project type also provides the mechanisms to generate data warehouse application packages to deploy on the DWE server. Data warehouse projects can also contain your physical data models.

Data warehouse projects sometimes depend on metadata (for example, a physical data model) that is stored in data design projects. You can link a data warehouse project to a data design project by right-clicking the data warehouse project and selecting **Project References** from the context menu.

Note: If you delete a linked physical data model (.dbm file) from a data warehouse project, you are deleting the original model file, not just the link to that model. Rather, you should remove the project reference to that data model.

Then you can select from the list of available data design projects. When a warehouse project refers to a data design project, you have access to the metadata in that project when you are designing data flows and mining flows. Linked resources are indicated within the Project Explorer. For example, within the Databases folder of a data warehouse project, we have a resource named DWESampleTutorial.dbm(linked from Tutorial-Data Model OLAP).

Important: Do not rename or copy and paste a data warehouse project. These actions have unpredictable results and are likely to cause problems with the objects inside the project, such as subflows and application profiles.

You can import data flows and other objects into a data warehouse project by selecting **File → Import → File system**. Be sure to identify the correct folder name for the object that you want to import. For example, if you are importing a data flow, specify the data-flows folder as the target directory in your project. If you do not select the correct folder, the results of the import operation are unpredictable.

Add database connections

Every time the DWE Design Studio is launched, the Database Explorer will display all local DB2 databases, any remote DB2 databases that have been cataloged on the DWE client machine, and any previously defined non-DB2 database connections. Before you can browse or explore a new database, you must define a database connection to the data source.

Note: The database connections that you define in the Database Explorer view are used during the design or development phase of your data warehouse project. They define a connection from your Design Studio client to the database server. When you are ready to deploy your project, you must define a run-time database connection via the Admin Console. For details on how to do this refer to 9.3.3, “Managing warehouse resources” on page 405.

The database connections are represented in the Database Explorer view by a node. The connections are initially disconnected (no [+] sign on their left side). You can connect to a database by right-clicking its connection node and choosing **Reconnect**. You will then be prompted for a user ID and password. Once the connection has been made, you can expand the tree to explore its schemata, tables, and other objects.

If you wish to add another remote database or non-DB2 database to your Database Explorer view, right-click the **Connections** node and select **New Connection**. A dialog box will be displayed to collect the connection information such as database vendor and version, database name, port number, JDBC driver class location, and user ID and password. You can also specify filters for the database connection if it is appropriate to do so. Once this information is correctly supplied, the database will be added to the Connections list. For more details on configuring database connections refer to:

http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp?topic=/com.ibm.datatools.connection.ui.doc/topics/cdbconn_overview.html

Browse databases and explore data

Once you are successfully connected to a database in the Database Explorer, there are many ways to explore your data from within DWE Design Studio. You can explore schemata, table relationships, table content, or value distributions in your data. This is often the first step in creating a mining project; by exploring your database objects, you can select the appropriate data for mining analyses.

► Exploring database schemata

To expand a database you wish to explore, double-click its icon or click the [+] to the left of the database name. You can view items such as storage diagrams, schemas, and tables; and view relationships between tables and columns. Utilize the Properties view after selecting a table to display this information.

► Viewing sample content

Inspect the content of single tables by expanding the explorer tree and viewing the schemata and tables. Right-click a table and choose **Data** → **Sample Contents**. The Data Output view will display a sampling of the data. The number of rows selected for the sample is controlled by the output preference setting, which is found under the data category, as described in “Customize the environment” on page 53.

► Exploring value distributions

If you have enabled the database for data mining, you can look at data value distributions for a particular table. This feature also shows other statistical information. Right-click the table name and choose **Distributions** and then choose **Multivariate**, **Univariate**, or **Bivariate**.

The multivariate option allows you to visually explore value distributions and relationships between the columns of a table. For each field in the table, a chart is displayed that shows the distribution of the values in the chart. Multiple fields can be compared, as shown in Figure 3-17, by selecting the field name in the statistics table that is displayed above the chart.

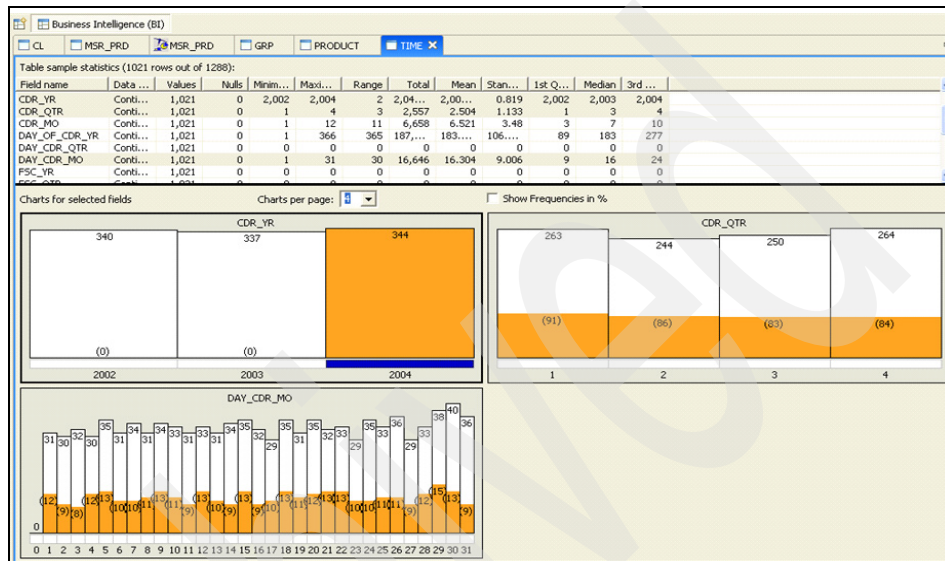


Figure 3-17 Multivariate value distributions

For more details on the value distribution options refer to Chapter 7, “DWE and data mining” on page 237.

► Editing, loading, and extracting data

In addition to being able to sample data contents, there are other actions that are available from the Data context menu. To get to the Data context menu, expand a database tree within the Database Explorer view, select a table, right-click, and chose **Data**. Actions that are available from the context menu are:

- Edit - With the proper database authority, you may edit the contents of the table in an editor.
- Load - Load the table with data from a flat file.
- Extract - Write the contents of the table to a flat file via the Extract option.

Work with databases offline

Database connection information can be saved locally to allow database objects to be viewed from DWE Design Studio without having an active connection to the

database. Not all features are available without an active connection. Capabilities that can be performed include viewing database objects and their properties and creating and viewing overview diagrams. In order to work with a database connection offline:

1. While connected, refresh your database connection.
2. Right-click the active database connection and select **Save Offline**. The connection information will be saved to your local Design Studio client.
3. Right-click the same active database connection and choose **Disconnect**.
4. Right-click the same connection again and choose **Work Offline**. The locally saved database information will be retrieved.
5. To return to the active connection, right-click and choose **Disconnect**, then right-click the connection and choose **Reconnect**.
6. If you wish to work disconnected again, we recommend that you refresh your local connection by re-performing these steps to be sure that you have any changes that were made to the database since you were last connected.

Customize the environment

You can modify many aspects of the behavior and appearance of your Design Studio workbench. For example, you can change the fonts and colors that are used, control the placement of tabs, and configure the behavior of plug-ins that you may have installed into your Eclipse environment. To customize your DWE Design Studio environment, from the menu select **Window** → **Preferences**. The preferences dialog allows you to set preferences for all tools that you have installed into your DWE Design Studio framework, as shown in Figure 3-18.

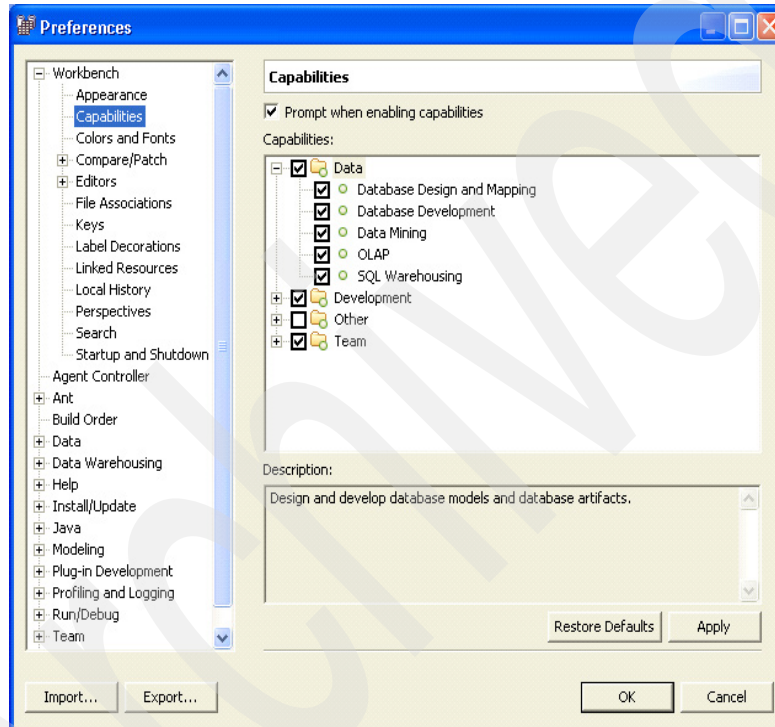


Figure 3-18 Setting preferences and capabilities for the DWE Design Studio

The tree on the left of the dialog displays all of the categories that can be customized. Some important preferences to review are:

- **Capabilities** - Due to the extensible nature of the Eclipse platform and the ability to combine very different tools into a single platform (such as Java development and data development tools), the number of available commands, views, and perspectives can be high. The UI can be simplified and controlled by enabling or disabling capabilities. Disabling a capability will hide all menu options, views, and creation wizards related to this function. The Capabilities category is under the Workbench heading.

- ▶ *Workbench* - Configure the general appearance and behavior of the workbench. This is where you can set fonts and colors, configure keyboard shortcuts, and configure startup and shutdown behavior.
- ▶ *Data* - Configure the preferences for data modeling and data exploration features. This is where you can control the number of rows returned when the option to sample contents is run, set the notation type used in the model diagrams, and configure the naming standards for physical tables, columns, indexes, and various constraints.
- ▶ *Data Warehousing* - Set the preferences for the SQLW and data mining features, such as the colors used for links, operators, and operator ports.
- ▶ *Modeling* - Define the appearance of the data model objects including constraints, comments, and notes. This is also where validation rules for constraints are enabled and disabled.
- ▶ *Team* - Configure CVS or any other control version system solution you may be using.

Model data structures

As part of the process of developing your BI solution, you will be designing and modifying physical database models and defining schemas and storage specifications. To get started, open the data design project you wish to work in, and launch the New Data Model Wizard by right-clicking the **Data Models** folder and choosing **New** → **Physical Data Model**. Choose the destination project folder, provide a name for your data model, and continue to follow the prompts from the wizard. Models can be created from scratch or reverse engineered from existing DDL or databases. Other tasks that are performed as part of this phase in the development life cycle include designing tablespaces and containers for DB2 databases, comparing data objects and models with each other or with other objects, analyzing designs to confirm that standards are complied with, and deploying the model. For more details about physical data modeling refer to Chapter 4, “Developing the physical data model” on page 57.

Create OLAP models

An OLAP model reflects the dimensionality of data that is typically found in a star schema-like structure to reflect objects like facts, measures, dimensions, and hierarchies. This metadata is very important to Business Intelligence tools. You can create this extra metadata by defining OLAP models and cubes as part of the physical data model. The DWE Design Studio also has wizards to help optimize access to relational OLAP cubes in DB2. For greater detail on OLAP modeling refer to Chapter 5, “Enabling OLAP in DB2 UDB” on page 79.

Design data flows and control flows

Data flows and control flows are housed within the data warehouse project type. Data flows model data transformation steps that move data from a relational table or file, perform some processing, and insert some type of relational or file target. From the Data Flow models, SQL-based code is generated to accomplish the tasks defined in the flow.

A control flow determines the processing sequence of data flows, mining flows, and other types of common data processing tasks such as executing OS scripts, DB2 scripts, FTP, and e-mail. It allows the construction of success/failure paths and iterative loops.

Create data mining models

DWE Data Mining functions are SQL-based tasks for the creation and application of data mining algorithms and execute within the DB2 database engine, therefore not requiring data to be extracted from the database. Data mining models can be created using DWE Data Mining flow, which allows the graphical development of data mining data preparation, the creation of data mining models, visualization of data models, and the application (or scoring) of the model.

3.3.5 Team component

If you have many developers working on your data warehousing projects within DWE Design Studio, you may want to set up a version control system. A version control system keeps track of changes to files and allows developers who may be physically separated to collaborate on design projects. The DWE Design Studio offers integrated support for two version control solutions, CVS and IBM ClearCase®. The integration points allow you to perform version control tasks directly from your database project explorer. In this section we provide a brief overview of the CVS and ClearCase options.

Concurrent Versions System (CVS)

The Eclipse framework supports the CVS standard. CVS manages a set of related files, such as DWE Design Studio project files, in a repository that is running on a CVS server. In order to obtain a copy of the files you wish to work on, you must check the files out from the CVS server. Once you are done with your work, update your changes back to the server by committing your files via a check-in process. The CVS server also manages conflicts, in the case that two different developers have made changes to the same set of project files. CVS provides a history of the work done by team members, and provides the mechanisms to coordinate and integrate work. If you have a CVS server set up within your network environment, you can enable the Teaming perspective/views.

A Concurrent Versions System repository is a persistent data store that coordinates multi-user access to projects and their contents. Projects in a repository can be of two forms: immutable (a project version) or modifiable (a project in a branch). Communication between the repository and Workbench clients is possible over local or wide area networks. Currently the Workbench supports two internal authentication protocols for CVS: pserver and ssh. Authentication protocols provided by external tools can also be used by CVS. For more information go to:

<http://www.cvshome.org>

The workbench is shipped with a built-in client for the Concurrent Versions System. With this client you can access CVS repositories.

Developing the physical data model

In “The BI solutions development life cycle” on page 30 we explained the concept of the BI solutions development life cycle. Recall that the first step in the life cycle is to design a physical model of the data structures needed to support the BI solution.

But what is a physical model of the data? We assume that you know the answer to this question, but also realize that all readers will not. To better understand what it is, or to get an idea of what it is, requires at least a brief look at data modeling concepts. It is a complex topic whose description could take an entire book. This is simply a terse description of a few basic terms to put them into perspective.

The data infrastructure for any business is built around data models. There are actually three types of data models: conceptual, logical, and physical. Here we take a very brief look at each:

- ▶ **Conceptual data model:** This is a high-level business view of the data. It identifies the data entities used in the business and their relationships to each other. This aids in the understanding and development of the logical and physical data models.
- ▶ **Logical data model:** This is where the data model actually gets defined. Here is where the data entities are identified in detail along with any existing

relationships between them. These are normalized entities that are described by specific attributes and data types, and candidate keys that will be used for direct access to the data.

- **Physical data model:** This is basically the physical implementation of the logical model in a specific database management system. It identifies the implementation details in terms of such things as the configuration, keys, indexes selected, and constraints, and the means by which it is physically stored in the selected database structures.

This is a major step in that life cycle. To help with this process, the DB2 Design Studio includes a subset of the functionality provided in the Rational Data Architect (RDA) product.

In this chapter we describe the physical data modeling capabilities within the BI Design Studio. DWE includes the physical data modeling and corresponding SQL generation capabilities to help implement changes to the physical model.

4.1 Physical data model

Physical data models define the internal schema of the data sources within your warehouse environment. Data models outline data tables, the columns within those tables, and the relationships between tables. The DWE Design Studio includes the components needed to allow you to create a physical model and then generate the SQL appropriate for your implementation target. Physical models are constrained to the concept of the target, for example, DWE physical models are constrained to the relational model. You can only model objects that are supported by the target database. For example, the ability to model MQTs only applies to DB2 targets.

The physical models that you create in Design Studio can be used to create brand new schemas or to update schemas that already exist in the target database.

Using DWE Design Studio, within your physical data models, you can define data elements for the target database such as:

- ▶ Databases (one per data model)
- ▶ Tables
- ▶ Views
- ▶ Primary keys and foreign keys
- ▶ Indexes
- ▶ Stored procedures and functions
- ▶ DB2 Tablespaces and buffer pools

The physical data models that you create and work within Design Studio are implemented as an entity relationship diagram. The models are visually represented using either Information Engineering (IE) notation or Unified Modeling Language (UML) notation. Before you begin your modeling work, you should configure Design Studio to use the notation that you prefer. This configuration is set in the Data → Diagram capability. For more details on configuring and customizing this setting refer to 3.3.4, “Common tasks” on page 48.

Physical model structure

Physical models are displayed in the Data Project Explorer view of the Design Studio. They are identified by a .dbm extension at the end of the model name. All physical models share a common structure, regardless of whether all objects are present within the model. In Figure 4-1 on page 60 we show the components of a physical data model. There is one database per physical model. Within the database, you may have storage diagrams (for DB2 database targets only), SQL statements, and numerous schemas. Each schema has a logical folder for diagrams, a folder for OLAP objects (for DB2 database targets only), and tables.

If you expand the table in the Data Project Explorer view, you will see definitions for columns, including primary key definitions, constraint specifications, and indexes.

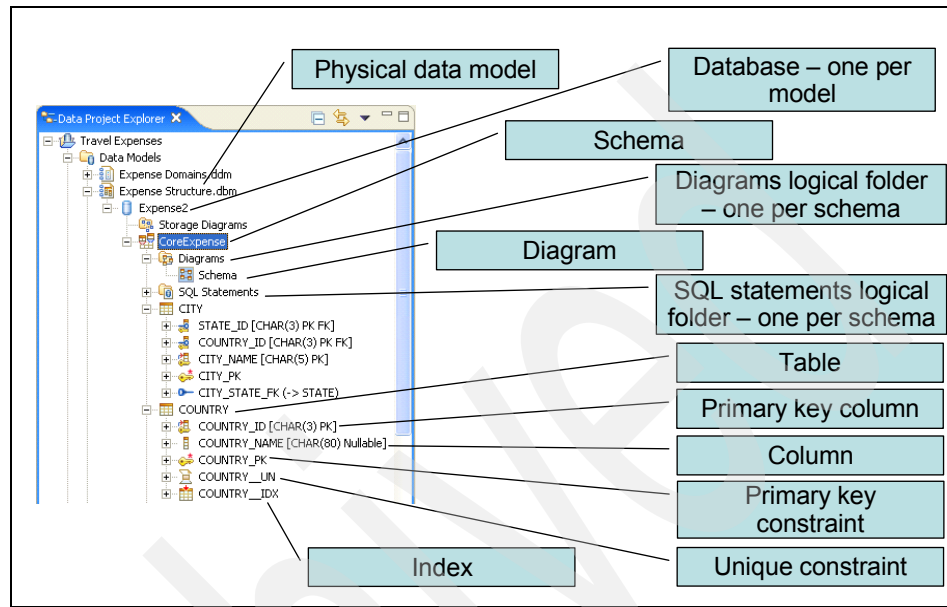


Figure 4-1 Physical model structure

4.2 Creating the physical data model

There are several ways to create physical data models with Design Studio. For example, models can be created from templates or from reverse engineering. There are also multiple ways to reverse engineer a data model. For example:

- ▶ From an empty template
- ▶ From a database using the Physical Model wizard
- ▶ From DDL using the Physical Model wizard
- ▶ Dragging and dropping from the Database Explorer view to the project in the Data Project Explorer view

4.2.1 From a template

You can design a new data model by creating a model from an empty template, which is provided with DWE Design Studio. To create a model this way, highlight the **Data Models** folder in the project, right-click, and choose **New** → **Physical**

Data Model. This will launch the New Physical Data Model wizard, as shown in Figure 4-2. This wizard can also be launched from the main Design Studio menu by selecting **File** → **New** → **Physical Data Model**.

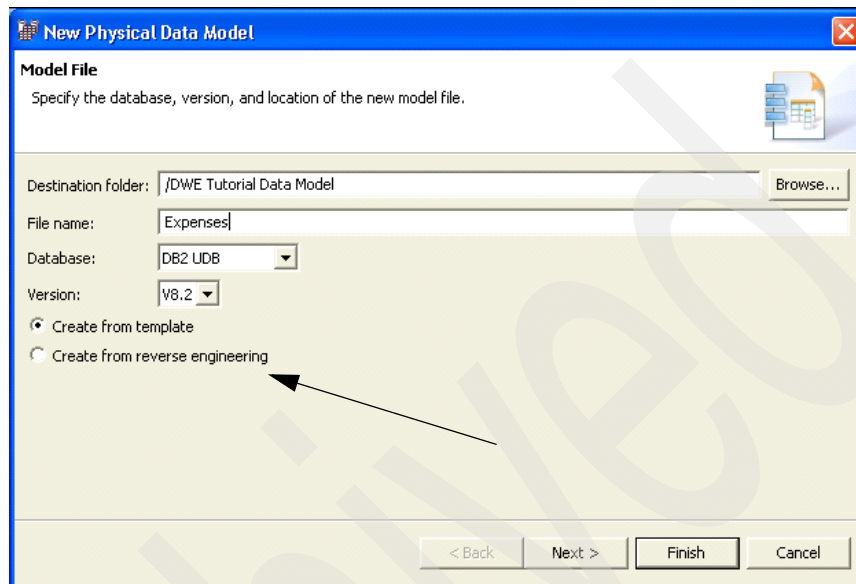


Figure 4-2 Physical Data Model wizard

To use the template approach, select the radio button labeled **Create from template**, as shown in Figure 4-2. Design Studio includes a modeling template called *Empty Physical Model Template*, which will be shown on the next page of the wizard. Choose the template and click **Finish**.

From the blank design template you can then use the visual diagram editor with its palette to design the model or you may use the Data Project Explorer to add objects to the model. For more information refer to 4.3.2, “Using the Diagram Editor” on page 67.

4.2.2 From an existing database

Another approach to creating physical data models is to reverse engineer from a database connection or database definition. The steps to reverse engineer start out similar to the template approach. Launch the New Physical Model wizard by selecting **File** → **New** → **Physical Data Model** from the main menu or by right-clicking either the project or the Data Models folder within the Data Project Explorer view and choosing **New** → **Physical Data Model**. Provide the details such as destination folder, model name, and database type and version. Then

select the radio button that is labeled **Create from reverse engineering** and click **Next**. The next screen provides the choice to reverse engineer from either a database or a DDL script. If you choose the database option, you will be prompted for database connection information. Then you can either use an existing connection or define a new one. After providing the database connection information, including user ID and password, you will be presented with a list of schemas that can be reverse engineered. You can then select a schema, as depicted in Figure 4-3.

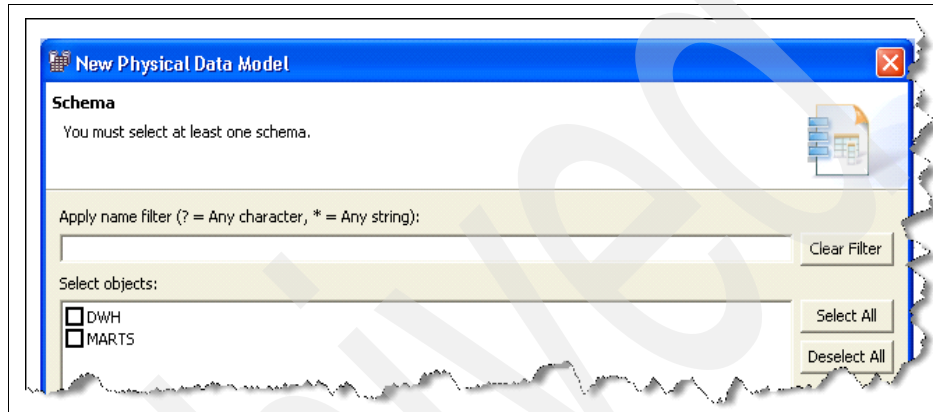


Figure 4-3 Selecting schemas to reverse engineer

Note: If there is a filter defined as part of the database connection, the filter is applied every time the database connection is used. Therefore, this will impact the list of schemas that you will see within the New Physical Data Model wizard. To verify whether filters are in use, review the objects associated with the database connection in the Database Explorer view. If the schema folder is labeled *Schemas[Filtered]*, then filters are enabled. To modify or review the filters, select the **Schema** folder in the Database Explorer, right-click, and choose **Filter**. You may then make any changes necessary.

As illustrated in Figure 4-4, the schema object list at this step in the wizard may also be filtered dynamically. This filter specifies the qualifiers to be included in the list.

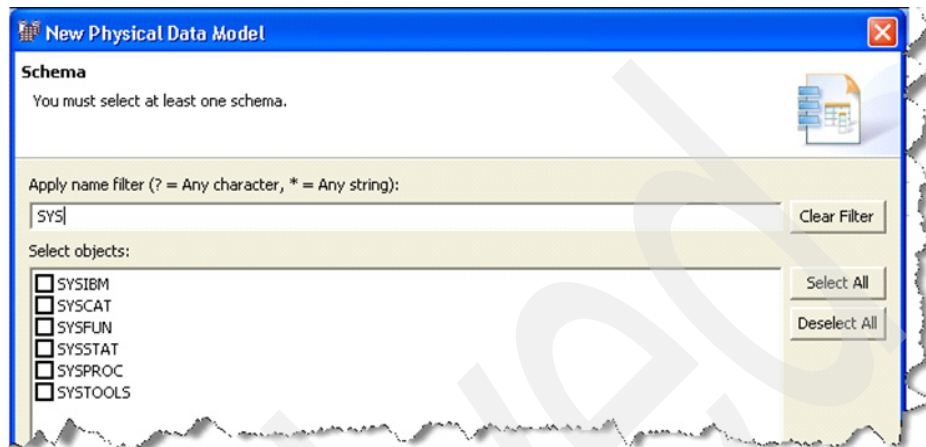


Figure 4-4 Filtering schemas to reverse engineer

After selecting the schemas you wish to use in the model, click **Next** to be presented with a list of the database elements you would like to include. Check all database object types of interest and click **Next**. The final screen of the wizard provides options to create an overview diagram and to infer implicit relationships. If you do not select these options, the components may be added to your model later.

4.2.3 From DDL

Another of the options available within the Physical Model wizard is to reverse engineer from database definition language (DDL). To do this launch the wizard, again by selecting **File** → **New** → **Physical Data Model** from the main menu, or by right-clicking either your project or the Data Models folder within the Data Project Explorer view, and choosing **New** → **Physical Data Model**. Provide the details such as destination folder, model name, and database type and version. Then select the radio button that is labeled **Create from reverse engineering** and click **Next**. At the next prompt, choose **DDL script** as the source, and then identify the location of the DDL file, as shown in Figure 4-5.

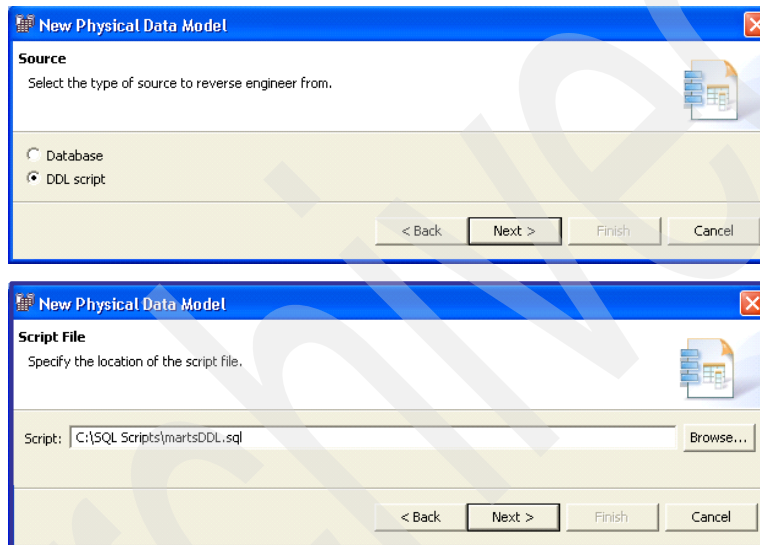


Figure 4-5 Reverse engineering from existing DDL

Similar to the steps to reverse engineer from a database connection, on the option screen you may choose to have an overview diagram created and implicit relationships inferred. If you do not choose these options at this time, those components may be created later from the Data Project Explorer.

4.2.4 From the Database Explorer

Perhaps the easiest method to create a new physical model from an existing database is to drag and drop the objects from the Database Explorer to the Data Project Explorer. To use this method, verify that there is an active connection to the source database within the Database Explorer view. Expand the database connection and select the objects to be reverse engineered. The level of granularity available ranges from database to table. Once you have selected the

objects, drag them up to the Data Project Explorer and drop them onto an existing project, as shown in Figure 4-6. The resulting physical model name will be the name of the database connection. A number will be added to the end of the model name for uniqueness, if necessary.

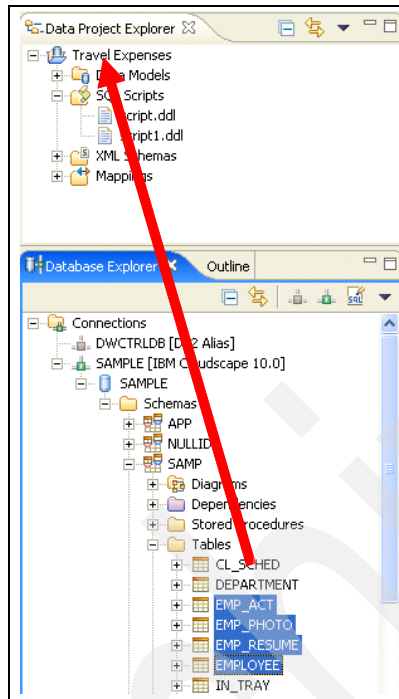


Figure 4-6 Drag and drop reverse engineer process

4.3 Working with diagrams

DWE Design Studio uses entity relationship diagrams to provide a visual representation of the physical data models within your projects. Entity relationship diagrams are a useful mechanism for understanding the data elements within your projects and communicating that information to others. Within DWE Design Studio, you may have multiple diagrams per schema within the projects. It is often helpful to organize the data models into multiple subject areas, especially when working with a large, complex data model.

In addition to the value that the diagrams bring to the projects by simplifying and improving the understanding of complex data environments, the diagrams can also be used to edit and modify the physical data model.

4.3.1 Creating a diagram

The DWE Design Studio provides several ways to add the entity relationship diagrams to the projects.

If you have chosen to reverse engineer the data model, you may opt to have the wizard create an overview diagram for you. The wizard provides prompts that allow you to specify what elements you wish to have included in the diagram.

You may still use diagrams in the data projects, even if you do not create the data models via the reverse engineering approach. New diagrams may be created from any Diagrams folder in the Project Explorer. Simply select the **Diagrams** folder, right-click, and choose **New Overview Diagram**. You will be prompted to select which elements from the current schema you wish to include in the diagram.

You may also wish to create a blank diagram, rather than including existing schema elements. To create a blank diagram right-click the **Diagrams** folder in the Project Explorer and choose **New Blank Diagram**.

4.3.2 Using the Diagram Editor

An overall view of the Diagram Editor is shown in Figure 4-7. The two main components to the Diagram Editor are the drawing area, or canvas, and the palette. Elements may be added to the diagram from either the palette or the Data Project Explorer. To use the Data Project Explorer, simply drag elements from the Data Models folder onto the diagram canvas.

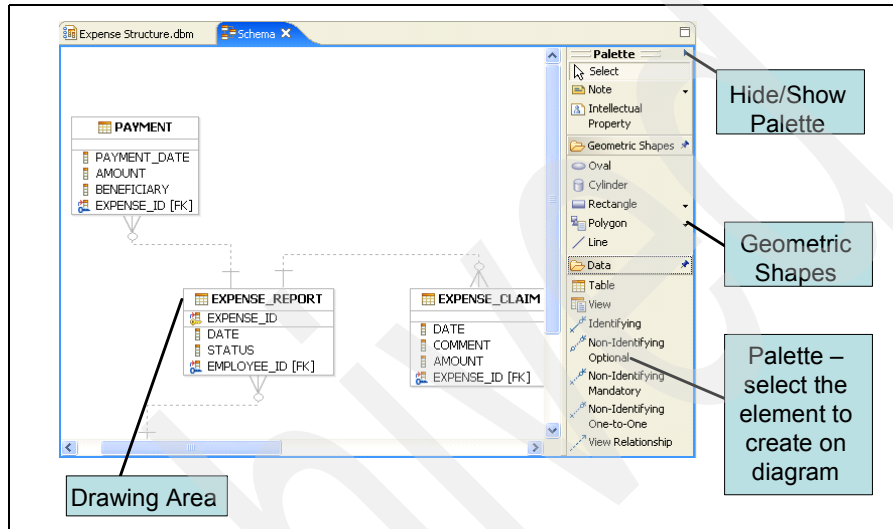


Figure 4-7 Visual Diagram Editor

Items from the palette may be placed on the drawing area by clicking an element to select it and then moving the mouse to the drawing area and clicking. Elements that are added to the diagram in this manner are provided with a default name, but you may change the name to a more meaningful name, as shown in Figure 4-8. As you can also see in Figure 4-8, when elements are added to the canvas, that they are also added to the Data Project Explorer.

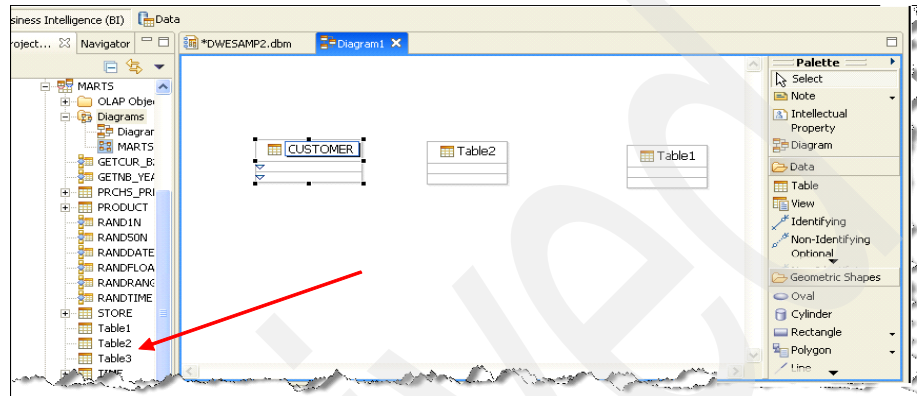


Figure 4-8 Adding palette elements to the diagram

Once tables have been added to the diagram, you may add columns to the tables from the visual diagram. When you select a model element action bars will appear, providing context aware pop-ups, as shown in Figure 4-9. Options that are available via the action bar include the ability to add a new key, column, index, or trigger. The palette can be used to establish identifying and non-identifying relationships between the various tables that make up each diagram. As you are using this approach, you may also utilize the Properties view to further define the objects that you are creating.

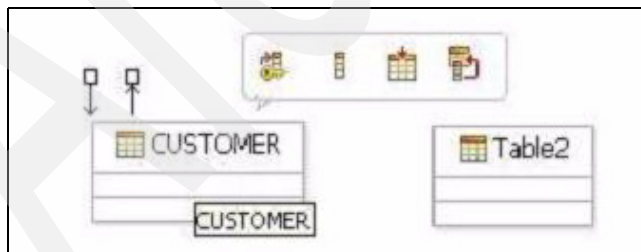


Figure 4-9 The Action bar in the Diagram Editor

4.4 Editing physical data models

DWE Design Studio provides two approaches to editing physical data models. Recall that the physical data models that you create are visually implemented as an entity relationship diagram. Once you have created a diagram, you may make changes to the physical model by editing the diagram. Refer to 4.3.2, “Using the Diagram Editor” on page 67 for more details about this approach. Another approach is to utilize the options available within the Data Project Explorer to modify or edit the physical model.

4.4.1 Using the Data Project Explorer

The Data Project Explorer provides an easy way to modify or edit the diagrams of the physical data models. Data model objects that have already been defined may simply be dragged onto the diagram canvas. Object properties may also be modified within the Data Project Explorer by selecting them and then utilizing the Properties view to modify various settings.

You can also use the Data Project Explorer to create new objects in the models, using the following steps:

1. Select the database within the Data Project Explorer, right-click, select **Add Data Object**, then schema, bufferpool, tablespaces, or dependencies.
2. Select the schema within the Data Project Explorer, right-click, select **Add Data Object**, then table, view, function, or stored procedure.
3. Select the table within the Data Project Explorer, right-click, select **Add Data Object**, then column check constraint, unique constraint, foreign key, index, trigger, or dependency.

4.5 Model analysis

DWE Design Studio includes a rule-based model analysis tool that allows you to evaluate adherence of the data model to design and optimization best practices. The model analysis utility provides a mechanism to improve the integrity and quality of your physical models, and helps to assure that the physical model is valid. We recommend that physical data models are validated before they are deployed, in order to be sure that there are not any problems in the design.

Model analysis may be performed against databases or schemas. To launch the wizard, select the database or schema you are interested in analyzing, right-click, and choose **Analyze Model**.

Models may be analyzed against the following categories of rules:

► OLAP constraints. These are categorized as follows:

– OLAP base rules:

- Attribute
- Attribute relationship
- Cube
- Cube dimension
- Cube facts
- Cube hierarchy
- Cube level
- Cube model
- Dimension
- Facts
- Hierarchy
- Join
- Level
- Measure
- OLAP object

– OLAP optimization rules:

- Dimension
- Join

► Syntax check: These are categorized as follows:

- Data types
- Key constraints and indexes
- Object names
- SQL statement

Within each of these categories, there are several rules that you may choose to include in the analysis, such as key constraints, object naming standards, and several rules related to OLAP best practices. Figure 4-10 shows an example of the Model Analysis wizard. After selecting the set of rules that are of interest, click **Finish** in order to start the analysis.

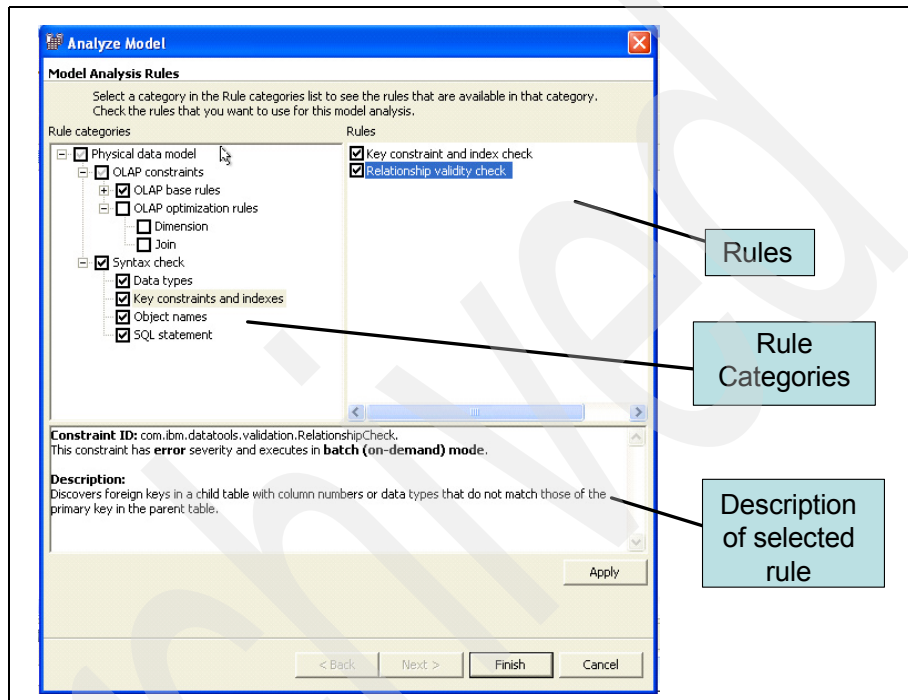


Figure 4-10 Model Analysis

The results of the Model Analysis utility are displayed in the Problems view. An example of the output is shown in Figure 4-11. Double-click an item in the Problems view in order to navigate to the model objects in the Project Explorer that are affected. You may then take corrective action and re-run the model analysis to verify that all problems were corrected.












| Properties | Data Output | Problems | Job Status | Console | Model Report |
|---|---|-----------------------|-------------|---------|--------------|
| 80 errors, 0 warnings, 5 infos | | | | | |
| | Description | Resource | In Folder | | |
|  | Element CL has a name conflict with one or more other elements in DWH | DWESampleTutorial.dbm | Tutorial-De | | |
|  | Element GRP has a name conflict with one or more other elements in DWH | DWESampleTutorial.dbm | Tutorial-De | | |
|  | Element GRPUNQ has a name conflict with one or more other elements in DWH | DWESampleTutorial.dbm | Tutorial-De | | |
|  | Element IDX503040149450000 has a name conflict with one or more other elements in DWH | DWESampleTutorial.dbm | Tutorial-De | | |
|  | Element IP has a name conflict with one or more other elements in DWH | DWESampleTutorial.dbm | Tutorial-De | | |
|  | Element ITM_TXN has a name conflict with one or more other elements in DWH | DWESampleTutorial.dbm | Tutorial-De | | |
|  | Element ITMMKTID has a name conflict with one or more other elements in DWH | DWESampleTutorial.dbm | Tutorial-De | | |
|  | Element ITMTXN1 has a name conflict with one or more other elements in DWH | DWESampleTutorial.dbm | Tutorial-De | | |
|  | Element ITMTXN2 has a name conflict with one or more other elements in DWH | DWESampleTutorial.dbm | Tutorial-De | | |
|  | Element ITMTXN3 has a name conflict with one or more other elements in DWH | DWESampleTutorial.dbm | Tutorial-De | | |
|  | Element MKT_BSKT_TXN has a name conflict with one or more other elements in DWH | DWESampleTutorial.dbm | Tutorial-De | | |

Figure 4-11 Model Analysis output

Continue to correct any issues that the model analysis identifies until all errors are cleared. Once the model has been successfully validated, you are ready to deploy it.

4.6 Deploying the data model

After the physical model has been designed and validated, you are ready to deploy the model into your environment. The deployment process results in the schemas, tables, and other objects that were modeled being created in the target database. Once the physical model has been deployed, you may begin populating the structures with data.

4.6.1 Using Design Studio to deploy a physical data model

Physical data models may be deployed from DWE Design Studio via the Generate DDL wizard, which generates context-driven DDL. When you generate the DDL code, you may choose a database, schema, table, bufferpool, or tablespace as the root for the code generation. To launch the DDL wizard, select the desired data element, right-click, and choose **Generate DDL**. Alternatively, to launch the wizard from the menu, with the data element selected, click **Data** →

Generate DDL. Respond through the prompts of the wizard, indicating what model elements and objects you would like to include in the DDL script. Refer to Figure 4-12 for an example of the Generate DDL wizard.

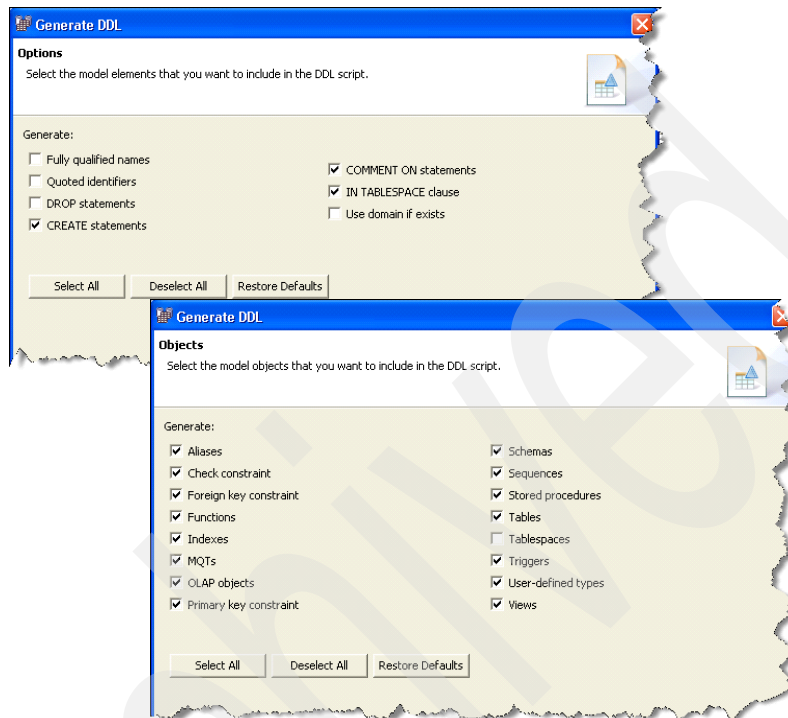


Figure 4-12 Generating DDL

Once you have selected the objects you wish to deploy, you will be presented with a summary of the DDL script, with options to save the DDL file or execute the DDL script on the server. These options are shown in Figure 4-13. If you choose to run the DDL on the server, you will be prompted to select a database connection or create a new database connection. The DDL script will be saved in the SQL Scripts logical folder within your data design project.

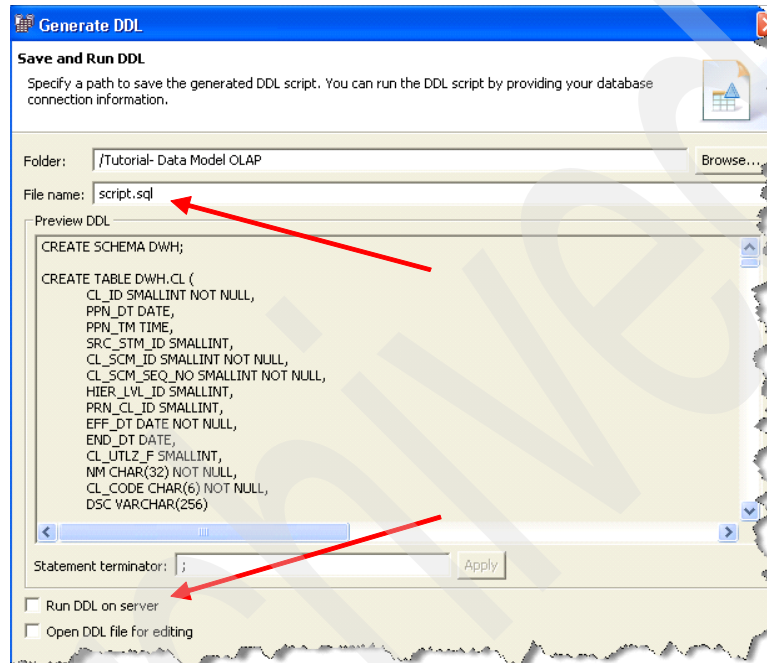


Figure 4-13 Options for saving and running DDL

The DDL scripts that are located in the SQL Scripts folder may be saved for later execution, and they may also be modified before being executed. To edit the DDL, select the file within the SQL Scripts folder, right-click, and choose **Open With** → **SQL Editor**. This will cause the file to be opened with a text editor. When you are ready to run the DDL, select the file, right-click, and choose **Run SQL**. You may review the results in the Data Output view, which includes status information, messages, parameters, and results.

4.6.2 Using the Admin Console to deploy a physical model

The DWE Admin Console may also be used to deploy physical models. For more details on utilizing this approach to model deployment refer to 9.2.2, “Deployment using the DWE Administration Console” on page 385.

4.7 Maintaining model accuracy

It is important to manage changes to the models that may happen over time. As business requirements change, there is often an adjustment that ultimately needs to be made to the physical data models. DWE Design Studio has the ability to help you manage that type of change. It is helpful to be able to compare the differences between two physical data models, and also compare the difference between a physical data model and the database where it was deployed. Once changes are identified, it is important to be able to synchronize the two models being compared. It is also important to be able to analyze the impact of a change to the model before the change is implemented.

4.7.1 Comparing objects within the physical data model

DWE Design Studio provides object-based compare and sync capabilities as a mechanism to assist and simplify the task of maintaining model accuracy. From the Database Explorer, you may select a database object, right-click, and choose **Compare With** → **Another Data Object**. You will then be prompted to select the object to use for comparison. Another way to achieve the same comparison is to highlight two data objects, right-click, and choose **Compare With** → **Each Other**. Within the Project Explorer, there is a third comparison available if the source of the model was created by reverse engineering. If that is the case, then select an object, right-click, and choose **Compare** → **Original Source**. This option is only available from the Project Explorer, not the Database Explorer.

These options under the Compare With command are helpful if you need to compare physical objects with each other, for example, to compare objects from a test database to objects in a production database. Or you may need to compare the baseline database objects from the Database Explorer with changed objects from your project before deploying changes.

4.7.2 Visualizing differences between objects

The results of the object comparison are displayed in two views, which are connected. The upper view is called Structural Compare. The Structural Compare shows the differences in a tree format. The first column is a tree. Each entry in the tree represents a part of the model where a difference was found. The second column in the Structural Compare output shows the first object as input to the Compare command, and the third column shows the second object to which it was being compared. A copy of the differences may be saved by clicking the **Export** button in the upper right portion of the Structural Compare view. The file that results from this option is an XML file.

If two different items that appear on different rows in the comparison output need to be compared to each other, you may use the Pair/Separate buttons to facilitate their comparison and align the comparison results. This is helpful when the column names are different, but the objects represent the same data.

As you work down through the tree in the Structural Compare, differences will be highlighted in the lower view, which is called the Property Compare. This shows the Properties view. An example of the output from the Compare Editor is shown in Figure 4-14.

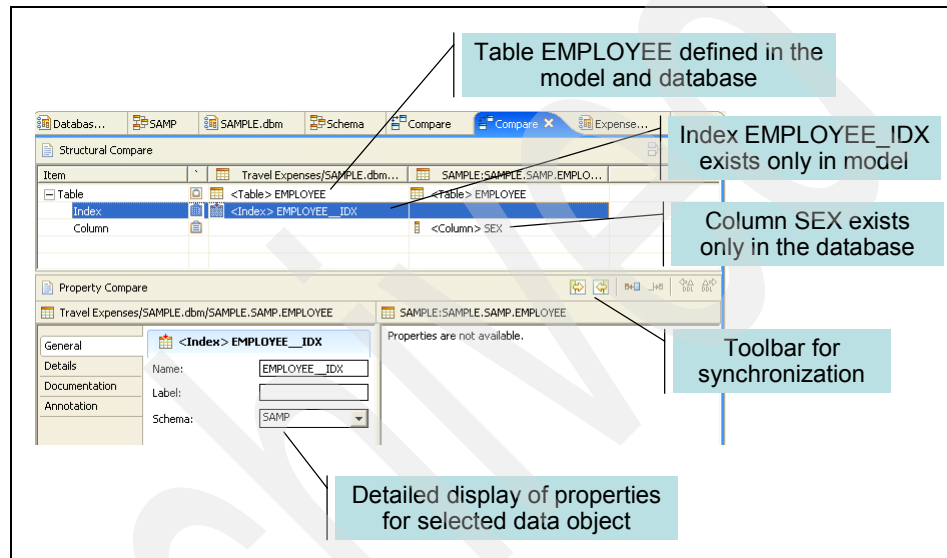


Figure 4-14 The Compare Editor

4.7.3 Synchronization of differences

As you view the differences between the objects you have compared, you may use the Copy From buttons, which are located on the right side of the toolbar that separates the Structural Compare and the Property Compare. These buttons allow you to easily implement changes from one model or object to another. You may copy changes from left to right, or from right to left. As you use the Copy From buttons, the information displayed in the Structural Compare is updated to reflect the change.

Another way that you can implement changes to bring your models in sync with each other is to edit the values that are displayed in the Property Compare view.

Any changes that are made may be undone or redone by clicking the menu option **Edit** → **Undo** and **Edit** → **Redo**.

Once you have reviewed all of the differences and have decided what needs to be done to bring your models in sync with each other, you are ready to generate delta DDL. This DDL can be saved for later execution or may be executed directly from DWE Design Studio.

4.7.4 Impact analysis

DWE Design Studio also provides a mechanism for performing impact analysis. It is beneficial to understand the implications of changes to models before they are implemented. The impact analysis utility will show all of the dependencies for the selected object. The results are visually displayed, with a dependency diagram and a Model Report view added to the Output pane of Design Studio.

The impact analysis discovery can be run selectively. To launch the utility, highlight an object, right-click, and choose **Analyze Impact**. Choose the appropriate options, as shown in Figure 4-15.

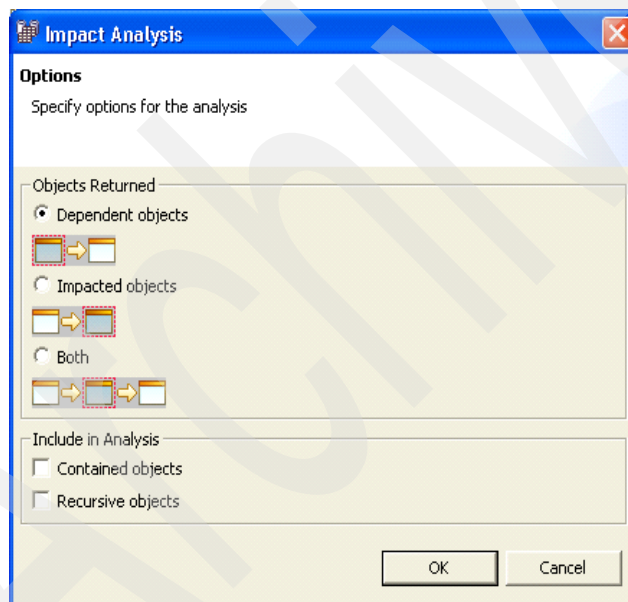


Figure 4-15 Impact Analysis options

The results of the impact analysis are shown in a dependency diagram, as illustrated in Figure 4-16.

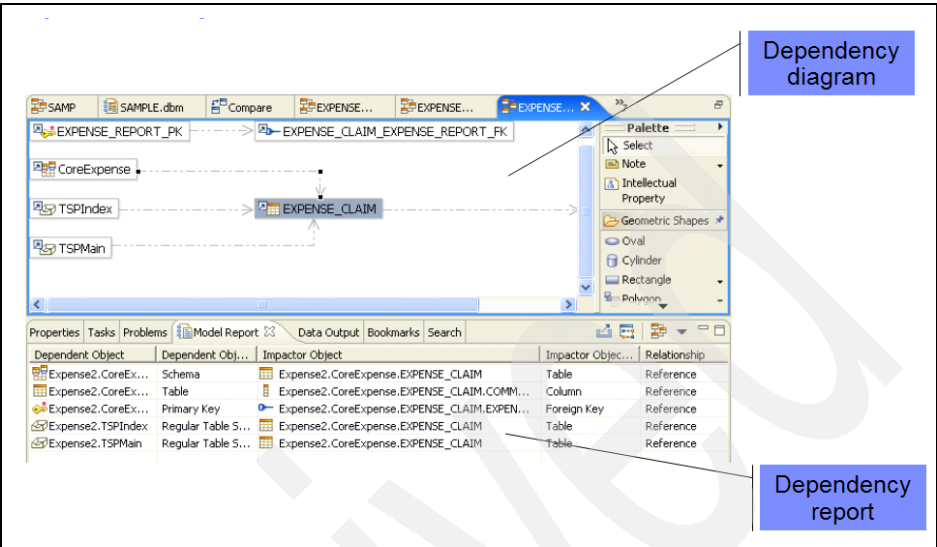


Figure 4-16 Impact analysis report

4.8 Summary

In this chapter we have shown how to create and edit physical models. We have also explored best practices around model validation and analysis. Once these steps have been concluded, you will have deployed your physical model to the database target. You are now ready to begin populating those structures with data. These procedures are outlined in Chapter 5, “Enabling OLAP in DB2 UDB” on page 79.

Enabling OLAP in DB2 UDB

Online Analytical Processing (OLAP) is a popular and powerful data analytical method. In this chapter we introduce the OLAP capabilities provided in DB2 Data Warehouse Enterprise Edition V9.1. An overview of modeling and deploying OLAP functionality using DWE Design Studio is discussed and is referred to simply as DWE OLAP.

DB2 UDB V8.1 became OLAP aware with the introduction of DB2 Cube Views. DB2 Cube Views was offered as a separate product to DB2 UDB, and also bundled with DB2 Data Warehouse Edition V8.

In DB2 Data Warehouse Edition V9.1, DB2 Cube Views has been integrated into the DWE Design Studio, enabling a common integrated data warehouse design environment. DWE V9.1 reference materials still refer to DB2 Cube Views and should be considered synonymous with DWE OLAP.

5.1 Implementing OLAP

Online analytic processing is key component of many data warehouse or business intelligence projects and often is a core requirement from users. OLAP enables users (business analysts, managers, and executives) to gain insight into data through a fast, consistent, and interactive interface, thereby enabling increased productivity and faster decision making.

Once the data (facts and dimensions) are loaded into an OLAP server, BI tools provide the capability to analyze data by simply dragging and dropping dimensions and facts into the appropriate locations. Users can interrogate data at different levels of summarization along the various dimensions. OLAP enables users to easily navigate the data, and explore and build new reports with little assistance from IT professionals.

5.1.1 OLAP architectures

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 approaches to OLAP databases are called MOLAP, ROLAP, and HOLAP. The various OLAP architectures are briefly described below, but be aware that OLAP products from different vendors with the same architectures can still vary greatly in functionality and capabilities.

- MOLAP is the term for *Multidimensional OLAP*. The database is stored in a special, typically proprietary, structure that is optimized (often through some pre-calculation of data values) for very fast query response time and multidimensional analysis. The database usually has a proprietary OLAP API that BI tools must support. Various OLAP APIs often offer several OLAP formulas not found in standard SQL that are frequently used when performing financial analysis.

MOLAP databases require time to load the data, time for the pre-calculations and data sparsity/compression techniques to manage possible data explosion that can greatly increase disk usage. This implementation typically has limitations when it comes to large volumes of data (in the 100 GB range, as an example) and large numbers of concurrent users. It is not uncommon to have many MOLAP databases on many servers to address large data and user scalability requirements.

- ROLAP is the term for *Relational OLAP*. The database is a standard relational database and the database model is often a star or snowflake schema. A primary benefit of this implementation is the significant scalability capabilities offered by relational databases. Access is provided through

standard SQL, which enables access to the data by a broader set of tools and applications.

ROLAP performance has typically been considered slower than MOLAP since each query is an SQL query (or multiple SQL queries) and the query time is largely governed by the complexity of the SQL used, the number and size of the tables that must be joined, and the level of aggregation required to satisfy the query.

- *HOLAP* is the term for *Hybrid OLAP*. As the name implies, it is a hybrid of ROLAP and MOLAP. A HOLAP database typically stores high-level aggregates in a MOLAP database while lower-level detail data is stored in a relational database. By storing the lower levels of a dimension in relational format instead of in MOLAP, the MOLAP database size is reduced and therefore the time required to load and 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 HOLAP database developer determines what level of granularity resides in which database. From a user perspective the line between what is stored in MOLAP and what is stored in relational is not always seamless or as flexible to the users as they drill through from the MOLAP cube into the underlying relational data. HOLAP databases use the proprietary OLAP API of the MOLAP database.

5.1.2 DB2 UDB and OLAP

Clients have strived to reduce the number of BI tools, but different requirements and organizational issues often result in clients needing to support a combination of BI reporting and OLAP solutions from different vendors. IBM recognized the need and importance of supporting the variety of OLAP solutions as well as other BI tools (reporting, statistical, and data mining). To support this they have delivered many data warehouse features (such as materialized query tables, multi-dimensional clustering, hash and range partitioning, compression, query re-write, optimizer enhancements, and cube and rollup functions) within DB2 UDB that greatly improves query and OLAP performance. These features are discussed in more detail in Chapter 10, “DB2 UDB Data Warehouse features” on page 465.

The wide variety of OLAP architectures and tools on the market today exist because no single OLAP solution has met all of the OLAP requirements. MOLAP vendors have extended their architecture to create a HOLAP solution to address

scalability issues. However, they still lack the scalability and flexibility provided by a relational database such as DB2 UDB. DB2 UDB has incorporated many OLAP features to address performance. Similar to MOLAP solutions, DB2 UDB can pre-compute results and deliver excellent performance.

The following key industry trends and directions have influenced the IBM OLAP strategy and the development of DB2 Cube Views.

- ▶ *Database parallelism* - IBM has been adding more BI and OLAP functionality into the DB2 database management system. The parallelism of the shared-nothing database architecture of DB2 UDB enables these functions to be performed in parallel over potentially hundreds of processors, as opposed to MOLAP engines, which typically reside on smaller servers with 1, 2, or 4 CPUs.
- ▶ *Server and datamart consolidation* - The proliferation of departmental servers and the increasing cost to manage these many systems and data feeds has led to IT consolidation. Through various database and hardware partitioning features, different types of workload can all be handled within a single DB2 database system.
- ▶ *Real-time analytics* - As data warehouse environments evolve, requirements for real-time or near real-time analysis is growing. The infrastructure to enable real-time and trickle feeds into the data warehouse has matured, and can now make it possible for real-time analytics. Real-time analytics is limited if data must subsequently be sent to additional datamarts or OLAP servers.

More details can be found in the following IBM Redbooks:

- ▶ *Preparing for DB2 Near-Realtime Business Intelligence*, SG24-6071-00
- ▶ *Data Mart Consolidation: Getting Control of Your Enterprise Information*, SG24-6653
- ▶ *DB2 UDB's High-Function Business Intelligence*, SG24-6546-00

5.2 DWE OLAP

DWE OLAP, also known as DB2 Cube Views, is an add-on feature of DB2 UDB that specifically improves the ability of DB2 UDB to perform OLAP processing. With DWE OLAP, the database becomes multidimensionally aware by adding OLAP metadata to the DB2 UDB catalog tables. The OLAP metadata describes the dimensional structure of the relational tables and provides the foundation for DB2 UDB to be optimized for OLAP tools and applications.

Based on the OLAP metadata, the DWE OLAP Optimization Advisor will analyze the dimensional model and recommend aggregate tables, which are known as

DB2 Materialized Query Tables (MQTs). The MQTs contain pre-calculated data that maps to the OLAP structures to dramatically improve query performance for OLAP tools and applications.

DWE OLAP is a unique integrated approach to OLAP and improves DB2 query performance. DWE OLAP:

- ▶ Enables the DB2 UDB Optimizer to rewrite incoming queries to take advantage of the MQTs recommended by DWE OLAP.
- ▶ Enables applications and tools to process the dimensional data in DB2 naturally, productively, and efficiently.
- ▶ Enhances all queries that use aggregate data including OLAP style queries, query extracts to load MOLAP cubes, drill-through queries from MOLAP, and HOLAP tools and ad-hoc analysis against relational tables in DB2 UDB.

Organizations will benefit from DWE OLAP by:

- ▶ Leveraging existing SQL skills in the deployment of data warehouse projects
- ▶ Supporting numerous tools to ensure interoperability of components
- ▶ Enabling centralized management of OLAP metadata
- ▶ Reducing rollout costs of IBM Business Partner tools
- ▶ Increasing overall performance of the data warehouse

5.2.1 DWE OLAP metadata

DWE OLAP describes metadata objects similar to multidimensional model structures, and in particular star schemas and snowflake schemas. However, the objects of interest in DWE OLAP can be a particular grouping in, or a subset of, a star schema or snowflake schema. Figure 5-1 shows an example of the mapping of DWE OLAP metadata to a multidimensional model.

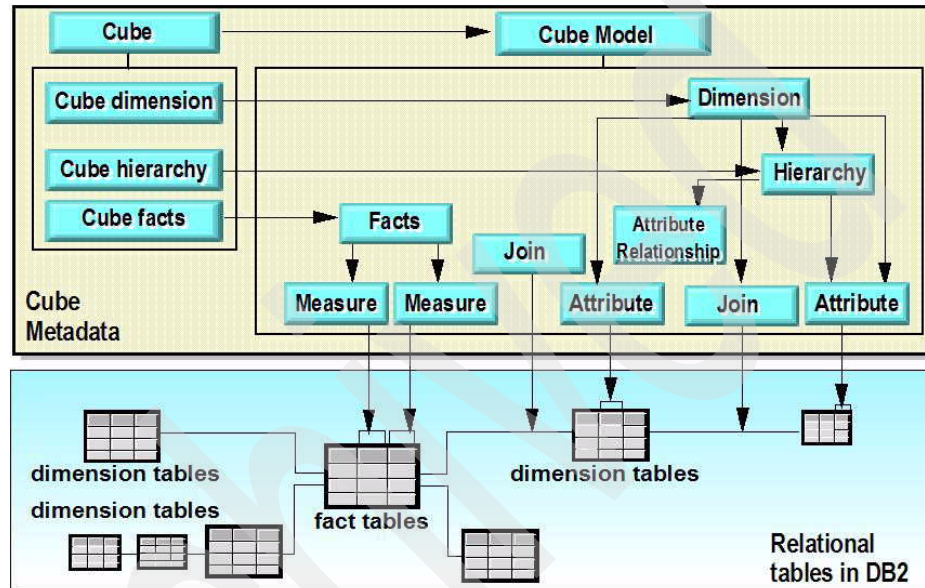


Figure 5-1 DWE OLAP metadata

The following metadata objects are used in a DWE OLAP model:

- *Cube model* - representation of a logical star or snowflake schema. The cube model groups relevant dimension objects around a central facts object. It describes all data related to a collection of measures. Typically, the cube model relates to a star schema or snowflake schema in the database. The cube model references a single facts object and one or more dimensions.
- *Facts object* - groups related measures that are interesting to a specific application. In a cube model, a facts object is used as a center of a star or snowflake schema. The facts object stores information about the attributes that are used in fact-to-dimension joins, and about the attributes and joins that are used to map the additional measures across multiple database tables. Therefore, in addition to a set of measures, a facts object stores a set of attributes and a set of joins.

- ▶ *Measure* - defines a measurement entity and is used in facts objects. Measures typically point to numeric columns in a fact table.
- ▶ *Dimension* - a collection of related attributes that together describe an aspect of the data. A dimension can reference attributes from one or more dimension tables. However, if you use attributes from multiple dimension tables, the tables must have joins between them and the dimension must reference those joins. A dimension also references one or more hierarchies and can reference relationships between attributes.
- ▶ *Attribute* - maps to either a single column in a table or an expression that is a combination of a set of columns or other attributes.
- ▶ *Join* - analogous to a table join in a SQL query. A join has a type and cardinality. Joins can be used in dimensions to join dimension tables together, or in a cube model to join the dimensions of the cube model to its facts object, or within a facts object to join multiple fact tables.
- ▶ *Hierarchy* - defines relationships among a set of one or more attributes that are grouped by levels in the dimension of a cube model.
- ▶ *Level* - consists of a set of attributes that are related and work together as one logical step in a hierarchy ordering. The relationships between the attributes in a level are usually defined with a functional dependency.
- ▶ *Cube* - a specific instance or subset of a cube model. The cube facts and cube dimensions are a subset of those that are referenced in the cube model. A cube is the closest to an OLAP conceptual cube. Most cubes can be retrieved with a single SQL statement.
- ▶ *Cube facts* - reference a subset of the measures from the facts object from which they are derived.
- ▶ *Cube dimension* - references a subset of the attributes of the dimension from which it is derived. It also references a single cube hierarchy.
- ▶ *Cube hierarchy* - references a subset of the attributes of the hierarchy from which it is derived, where the order of the attributes must be in the same order as their order in the hierarchy.
- ▶ *Cube level* - a subset of a level that is used in a cube. A cube level references the level from which it is derived, and inherits the level key attributes and default attributes that are defined for the parent level.

5.2.2 Materialized Query Tables (MQTs)

Materialized query tables are a powerful way to improve response time for complex queries. MQTs are aggregates of base data stored in the database. The DB2 UDB Optimizer is able to recognize that a specific query requires an

aggregate and, if it has a relevant MQT available, can rewrite the query to run against the MQT instead of the base data.

Figure 5-2 shows a DB2 Graphical Explain for a query with no MQT and with an MQT. In both cases, the same SQL query against the base fact table is issued. The data comes from the fact table in the scenario with no MQT. When an MQT is present, DB2 UDB automatically rewrites the query to use the MQT instead of the base fact table. The query cost is dramatically lower when the MQT is used and performance can be improved by orders of magnitude since values are already pre-calculated.

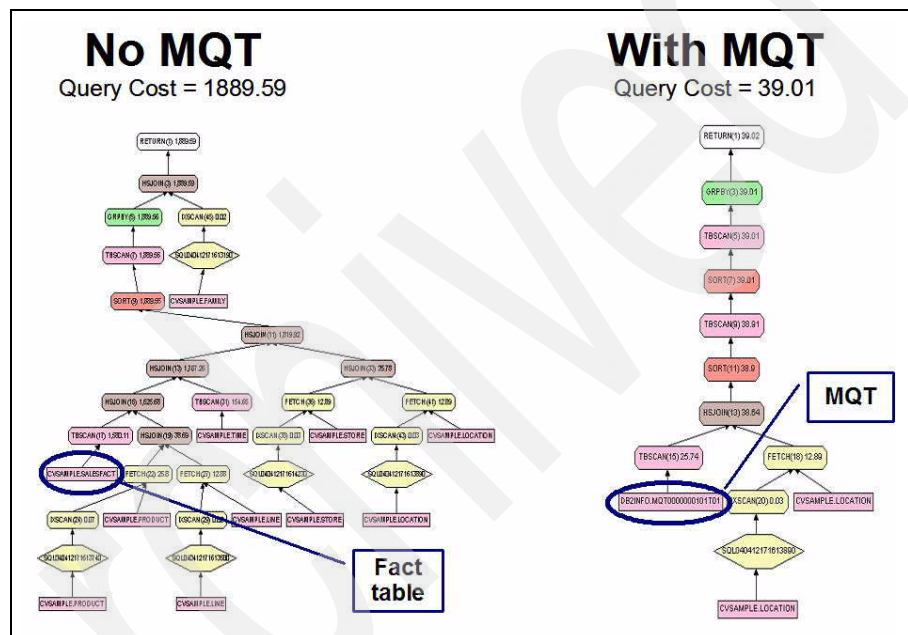


Figure 5-2 MQT explain comparison

The DB2 UDB Query Optimizer does not require the queries to be exact matches of an MQT. The optimizer may determine that a portion of the query can be resolved using an MQT, as seen in the example above. The optimizer can also use lower level aggregate MQTs to calculate higher level aggregates.

MQTs behave like regular tables in many ways and the same guidelines for optimizing data access using *tablespace* definitions, creating indexes, and issuing RUNSTATS apply to MQTs.

Important tip: Once MQTs are defined and built within DB2 UDB, users and BI tools should not construct queries to run directly against them. The SQL queries should be constructed against the base tables and not the MQTs. DB2 UDB will determine how the query should be rewritten and the optimal access path. In this manner, database administrators can drop and recreate new MQTs to reoptimize the environment as the data warehouse evolves with new tables and more data and different query workloads. Whenever a major update in the OLAP metadata occurs, the DWE OLAP Optimization Advisor should be re-run to determine whether new MQTs are recommended.

5.2.3 Optimization Advisor

DWE OLAP provides an Optimization Advisor that recommends MQTs based on the DWE OLAP metadata and environmental information, as shown in Figure 5-3. This relieves the database administrator from having to spend time and effort trying to determine the most effective MQTs to create.

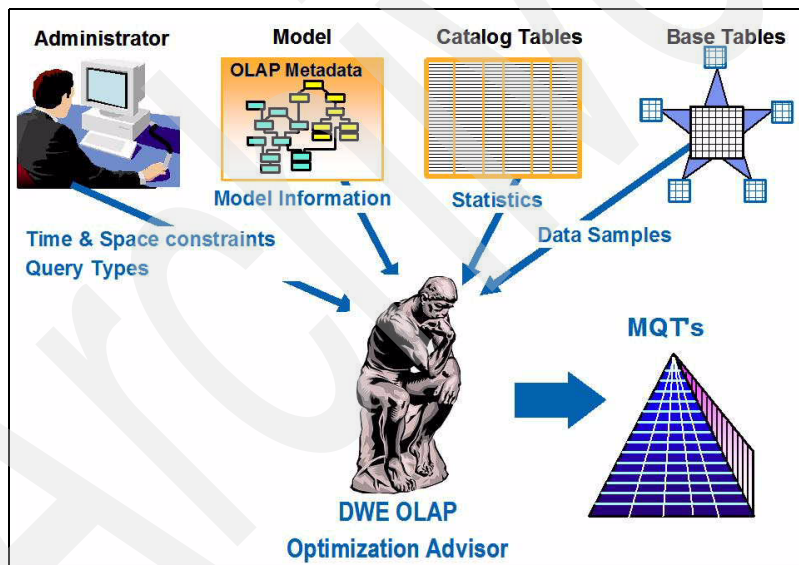


Figure 5-3 DWE OLAP Optimization Advisor

Several factors are taken into consideration by the Optimization Advisor to determine the best MQT (or set of MQTs) that can be recommended for a given environment:

- ▶ Time: How long do you want the advisor to think?
- ▶ Space: How much disk space do you want to provide for the MQTs?

- ▶ Query types: For what kind of queries are you optimizing (drill down, report, extract, drill through)?
- ▶ The cube model: What dimensions, hierarchies, attributes, and measures?
- ▶ The data: The advisor gets useful information from the DB2 statistics for each table, so statistics must be up-to-date. The advisor can also optionally get useful information from sampling.

Important: The Optimization Advisor is affected by the catalog statistics and sample data. Therefore, MQT recommendations will be different between a test and production system if the catalog statistics and data vary greatly.

After running the Optimization Advisor, scripts to create and update the MQTs are generated. These can be run immediately or scheduled to run during off hours. Database administrators (DBAs) do not need to understand and write any SQL. However, a DBA can modify the scripts before they are run.

Important: Be aware that the initial MQT build can take several hours and consume many computing resources, so we recommend that you run the scripts off hours or over the weekend.

5.3 Modeling OLAP with OLAP Center

With DB2 Cube Views V8, all the OLAP modeling and optimization is done in the OLAP Center, which is depicted in Figure 5-4.

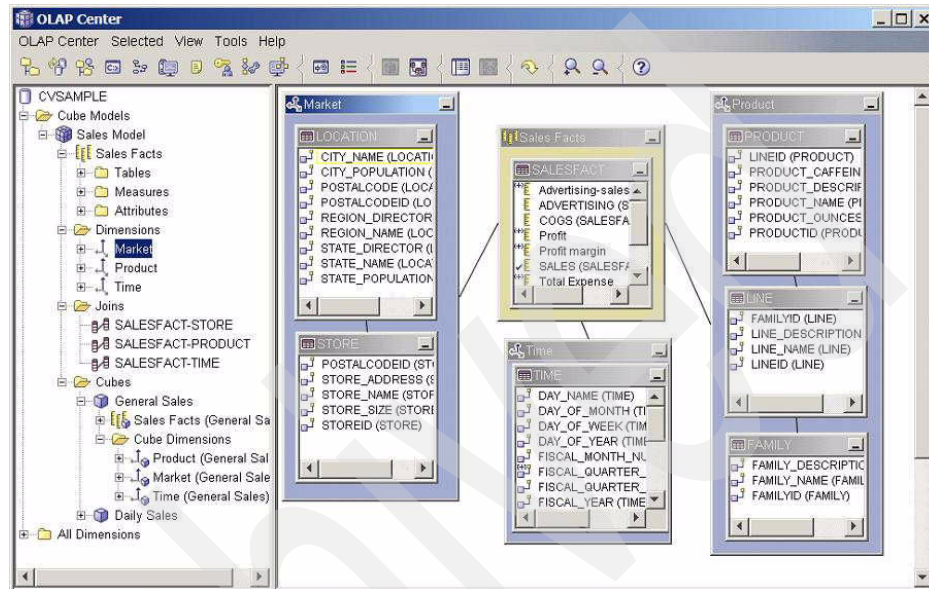


Figure 5-4 DB2 Cube Views OLAP Center

The OLAP Center, like the DB2 Control Center, is a stand-alone JAVA applet. It creates the DB2 Cube Views catalog tables for each database, and the OLAP metadata is entered or imported into DB2 UDB. The IBM Redbook *DB2 Cube Views: A Primer*, SG24-7002, extensively describes using the OLAP Center.

In DWE V9.1, the OLAP Center is replaced by the Design Studio. The OLAP Center will be phased out in a future release and clients should use the new Design Studio, as discussed in the next section.

Important: The OLAP Center in DB2 Cube Views V8 is still provided in DWE V9.1, and IBM will continue to support the OLAP Center throughout the DB2 Cube Views V8 lifetime. However, IBM will not introduce any further enhancements in the OLAP Center, so you should plan to migrate to DWE Design Studio.

5.3.1 Migrating from OLAP Center to Design Studio

Migration from the OLAP Center to the Design Studio in DWE V9.1 is easy. All the OLAP metadata from the OLAP Center resides in the OLAP-enabled database. Use the Design Studio Database Explorer to connect to the OLAP-enabled database and the OLAP objects will appear and be valid in the OLAP objects folder under the database schema.

If the DB2 V8 database was previously at maintenance Fixpack level 7 (FP7), FP8, or FP9 and DWE V9.1 (which includes DB2 V8 FP10) is installed, the OLAP metadata will need to be migrated to the FP10 level. Upon connecting to the database with Database Explorer and browsing the OLAP metadata, a message will appear notifying that the metadata is back-level and ask if the metadata should be migrated. On answering *yes* to the prompt, the metadata will be migrated. The metadata can also be migrated using the DWE Admin Console.

The import/export wizards in the Design Studio import and export an XML file in the OLAP Center format. This is the same behavior as the import wizard and export dialog in OLAP Center. Consequently, any XML file exchanged with the OLAP Center should work with Design Studio.

5.4 Modeling OLAP with DWE Design Studio

In DWE V9.1, DB2 Cube Views has been integrated into the DWE Design Studio, enabling a common integrated data warehouse design environment. Users familiar with the OLAP Center in DB2 Cube Views V8 will find entering the OLAP metadata within the DWE Design Studio very similar. The IBM Redbook *DB2 Cube Views: A Primer*, SG24-7002, discusses many valuable concepts and topics that are still applicable for DWE V9.1 and is highly recommended. The process of modeling the OLAP objects is similar, except that DWE V9.1 uses the Design Studio instead of the OLAP Center.

Within the Design Studio, creating an OLAP model primarily takes place using the Database Explorer, Data Project Explorer, and Project views, as seen in Figure 5-5. Users should review and be familiar with Chapter 3, “The DB2 DWE Design Studio” on page 29, and Chapter 4, “Developing the physical data model” on page 57, before working with DWE OLAP.

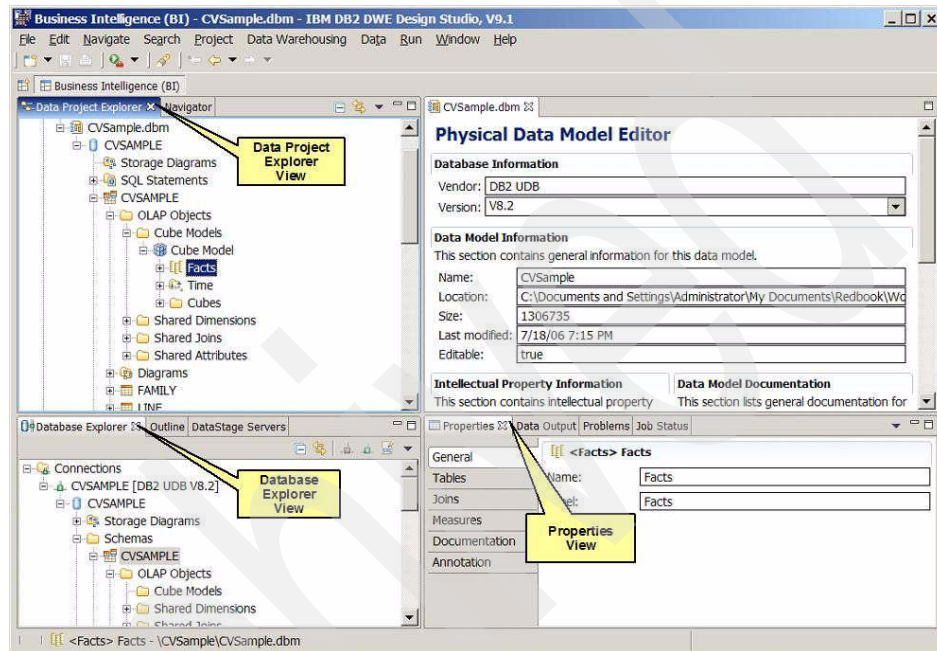


Figure 5-5 Design Studio - DWE OLAP views

5.4.1 Database Explorer

The Database Explorer provides a tree view that allows the user to browse the catalog contents of a relational database. Initially a connection to a database is created, then the connection node can be expanded to reveal a database icon, which can be further expanded to display nodes representing the catalog contents of the database, as shown in Figure 5-6.

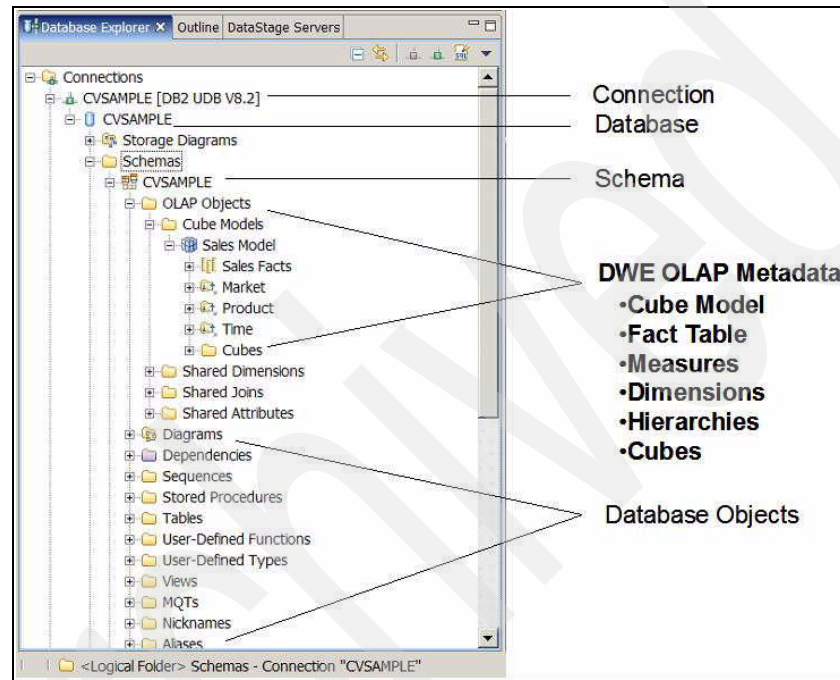


Figure 5-6 Database Explorer - DWE OLAP metadata

OLAP objects, like all other database objects, are displayed within the context of the database schema in which they are contained. Within the Database Explorer, existing OLAP objects can only be viewed. However, OLAP objects can be imported from an XML file and created as new physical objects in the Database Explorer. Otherwise, creating new OLAP objects takes place in the Data Project Explorer.

5.4.2 Properties view

The Properties view, as shown in Figure 5-7 on page 93, allows the user to view the properties of the selected object. The Properties view is applicable to objects in either the Database Explorer or Data Project Explorer. Properties for objects

selected in the Database Explorer are read only. Properties for objects selected in the Project Explorer are editable.

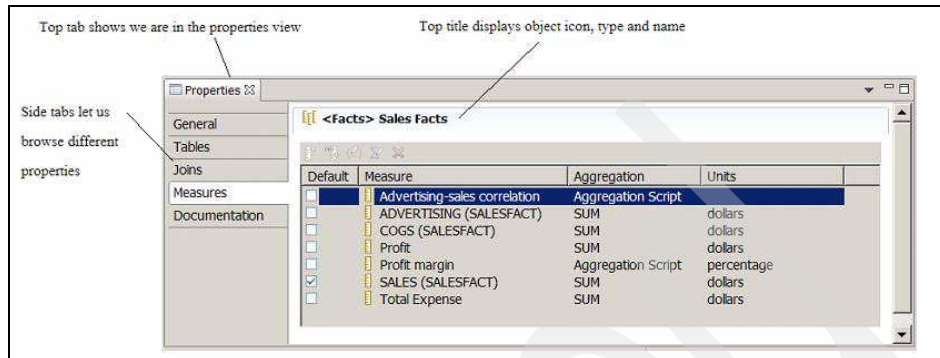


Figure 5-7 Properties view

5.4.3 Data Project Explorer view

The Data Project Explorer view in Design Studio is used to view projects and data models. A project can contain one or more physical data models, and each physical data model is a model of either a complete relational database or a part of a relational database such as a schema and its contents. Physical data models can contain OLAP objects as well as the traditional relational objects such as tables and views, as shown in Figure 5-8. As new projects are created, each project maps to a folder in the Windows file system while each physical data model maps to a file in the folder.

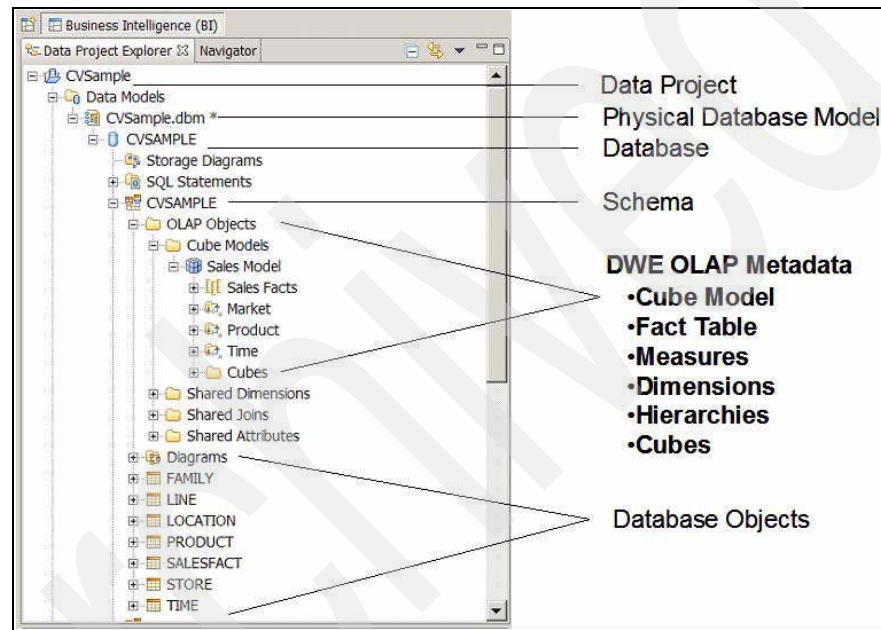


Figure 5-8 Data Project Explorer - DWE OLAP metadata

Context menus using the right mouse button are available from selected objects in the view to create, delete, and alter the objects. A physical data model is first needed in order to model OLAP metadata objects. A physical data model containing OLAP objects can be created by:

- ▶ Reverse engineering an existing DB2 database containing OLAP objects.
- ▶ Creating an empty physical model and then creating OLAP objects by:
 - Importing objects from a DWE OLAP XML file
 - Using the quick start wizard
 - Using the context menus (right-click)

- Manually copying objects from a database in the Database Explorer using either the clipboard or drag and drop

5.4.4 Creating the CVSAMPLE sample database

DWE OLAP provides data to create a sample database called CVSAMPLE. The sample data includes a set of tables that contain data about a fictitious company that sells beverages. A set of metadata objects that describe the sample data tables is also included. Most of the examples in the DWE Design Studio User Guide are based on the CVSAMPLE database and corresponding cube model.

To create and populate the CVSAMPLE database, perform the following steps:

1. Launch a DB2 command window on the database server. From Windows, the DB2 command window can be found in the **Start → Programs → IBM DB2 → Command Line Tools → Command Window**.
2. Create a sample database called CVSAMPLE by entering the following command:

```
db2 create db cvsample
```
3. Connect to the database by entering the command:

```
db2 connect to cvsample
```
4. Change to the `SQLLIB\samples\olap\cvsample` directory and then enter the following DB2 command to create the CVSAMPLE tables:

```
db2 -tvf CVSampleTables.sql
```

5.4.5 Preparing a DB2 UDB database for DWE OLAP

Setting up a database to be used with DWE OLAP includes:

- ▶ Registering the DWE OLAP stored procedure with the database
- ▶ Creating metadata catalog tables for DWE OLAP

This preparation is done manually using DB2 commands or from the Design Studio.

Using DB2 commands

The steps are:

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

```
db2 connect to cvsample
```
2. Change to the `SQLLIB\misc` directory and enter the following command:

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

Attention: Do *not* modify the db2mdapi.sql script, as the results will be unpredictable.

Using Design Studio

To use this:

1. Open the Design Studio. From a Windows system select **Start** → **Programs** → **IBM DB2 Data Warehouse Edition** → **Design Studio**.
2. Enter a workplace location. Note that this is where all the metadata will be stored in an XML file. These files should be backed-up on a regular basis.
3. Look at the Database Explorer in the lower left corner. Use the context menu by right-clicking **CVSAMPLE** and selecting **Reconnect** to connect to the database, as shown in Figure 5-9. Complete the login process by specifying your user ID and password. If the database CVSAMPLE is not there, highlight the **Connections** folder and use the context menu by right-clicking and selecting **New Connection**. Complete the wizard entry to establish a connection.

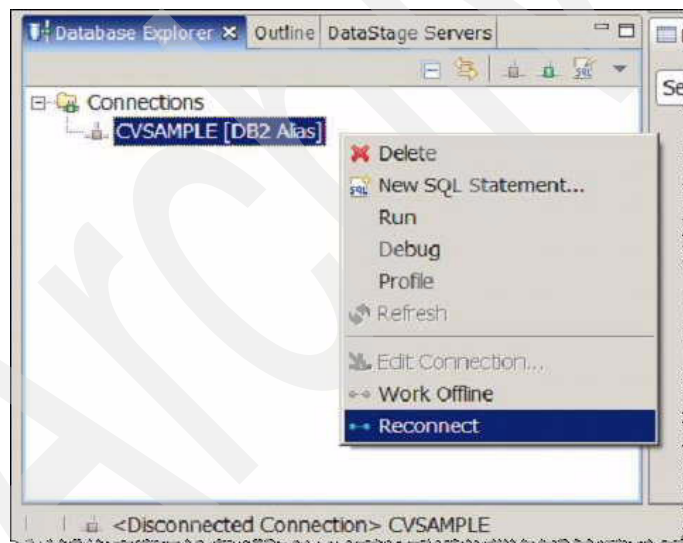


Figure 5-9 Design Studio - Database Explorer

4. Expand the tree folders **CVSAMPLE** → **Schemas** → **CVSAMPLE** → **OLAP Objects** and double-click the **Cube Model** folder, as seen in Figure 5-10 on page 97.

5. The OLAP Question message, shown in Figure 5-10, will appear. Select **Yes** and the DWE OLAP catalog tables will be created for this database.

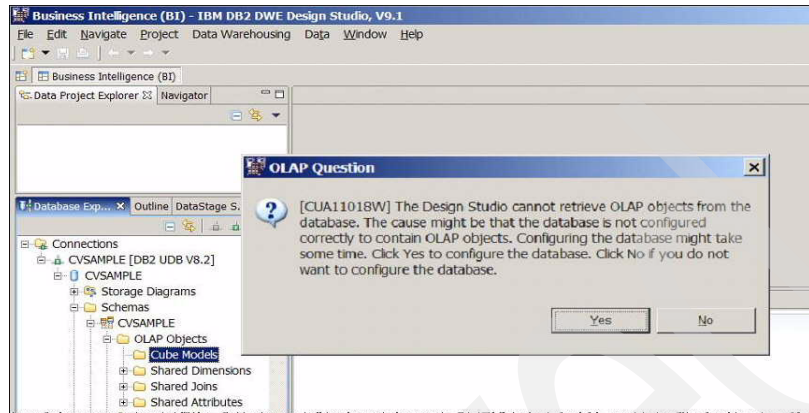


Figure 5-10 Creating the DWE OLAP catalog tables

Attention: By selecting **NO**, the DWE OLAP catalogs can be created later. Select the **OLAP Objects** folder, right-click, and select **Refresh**. Double-click the **Cube Models** and you will be prompted with the OLAP question again.

6. To validate that the DWE OLAP catalog tables have been created, right-click to bring up the context menu and select **Refresh**. Look in the schemas folder for the DB2INFO schema. New DWE OLAP catalog tables, as shown in Figure 5-11, should be present.

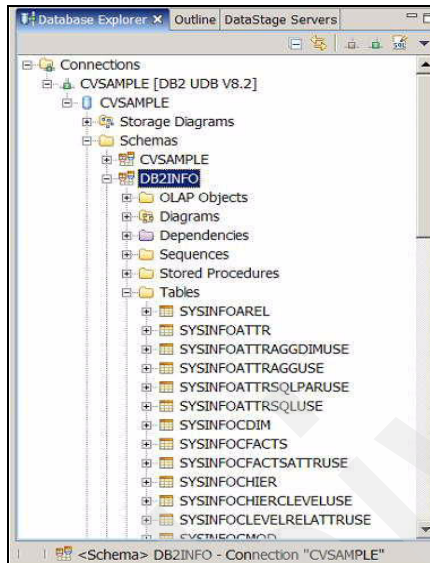


Figure 5-11 DWE OLAP catalog tables

5.4.6 Creating a new data project

A data project needs to be defined before an OLAP model can be created. There are several types of Data Projects, but for DWE OLAP use the data design project (OLAP).

1. From the Design Studio, select **File** → **New** → **Data Design Project (OLAP)**, as highlighted in Figure 5-12.

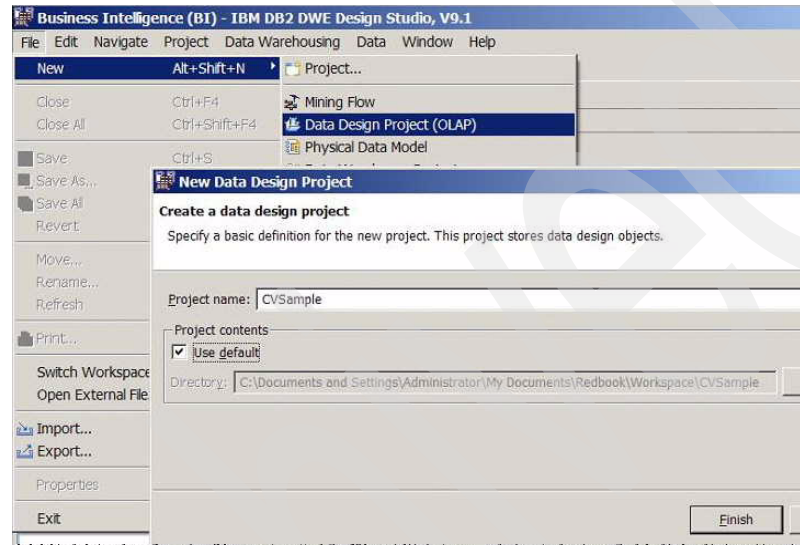


Figure 5-12 New data project

2. Type CVSample for the project name and click **Finish**.

5.4.7 Creating a new physical data model

Before the OLAP model is created, the physical data model of the underlying base tables (facts and dimensions) should be defined.

1. Select the **CVSAMPLE** project.

2. Right-click and use the context menu to select **New** → **Physical Data Model**, as shown in Figure 5-13.

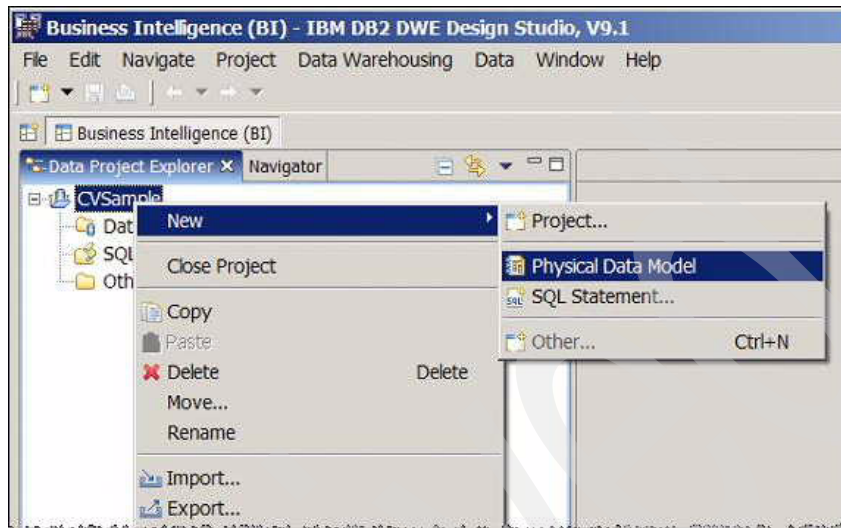


Figure 5-13 New physical data model

3. In the File Name box, enter CVSAMPLE, as shown on the left side of Figure 5-14.

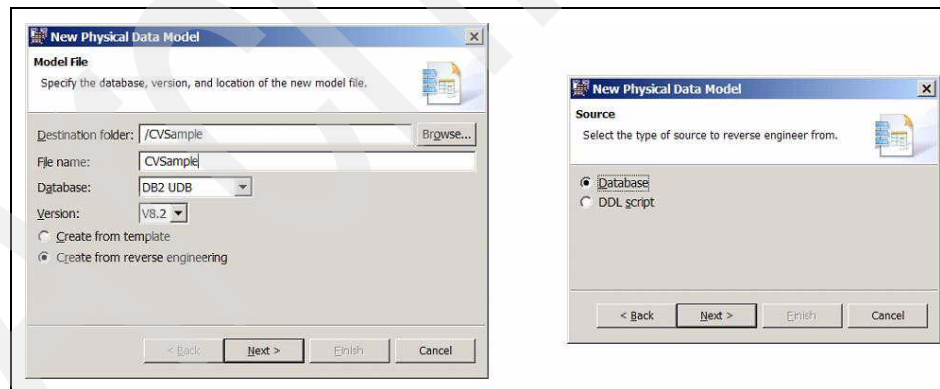


Figure 5-14 New physical data model options

4. Select the appropriate database and version. In this example, **DB2 UDB** and **v8.2**.
5. Select the **Create from reverse engineering** radio button and then **Next**.

6. From the new physical data model window, select **Database**, as seen on the right side of Figure 5-14 on page 100. This specifies that an existing database will be used.
7. As shown in Figure 5-15, select **Use an existing connection**, select the **CVSAMPLE** database, and click **Next**.

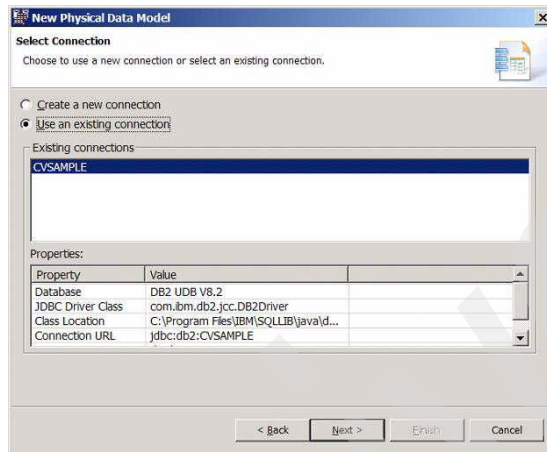


Figure 5-15 New physical data model connections

8. As shown on the left-side of Figure 5-16, click **Select All** to import all schemas, then click **Next**.
9. Accept the default Database Elements, as shown on the right side of Figure 5-16, and click **Next**.

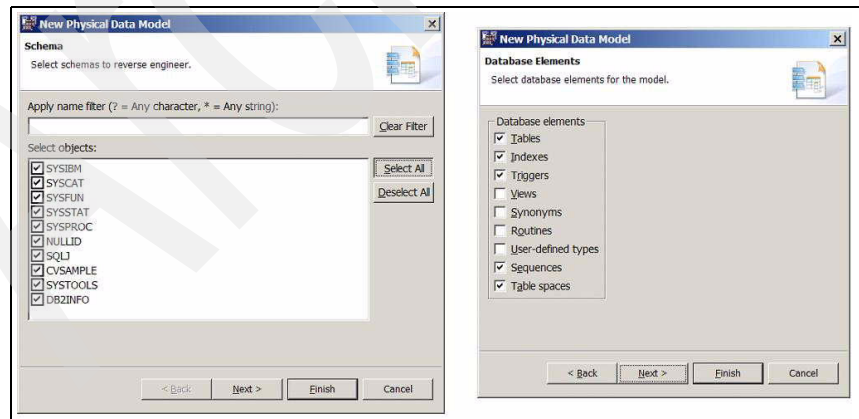


Figure 5-16 New physical data model options

10. Check the **Generate Diagrams Overview** box and click **Finish**, as shown in Figure 5-17.

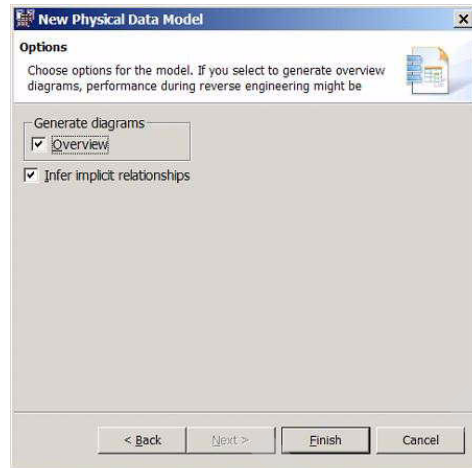


Figure 5-17 New physical data model options

5.4.8 Quick Start Wizard

DWE OLAP provides a Quick Start wizard to define OLAP objects within a physical data model. This can be opened by using the context menu, as shown in Figure 5-18.

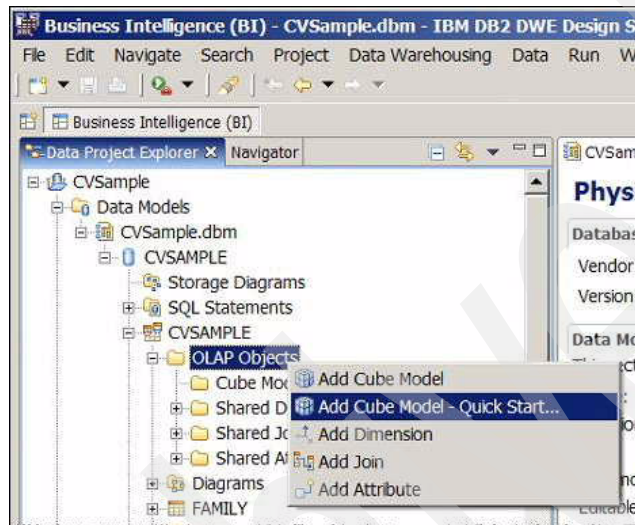


Figure 5-18 Quick Start wizard

The Quick Start wizard creates the OLAP objects that it can logically infer from the relational star schema or snowflake schema. Based on the fact table, the wizard detects the corresponding dimensions, joins, attributes, and measures.

Before using the Quick Start wizard, 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. The wizard completes the cube model by creating the dimensions, attributes, and joins using the RI constraints.

Attention: The generated cube model does not contain any hierarchies, levels, or cubes, so these have to be created by the user after the Quick Start wizard has finished.

5.4.9 Defining cube models

To create a complete cube model without using the Quick Start wizard, create an empty cube model and then add facts objects, dimensions, hierarchies, and levels for each dimension in that cube model. The cube model is the highest level

OLAP object and represents a relational star schema, including information about all tables, columns, joins, and the relationship between each of these objects.

To create a cube model:

1. Navigate down the tree and select the **Cube Model** folder. Use the context menu and select **Add Cube Model**, as shown in Figure 5-19.

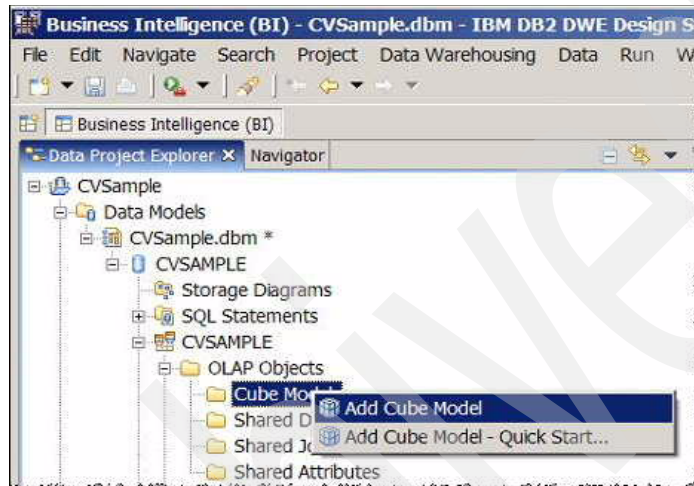


Figure 5-19 Add new cube model

2. Enter Sales Model as the name of the cube model.

Tip: Go to 5.7.1, “Import” on page 133, and read the instructions on importing the CVSample metadata to see a full example of an OLAP model in DWE OLAP.

5.4.10 Defining the fact table and measures

The first step within a cube model is to identify the fact table. The process of creating the facts object also includes identifying and defining the measures, aggregations, and attributes.

To create the fact table and measures:

1. In the Data Project Explorer, expand the cube model. Select the **Table** folder under the Facts folder, as shown in Figure 5-20. Use the context menu and select **Add Existing Tables**.

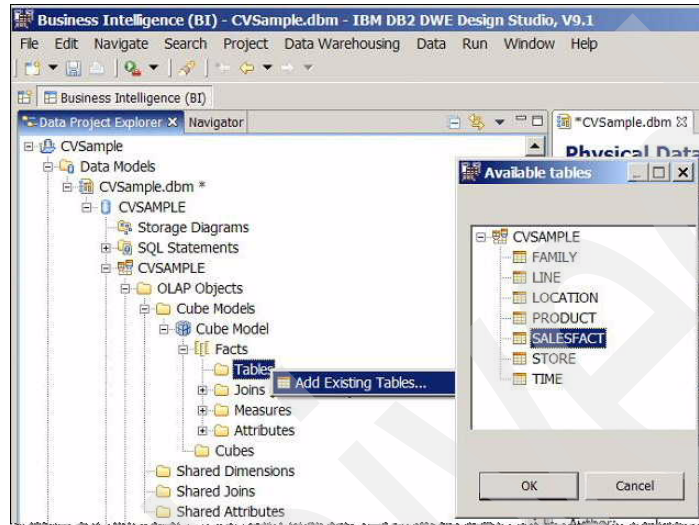


Figure 5-20 Add new cube model - fact table

2. Select the **SALESFACT** table and click **OK**.

Note: If more than one table is needed to build the fact object, then the join between fact tables needs to be defined.

3. Select the **Measures** folder, as shown in Figure 5-21. Use the context menu and select **Add Measure from Columns**.

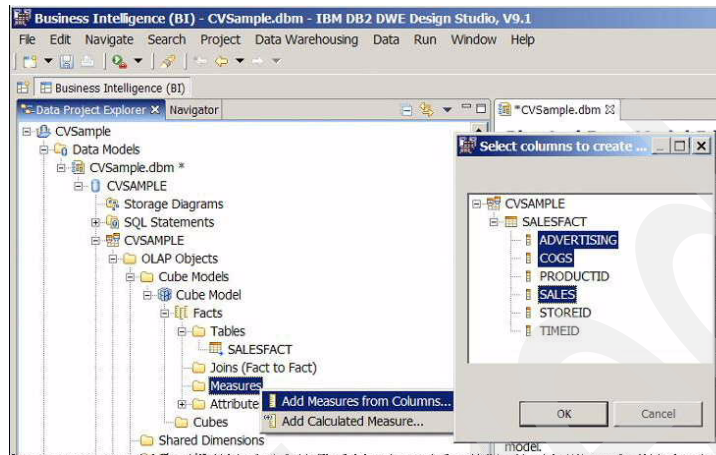


Figure 5-21 Add new cube model - measures

4. Expand the tree and select the **SALESFACT** table. Holding the **CTL** key down, select **ADVERTISING**, **COGS**, and **SALES**, then click **OK**.

Note: Calculated measures can be calculated based on existing measures from the fact table. For example, PROFIT is calculated as SALES- COGS. Click **Add Calculated Measure** to launch the SQL expression builder.

5.4.11 Defining dimensions

Dimensions provide a way to categorize a set of related attributes that together describe one aspect of a measure. Dimensions in cube models organize the data in the facts object according to logical categories such as region, product, or time.

To create dimensions:

1. In the Data Project Explorer, expand the schema. Select the **Cube Model** folder.

2. Use the context menu and select **Add Data Object** → **Dimension**, as shown in Figure 5-22.

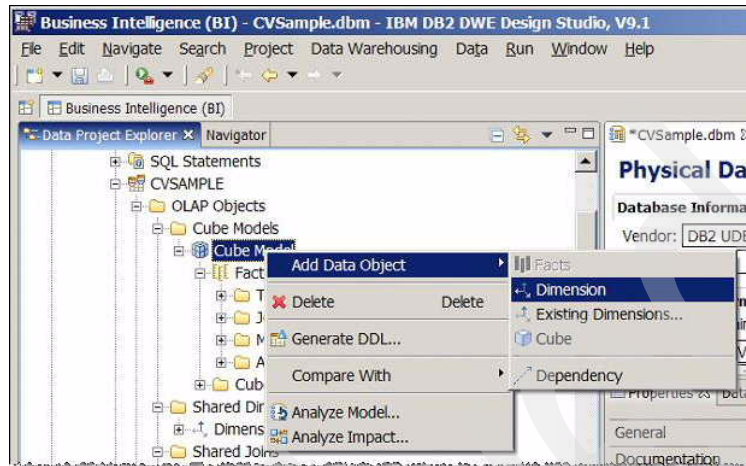


Figure 5-22 Add new cube model - dimensions

3. Type the name Time for the dimension.
4. Select the **Tables** folder and use the context menu to **Add Existing Table**, as shown in Figure 5-23.

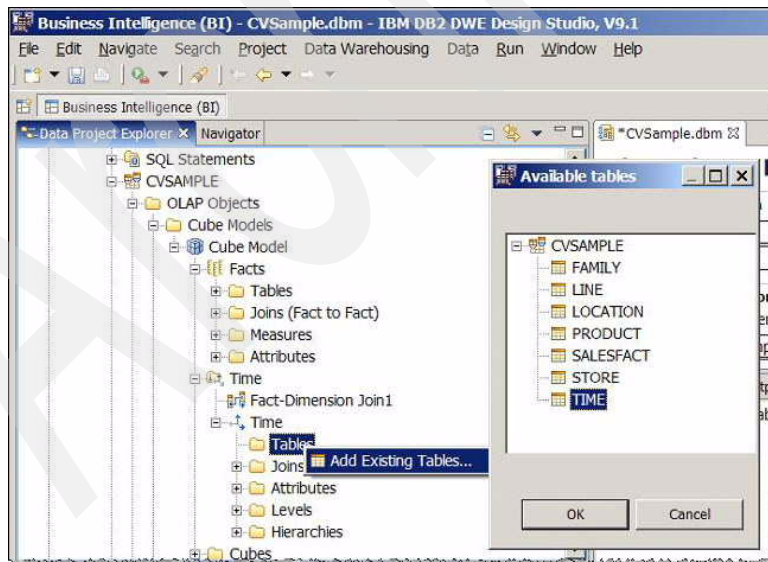


Figure 5-23 Add new cube model - dimensions

5. Select the **TIME** table and click **OK**.

Tip: Select the specific dimension and use the Properties view to make changes to that dimension. If you cannot see the Properties view, click **Window** → **Show View** → **Properties**.

On the General page, when you specify a name for the dimension information folder, the dimension contained in the folder automatically takes the same name. In the other pages of the Properties view, table selections for the dimension, joins between the tables in a dimension, attributes for the dimension, and the type of the dimension can be modified after they have been defined using the context menus.

5.4.12 Defining joins

The relationship between a dimension and the fact object must be defined for each dimension in a cube model. The relationship is defined by a fact-to-dimension join.

To create a fact-to-dimension join:

1. In the Data Project Explorer, expand the folder for the existing dimension and select the **Fact-Dimension Join** object, as shown in Figure 5-24.

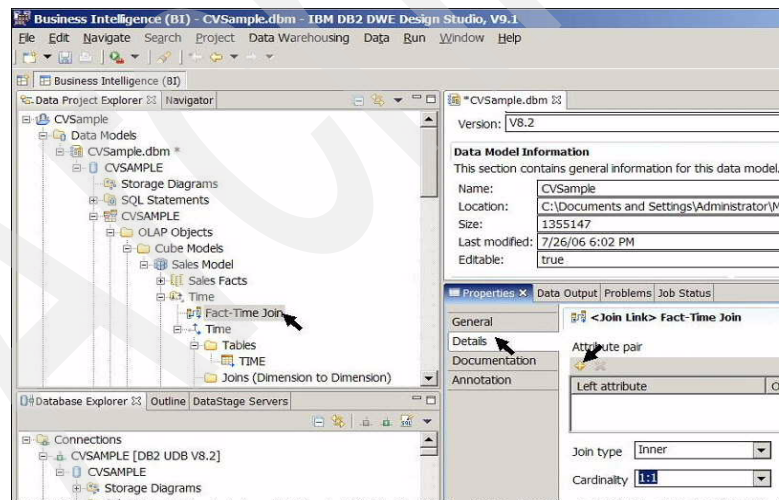


Figure 5-24 Adding joins

2. Rename the join as Fact-Time Join using the F2 key, or from the Properties view select the **General** page and enter the name.
3. From the Properties view, select the **Details** page.
4. To define the join, add an Attribute Pair by clicking the green plus button (+).
5. As shown in Figure 5-25, select the **TIMEID** from each table as the key. Select **INNER JOIN** as the type of join and select **1: Many** as the cardinality.

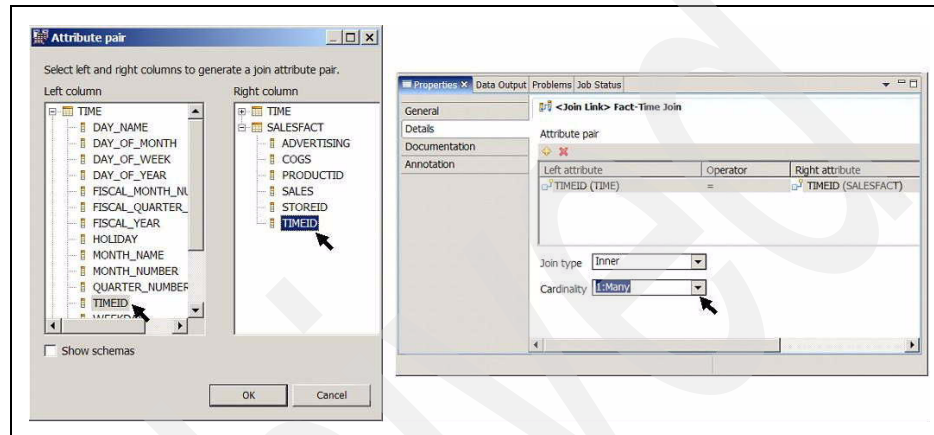


Figure 5-25 Adding joins

Tip: Dimension-to-dimension joins are used if a snow-flake schema is being modeled. Before you can create a dimension-to-dimension join, you must create a dimension and add two or more tables to the dimension.

5.4.13 Defining attributes

Attributes are either a single column in a table or an expression that is a combination of a set of columns or other attributes.

To create an attribute in a dimension:

1. In the Data Project Explorer, expand the folder for the dimension. Select the **Attributes** folder and use the context menu to select **Add Attributes from Columns**, as shown in Figure 5-26.

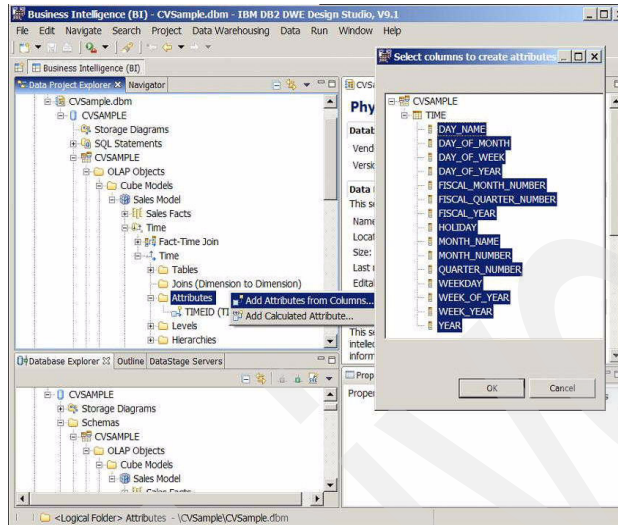


Figure 5-26 Adding attributes

2. In the Select Columns to Create Attributes window, expand the table and select all of the columns by holding down the Shift key and clicking **OK**.

Creating calculated attributes will bring up the OLAP SQL Expression® Builder window to create the source expression. The SQL Expression Builder provides lists of available attributes, columns, operators, functions, and constants.

5.4.14 Defining levels

The level object represents a set of information (attributes) that is logically related at the same level of detail and uniquely identifies the data in the level. Year, quarter, month, week, and day would be examples of different levels. Levels will be used in defining dimensional hierarchies.

To create a level for a hierarchy:

1. In the Data Project Explorer, expand a dimension for a cube model. Select the **Levels** folder and use the context menu to select **Add Level**, as shown in Figure 5-27.

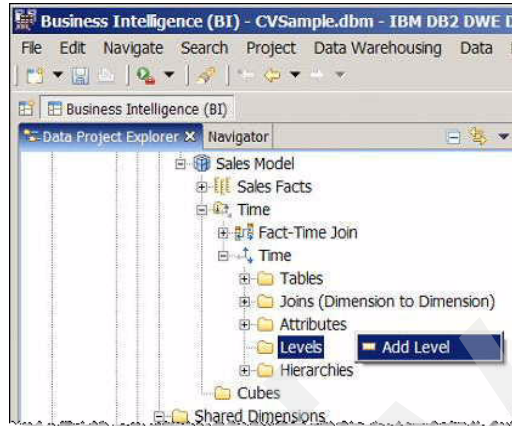


Figure 5-27 Adding levels

2. Make sure that you can see the Properties view, as shown in Figure 5-28. If you cannot see the Properties view, click **Window** → **Show View** → **Properties**.

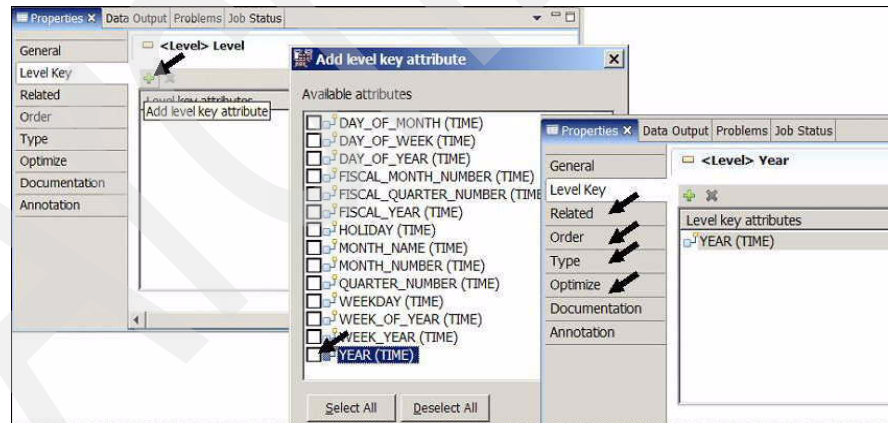


Figure 5-28 Adding Levels Properties view

3. Select the **Level Key** page and select the green plus button (+) to add a level key attribute.
4. In the Add level key attribute window, select **YEAR(TIME)** and click **OK**.

5. Select the **General** page and rename the level as **YEAR** in the Name field. The label field will automatically be updated.
6. Browse the remaining pages in the Properties view. Details on each page can be found in the online help (**Programs** → **DB2 Data Warehouse Edition** → **DB2 and DWE Information Center**) or the *Design Studio User's Guide Version 9.1 Linux, UNIX, and Windows*, SC18-9802.

5.4.15 Defining hierarchies

Hierarchies define relationships between two or more levels within a given dimension of a cube model. For example, year-quarter-month could be a hierarchy within a time dimension. Defining these relationships provides a navigational and computational means of traversing a given dimension.

Note: Defining hierarchies with various levels requires that the levels first be defined. In the following examples, the *CVSampleMetadata.xml* file has been imported as described in 5.7.1, “Import” on page 133.

To create a hierarchy for a dimension:

1. In the Data Project Explorer, expand the dimension folder and select the **Hierarchies** folder. Use the context menu and select **Add Hierarchy**, as shown in Figure 5-29.

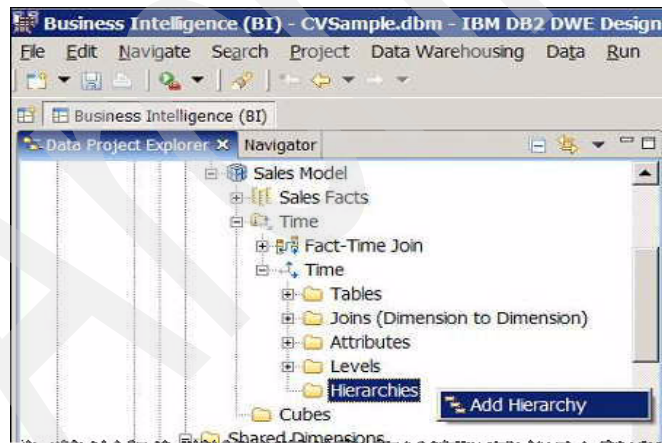


Figure 5-29 Adding hierarchies

2. Make sure that you can see the Properties view, as shown in Figure 5-30. In that view, select the existing time dimension to examine the defined hierarchies.

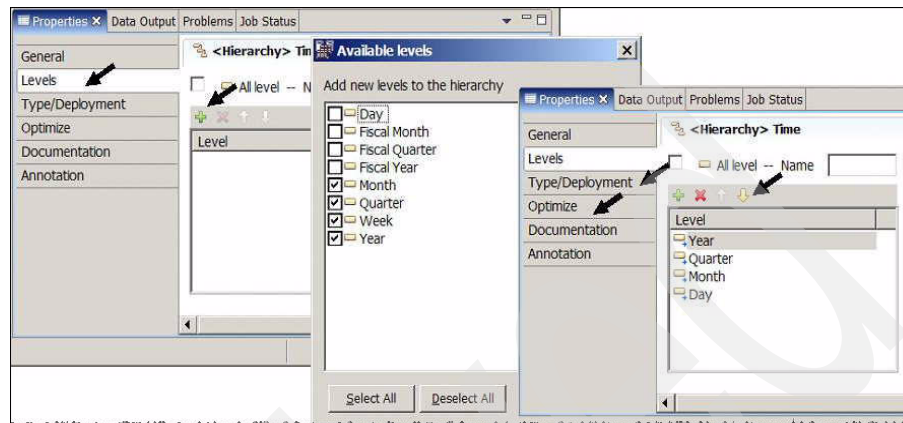


Figure 5-30 Adding Hierarchies Property view

3. Select the **Level Page** and click the green plus button (+) to add a new level.
4. In the Available levels window, check the desired levels and click **OK**.
5. Use the up and down arrows to order the levels.
6. Browse the remaining page tabs in the Properties view. Details about each page can be found in the online help (**Programs → DB2 Data Warehouse Edition → DB2 and DWE Information Center**) or *Design Studio User's Guide Version 9.1 Linux, UNIX, and Windows*, SC18-9802.

5.4.16 Defining cubes

After the cube model is defined, individual cube objects that contain all or a subset of the cube model objects are created. A cube precisely defines an OLAP cube and includes cube facts, cube dimensions, cube hierarchies, cube levels, and cube measures.

Cubes specify regions of the cube model that are significant. The cube object can be considered similar to the various MOLAP cubes that are often built for a specific subject area. A cube can define a subset of the data used by a vendor BI tool.

One or more cubes can be derived from a cube model. A cube has a cube fact as the central object and cube dimensions. The cube fact (measures) and cube dimensions are 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.

Note: A cube model must have a fact object and at least one dimension with a hierarchy before a cube can be defined. In the following example, the CVSampleMetadata.xml file has been imported as described in 5.7.1, “Import” on page 133.

Here is an example of creating a cube:

1. In the Data Project Explorer, expand the schema that contains the cube model from which you want to derive the cube. Select the cube folder and use the context menu to select **Add Cube**, as shown in Figure 5-31.

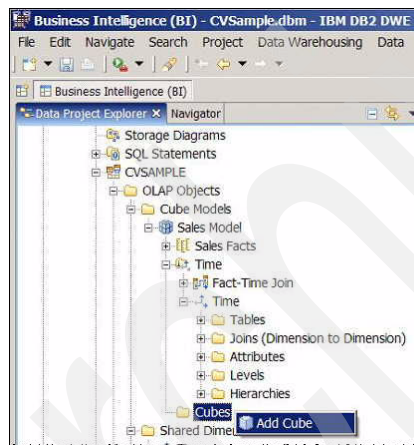


Figure 5-31 Adding a new cube

2. Make sure that you can see the Properties view, as shown in Figure 5-32. In the example below, expand the existing **General Sales cube** and select the **Cube Facts** object.

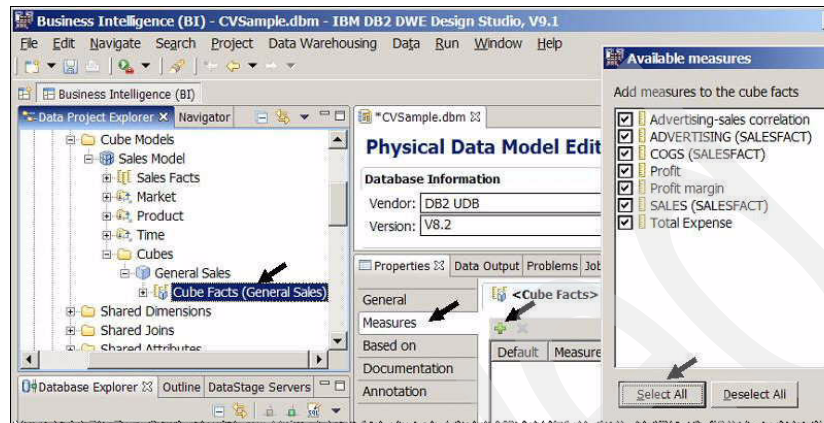


Figure 5-32 Adding a cube fact

3. In the properties view, select the **Measures** page and click the green plus button (+) to add new measures. Click the red x to remove measures.
4. In the Available measures window, select the measures for this particular cube. Click **OK**.
5. Select **General Sales cube**, as shown in Figure 5-33.

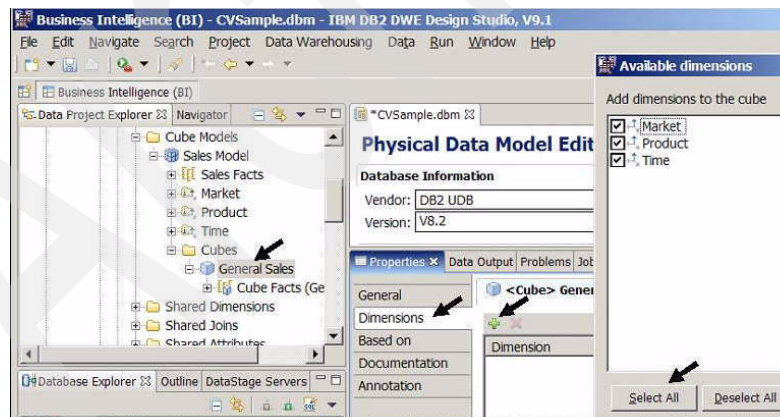


Figure 5-33 Adding a cube dimension

6. In the properties view, select the **Dimensions** page. Click the green plus button (+) to add new dimensions. Click the red x to remove a dimension.

7. In the Available dimensions window, select new dimensions and click **OK**.
8. Select the **MARKET** dimension, as shown in Figure 5-34.

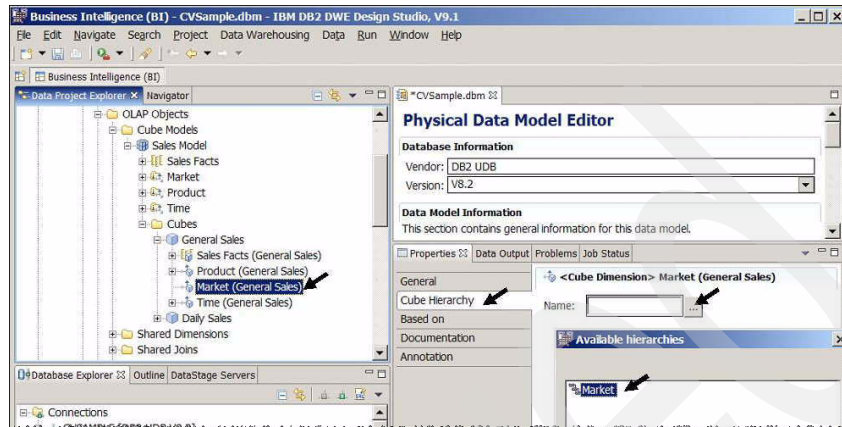


Figure 5-34 Adding a cube hierarchy

9. In the Properties view, select the **Cube Hierarchy** page and click the **...** button to see a list of available cube hierarchies.
 10. Select a hierarchy and click **OK**.
- Repeat steps 8–10 for each dimension.

5.4.17 Analyze OLAP objects for validity

After creating a physical data and OLAP model, the models can be validated using the Analyze Model option available from the context menus displayed from the Data Project Explorer, as shown in Figure 5-35.

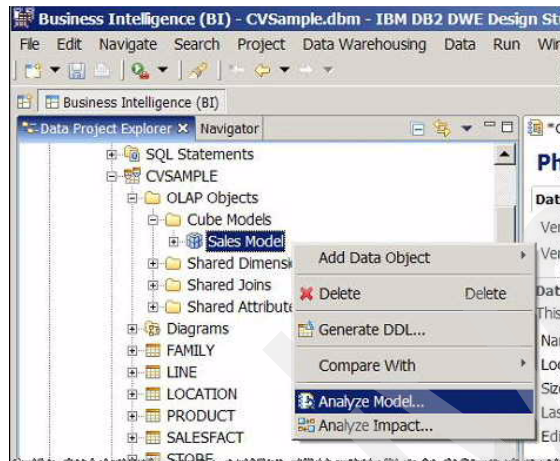


Figure 5-35 Analyze Model

The Analyze Model dialog can be used to check OLAP base rules or OLAP optimization rules. If all OLAP base rules are followed, then the OLAP objects are valid to be deployed to a DB2 database. Furthermore, if all the OLAP optimization rules are followed, then the OLAP cube models are good for the Optimization wizard to be run and produce optimization recommendations. If any rules are broken by the model, the Properties view in Design Studio displays the rule that was broken and the object.

A cube model will validate successfully only after you add the following mandatory components:

- ▶ At least one facts object
- ▶ At least one dimension
- ▶ A hierarchy defined for at least one dimension
- ▶ Joins between the existing facts object and dimensions
- ▶ Attributes that reference existing table columns

5.5 Deploying objects to DB2 UDB databases

Once a physical data model and its OLAP objects have been created and validated, the objects can be deployed to a DB2 UDB database. This can be done in two different ways:

- ▶ By using the Generate DDL wizard to create a DDL SQL script, which will deploy all the objects from the model to a DB2 database.
- ▶ By comparing the data model with an existing DB2 database and creating a DDL script that, when run, will just write out the differences to the target DB2 database. This option is useful when the model was initially created by reverse engineering the DB2 UDB database.

5.5.1 Generate DDL

The Generate DDL wizard, shown in Figure 5-36, can be used to create and optionally run a DDL script. The script will create the set of objects contained in a physical data model. The contents of the script can be tailored by selecting various options provided in the Generate DDL wizard pages. The wizard is opened by clicking an object in the physical database and selecting the **Generate DDL** option from the context menu. If the OLAP objects check box is selected from the second page of the wizard, the generated script will contain calls to the MD_MESSAGE stored procedure to create the OLAP objects that have been modeled in the physical data model.

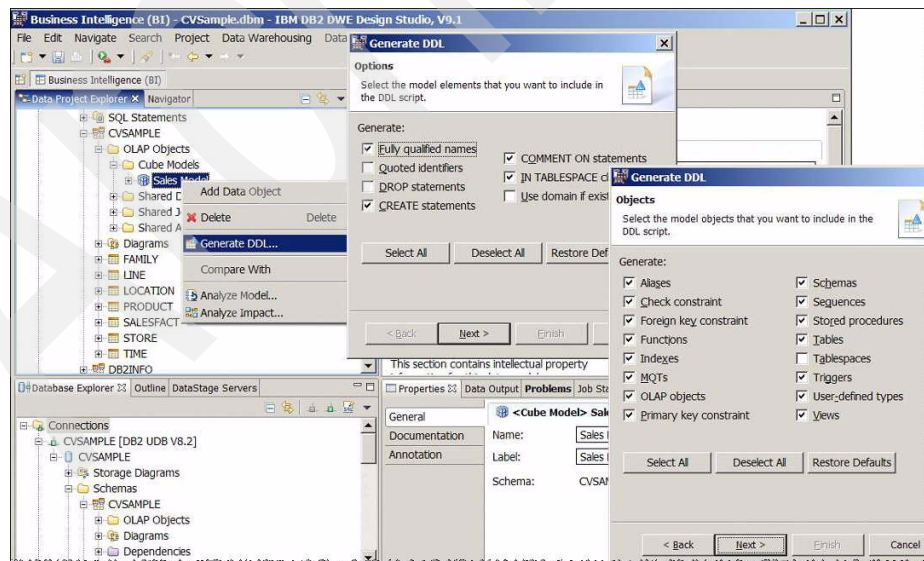


Figure 5-36 Generate DDL

If generating DDL for a mixture of relational and OLAP objects, make sure that the **Fully qualified names** check box is selected. Because there are two forms of OLAP metadata, you will be prompted for the version of OLAP metadata you want the DDL script to contain. You can choose to execute DDL immediately, save to a script, or both.

5.5.2 Compare and Sync Editor and Delta DDL

If the physical data model being deployed was initially created by reverse engineering an existing DB2 UDB database, then only the delta changes made to the physical data model need to be deployed to the target DB2 UDB database. To accomplish this deployment the user must first compare the physical data model with the existing database, as shown in Figure 5-37.

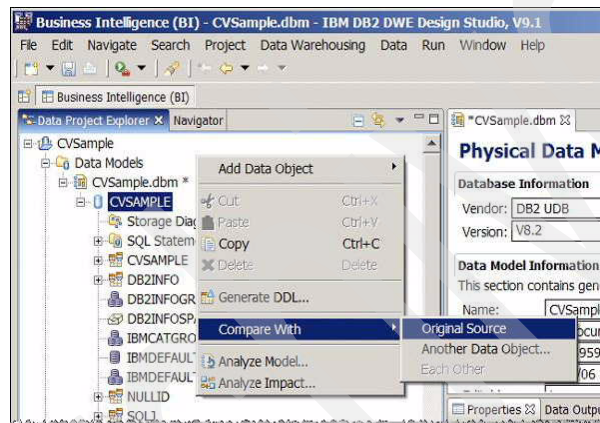


Figure 5-37 Compare objects

This opens the Compare and Sync Editor, which shows the differences between the physical data model and the existing database, as shown in Figure 5-38.

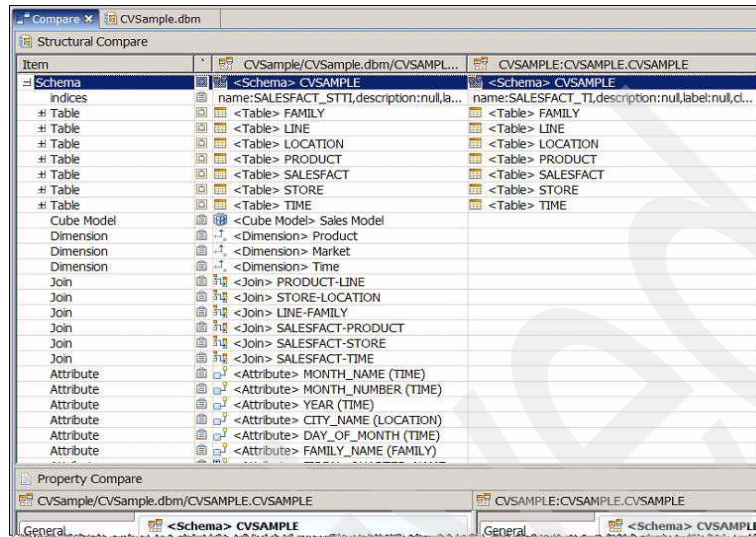


Figure 5-38 Compare models report

Next copy the changes from the physical data model to the existing DB2 UDB database by clicking the appropriate copy button. When objects are copied to an existing DB2 UDB database, this causes the Generate delta DDL button to be enabled. Clicking this will display the Generate Delta DDL wizard, which will allow you to create a DDL script that, when executed, will write out the changes to the target DB2 UDB database. The wizard allows you to run the script immediately or save it and run it later.

5.5.3 DWE OLAP Optimization Advisor

The DWE OLAP Optimization Advisor provides recommendations about which MQTs and indexes to build in order to improve SQL queries executed against the star schema tables. Because the Optimization Advisor function does sampling on rows in the tables, the Optimization Advisor can only be run against a cube model in the Database Explorer — that is, a cube model that exists in an existing DB2 UDB database. Consequently, the Optimization Advisor can be launched from the context menu of a cube model in the Database Explorer, as shown in Figure 5-39. The Optimization Advisor should be run after the DWE OLAP model has passed validation.

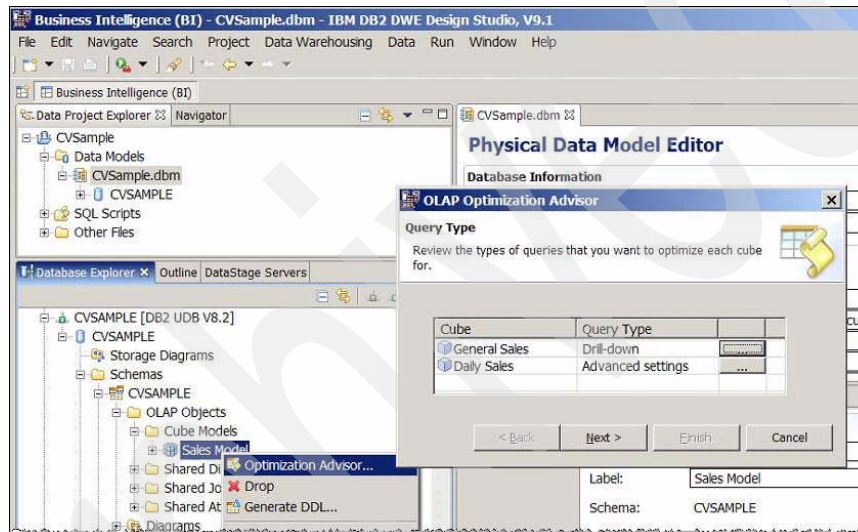


Figure 5-39 Optimization Advisor

As shown in Figure 5-40, the Optimization Advisor allows the modeler to specify a query type for each cube. The options include:

- ▶ Drill-down queries: typically access a subset of the data at the top of a cube
- ▶ Report queries: equally likely to access any part of the cube
- ▶ MOLAP extract queries: load data from the lowest levels of the cube dimensions into a MOLAP (Multidimensional OLAP) data store
- ▶ Advanced settings: indicate that optimization slices have been specified that are expected to be queried most frequently

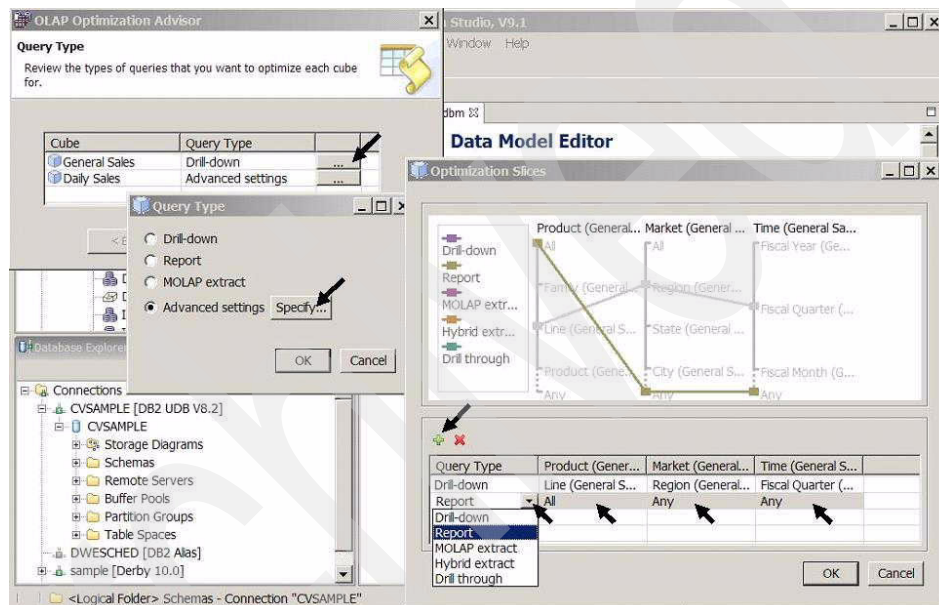


Figure 5-40 Optimization Advisor advanced settings slices

An optimization slice is a method of defining critical cube regions that have the most query activity and are prioritized for inclusion in the MQTs recommended by the Advisor. Click **Specify** button on the query type selection screen to display a screen to graphically manipulate an aggregation level for a cube dimension. At least one optimization slice must be specified per cube by pressing the green “plus” button (+). Use the following guidelines when specifying an option for each cube dimension:

- ▶ Specify a specific level in a cube dimension, such as month in the time cube dimension, if you know that the specified level is important or frequently queried.
- ▶ Specify *All* in a cube dimension if the highest aggregation of the cube dimension is important or frequently queried.

- Specify *Any* in a cube dimension if no level is significantly more important than any other level in that cube dimension, many levels in that cube dimension are queried, or you do not know how often each level in that cube dimension is queried.

After the query types and optimization slices are determined, several MQT options must be specified. The decisions (shown in Figure 5-41) need to be made and typically involve a database administrator. They include information such as:

- Whether to create the MQTs as immediate refresh or deferred refresh
- Which tablespace to use for the MQTs
- Which index tablespace to use for the MQTs
- How much disk space the Optimization Advisor can use when determining MQT recommendations
- How much time the Optimization Advisor can take when determining MQT recommendations
- Whether the Optimization Advisor uses data sampling in determining MQT recommendations

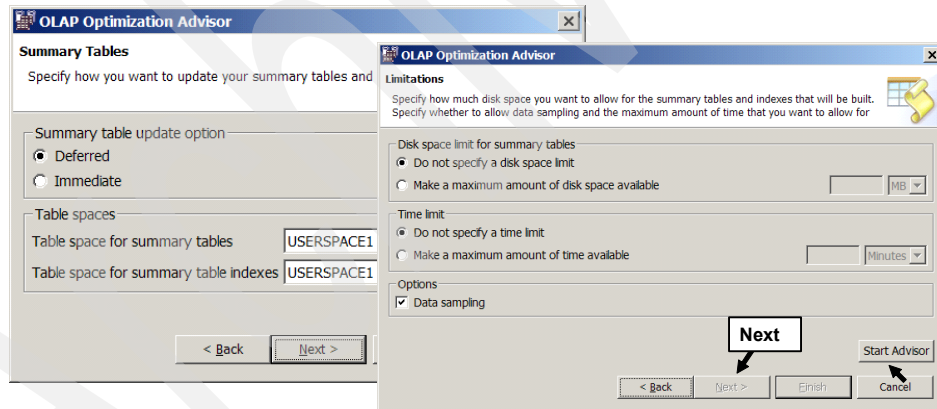


Figure 5-41 Optimization Advisor recommendation options

Click **Start Advisor** to start the optimization process.

After the optimization process is complete, click **Next** and the Optimization Advisor generates two SQL scripts: one to create the MQTs and another to refresh the MQTs, as shown in Figure 5-42. Notice that the Optimization Advisor will provide an estimate of the disk space needed for the MQTs.

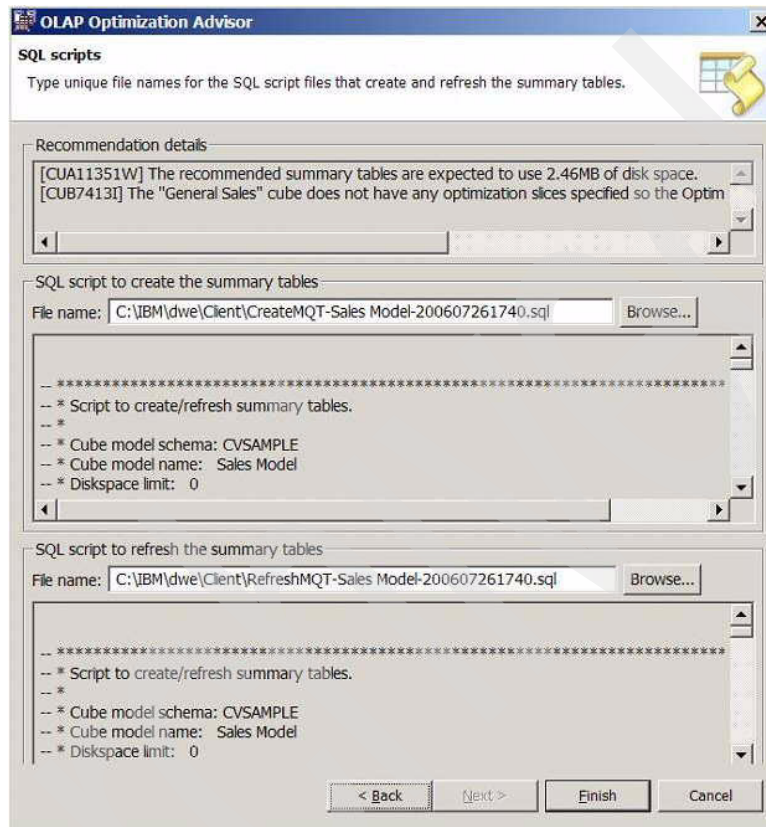


Figure 5-42 Optimization Advisor recommendations

Click the **Browse** button to specify where the SQL scripts will be saved. Click the **Finish** button to complete the process of saving the SQL scripts. Example 5-1 is an example of the SQL script to create the MQTs (summary tables).

Example 5-1 SQL script to create MQTs

```
-- *****
-- * Script to create/refresh summary tables.
-- *
-- * Cube model schema: CVSAMPLE
-- * Cube model name: Sales Model
-- * Diskspace limit: 0
-- * Time limit: 0
-- * Sampling: Yes
-- * Refresh type: Refresh deferred
```

```

-- * Tablespace name:  USERSPACE1
-- * Indexspace name:  USERSPACE1
-- *****

DROP TABLE DB2INFO.MQT0000000001T01;

DROP TABLE DB2INFO.MQT0000000001T02;

DROP TABLE DB2INFO.MQT0000000001T03;

UPDATE COMMAND OPTIONS USING c OFF;

CREATE TABLE DB2INFO.MQT0000000001T01 AS
(SELECT
SUM(T1."SALES") AS "SALES (SALESFACT)",
SUM(T1."COGS") AS "COGS (SALESFACT)",
SUM(T1."ADVERTISING") AS "ADVERTISING (SALESFACT)",
SUM(T1."COGS" + T1."ADVERTISING") AS "Total Expense",
SUM(T1."SALES" - (T1."COGS" + T1."ADVERTISING")) AS "Profit",
T5."LINEID" AS "LINEID (LINE)",
T6."STATE_NAME" AS "STATE_NAME (LOCATION)",
T4."FISCAL_YEAR" AS "FISCAL_YEAR (TIME)",
'Qtr ' CONCAT (cast( T4."FISCAL_QUARTER_NUMBER" as char(1))) AS "FISCAL_QUARTER_NAME",
T4."FISCAL_MONTH_NUMBER" AS "FISCAL_MONTH_NUMBER (TIME)"

FROM
"CVSAMPLE"."SALESFACT" AS T1,
"CVSAMPLE"."STORE" AS T2,
"CVSAMPLE"."PRODUCT" AS T3,
"CVSAMPLE"."TIME" AS T4,
"CVSAMPLE"."LINE" AS T5,
"CVSAMPLE"."LOCATION" AS T6

WHERE
T1."STOREID"=T2."STOREID" AND
T1."PRODUCTID"=T3."PRODUCTID" AND
T1."TIMEID"=T4."TIMEID" AND
T3."LINEID"=T5."LINEID" AND
T2."POSTALCODEID"=T6."POSTALCODEID"

GROUP BY
T5."LINEID",
T6."STATE_NAME",
T4."FISCAL_YEAR",
'Qtr ' CONCAT (cast( T4."FISCAL_QUARTER_NUMBER" as char(1))),
T4."FISCAL_MONTH_NUMBER")

DATA INITIALLY DEFERRED
REFRESH DEFERRED
ENABLE QUERY OPTIMIZATION
MAINTAINED BY SYSTEM
IN "USERSPACE1"
INDEX IN "USERSPACE1"
NOT LOGGED INITIALLY;

COMMENT ON TABLE DB2INFO.MQT0000000001T01 IS 'AST created for cube model CVSAMPLE.Sales Model';

COMMIT;

DECLARE C_MQT0000000001T01 CURSOR FOR
(SELECT
SUM(T1."SALES") AS "SALES (SALESFACT)",
SUM(T1."COGS") AS "COGS (SALESFACT)",
SUM(T1."ADVERTISING") AS "ADVERTISING (SALESFACT)",
SUM(T1."COGS" + T1."ADVERTISING") AS "Total Expense",
SUM(T1."SALES" - (T1."COGS" + T1."ADVERTISING")) AS "Profit",
T5."LINEID" AS "LINEID (LINE)",

```

```

T6."STATE_NAME" AS "STATE_NAME (LOCATION)",
T4."FISCAL_YEAR" AS "FISCAL_YEAR (TIME)",
'Qtr ' CONCAT (cast( T4."FISCAL_QUARTER_NUMBER" as char(1))) AS "FISCAL_QUARTER_NAME",
T4."FISCAL_MONTH_NUMBER" AS "FISCAL_MONTH_NUMBER (TIME)"

FROM
"CVSAMPLE"."SALESFACT" AS T1,
"CVSAMPLE"."STORE" AS T2,
"CVSAMPLE"."PRODUCT" AS T3,
"CVSAMPLE"."TIME" AS T4,
"CVSAMPLE"."LINE" AS T5,
"CVSAMPLE"."LOCATION" AS T6

WHERE
T1."STOREID"=T2."STOREID" AND
T1."PRODUCTID"=T3."PRODUCTID" AND
T1."TIMEID"=T4."TIMEID" AND
T3."LINEID"=T5."LINEID" AND
T2."POSTALCODEID"=T6."POSTALCODEID"

GROUP BY
T5."LINEID",
T6."STATE_NAME",
T4."FISCAL_YEAR",
'Qtr ' CONCAT (cast( T4."FISCAL_QUARTER_NUMBER" as char(1))),
T4."FISCAL_MONTH_NUMBER");

LOAD FROM C_MQT0000000001T01 OF CURSOR REPLACE INTO DB2INFO.MQT0000000001T01;
SET INTEGRITY FOR DB2INFO.MQT0000000001T01 ALL IMMEDIATE UNCHECKED;

CREATE INDEX DB2INFO.IDX0000000001T0101 ON DB2INFO.MQT0000000001T01("LINEID (LINE)");

RUNSTATS ON TABLE DB2INFO.MQT0000000001T01 AND INDEXES ALL;

COMMIT;

CREATE TABLE DB2INFO.MQT0000000001T02 AS
(SELECT
SUM(T1."SALES") AS "SALES (SALESFACT)",
SUM(T1."COGS") AS "COGS (SALESFACT)",
SUM(T1."ADVERTISING") AS "ADVERTISING (SALESFACT)",
SUM(T1."COGS" + T1."ADVERTISING") AS "Total Expense",
SUM(T1."SALES" - (T1."COGS" + T1."ADVERTISING")) AS "Profit",
T3."PRODUCTID" AS "PRODUCTID (PRODUCT)",
T4."STATE_NAME" AS "STATE_NAME (LOCATION)",
T4."CITY_NAME" AS "CITY_NAME (LOCATION)"

FROM
"CVSAMPLE"."SALESFACT" AS T1,
"CVSAMPLE"."STORE" AS T2,
"CVSAMPLE"."PRODUCT" AS T3,
"CVSAMPLE"."LOCATION" AS T4

WHERE
T1."STOREID"=T2."STOREID" AND
T1."PRODUCTID"=T3."PRODUCTID" AND
T2."POSTALCODEID"=T4."POSTALCODEID"

GROUP BY
T3."PRODUCTID",
T4."STATE_NAME",
T4."CITY_NAME")

DATA INITIALLY DEFERRED
REFRESH DEFERRED
ENABLE QUERY OPTIMIZATION
MAINTAINED BY SYSTEM

```



```

IN "USERSPACE1"
INDEX IN "USERSPACE1"
NOT LOGGED INITIALLY;

COMMENT ON TABLE DB2INFO.MQT0000000001T02 IS 'AST created for cube model CVSAMPLE.Sales Model';

COMMIT;

DECLARE C_MQT0000000001T02 CURSOR FOR
(SELECT
SUM(T1."SALES") AS "SALES (SALESFACT)",
SUM(T1."COGS") AS "COGS (SALESFACT)",
SUM(T1."ADVERTISING") AS "ADVERTISING (SALESFACT)",
SUM(T1."COGS" + T1."ADVERTISING") AS "Total Expense",
SUM(T1."SALES" - (T1."COGS" + T1."ADVERTISING")) AS "Profit",
T3."PRODUCTID" AS "PRODUCTID (PRODUCT)",
T4."STATE_NAME" AS "STATE_NAME (LOCATION)",
T4."CITY_NAME" AS "CITY_NAME (LOCATION)"

FROM
"CVSAMPLE"."SALESFACT" AS T1,
"CVSAMPLE"."STORE" AS T2,
"CVSAMPLE"."PRODUCT" AS T3,
"CVSAMPLE"."LOCATION" AS T4

WHERE
T1."STOREID"=T2."STOREID" AND
T1."PRODUCTID"=T3."PRODUCTID" AND
T2."POSTALCODEID"=T4."POSTALCODEID"

GROUP BY
T3."PRODUCTID",
T4."STATE_NAME",
T4."CITY_NAME");

LOAD FROM C_MQT0000000001T02 OF CURSOR REPLACE INTO DB2INFO.MQT0000000001T02;
SET INTEGRITY FOR DB2INFO.MQT0000000001T02 ALL IMMEDIATE UNCHECKED;

CREATE INDEX DB2INFO.IDX0000000001T0201 ON DB2INFO.MQT0000000001T02("PRODUCTID (PRODUCT)");

RUNSTATS ON TABLE DB2INFO.MQT0000000001T02 AND INDEXES ALL;

COMMIT;

CREATE TABLE DB2INFO.MQT0000000001T03 AS
(SELECT
SUM(T1."SALES") AS "SALES (SALESFACT)",
SUM(T1."COGS") AS "COGS (SALESFACT)",
SUM(T1."ADVERTISING") AS "ADVERTISING (SALESFACT)",
SUM(T1."COGS" + T1."ADVERTISING") AS "Total Expense",
SUM(T1."SALES" - (T1."COGS" + T1."ADVERTISING")) AS "Profit",
T5."LINEID" AS "LINEID (LINE)",
T2."STOREID" AS "STOREID (STORE)",
T4."YEAR" AS "YEAR (TIME)",
'Qtr ' CONCAT (cast(T4."QUARTER_NUMBER" as char(1))) AS "QUARTER_NAME",
T4."MONTH_NUMBER" AS "MONTH_NUMBER (TIME)"

FROM
"CVSAMPLE"."SALESFACT" AS T1,
"CVSAMPLE"."STORE" AS T2,
"CVSAMPLE"."PRODUCT" AS T3,
"CVSAMPLE"."TIME" AS T4,
"CVSAMPLE"."LINE" AS T5

WHERE
T1."STOREID"=T2."STOREID" AND
T1."PRODUCTID"=T3."PRODUCTID" AND

```

```

T1."TIMEID"=T4."TIMEID" AND
T3."LINEID"=T5."LINEID"

GROUP BY
T5."LINEID",
T2."STOREID",
T4."YEAR",
'Qtr ' CONCAT (cast(T4."QUARTER_NUMBER" as char(1))),
T4."MONTH_NUMBER")

DATA INITIALLY DEFERRED
REFRESH DEFERRED
ENABLE QUERY OPTIMIZATION
MAINTAINED BY SYSTEM
IN "USERSPACE1"
INDEX IN "USERSPACE1"
NOT LOGGED INITIALLY;

COMMENT ON TABLE DB2INFO.MQT0000000001T03 IS 'AST created for cube model CVSAMPLE.Sales Model';

COMMIT;

DECLARE C_MQT0000000001T03 CURSOR FOR
(SELECT
SUM(T1."SALES") AS "SALES (SALESFACT)",
SUM(T1."COGS") AS "COGS (SALESFACT)",
SUM(T1."ADVERTISING") AS "ADVERTISING (SALESFACT)",
SUM(T1."COGS" + T1."ADVERTISING") AS "Total Expense",
SUM(T1."SALES" - (T1."COGS" + T1."ADVERTISING")) AS "Profit",
T5."LINEID" AS "LINEID (LINE)",
T2."STOREID" AS "STOREID (STORE)",
T4."YEAR" AS "YEAR (TIME)",
'Qtr ' CONCAT (cast(T4."QUARTER_NUMBER" as char(1))) AS "QUARTER_NAME",
T4."MONTH_NUMBER" AS "MONTH_NUMBER (TIME)"

FROM
"CVSAMPLE"."SALESFACT" AS T1,
"CVSAMPLE"."STORE" AS T2,
"CVSAMPLE"."PRODUCT" AS T3,
"CVSAMPLE"."TIME" AS T4,
"CVSAMPLE"."LINE" AS T5

WHERE
T1."STOREID"=T2."STOREID" AND
T1."PRODUCTID"=T3."PRODUCTID" AND
T1."TIMEID"=T4."TIMEID" AND
T3."LINEID"=T5."LINEID"

GROUP BY
T5."LINEID",
T2."STOREID",
T4."YEAR",
'Qtr ' CONCAT (cast(T4."QUARTER_NUMBER" as char(1))),
T4."MONTH_NUMBER");

LOAD FROM C_MQT0000000001T03 OF CURSOR REPLACE INTO DB2INFO.MQT0000000001T03;
SET INTEGRITY FOR DB2INFO.MQT0000000001T03 ALL IMMEDIATE UNCHECKED;

CREATE INDEX DB2INFO.IDX0000000001T0301 ON DB2INFO.MQT0000000001T03("STOREID (STORE)");

RUNSTATS ON TABLE DB2INFO.MQT0000000001T03 AND INDEXES ALL;

COMMIT;

```

Tip: Review the IBM Redbook *DB2 Cube Views: A Primer*, SG24-700202, for additional information about the optimization process and implementing MQTs within DB2 UDB.

5.6 DWE OLAP optimization cycle

The Optimization Advisor in DWE OLAP provides a model-based advisor. The model-based approach is generally performed before you have detailed knowledge about how users will access the data. It allows the DBA or data modeler to easily define a metadata model of the facts and dimensions present in the database that users will query against. This metadata model is examined by the DWE OLAP Optimization Advisor, and one or more MQTs are recommended for the data warehouse.

The optimization cycle is iterative and requires continuous refinement and updates for the MQTs. The DWE Admin Console provides an environment in which the administrator can conveniently invoke the optimization advisor to fine-tune the cube model.

As users query the database, different parts of the OLAP model will get used more than others. Certain measures may be requested more often, different combinations of dimensions may be used more frequently, and different levels of aggregations within a dimension will be requested more than others. To achieve optimal performance, using both model-based and workload-based optimization techniques together is needed, as shown in Figure 5-43.

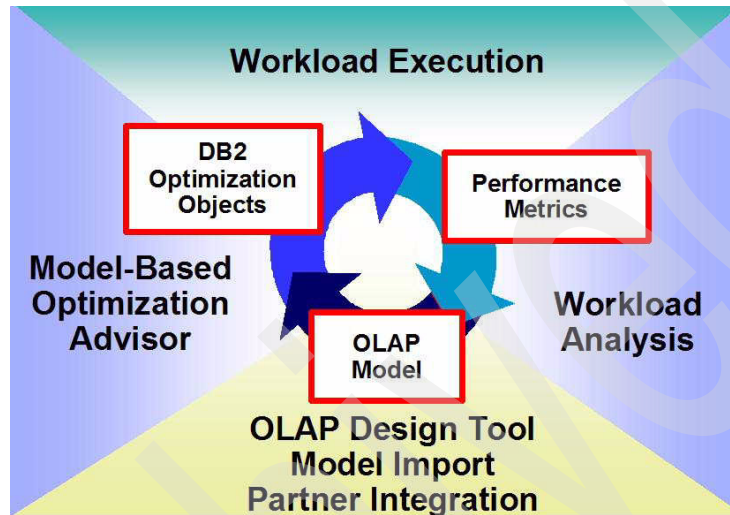


Figure 5-43 Optimization cycle

Once the DWE OLAP recommended MQTs are implemented, the actual query workload should be captured and stored for analysis using DB2 Query Patroller (DB2 QP). DB2 QP is discussed in more detail in Chapter 11, “Managing SQL queries with DWE” on page 511.

DB2 UDB provides the DB2 Design Advisor, which is a comprehensive workload-based advisor. This tool takes a collection of SQL statements and their frequencies, analyzes the common sub-expressions of the SQL, and factors in table statistics, existing indexes, existing MQTs, and Multidimensional Clustering (MDC) indexes in order to generate recommendations for data organization changes that will improve query performance of the given workload. The input workload can be obtained from a file, DB2 Query Patroller, or the statement heap. The output recommendations are new table indexes, Multidimensional Clustering (MDC) indexes, partitioning keys, and MQTs for the database. The DB2 Design Advisor develops costs for various plan alternatives with and without MQTs and develops a set of MQT recommendations. The DB2 Design Advisor may suggest that some existing database objects be dropped because they do not appear to improve the performance of the given workload.

The DWE OLAP Optimization and DB2 Design Advisors complement each other. The index recommendations from the DB2 Design Advisor may be used by the DWE OLAP Optimization Advisor and the DB2 Design Advisor may recommend MQTs for other queries that cannot be handled by the MQTs from the DWE OLAP Optimization Advisor. This process of using both advisors is iterative and should be repeated if there are significant changes in the cube model, the SQL workload, or data volumes.

We recommend the following steps when using both the DWE OLAP Optimization Advisor and DB2 Design Advisor:

1. Run the runstats utility on all tables.
2. Capture the SQL workload on the system with DB2 Query Patroller.
3. Run the DB2 Design Advisor to get initial index recommendations.
4. Review the index recommendations and implement them if there is an expected performance gain.
5. Create the DWE OLAP cube model.
6. Run the DWE OLAP Optimization Advisor to get MQT and index recommendations.
7. Review and implement the MQT and index recommendations.
8. Run the DB2 Design Advisor based on the captured workload to get index and MQT recommendations.
9. Review the index and MQT recommendations from the DB2 Design Advisor, and implement them if there is an expected performance gain.

Repeat steps 6 to 9 above as necessary.

5.7 Metadata exchange

DWE OLAP facilitates the exchange of metadata information between multiple third-party BI applications, ETL tools, modeling tools, and DB2 UDB through metadata bridges, as shown in Figure 5-44. There are many different scenarios on how to build the OLAP models in DWE OLAP using these metadata bridges. To facilitate building the starting cube model, the key is to determine whether information already captured in a BI application tool can be shared with DWE OLAP.

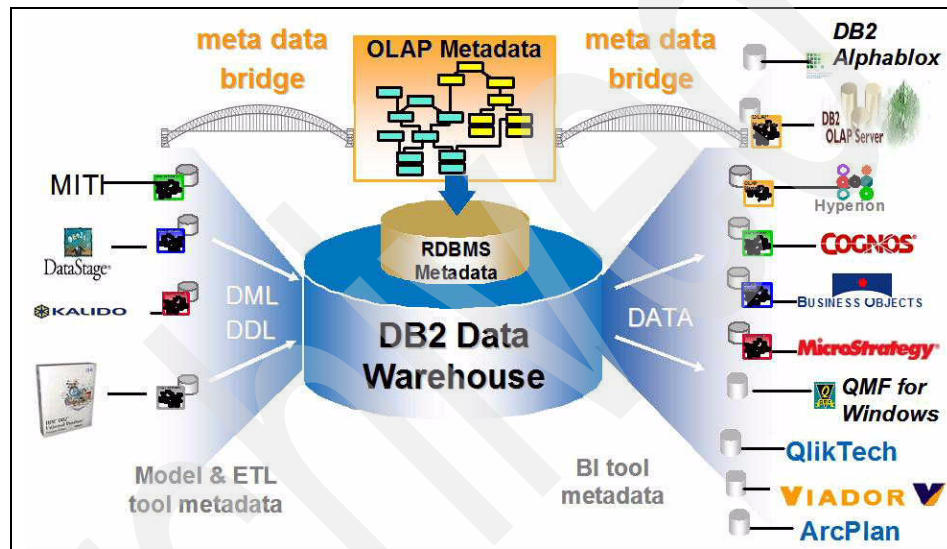


Figure 5-44 Metadata interchange

Metadata interchange between DWE OLAP and the third-party BI analytic tool is either 1-way or 2-way. A 1-way metadata interchange means unidirectional sharing (either export or import) of metadata between DWE and the third-party analytic tools. A 2-way metadata interchange means a bidirectional sharing (both import and export are supported) of metadata between DWE and third-party analytic tools. In some cases, a third-party BI or modeling tool must use an additional model bridge (such as the Meta Integration Model Bridge) to enable the 2-way metadata interchange. The metadata bridges are developed and maintained by the third-party BI analytical tool vendors.

The Import and Export dialog is used to interchange OLAP metadata objects from either a physical data model in the Data Project Explorer or directly from a DB2 UDB database in the Database Explorer. The dialogs are opened by clicking the **File** → **Export** menu option.

5.7.1 Import

OLAP objects can be created in a physical data model using the Import wizard, as shown in Figure 5-45. Since the measures and attributes in the OLAP XML file refer to columns in tables, the user should typically make sure the tables exist in the model before importing the OLAP objects. If the tables do not exist, Design Studio has a feature that will try to create tables in the model. This will at least allow the OLAP measures and attributes to be imported. However, these tables that Design Studio creates are not guaranteed to be identical to the tables the OLAP objects originally referred to in the database from which the OLAP objects were exported.

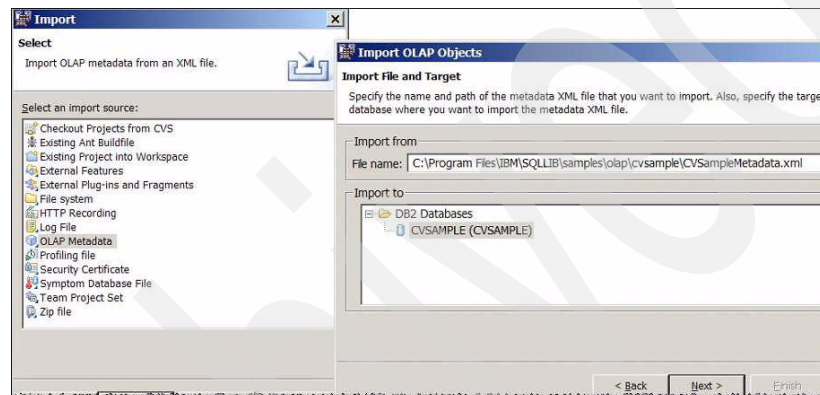


Figure 5-45 Import OLAP metadata

Tip: A sample cube model of the CVSAMPLE database is provided and can be found in the \SQLLIB\samples\olap\cvsample subdirectory in a file called CVSampleMetadata.xml. After creating the DB2 UDB CVSAMPLE database, use the Import dialog to import the CVSampleMetadata.xml file to see a full example of an OLAP model.

5.7.2 Export

OLAP metadata can be exported either from a physical data model in the Data Project Explorer or an existing DB2 UDB database in the Database Explorer, as shown in Figure 5-46. If exporting from a physical data model, make sure the metadata is valid.

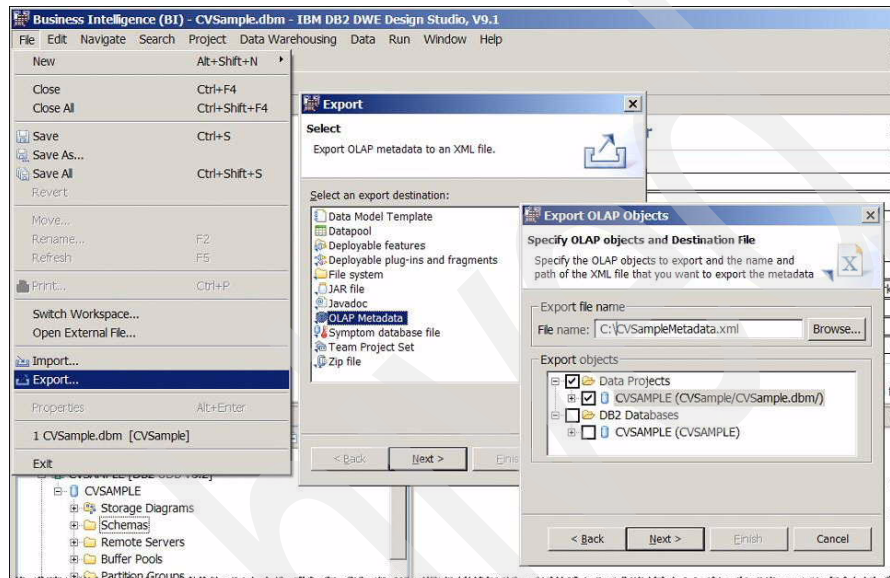


Figure 5-46 Export metadata

The final Export dialog allows the user to choose what version of the OLAP metadata to export, as shown in Figure 5-47. The version of the metadata selected should match the version accepted by any bridge utility the user intends to use to convert the OLAP metadata to some other format.



Figure 5-47 Export metadata versions

Data movement and transformation

In this chapter we discuss the data movement and transformation capabilities of DB2 Data Warehouse Edition (DWE). The component of DWE that provides these services is called the SQL Warehousing Tool (SQW).

You will gain an understanding of the development and runtime architecture of SQW, understand the overall life cycle of developing and executing the data movement and transformation flows, and learn how to develop data flows and sequence those into control flows and how to package your flow for deployment to one or more runtime environments.

6.1 DWE SQL Warehousing Tool (SQW)

In this section we provide an overview and architecture discussion of the SQL Warehousing Tool. It is important to understand, at a high level, the overall purpose and intent of the SQL Warehousing Tool in the context of building the Business Intelligence (BI) Platform. In this section we describe the overall function of SQW, the SQW Application life cycle, the definitions of source, target, and execution databases and show how to get started with a data warehouse project.

6.1.1 SQW overview

One important basic function in building the BI platform is having the ability to manage and move data around within the data warehouse while transforming it for various purposes. DWE provides this functionality for DB2 data warehouses with the component known as the SQL Warehousing Tool (SQW). SQW refers to the functions within the DWE Design Studio for developing data movement and transformation flows and functions within the DWE Runtime environment for the deployment, execution, and management of these flows.

Using SQW functions within the DWE Design Studio, you develop logical flow models, using a graphical flow editor, that represent the actions that you need to apply to the data as it moves from sources to targets. These flows are called *data flows*. The majority of data flows will have sources and targets that represent relational tables within the data warehouse, but these sources and targets can also be flat files and remote DB2 and non-DB2 relational data stores.

The operators in a data flow represent a higher level abstraction model of the actions that need to be applied to the data. From this model, DB2 SQL-based code is generated and organized into an execution plan called a data flow *Execution Plan Graph* (EPG). This generated code is primarily DB2 SQL and, at execution, will be executed by the DB2 database engine.

There are also specialized data flows called mining flows. Mining flows allow you to prepare data for data mining, execute a data mining algorithm creating a mining model, visualize the mining model, and applying the mining model. See Chapter 7, “DWE and data mining” on page 237, for a detailed discussion of DWE Mining.

Typically, you will develop a number of data or mining flows for a particular warehouse application that may have some type of dependencies in the order of execution such that certain flows will need to execute before other flows using some type of processing rules. DWE provides a graphical editor into which you

can define the order of execution of these related data flows and mining flows. These are called *control flows*.

In addition to sequencing data flows and mining flows, a control flow can execute other data processing functions such as operating system scripts, DB2 SQL scripts, batch executables, FTP, sending e-mail notifications, and others. A control flow contains data flow and non-data flow activities such as command execution and file wait, as examples. A control flow also generates a control flow EPG where the SQW runtime engine (DIS) reads and executes the appropriate nodes based on the execution result of the predecessor activity.

DWE SQW also provides a runtime server component that supports the deployment, execution, and management of data movement and transformation flows for a runtime environment such as a test or QA system, or a production system. The user interface for the runtime environment is via a Web browser using the DWE Administration Console, which provides access to the functions required to manage the DWE runtime environment, including the functions necessary to deploy, execute, and manage the SQW data movement and transformation flows.

DWE SQW can also be used as a complementary product to an ETL tool. If the ETL tool is IBM WebSphere DataStage (DataStage), there are specific integration points that allow the integration of a DataStage job into SQW flows. Then SQW functions will be executed by DB2 and DataStage jobs will be executed by the DataStage server. Conversely, SQW data flows can be integrated into DataStage jobs and, again, the SQW functions will be executed by DB2 and DataStage functions will be executed by the DataStage server.

6.1.2 Architecture

DWE SQW has architectural components in both the DWE development environment and the DWE runtime environment.

The development components for SQW are part of the DWE integrated development environment, the DWE Design Studio. See Chapter 3, “The DB2 DWE Design Studio” on page 29, for an overview of the DWE Design Studio.

As depicted in Figure 6-1, the elements of the Design Studio for SQW consist of a data flow editor and a control flow editor. These are graphical flow editors consisting of icons that represent various types of operations, connected by flow arrows that represent the flow of data or control between the operators. Flows are really models of the data movement and transformation processing. The information, or properties, that define these flow are stored in the underlying metadata structure for the DWE Design Studio, using the Eclipse Modeling Framework.

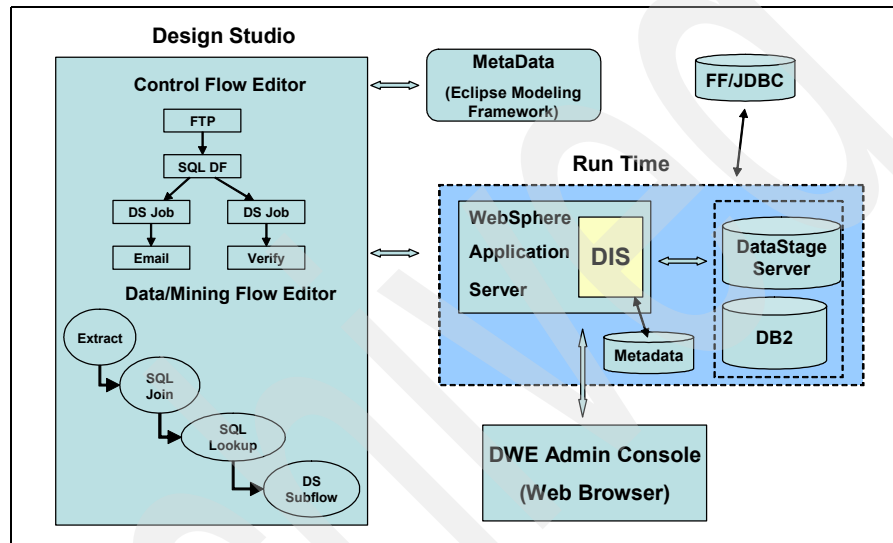


Figure 6-1 SQW architecture

The runtime environment is essentially a set of J2EE applications that execute within a WebSphere Application Server environment and the appropriate DB2 and DataStage servers. This is referred to as the Data Integration Service (DIS), which supports the capabilities to:

- ▶ Communicate with the user via the Web-based DWE Administration Console and execute administration requests made by the administrator.
- ▶ Monitor the metadata for scheduled process execution based on scheduled time or completion of an activity, and to submit the job to the appropriate location, a DB2 server or a DataStage server.
- ▶ Monitor executing jobs and log appropriate execution and completion metadata.

The metadata to support SQW in the runtime environment is stored in a DB2 relational database. This metadata consists of the information for the data flows,

mining flows, control flows, data connectivity information, FTP and DataStage server information, schedule information, runtime statistics, and so on.

Finally, there are the transformation servers. In the case of SQW jobs, this would be the DB2 database server that is designated as the execution database. For DataStage jobs that are integrated into SQW jobs, the DataStage server would be part of the architecture.

6.1.3 SQW warehouse application life cycle

SQW uses a *design once, deploy multiple* type of development paradigm. This means that you develop a set of data movement and transformation flows once within the DWE Design Studio and create a deployment package. This deployment package can then be deployed to multiple runtime environments without going back into the development environment. The deployment package, depicted in Figure 6-2, “Develop once, deploy multiple” on page 141, is developed once in the DWE Design Studio. It is an iterative process, and when finished a deployment package is created. It is first deployed to a system test or QA environment for testing and, if successfully tested, the same deployment package is subsequently deployed to production without having to make any modifications using the DWE Design Studio.

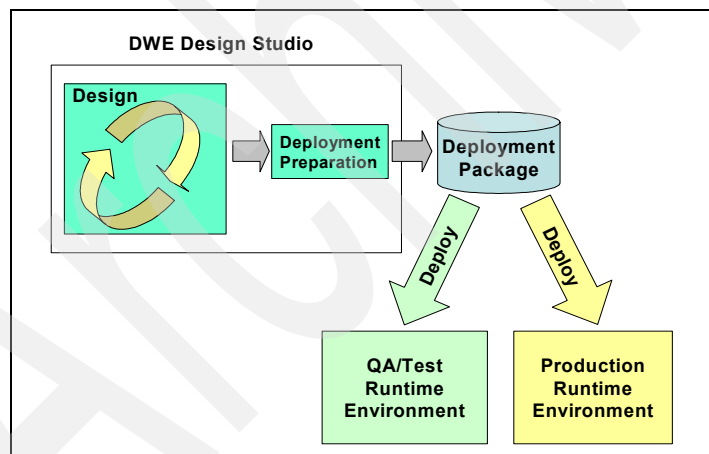


Figure 6-2 Develop once, deploy multiple

SQW development

All SQW development is performed using the DWE Design Studio in a data warehouse project. The primary components that you will develop in a data warehouse project are data flows, mining flows, and control flows.

Before developing data flows, information about the names and structure of the data sources and targets is needed as well as the data transformation specifications. The data source and target information is contained in the physical data models that are created in a data project. See Chapter 4, “Developing the physical data model” on page 57, for more information. By simply providing a reference link to one or more data models within the data warehouse project, the data model metadata is made available to the data flows. Data project references are added when a data warehouse project is created by checking the required data projects in the wizard or can be added at a later time by right-clicking the data warehouse project name and selecting project references from the menu.

Once there is access to the metadata, the data flows can be designed, developed, and validated. When successfully validated, the data flow can be test-run from within the DWE Design Studio. This process is depicted in Figure 6-3.

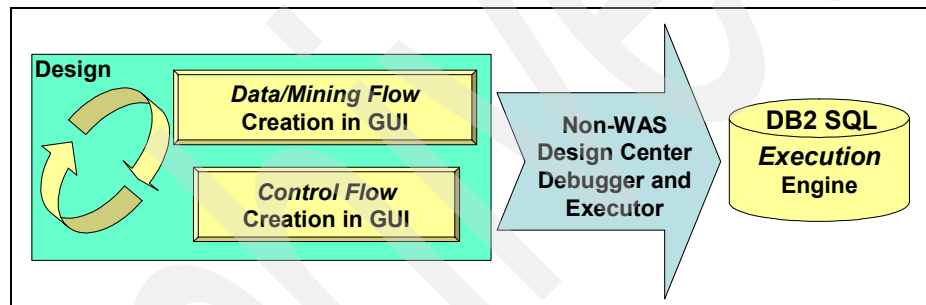


Figure 6-3 Iterative development and testing

When one or more data flows have been developed, they can be sequenced into one or more control flows for defining the processing rules. In a control flow, other data processing activities can be added for activities such as executing OS and SQL scripts, FTP, and e-mail notifications. Control flows can also be tested within the DWE Design Studio. The DWE Design Studio provides a debugger for both data flows and control flows that allows you to step through the operators in a data flow or control flow.

When a data flow or control flow is tested, the generated SQL is submitted by the DWE Design Studio directly to the DB2 execution database and does not require access to a DWE runtime server. This is because the DWE Design Studio has a subset of the runtime functionality to allow the submission of SQL flow activities to a DB2 execution database. However, this type of execution should be limited to unit type testing.

As with most development projects, development of data flows and control flows is an iterative process of developing, testing, modifying, testing, and so on until you are satisfied that the set of data flows and controls flows are ready to deploy to a runtime environment.

SQW deployment preparation

Once satisfied that the set of data flows and controls are development and unit tested sufficiently, they can be deployed to a runtime environment for system testing and subsequently to production. The last step of the development effort is to package a related set of control flows into a warehouse application and create a deployment package. This process is called *deployment preparation* and is accomplished within the DWE Design Studio using the *Data Warehouse Application Deployment Preparation* wizard.

The deployment preparation wizard helps you to:

1. Create an application profile for the warehouse application, which refers to the set of control flows that comprise this application and other deployment configuration information.
2. Generate the code based on the flow models included in the selected set of control flows.
3. Create the deployment package into a zip file that contains all of the necessary information to deploy this warehouse application into a runtime environment.

This deployment preparation is depicted in Figure 6-4.

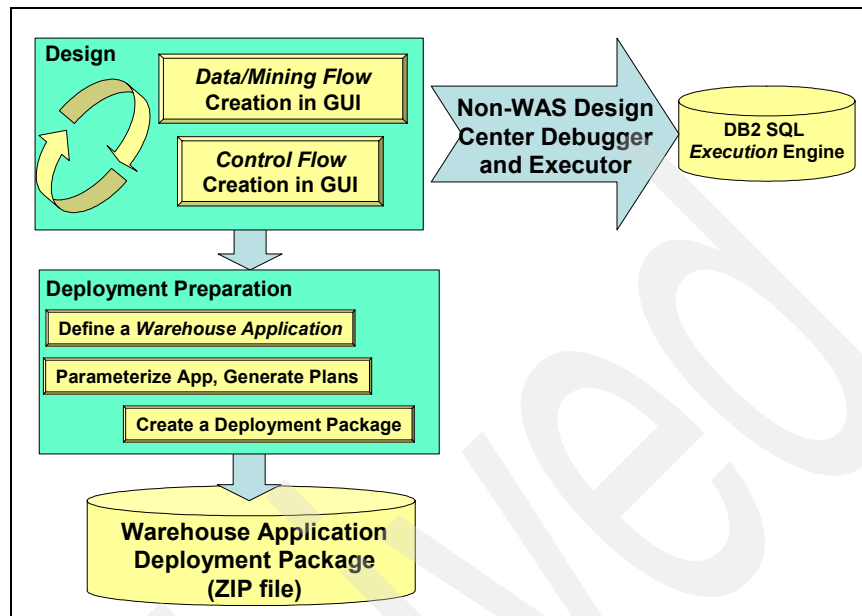


Figure 6-4 SQW Warehouse Application deployment preparation

The resultant zip file will be used by the administrators of the runtime environments to deploy the application.

SQW deployment

As depicted in Figure 6-2 on page 141, the same deployment package can be deployed to multiple runtime environments. A typical situation would be to first deploy to some type of system test runtime environment where the flows are tested in a simulated production environment before being deployed to a production runtime environment.

Deployment to a particular runtime environment is done by the administrator of the runtime environment using the Web-based DWE Administration Console. The administrator needs access to the deployment zip file either on the DWE Application Server or on the local system.

The deployment process consists of using the DWE Administration Console to:

1. Configure the runtime environment and create the appropriate database connectivity definition.
2. Install the warehouse application into the runtime environment using the deployment zip file.

3. Develop process schedules for the execution of the warehouse application processes.

In the remainder of this chapter we focus on the development and deployment preparation functions within the DWE Design Studio. For additional information about this topic see Chapter 9, “Deploying and managing DWE solutions” on page 365.

6.1.4 Source, target, and execution databases

Before going further it is important to understand the definition of the term *execution database* and its relationship to source and target databases. The *execution database* is the database that does the transformation work when the SQL code in a data warehouse application runs. This database must be a DB2 UDB database.

The execution database alias is a primary property of a data flow and all SQL code for that data flow will be run at the execution database. Different data flows can have different execution databases and the data flow SQL code will be submitted to the execution database defined in the data flow properties. An example would be if you need to move data from a warehouse database to a distributed data mart. Part of the processing may happen at the data warehouse database and part of the processing may happen at the distributed data mart database.

The notion of a remote or local database source or target is relative to the execution database where the SQL code is submitted. If the source or target database is the same database as the execution database, then they are local databases. Only DB2 databases can be local databases. When the target or source database is different from the execution database, then they are remote databases. A remote database is accessed by the Java Database Connectivity interface standard (JDBC) and may be a non-DB2 relational database.

In Figure 6-5 on page 146 we show four possible configurations of source schema, execution database, and target schema that can be supported with SQW. The database symbol with the dashed lines represents the DB2 execution database, the database symbols with the solid lines represent source or target schemas, and the shaded symbol represents the local database. The performance characteristics of these various configurations can be significantly different, as remote databases will incur some network overhead:

1. The first scenario occurs when the source schema and the target schema are local to the execution database. In this case all of the data is in one database and DB2 can process the table-to-table data movement and transformations effectively without the data flowing out of the database.

2. The second scenario occurs when the source data is not local to the execution database. This could be a database on a remote server, another database on the same server, a non-DB2 database, or a flat file. In this case the needed source data has to first be staged to the execution database before it can be processed.
3. The third scenario occurs when the target is not local to the execution database. This could be a database on a remote system, another database on the same server, or a non-DB2 database. In this case the data has to flow out of the execution database to the external target using remote inserts.
4. The fourth scenario occurs when both the source data and the eventual target are not local to the DB2 database. The source data has to be staged to the execution database for processing and then sent to the remote target for updating.

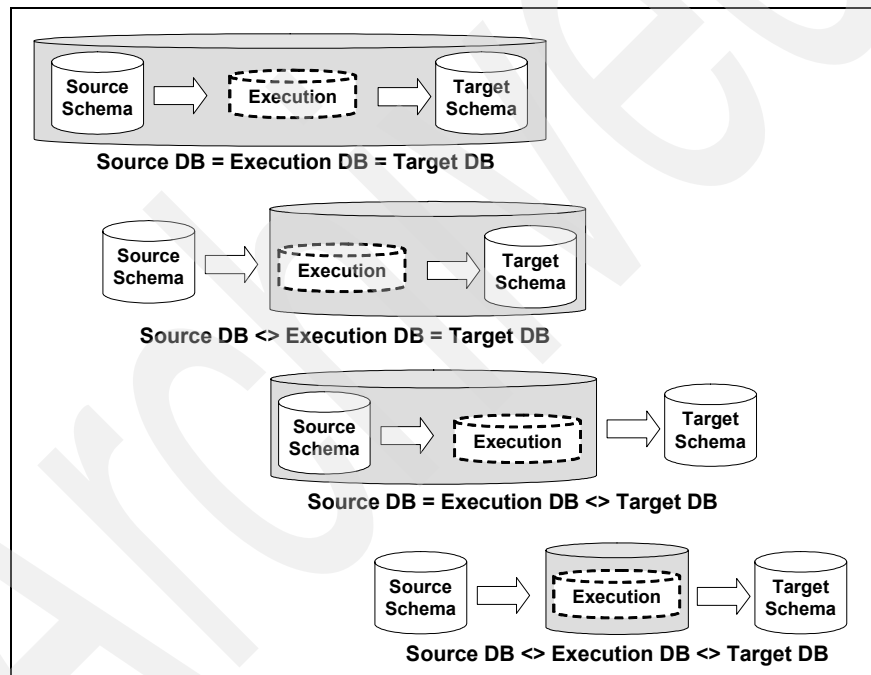


Figure 6-5 Configurations of source, execution, and target databases

While all of these scenarios are supported by SQW, the most effective performance will be gained by co-locating the target and execution database as shown in the first two scenarios.

6.1.5 Setting up a data warehouse project

The development of data flows, mining flows, and control flows takes place within the DWE Design Studio and is organized into data warehouse projects, which are containers for the metadata artifacts that represent the flows. In addition to data flows, mining flows, and control flows, there are also folders in the data warehouse project for other artifacts that support the development effort such as subflows, application profiles, and resource profiles. Figure 6-6 shows a data warehouse project folder structure. The data warehouse project folder is accessed from the *Data Project Explorer* tab of the DWE Design Studio.

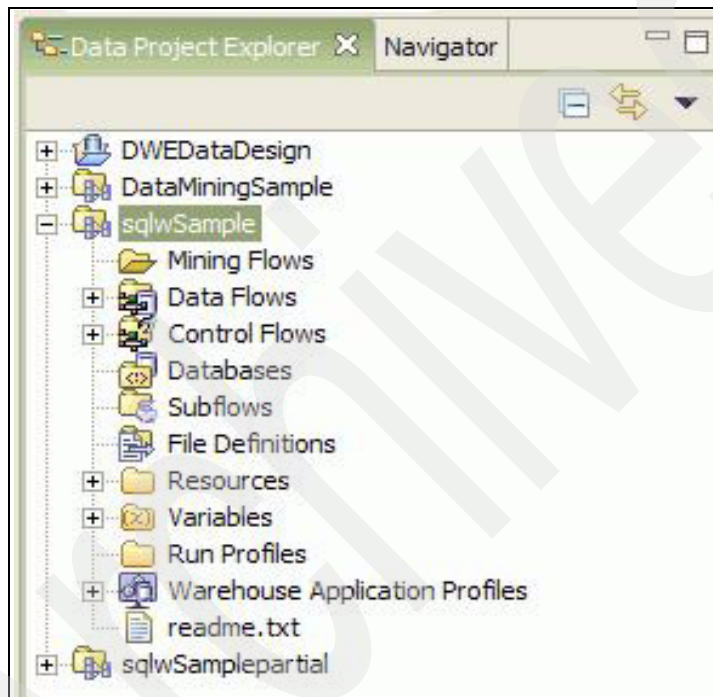


Figure 6-6 Data warehouse project, sqlwSample

Many times data warehouse projects depend on metadata representing the data object in source and target databases. This metadata is the physical data model and is contained in another type of project called a *data project*. See Chapter 4, “Developing the physical data model” on page 57, for more information about developing the physical data model. The metadata contained in a physical data model can simply be made available to a data warehouse project by creating a project reference link to the physical data model. In fact, the data warehouse project could need reference links to multiple physical data models.

The DWE Design Studio supports the ability to link to multiple physical data models from a single data warehouse project. A single physical data model may have references from multiple data warehouse projects.

Important: If a linked physical data model (.dbm file) is deleted from a data warehouse project, the original model file is deleted, not just the link to that model. Instead, use the remove option to remove the reference.

Data flows and other objects can be imported into a data warehouse project by selecting **Import** from the File menu, then **File system**. Be sure to identify the correct target folder type for the object being imported. For example, make sure to import data flows into the data flows folder in the data warehouse project.

Note: The actual development of data flows and control flows does not require a live connection to a database, but a live connection to a relational database is required to test a data flow or control flow. You may also want a live connection in the Database Explorer to view or manipulate physical objects in the database or to sample data contents.

6.2 Developing data flows

In this section we discuss in more detail the process of developing SQW data flows using the DWE Design Studio. We discuss the data flow operators, flow validation, code generation, testing, and debugging. For more information about this topic refer to the DB2 and DWE Information Center or the DB2 Data Warehouse Edition V9.1 Tutorial (a component of the DWE documentation). The tutorial also contains hands-on exercises in developing SQW flows.

6.2.1 Defining a data flow

Data flows model the SQL-based data movement and transformation activities that execute in the DB2 database. A data flow consists of activities that extract data from flat files or relational tables, transform the data, and load it into a relational table in a data warehouse, data mart, or staging area. Data can also be exported to flat files.

The DWE Design Studio provides a graphical editor with an intuitive way to visualize and design data flows. Graphical operators model the various steps of a data flow activity. By arranging these source, transform, and target operators in a canvas work area, connecting them, and defining their properties, models can be created that meet the business requirements. After creating data flows, you generate their SQL code, which creates the specific SQL operations that are

performed by the execution database when you run a data warehouse application. Figure 6-7 shows a simple data flow that selects data from a DB2 staging table, removes duplicates, sorts the rows, and inserts the results into another DB2 table. Discarded duplicates are put into a flat file.

A data flow consists of operators, ports, and connectors. Figure 6-7 shows a simple data flow that contains six operators. There are three types of operators: source, target, and transform operators. Source operators represent data that is being consumed by the data flow, target operators represent where and how data is being placed after it is transformed, and transform operators cause some type of change in the data. In Figure 6-7 there are three table source operators, one table target operator, and two transform operators. Operators have properties that define the specific behavior of that operator.

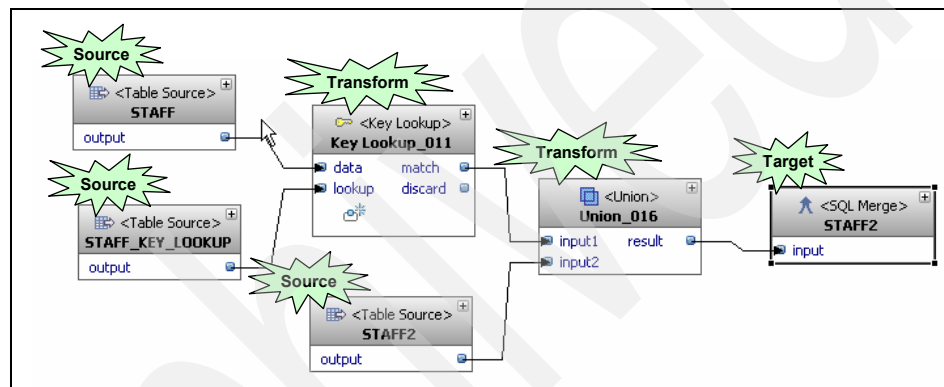


Figure 6-7 A simple data flow showing types of operators

Operators have ports that define the point of data input or output. The ports of the simple data flow in Figure 6-7 are circled in Figure 6-8 on page 150. Source operators only have output ports, target operators only have input ports, and transform operators have both input and output ports. Some operators may have multiple input or output ports and some may have a variable number of ports indicated by a small icon under the last port, as seen in the input ports section of the key lookup operator in Figure 6-8 on page 150 (pointed to by the arrow). Clicking this icon adds an additional port.

Ports also have properties. The primary property of a port is the data layout definition, or virtual table, for the data flowing through that port.

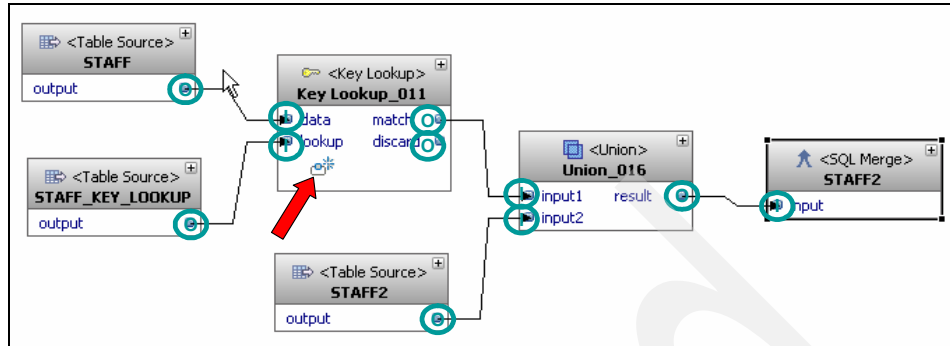


Figure 6-8 A simple data flow showing the ports

Operator ports are connected via connectors that direct the flow of data from an output port to an input port, as seen in Figure 6-9. One output port may have multiple connectors feeding multiple input ports. Connectors also define the column, or field, mapping between the output port and input port.

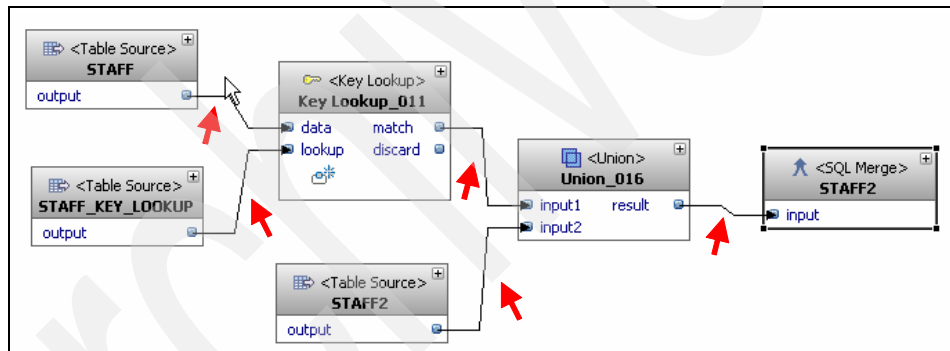


Figure 6-9 A simple data flow showing connectors

6.2.2 Data flow editor

In the DWE Design Studio, data flows are developed using a number of common DWE Design Studio functions, such as the Data Project Explorer; views such as Properties, Data Output, and Problems; and, optionally, the Data Explorer. However, the actual creation and editing of data flows occurs in a specific graphical editor called the Data Flow Editor, depicted in Figure 6-10.

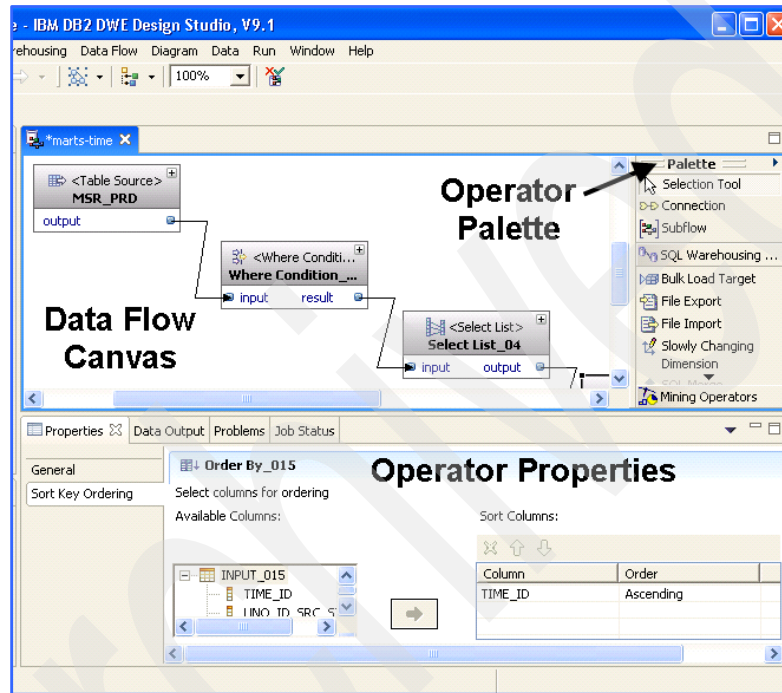


Figure 6-10 Data flow editor and properties

As with most graphical editors in the Eclipse environment, the data flow editor follows the drag, drop, connect, and set properties paradigm of development. When you create a new data flow or open an existing data flow, the data flow editor opens in a new tab in the editor area of the DWE Design Studio. The data flow editor consists of a canvas onto which graphical elements are drawn and an operator palette that contains all of the graphical elements that are relevant to building a data flow. In addition, you will use the common properties view tab to define the characteristics of the objects on the canvas. As you validate and test run the data flows, you will also make use of the Problems view tab and the Data Output tab.

Here are a few things to consider when developing data flows:

- ▶ Be familiar with the common functions in the DWE Design Studio, such as:
 - Working with perspectives
 - Working with views
 - Using the Data Project Explore
 - Using the Database Explorer
 - Dragging and dropping from the Data Project Explore and palette to the canvas
- ▶ Orient the data flow from left to right. This makes a neater diagram because the output ports are on the right side of an operator and the input ports are on the left side.
- ▶ Work with just a few operators at a time.
- ▶ Operators have properties that must be defined.
- ▶ Operator input and output ports have properties that may have to be manually defined or modified. The properties of these ports are the schemas (column definitions) of the data that flows between the operators. These schemas may have to be managed as the flow progresses.
- ▶ A data flow itself has properties with the execution database. Where this data flow is to execute is the most important.

6.2.3 Data flow operators

Data flow operators represent the source, transform, and target steps in a data flow. Operators, graphical objects that you can select from a palette and drop in the canvas work area, form the nodes in a data flow diagram. Each type of operator has a specific set of input and output data ports, one or more lists of data columns, and a number of properties that define exactly how each operator moves or transforms data.

There are four categories of data flow operators:

- ▶ Source operators that causes data to be brought into the data flow from a persistent source, typically a relational table
- ▶ Target operators that cause data processed in the data flow to be persistent, typically in a relational table
- ▶ Transform operators that perform some type of action on the data as it moves from source to target

- Data mining operators that are used to apply data mining models in some form integrated within a data flow to do batch scoring or predicting against data that is already in or coming into the data warehouse

Operators can be expanded or collapsed to show more or less detail, as seen in Figure 6-11. The first two operators have been expanded and you can now see the data layout properties for each port and how the individual columns of the virtual tables map between the output port and the input port.

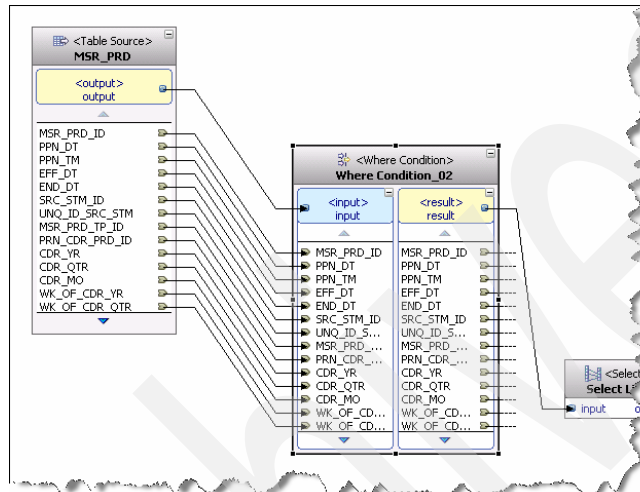
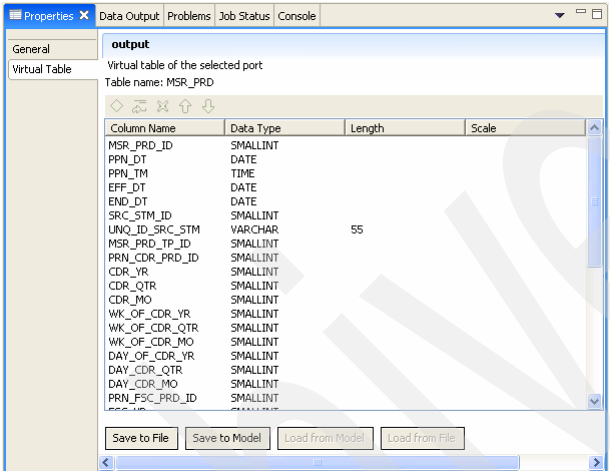


Figure 6-11 Expanded data flow operators

Properties and ports

Figure 6-8 on page 150 shows data flow operators that have ports that represent the data flowing into or out of a data flow operator, and Figure 6-11 shows an expansion of an operator to reveal the columns of the port. The port represents a virtual table that has a property page that defines the virtual table. Figure 6-12 is the properties page of the output port of the MSR_PRD table source operator in Figure 6-11 on page 153. The Virtual Table property tab defines the data layout of that port. Note that the data types are based on DB2 data types.



| Column Name | Data Type | Length | Scale |
|----------------|-----------|--------|-------|
| MSR_PRD_ID | SMALLINT | | |
| PRN_DT | DATE | | |
| PRN_TM | TIME | | |
| EFF_DT | DATE | | |
| END_DT | DATE | | |
| SRC_STM_ID | SMALLINT | | |
| UNQ_ID_SRC_STM | VARCHAR | 55 | |
| MSR_PRD_TP_ID | SMALLINT | | |
| PRN_CDR_PRD_ID | SMALLINT | | |
| CDR_YR | SMALLINT | | |
| CDR_QTR | SMALLINT | | |
| CDR_MO | SMALLINT | | |
| WK_OF_CDR_YR | SMALLINT | | |
| WK_OF_CDR_QTR | SMALLINT | | |
| WK_OF_CDR_MO | SMALLINT | | |
| DAY_OF_CDR_YR | SMALLINT | | |
| DAY_CDR_QTR | SMALLINT | | |
| DAY_CDR_MO | SMALLINT | | |
| PRN_FSC_PRD_ID | SMALLINT | | |

Figure 6-12 Properties of a port: virtual table

In most cases, when connecting an output port to an input port, the virtual table properties are propagated from the output port to the input port, resulting in both ports having the same virtual table layout. However, if the input port already has the virtual table properties defined, then a dialog will prompt you to decide how to map the data columns from the output port to the import port, as shown in Figure 6-13 on page 155. You may also remap the column connections by right-clicking the connection and selecting **Edit**. Or you can expand the operators and manually map individual columns.

Tip: The Propagate Columns option is useful to force the virtual table layout from the output port to the input port even if the input port already has a virtual table layout defined. This is handy when you have mapped the wrong output port to the input port and you delete the connection.

There are two common reasons that an input port may have the data layout properties already defined. First is that the operator with the input port is a target operator and the data layout properties are populated from the metadata of the

target operator. In this case you will want to map by name, position, or manually. Second is that, for non-target operators, the ports had been connected but the connector was deleted, which will leave the input port data layout properties defined. In this case, you would likely just force the propagation of the data layout properties from the output port to the input port.

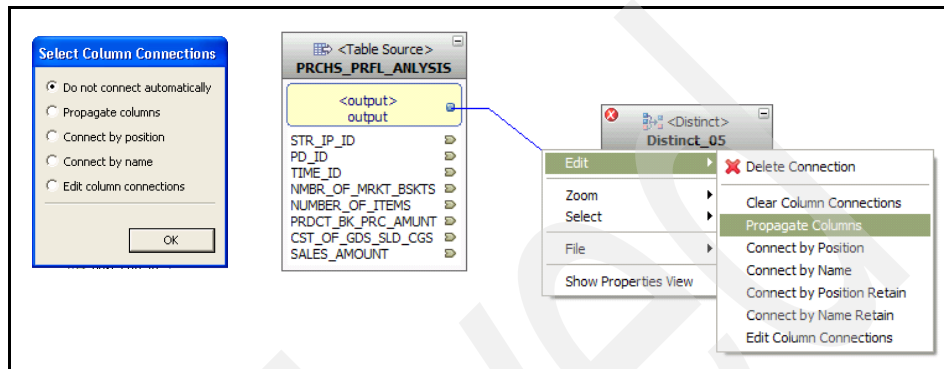


Figure 6-13 Selecting or modifying virtual table column connections

Source operators

Source operators represent some type of data that is input to your data flow. There are three types of source operators:

- ▶ Relational table
- ▶ File import
- ▶ SQL replication

These source operators are depicted in Figure 6-14.

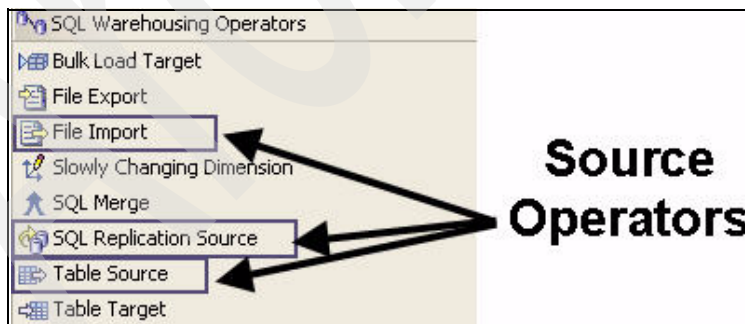


Figure 6-14 SQW source operators the operator palette

Table source operator

A table source operator represents a connection to a relational table in a database management system and corresponds closely to the SELECT clause of an SQL statement, as depicted in Figure 6-15 on page 156. Table source operators can be local or remote relative to the execution database of a data flow as defined in 6.1.4, “Source, target, and execution databases” on page 145.

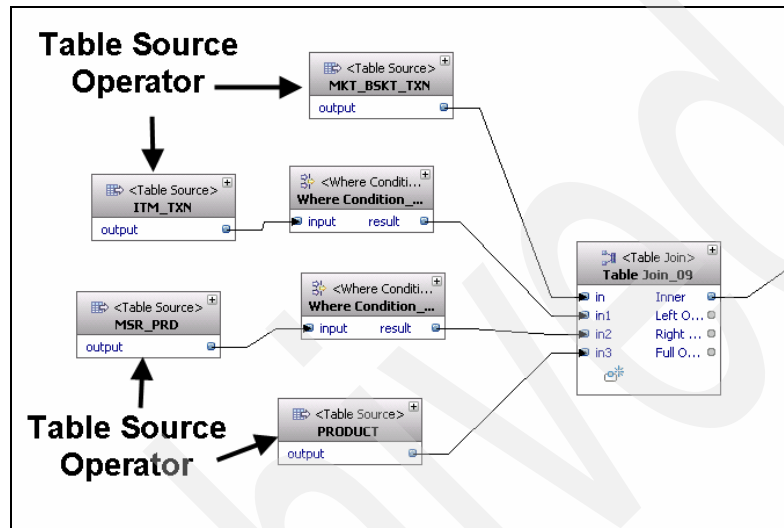


Figure 6-15 Relational table source operators

A table source operator provides column information, including column names and data types, to other operators in the data flow and is typically populated from the physical data model that is referenced by a data warehouse project. A table source can be added to the canvas by selecting the target source operator in the palette and providing the table information or by selecting a table definition in the physical data model within the database folder of the data warehouse project then, when prompted, selecting the source property.

Figure 6-16 on page 157 shows the two property pages for a table source. The General tab contains the source database schema and table name, which can be populated by selecting the ellipsis (...) next to the Source database table field, which will present the list of referenced physical data models and the tables they contain. The table name and schema name can also be variables. This is particularly helpful in the case that schema name varies between the runtime environments to which this data flow will be deployed. A variable can be used for the schema name and changed at deployment time for each runtime environment. For example, use the schema QA for the tables in the test environment and schema PROD for the table in the production environment. Using a variable will allow the schema to be set at deployment time, for example,

to the appropriate value for that environment without having to return to the DWE Design Studio. See 6.4, “Variables in data flows and control flows” on page 222, for more information.

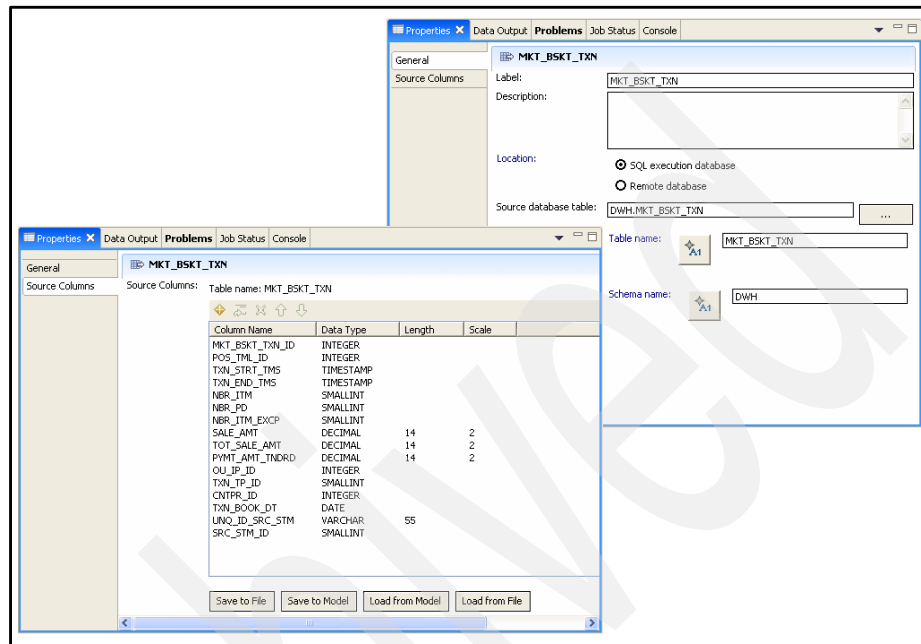


Figure 6-16 Table source properties

In the DWE Design Studio connections to databases, both local and remote are made via the Database Explorer. Before a dataflow is tested all of the required databases must have active connections in the Database Explorer.

In the runtime environment, remote connections are managed by the DWE Application Server component for extracting data from remote tables and the data will flow through the DWE Application Server as a gateway. Data from local data sources will stay local to the execution database and will not flow through the DWE Application Server.

A table source can be a federated nickname, in which case it will be treated as a local table source for the purposes of the data flow and the data will flow directly from the remote database to the local database. Federated data source connectivity is configured at the database server not at the DWE Application Server. DB2 and Informix remote sources can be defined using the built-in federation capability of DB2 while other sources will require the purchase of the appropriate WebSphere Information Integrator software and configuration.

File import operator

A file import operator makes data in a file available to a data flow by presenting it to a downstream operator, such as a virtual table via the output port. It may be connected directly to input ports of transform or target operators as in Figure 6-17. Supported file types are ASCII delimited, ASCII fixed format, and PC/IXF.

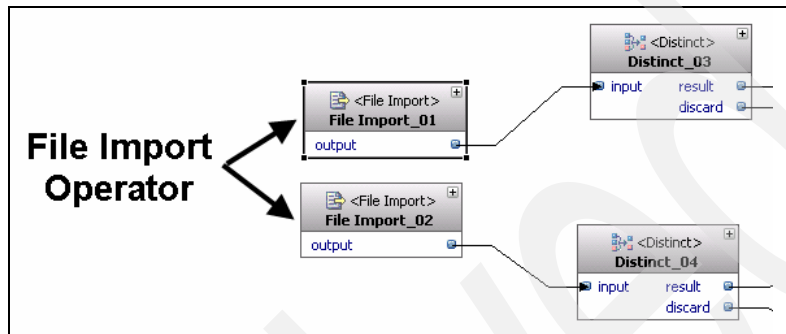


Figure 6-17 File import operators

File import properties are different depending on the file type, including:

- ▶ General property tab: fully qualified file path and name, which may be a variable.
- ▶ File format tab: specifies specifics of the file, which will vary depending on the file type selected including:
 - Delimited format
 - Field and string delimiters
 - Format for date, time, and time stamp fields
 - Field list, including field name and data type
 - IBM-defined code page ID
 - Fixed-length format
 - Format for date, time, and time stamp fields
 - Null indicator character
 - Field list including field name, start and end position, and null indicator position
 - IBM-defined code page ID
 - PC/IXF format
 - Field list including field name and data type

- For all formats, an advanced field to contain additional file type modifiers in the style of the DB2 Load utility
- For all formats, the file layout can be save to and loaded from a file format file
- ▶ Column select tab: allows you to select the columns from the file format to flow through the output port
- ▶ Advanced options tab:
 - Savecount: creates a consistency point after the specified number of records
 - Rowcount: specifies the maximum number of rows to load
 - Warning count: stops the load after the specified number of warnings
 - Messages file and tempfiles path

SQL replication source

SQL replication is the process of taking committed changes to tables in a source database, storing them in a staging table, and replicating the changes to a target table. This is controlled by a set of control tables, a program at the source database called capture, and a program at the target database called apply. Data can be captured and staged once for delivery to multiple targets and in different formats. A subset of the data can be captured in a table and the data can be manipulated at the source or at the target.

The job of the capture program is to run at the source identifying and recording changes in the source tables as they happen by examining the DB2 logs. Once changes are committed, the apply program will save the committed changes to the staging tables. The job of the apply program is to retrieve the changes from the staging tables and update the target tables. Both programs use a set of control tables to manage their tasks.

DWE supports DB2 LUW replication sources and targets, as depicted in Figure 6-18, while the full IBM replication solution supports many other types of sources and targets.

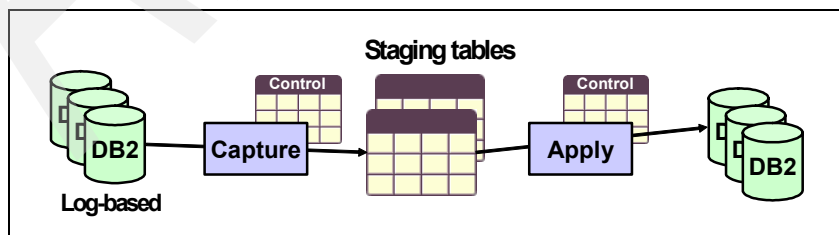


Figure 6-18 The DWE replication architecture

There are two types of target tables supported in SQW, user copy and point-in-time. The user copy replication target is the most common for basic data replication. The structure of a user copy table can be the same as the source table or a subset of the source columns, and you may have before-images or after-images and calculated columns. A user copy also requires that each row be uniquely identified via a unique index or primary key. A point-in-time table is similar to a user table, but a time stamp column is added to indicate when the source change was committed.

To use the SQL replication source operator:

1. Create replication control tables using the DB2 Replication Center.
2. Add the SQL replication operator to the data flow canvas.
3. Select the source connection and source schema and table plus the target schema and table. Note that the replication target database is the execution database.
4. Define the registration settings indicating the table you want to register, the columns you want to capture, and other registration properties and options that determine how registered data from this source will be handled and stored.
5. Define the subscription set that establishes the relationship between source tables and target tables. Specify which target table subscribes to the source data and then define how the replicated data should appear at the target.

The output port of a SQL replication operator can be used to feed the replicated data to downstream operators.

When code is generated for the SQL replication operator, there are three parts. The first part is executed at deployment time and will update the replication control tables with the information to define this replication scenario. There is also a corresponding set of code executed at undeployment that will remove the information from the control tables. Then the execution code will essentially cause an execution of the apply program to apply the defined replication. When doing a test run in the DWE Design Studio, the deployment code, execution code, and undeployment code are all executed.

Target operators

Target operators represent some type of persistent store into which the data flow will update, or update data, and identifies the techniques with which they will be updated. There are three categories of target operators:

- ▶ File export target
- ▶ Generic relational table target
- ▶ DB2 specific target operators for bulk loading, performing an SQL merge (also known as an upsert) and for handling slowly changing dimensions

These target operators are depicted in Figure 6-19.

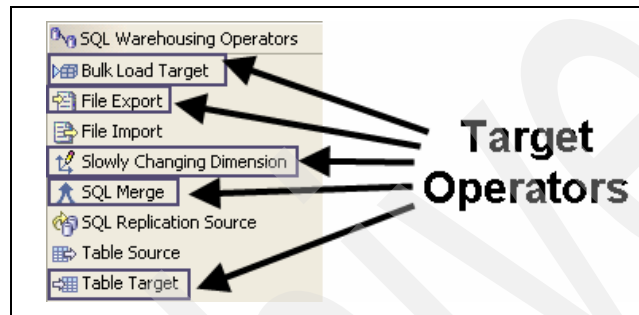


Figure 6-19 SQW target operators in the operator palette

File export operator

The file export operator will export data to delimited flat files. In addition to the obvious use of a file export operator to export data from a relational table, they might also be used to receive discarded rows from other operators such as the distinct or key lookup operator, as shown in Figure 6-20.

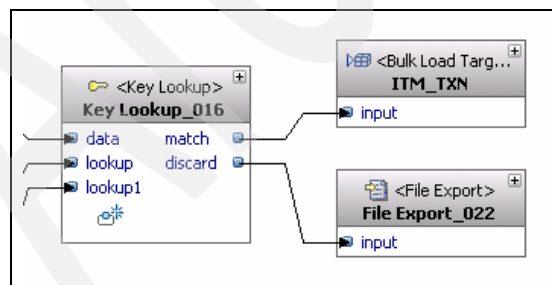


Figure 6-20 File export operator to hold key lookup discards

File export properties include the file location and name, its code page, and its delimiter characters. If desired, you can use a variable to define the file. The file

need not exist before the data flow is run, and the file fields, or columns, are determined automatically by the output schema of the preceding operator.

Tip: File export operators can also be useful while debugging a data flow when you may want to temporarily persist the data flowing through an output port. Taking advantage of the capability to connect one output port to multiple input port allows the addition of a file export operator to any output port even if there is already an existing connection. See Figure 6-21 for an example.

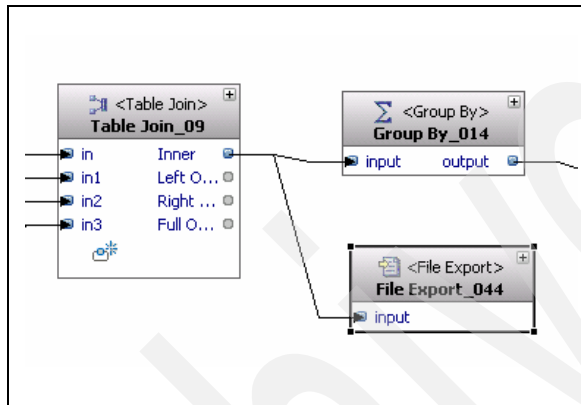


Figure 6-21 Using a file export operator to help debug

Table target operator

The table target operator emulates standard INSERT, UPDATE, and DELETE operations against a relational table. Table target operators can be local or remote relative to the execution database of a data flow as defined in 6.1.4, “Source, target, and execution databases” on page 145.

A table target can be added to the canvas by selecting the table target operator in the palette and then setting the table information, or by selecting a table definition in the physical data model within the database folder of the data warehouse project and then, when prompted, selecting the target property.

In the DWE Design Studio connections to databases, both local and remote, are made via the Database Explorer. Before a dataflow is tested all of the required databases must have active connections in the Database Explorer.

In the runtime environment, remote connections are managed by the DWE Application Server component for sending data to remote tables, and the data will flow through the DWE Application Server as a gateway. Data from local data

sources will stay local to the execution database and does not flow through the DWE Application Server.

Target table operators provide three ways to update the data in a target table:

- ▶ INSERT adds rows from the input data set to the target table without any changes to existing rows.
- ▶ UPDATE matches rows from the input data set with those in the target table. If the match is found, the row is updated; if no match exists, no action is taken.
- ▶ DELETE checks the target table for rows that meet a specified condition and deletes those rows.

Figure 6-22 shows the starting contents of the STAFF2 table. We also have three input tables, one of which will be used to insert into STAFF2, one of which will be used to update rows in STAFF2, and one of which will be used to delete rows from STAFF2. While we use source table operators as input into the target table operator, it could be an output port from any other operator.

| ID | NAME | DEPT | JOB | YEARS | SALARY | COMM |
|-----|-----------|------|-------|-------|----------|---------|
| 50 | Hanes | 15 | Mgr | 10 | 20659.80 | - |
| 70 | Rothman | 15 | Sales | 7 | 16502.83 | 1152.00 |
| 110 | Ngan | 15 | Clerk | 5 | 12508.20 | 206.60 |
| 170 | Kermisch | 15 | Clerk | 4 | 12258.50 | 110.10 |
| 10 | Sanders | 20 | Mgr | 7 | 18357.50 | - |
| 20 | Pernal | 20 | Sales | 8 | 18171.25 | 612.45 |
| 80 | James | 20 | Clerk | - | 13504.60 | 128.20 |
| 190 | Sneider | 20 | Clerk | 8 | 14252.75 | 126.50 |
| 30 | Marenghi | 38 | Mgr | 5 | 17506.75 | - |
| 40 | O'Brien | 38 | Sales | 6 | 18006.00 | 846.55 |
| 60 | Quigley | 38 | Sales | - | 16808.30 | 650.25 |
| 120 | Naughton | 38 | Clerk | - | 12954.75 | 180.00 |
| 180 | Abrahams | 38 | Clerk | 3 | 12009.75 | 236.50 |
| 90 | Koonitz | 42 | Sales | 6 | 18001.75 | 1386.70 |
| 100 | Plotz | 42 | Mgr | 7 | 18352.80 | - |
| 130 | Yamaguchi | 42 | Clerk | 6 | 10505.90 | 75.60 |
| 200 | Scoutten | 42 | Clerk | - | 11508.60 | 84.20 |
| 140 | Fraye | 51 | Mgr | 6 | 21150.00 | - |
| 150 | Williams | 51 | Sales | 6 | 19456.50 | 637.65 |
| 220 | Smith | 51 | Sales | 7 | 17654.50 | 992.80 |
| 230 | Lundquist | 51 | Clerk | 3 | 13369.80 | 189.65 |
| 250 | Wheeler | 51 | Clerk | 6 | 14460.00 | 513.30 |
| 270 | Lea | 66 | Mgr | 9 | 18555.50 | - |
| 280 | Wilson | 66 | Sales | 9 | 18674.50 | 811.50 |
| 310 | Graham | 66 | Sales | 13 | 21000.00 | 200.30 |
| 320 | Gonzales | 66 | Sales | 4 | 16858.20 | 844.00 |
| 330 | Burke | 66 | Clerk | 1 | 10988.00 | 55.50 |
| 290 | Quill | 84 | Mgr | 10 | 19818.00 | - |
| 300 | Davis | 84 | Sales | 5 | 15454.50 | 806.10 |
| 340 | Edwards | 84 | Sales | 7 | 17844.00 | 1285.00 |
| 350 | Gafney | 84 | Clerk | 5 | 13030.50 | 188.00 |

Figure 6-22 Starting contents of STAFF2 table

INSERT

Note that the starting contents of STAFF2 do not contain any rows for DEPT 10. Our first table, STAFF10, contains rows for DEPT that need to be inserted into STAFF2. See the update input data, data flow, and properties in Figure 6-23.

To create this data flow, we simply add a table source operator representing STAFF10 and a table target operator representing STAFF2 and connect the ports. We select the INSERT SQL operation in the General properties tab for the STAFF2 target table operator and make sure that the input columns are mapped to the STAFF2 columns correctly in the Map properties tab.

After executing the data flow, the rows from STAFF10 are inserted into STAFF2, as depicted in Figure 6-23. STAFF2 now has four rows for DEPT10.

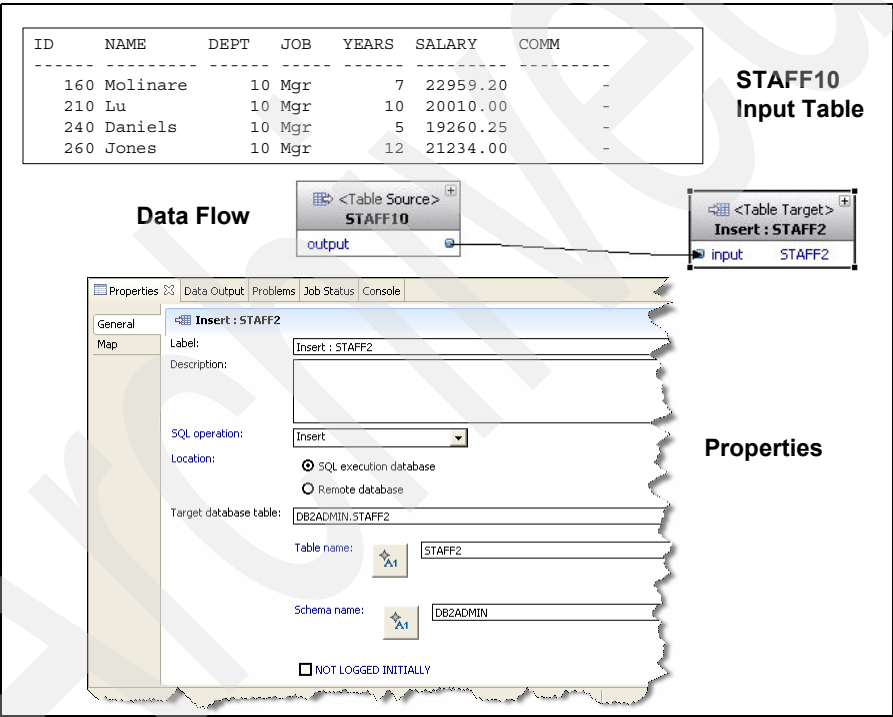


Figure 6-23 Target table operator - INSERT

UPDATE

In this data flow, the table STAFF15 has eight rows and has updates for the COMM column. All values have been set to 99. However, the updates are only to be applied to DEPT15 and not any other department. See the update input data, data flow, and properties in Figure 6-24.

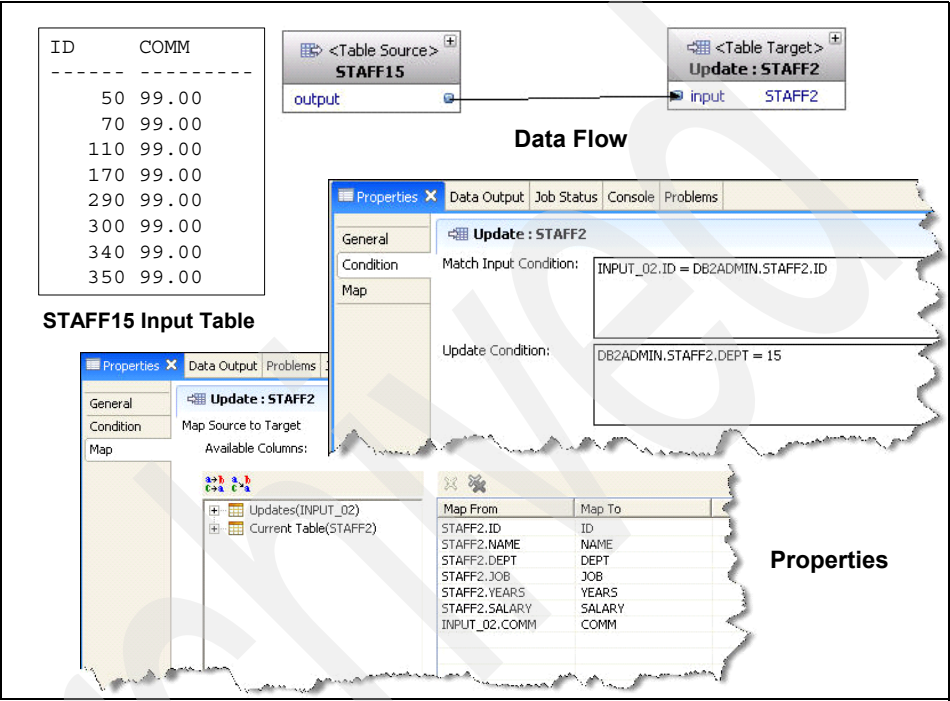


Figure 6-24 Target table operator - UPDATE

Creating this update data flow is the same as creating the previous insert data flow, with the exception of the properties for the table target operator. To instruct the operator to perform an update we set the SQL Operation field in the General properties tab to UPDATE. This adds a new properties tab (Condition tab) in which to set the conditions as to how the update is to be done.

The Condition tab has two condition properties. The *match input condition* specifies the condition that checks for existing rows in the target table with keys that match the keys in each input row. When a match is found, an update is performed. If no match is found, the input record is ignored. In our example, the match was on the column ID of both the input table and the target table. Optionally, the *update condition* could be used to specify a subset of rows from the target table that are eligible to be updated, such as only those target rows for DEPT 15, as set in the update condition.

On the Mapping property tab, resultant columns may be mapped from either the input table or the target table. We are only mapping the updated COMM column from the input table and the rest are mapped from the target table.

After the execution of this data flow, the COMM column for the four DEPT 15 rows in the input table will have been updated to 99.00 in STAFF2, but not the input rows for any other department. See the resulting STAFF2 table in Table 6-1.

DELETE

The delete data flow deletes from the target table all matching rows from the input table. In this example, we want to delete all rows from STAFF2 that have a matching record in STAFF20, which contain the four rows in Figure 6-25. Creating the data flow is similar to creating the insert and update data flows. In the General property tab, the DELETE SQL operation is selected and we define the delete condition on the Condition property tab.

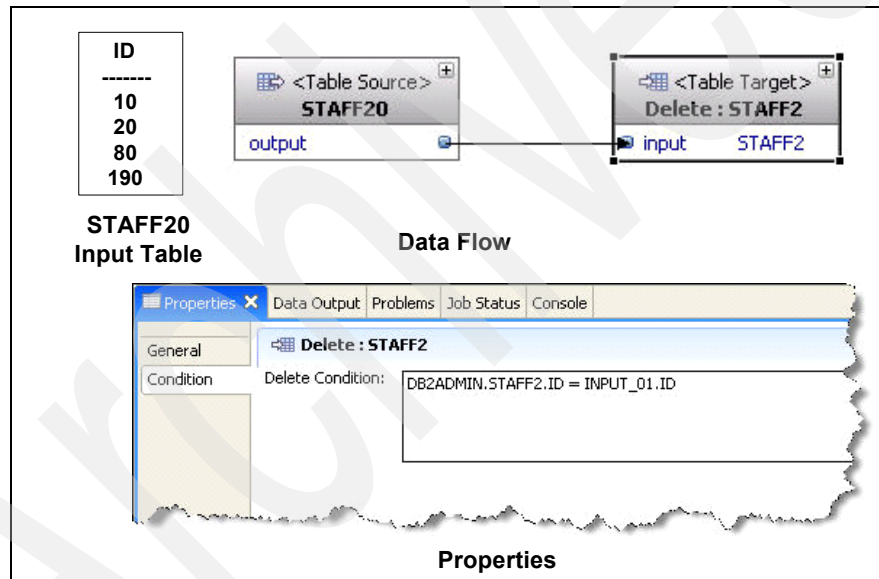


Figure 6-25 Target table operator - DELETE

After the execution of this data flow, we see that the four rows matching the ID column of STAFF20 have been deleted from STAFF2 in Table 6-1.

Table 6-1 Ending contents of STAFF2 table

| ID | Name | Dept | Job | Years | Salary | Comm |
|-----|----------|------|-----|-------|----------|------|
| 160 | Molinare | 10 | Mgr | 7 | 22959.20 | - |

| ID | Name | Dept | Job | Years | Salary | Comm |
|-----|-----------|------|-------|-------|----------|---------|
| 210 | Lu | 10 | Mgr | 10 | 20010.00 | - |
| 240 | Daniels | 10 | Mgr | 5 | 19260.25 | - |
| 260 | Jones | 10 | Mgr | 12 | 21234.00 | - |
| 50 | Hanes | 15 | Mgr | 10 | 20659.80 | 99.00 |
| 70 | Rothman | 15 | Sales | 7 | 16505.83 | 99.00 |
| 110 | Ngan | 15 | Clerk | 5 | 12508.20 | 99.00 |
| 170 | Kermisch | 15 | Clerk | 4 | 12258.50 | 99.00 |
| 30 | Marenghi | 38 | Mgr | 5 | 17506.75 | - |
| 40 | O'Brien | 38 | Sales | 6 | 18006.00 | 846.55 |
| 60 | Quigley | 38 | Sales | - | 16808.30 | 650.25 |
| 120 | Naughton | 38 | Clerk | - | 12954.75 | 180.00 |
| 180 | Abrahams | 38 | Clerk | 3 | 12009.75 | 236.50 |
| 90 | Koonitz | 42 | Sales | 6 | 18001.75 | 1386.70 |
| 100 | Plotz | 42 | Mgr | 7 | 18352.80 | - |
| 130 | Yamaguchi | 42 | Clerk | 6 | 10505.90 | 75.60 |
| 200 | Scoutten | 42 | Clerk | - | 11508.60 | 84.20 |
| 140 | Fraye | 51 | Mgr | 6 | 21150.00 | - |
| 150 | Williams | 51 | Sales | 6 | 19456.50 | 637.65 |
| 220 | Smith | 51 | Sales | 7 | 17654.50 | 992.80 |
| 230 | Lundquist | 51 | Clerk | 3 | 13369.80 | 189.65 |
| 250 | Wheeler | 51 | Clerk | 6 | 14460.00 | 513.30 |
| 270 | Lea | 66 | Mgr | 9 | 18555.50 | - |
| 280 | Wilson | 66 | Sales | 9 | 18674.50 | 811.50 |
| 310 | Graham | 66 | Sales | 13 | 21000.00 | 200.30 |
| 320 | Gonzales | 66 | Sales | 4 | 16858.20 | 844.00 |
| 330 | Burke | 66 | Clerk | 1 | 10988.00 | 55.50 |
| 290 | Quill | 84 | Mgr | 10 | 19818.00 | - |

| ID | Name | Dept | Job | Years | Salary | Comm |
|-----|---------|------|-------|-------|----------|---------|
| 300 | Davis | 84 | Sales | 5 | 15454.50 | 806.10 |
| 340 | Edwards | 84 | Sales | 7 | 17844.00 | 1285.00 |
| 350 | Gafney | 84 | Clerk | 5 | 1303.50 | 188.00 |

Bulk load target operator

The bulk load target operator can be used for the DB2 Load Utility to load the results of a data flow into a DB2 relational table, as seen in Figure 6-26. Bulk loads typically outperform SQL inserts on large data volumes. The target table must be a local table. Data can be loaded to the target table with the INSERT load mode property or the table contents can be replaced with the REPLACE load mode property.

Advanced options that can be set are the recovery mode, warning count, messages file, and the path for temporary files. Other DB2 Load Utility options can be adding entries to the Advance Options box. However, the syntax is not validated by the DWE Design Studio.

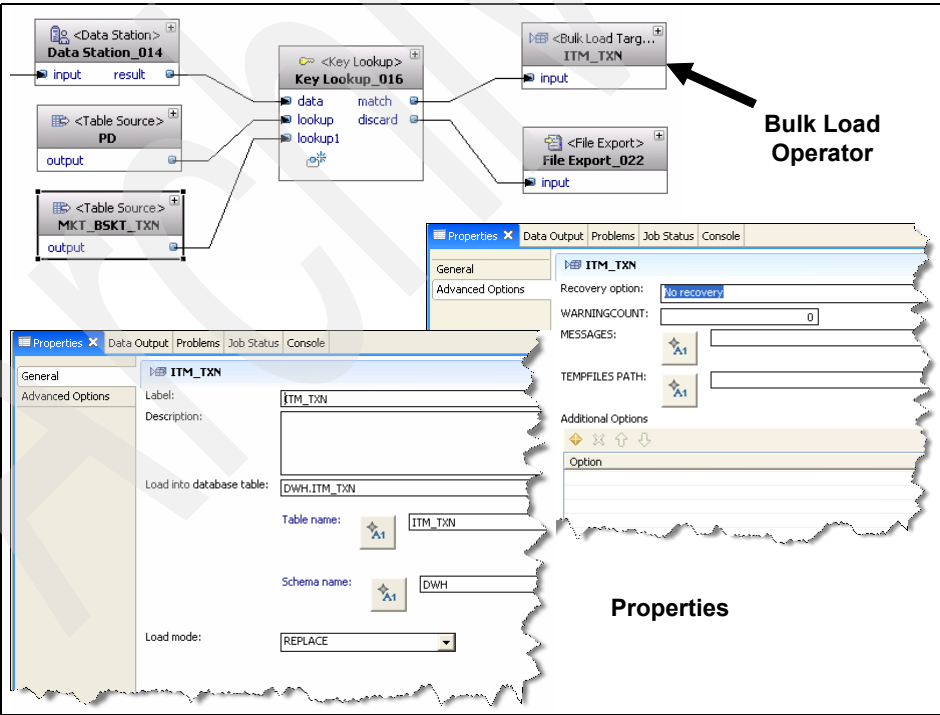


Figure 6-26 Bulk load target operator

SQL merge target operator

The SQL merge operator adds data to a target DB2 relational table using the DB2 merge statement, which emulates a combination of UPDATE and INSERT operations and is commonly known as an *upsert*.

The SQL merge operator loads a DB2 table by checking the target table for a row that matches a row from the input data set. Two conditions define the operation of a SQL merge operator. The first condition, called the *input match*, matches rows from the input with existing rows in the target table. If no matching row is found, the row is inserted as a new row. If an input match is found, the second condition, called the *target table match*, checks a value in each input row against the associated row in the target table. If the condition matches, the row in the target table is updated. Separate column mapping properties allow you to choose one set of columns for any rows that meet the insert criteria and a different set of columns for the updates to act upon.

In Figure 6-27 on page 171, there is an example data flow showing one usage of the SQL Merge operator. There is a target table, STAFF_TARGET, into which to merge a set of data from a staging table, STAFF_STAGE. The STAFF_STAGE table contains a set of rows with the staff key column ID and SALARY updates. The updates are to be applied to the STAFF_TARGET with the following rules:

- ▶ If a corresponding staff ID already exists in the target table, just replace the SALARY column in the target table with the SALARY from the staging table.
- ▶ Update an existing row only if the existing SALARY is less than \$18,000.
- ▶ If there is not a corresponding staff ID row in the target table, insert a new row with the ID and SALARY values in the target table and let the JOB column default to NULL.

The Update Map property tab defines how the target table is updated when the match condition is met. In this case, only map the SALARY column from the staging table to the SALARY column in the target table.

The Insert Map property tab defines how to add rows to the target table when the update match condition is not met. In this case, map the two input columns, ID and SALARY, from the staging table to the corresponding columns in the target table.

The Conditions property tab defines the *input match condition* and an optional filter condition on the target table, *target table match condition*. Only when both of these conditions are met will the match qualify and the target row be updated. In this example, the match condition will be an equality between the ID columns of both tables. However, if there is a match on ID, only update the target table if the existing value in the target table for SALARY is less than \$18,000.

Examining the results of the merge, we note that:

- ▶ IDs 40, 50, 60, and 70 in the staging table had corresponding rows in the target table and passed the input match condition.
- ▶ IDs 40 and 50 were not updated since their existing SALARY is greater than \$18,000, and therefore failed the target table match condition and were not updated.
- ▶ IDs 50 and 60 were updated in the target table as they met both conditions.
- ▶ IDs 80, 90, and 100 did not have corresponding rows in the target table and failed the input match condition. Therefore they were inserted into the target table.

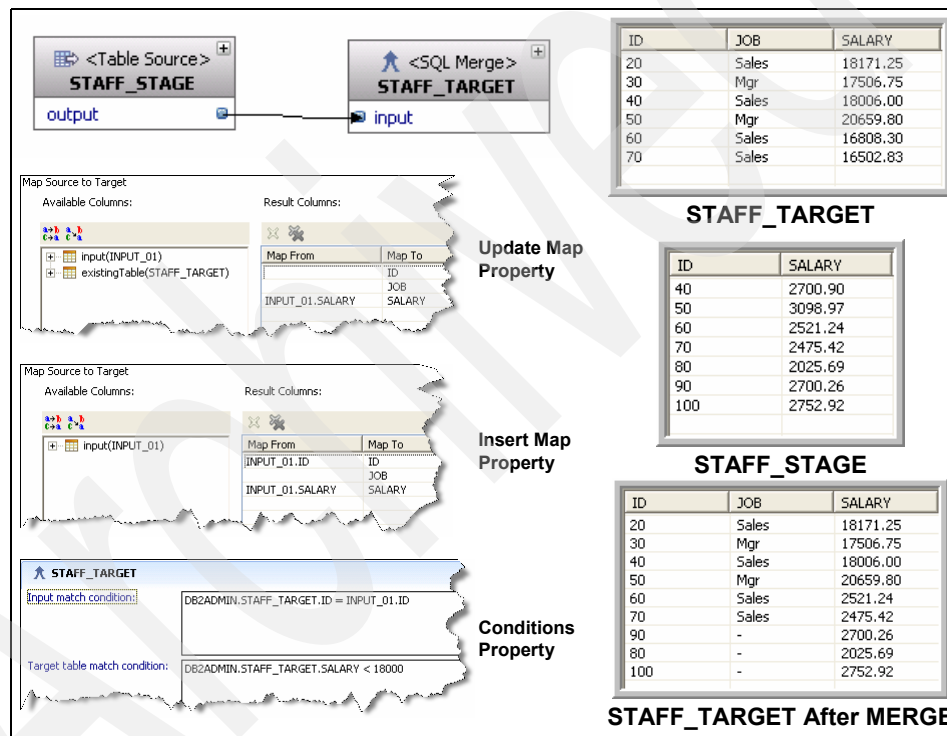


Figure 6-27 SQL merge target operator

Slowly changing dimension target operator

The slowly changing dimension (SCD) target operator provides a method for maintaining dimension tables in a data warehouse. Dimension tables contain attributes describing the business, such as products, clients, organization, time, and many others. Dimension tables are used in a star schema data model with fact tables that contain data that can be measured. The fact table contains

foreign keys to the related dimension tables so that the fact table can be joined with one or more of the dimension tables to view data by any of the various attributes in the dimension tables. A dimensional model such as this is used in Online Analytical Processing (OLAP) applications.

Dimension tables are relatively static over time but do pose challenges when values change. One very important purpose of a data warehouse is to maintain an accurate historical representation of the business. Therefore the data should be maintained such that changes can be tracked. However, there can be thousands of dimensional attributes that can change, and the data warehouse developers must decide whether a change to a particular attribute is important enough to the business to maintain the history of that change to feed into the update strategy for each dimension.

There are three standard dimension update strategies that have been defined:

- ▶ Type 1: This strategy essentially requires the entire row to be replaced with the updated row, keeping the same surrogate key, and does not maintain any history.
- ▶ Type 2: This strategy creates a new row containing the modified data with a new surrogate key and the same natural key. Some mechanism is used to indicate which record is current, typical with a pair of dates that indicate the effective date range. However, this technique indicates that some attribute changed in the dimension record, but not which attribute.
- ▶ Type 3: This strategy attempts to keep a limited amount of history for specific attributes by adding columns to the dimension table for current value and previous value. Of course, it is not limited to two and can be extended to some number of attributes in a series. An effective date may also be associated with the attribute series.

The SCD target operator supports type 1 and type 2 slowly changing dimension strategies. Only one SCD type can be defined for an SCD operator.

Here we illustrate a type 2 SCD operation to update a store dimension table, STORE_DIM, using the data values in a staging table, STORE_DIM_STAGE. The tables are depicted in Figure 6-28. Note that the STORE_DIM table has a surrogate key, STORE_KEY, which is generated using a DB2 sequence called STORE_SURROGATE_KEY provided using the NEXTVAL statement in the surrogate key expression property.

| STORE_KEY | STORE_CODE | STORE_TYPE | MANAGER_ID | STORE_SU... | STORE_RE... | STORE_DIS... | STORE_NAME | EFFECTIVE... | END_DATE |
|-----------|------------|---------------|------------|-----------------|---------------|-----------------|---------------|--------------|----------|
| 1 | 1199 | Nghbrhd M... | 16 | Subdivision ... | Region 45 ... | District 127... | ValueTrend... | 1/1/06 | 12/31/99 |
| 2 | 1095 | ValueTrend... | 21 | Subdivision ... | Region 45 ... | District 127... | ValueTrend... | 1/1/06 | 12/31/99 |
| 3 | 375 | Nghbrhd M... | 17 | Subdivision ... | Region 24 ... | District 226... | ValueTrend... | 1/1/06 | 12/31/99 |
| 4 | 681 | Supercente... | 19 | Subdivision ... | Region 24 ... | District 226... | ValueTrend... | 1/1/06 | 12/31/99 |
| 5 | 782 | Supercente... | 22 | Subdivision ... | Region 44 ... | District 38 ... | ValueTrend... | 1/1/06 | 12/31/99 |
| 6 | 554 | ValueTrend... | 26 | Subdivision ... | Region 44 ... | District 38 ... | ValueTrend... | 1/1/06 | 12/31/99 |
| 7 | 278 | Nghbrhd M... | 25 | Subdivision ... | Region 22 ... | District 477... | ValueTrend... | 1/1/06 | 12/31/99 |
| 8 | 116 | Supercente... | 18 | Subdivision ... | Region 22 ... | District 477... | ValueTrend... | 1/1/06 | 12/31/99 |
| 9 | 875 | ValueTrend... | 20 | Subdivision ... | Region 24 ... | District 490... | ValueTrend... | 1/1/06 | 12/31/99 |
| 10 | 1414 | Nghbrhd M... | 23 | Subdivision ... | Region 44 ... | District 84 ... | ValueTrend... | 1/1/06 | 12/31/99 |
| 11 | 835 | Nghbrhd M... | 24 | Subdivision ... | Region 44 ... | District 84 ... | ValueTrend... | 1/1/06 | 12/31/99 |

STORE_DIM Before

| STORE_CODE | STORE_TYPE | MANAGER_ID | STORE_SU... | STORE_RE... | STORE_DIS... | STORE_NAME |
|------------|-----------------|------------|-----------------|---------------|-----------------|---------------|
| 1095 | Midsize | 21 | Subdivision ... | Region 45 ... | District 127... | ValueTrend... |
| 681 | Supercenter ... | 119 | Subdivision ... | Region 24 ... | District 226... | ValueTrend... |
| 116 | Midsize | 18 | Subdivision ... | Region 22 ... | District 477... | ValueTrend... |
| 835 | Nghbrhd Mkt ... | 124 | Subdivision ... | Region 44 ... | District 84 ... | ValueTrend... |

STORE_DIM_STAGE

| STORE_KEY | STORE_CODE | STORE_TYPE | MANAGER_ID | STORE_SU... | STORE_RE... | STORE_DIS... | STORE_NAME | EFFECTIVE... | END_DATE |
|-----------|------------|-------------|------------|-----------------|---------------|-----------------|---------------|--------------|----------|
| 1 | 1199 | Nghbrhd Mkt | 16 | Subdivision ... | Region 45 ... | District 127... | ValueTrend... | 1/1/06 | 12/31/99 |
| 2 | 1095 | ValueTrend | 21 | Subdivision ... | Region 45 ... | District 127... | ValueTrend... | 1/1/06 | 6/22/06 |
| 3 | 375 | Nghbrhd Mkt | 17 | Subdivision ... | Region 24 ... | District 226... | ValueTrend... | 1/1/06 | 12/31/99 |
| 4 | 681 | Supercenter | 19 | Subdivision ... | Region 24 ... | District 226... | ValueTrend... | 1/1/06 | 6/22/06 |
| 5 | 782 | Supercenter | 22 | Subdivision ... | Region 44 ... | District 38 ... | ValueTrend... | 1/1/06 | 12/31/99 |
| 6 | 554 | ValueTrend | 26 | Subdivision ... | Region 44 ... | District 38 ... | ValueTrend... | 1/1/06 | 12/31/99 |
| 7 | 278 | Nghbrhd Mkt | 25 | Subdivision ... | Region 22 ... | District 477... | ValueTrend... | 1/1/06 | 12/31/99 |
| 8 | 116 | Supercenter | 18 | Subdivision ... | Region 22 ... | District 477... | ValueTrend... | 1/1/06 | 6/22/06 |
| 9 | 875 | ValueTrend | 20 | Subdivision ... | Region 24 ... | District 490... | ValueTrend... | 1/1/06 | 12/31/99 |
| 10 | 1414 | Nghbrhd Mkt | 23 | Subdivision ... | Region 44 ... | District 84 ... | ValueTrend... | 1/1/06 | 12/31/99 |
| 11 | 835 | Nghbrhd Mkt | 24 | Subdivision ... | Region 44 ... | District 84 ... | ValueTrend... | 1/1/06 | 6/22/06 |
| 12 | 1095 | Midsize | 21 | Subdivision ... | Region 45 ... | District 127... | ValueTrend... | 6/22/06 | 12/31/99 |
| 13 | 681 | Supercenter | 119 | Subdivision ... | Region 24 ... | District 226... | ValueTrend... | 6/22/06 | 12/31/99 |
| 14 | 116 | Midsize | 18 | Subdivision ... | Region 22 ... | District 477... | ValueTrend... | 6/22/06 | 12/31/99 |
| 15 | 835 | Nghbrhd Mkt | 124 | Subdivision ... | Region 44 ... | District 84 ... | ValueTrend... | 6/22/06 | 12/31/99 |

STORE_DIM After

Figure 6-28 SCD example - data tables

Figure 6-29 shows the dataflow and operators properties. Note that the surrogate key expression does not have to be a DB2 sequence but, if it is, it can also be provided via connecting a sequence operator to the sequence column input port. Look at “Sequence operator” on page 190 to see this SCD data flow using a sequence operator to provide the surrogate key instead of an expression.

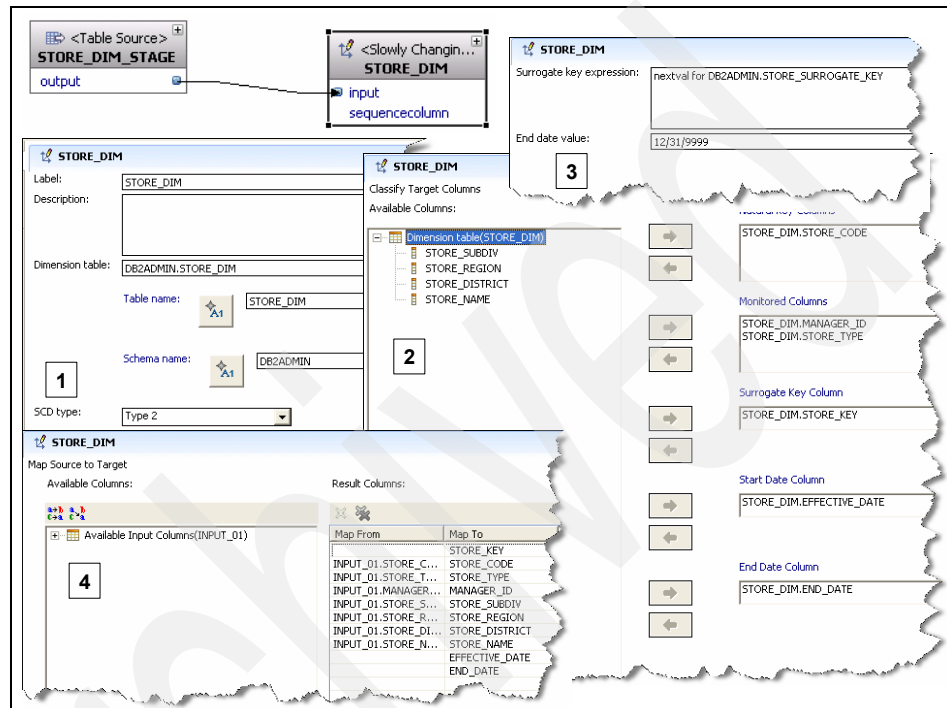


Figure 6-29 SCD data flow and operator properties

Transform operators

Transform operators represent some type of transformation action taken against the data flowing into the operator. As shown in Figure 6-30 on page 175, there are two categories of data flow transform operators. *SQL Warehousing* transforms represent typical SQL-based types of data movement and transformation functions. *Data mining* transforms allow data mining models to be applied to data flowing through a data flow. Data mining model creation is accomplished with a mining flow, which is discussed in Chapter 7, “DWE and data mining” on page 237.

Most transform operators have both input and output ports, but a few may have only an output port, and some may have multiple input ports or output ports, sometimes as a fixed number or sometimes as a variable. If ports can be added, there will be an icon at the bottom of the port list with the operator icons.

SQL Expression Builder

Before getting into the details of the transformers, it is helpful to understand the SQL Expression Builder, as it is a common component that is used by many of the transform operators. It is also referred to as the SQL Condition Builder. This provides a point-and-click Wizard for building SQL expressions.

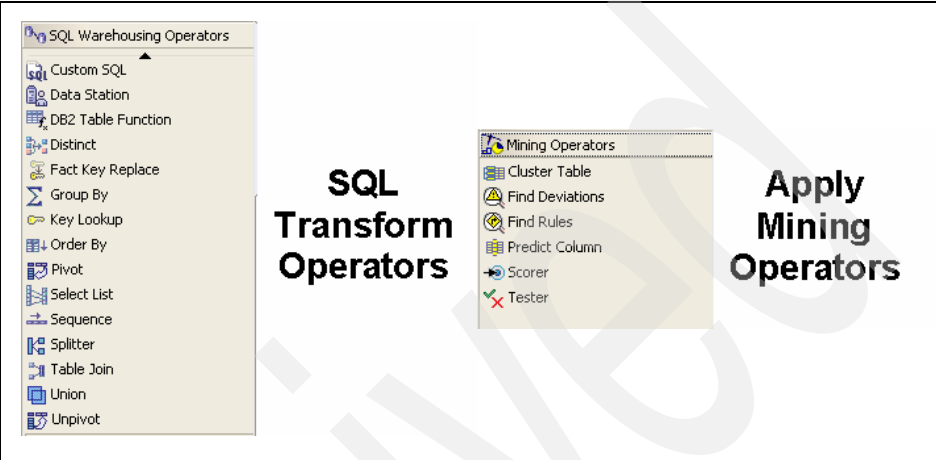


Figure 6-30 SQW transformation operators

Figure 6-31 shows the SQL expression builder dialog. There are four select boxes along the top that contain the table input columns, keywords, operators, and functions that can be used in building the SQL expression. The bottom text box contains the text of the expression.

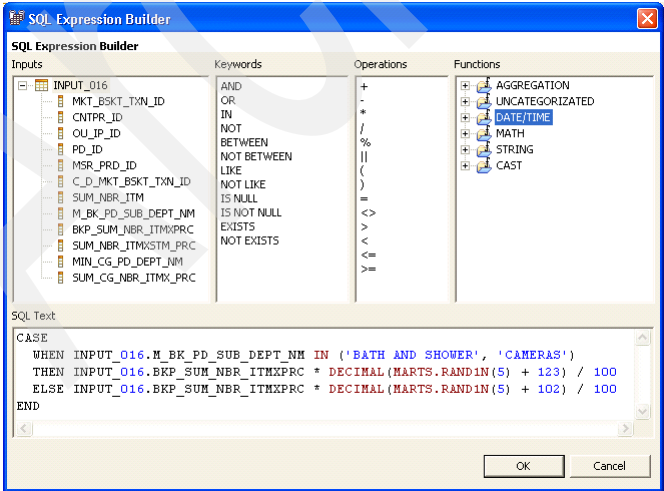


Figure 6-31 SQL expression builder

An expression is built by double-clicking the various elements in the select boxes, or they can be typed in manually. Any valid column function that is in the execution database can be manually added to the SQL text. This is useful for user-defined functions (UDFs) in DB2 and for other valid functions that might not be in the list of common functions. Aggregation functions are only available when the expression builder is launched from a *group by* or *select list* operator.

Select list operator

The select list operator emulates the SELECT clause of an SQL statement. It can be used to add, drop, split, modify, or combine columns of a data set. Columns can be added and modified using scalar functions, arithmetic, and constant values. Certain scalar functions, such as CONCATENATE and DATE, and column expressions containing arithmetic functions enable you to effectively combine values from multiple columns into a single column. Other functions enable you to change the data type of a column. From the column list property of the select list operator you can access the expression builder, which makes it easier to create complex column expressions by providing interactive lists of available input columns, scalar functions, and boolean and arithmetic operators.

The Select List property tab, depicted in Figure 6-32, is where the mapping from the input to the output and the transformation functions are defined. In the example, all of the columns from the input table are being mapped to the output table, but it also could have been just a subset of the input columns. We transform the SEX column by replacing the values M and F with MALE and FEMALE. There is also a new column, TOTAL_SALARY, that is a calculated value summing SALARY, BONUS, and COMM. To invoke the expression builder, highlight a cell under the Expression column and click the ellipsis button that appears.

| Expression | Column Name | Data Type | Length | Scale |
|---|--------------|-----------|--------|-------|
| INPUT_01.EMPNO | EMPNO | CHAR | 6 | |
| INPUT_01.FIRSTNAME | FIRSTNAME | VARCHAR | 12 | |
| INPUT_01.MIDINIT | MIDINIT | CHAR | 1 | |
| INPUT_01.LASTNAME | LASTNAME | VARCHAR | 15 | |
| INPUT_01.WORKDEPT | WORKDEPT | CHAR | 3 | |
| INPUT_01.PHONENO | PHONENO | CHAR | 4 | |
| INPUT_01.HIREDATE | HIREDATE | DATE | | |
| INPUT_01.JOB | JOB | CHAR | 8 | |
| INPUT_01.EDLEVEL | EDLEVEL | SMALLINT | | |
| CASE INPUT_01.SEX WHEN 'M' THEN 'MALE' WHEN 'F' THEN 'FEMALE' ELSE '' END | SEX | CHAR | 4 | |
| INPUT_01.BIRTHDATE | BIRTHDATE | DATE | | |
| INPUT_01.SALARY | SALARY | DECIMAL | 9 | 2 |
| INPUT_01.BONUS | BONUS | DECIMAL | 9 | 2 |
| INPUT_01.COMM | COMM | DECIMAL | 9 | 2 |
| INPUT_01.SALARY + INPUT_01.BONUS + INPUT_01.COMM | TOTAL_SALARY | DECIMAL | 12 | 2 |

Figure 6-32 SELECT LIST operator - mapping

A number of other operators also contain basic functionality of the *select* operator including:

- ▶ Group by
- ▶ Key lookup
- ▶ Splitter (each output port has a select list)
- ▶ SQL merge
- ▶ Table join
- ▶ Table target

Group by operator

The GROUP BY operator groups rows in a data set and summarizes values in designated columns using COUNT, SUM, MIN, MAX, AVG, and other functions, thereby emulating the SQL GROUP BY function. A GROUP BY operator is similar to a SELECT LIST operator, except that it provides the additional functionality of a GROUP BY property.

As depicted in Figure 6-33 on page 178, a GROUP BY operator has the properties that you would expect. The Select List property is equivalent to the select statement in a SQL GROUP BY statement, which selects the result columns and applies the appropriate function to the calculated columns. The group by property specified the grouping columns and the *having* property specifies any further filtering on the resulting data.

Tip: If you do not provide the having condition, you will receive a warning during validation that the condition is not set. If you do not like to see warnings, then simply add a condition of 1=1, as depicted in Figure 6-33.

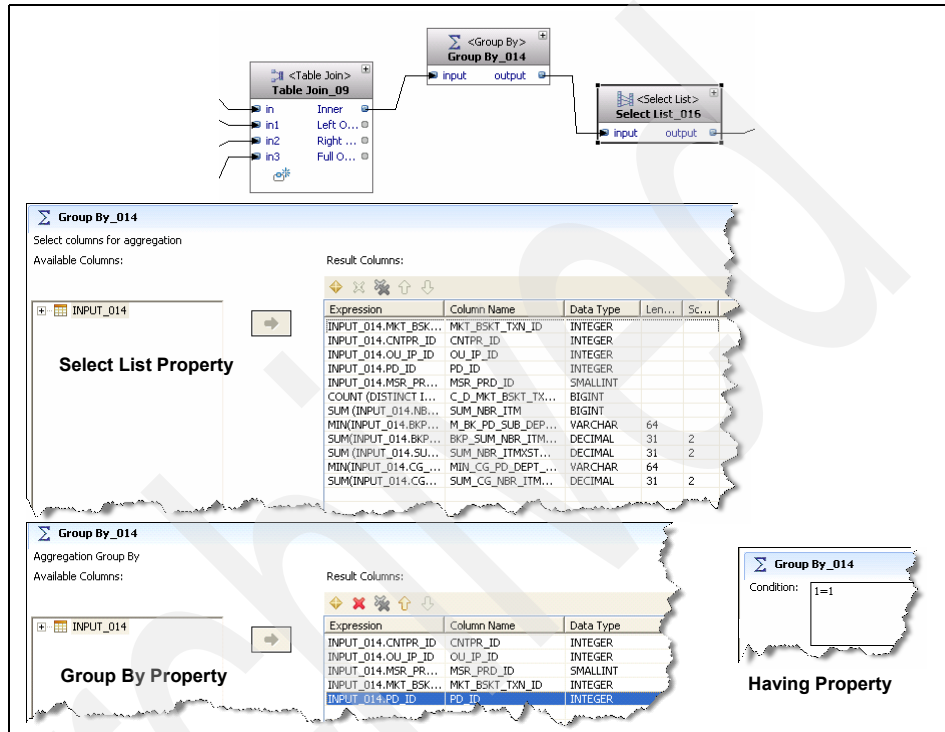


Figure 6-33 GROUP BY operator

Table join

The TABLE JOIN operator does exactly what the name indicates, it performs joins between two or more relational tables. It supports inner joins, left outer joins, right outer joins, full outer joins, and cross joins.

By default the TABLE JOIN operator has two input ports and four output ports. You may add any number of input ports by clicking the *Add a new port* icon found underneath the last input port. The four output ports represent the join types: inner, left outer, right outer, and full outer, and the output ports that have connections define the type of joins to be done. If you only have two input ports, then you may use any combination of the four output ports. However, if there are more than two input ports, then outer joins are not allowed.

As shown in Figure 6-34, the properties for the TABLE JOIN operator are basically the join condition and the mapping of columns from input ports to the result. The expression builder can be invoked to build the join condition.

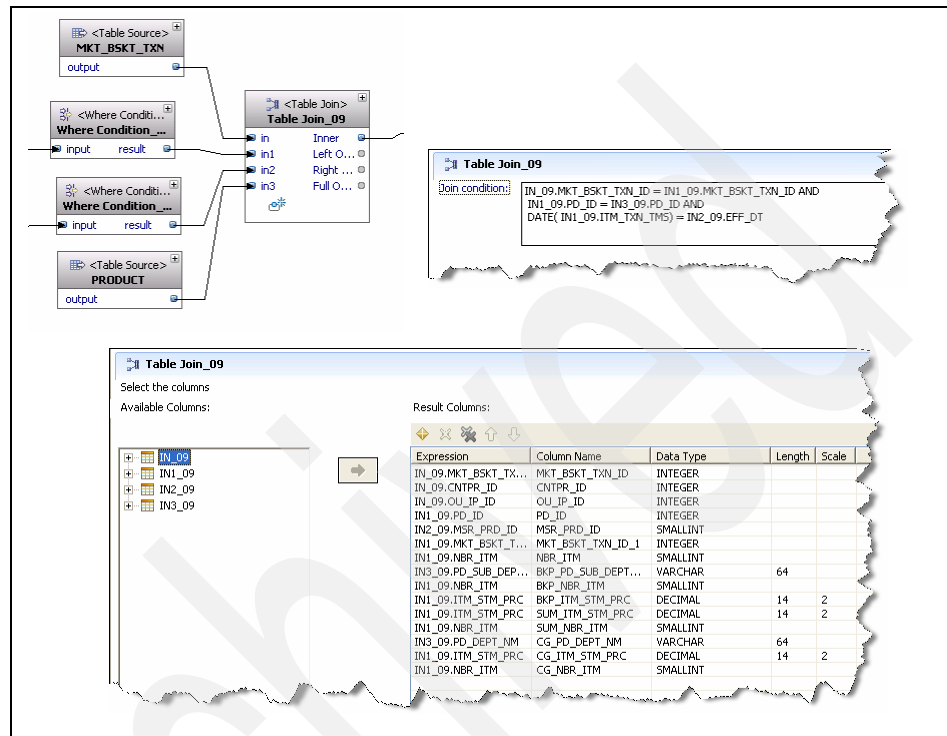


Figure 6-34 TABLE JOIN operator

Splitter operator

The SPLITTER operator takes one input and multiple outputs based on specified criteria. Each output may be different in terms of columns and rows and does not need to contain an exclusive set of rows.

The property of the SPLITTER is primarily the filter condition associated with each output port. The mapping of columns is handled via the port connections. In Figure 6-35 the STAFF table is split into three outputs based on DEPT.

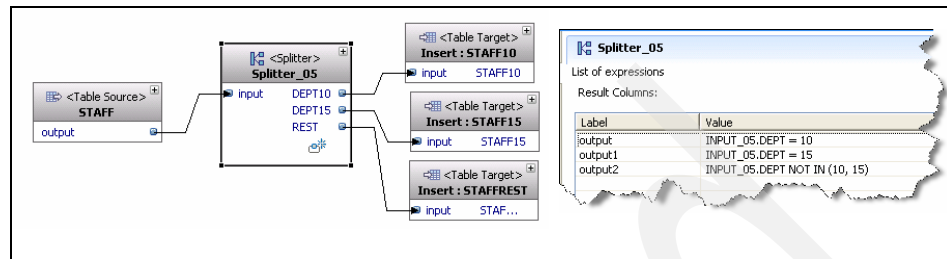


Figure 6-35 SPLITTER operator

Key lookup operator

The KEY LOOKUP operator is used to compare keys from a single input table with keys in one or more lookup tables and discard input table rows that do not have matching rows in the lookup tables. Rows that successfully match are sent to the output port. Otherwise they are sent to the discard port. Using the select list properties can select a subset of columns, add columns, and use expressions from the input table or lookup tables.

In Figure 6-36 the KEY LOOKUP operator has three inputs: the data table and two lookup tables. Ports for additional lookup tables can be added by clicking the icon just below the input port list. There is one condition in the condition list property tab for each lookup table stating the matching condition with the input table. The select list is where the result columns are selected from any of the input tables or derived columns.

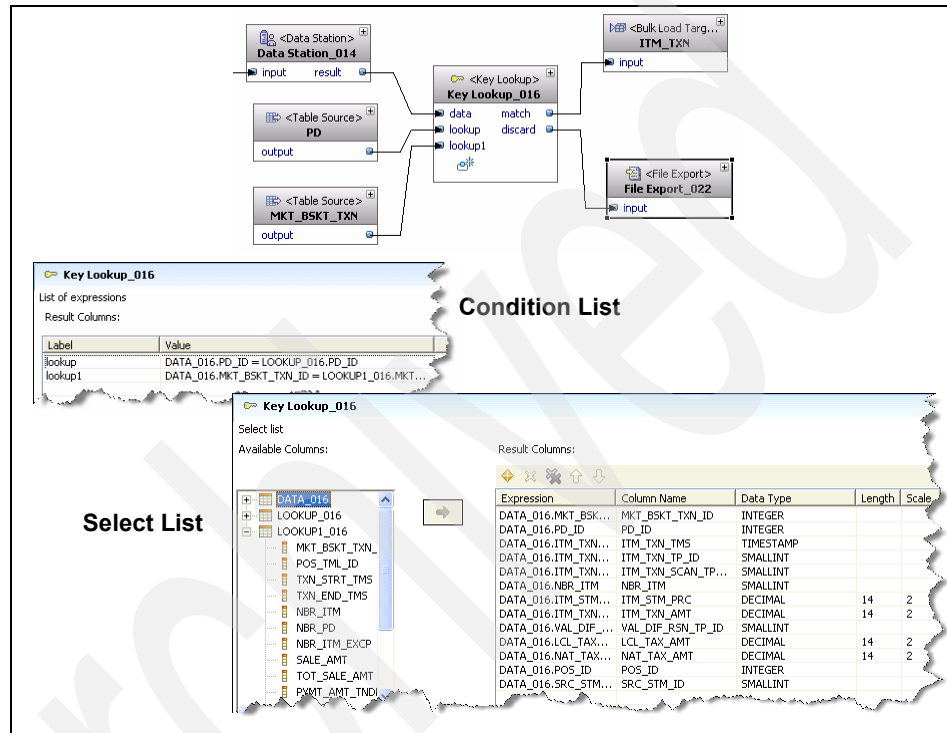


Figure 6-36 KEY LOOKUP operator

Distinct operator

The DISTINCT operator simply removes duplicate rows from the input table, passing unique rows to the result port and duplicates to the discard port. When there are duplicates, the first row found is passed to the result port, the rest to the discard port, and there is no guarantee in the order of processing.

Figure 6-37 shows a DISTINCT operator that has a file connected to the input port. The column select property tab specifies which columns are checked for determining uniqueness that, in the example, is a subset of the input columns. Note that the entire row is passed to the appropriate output port.

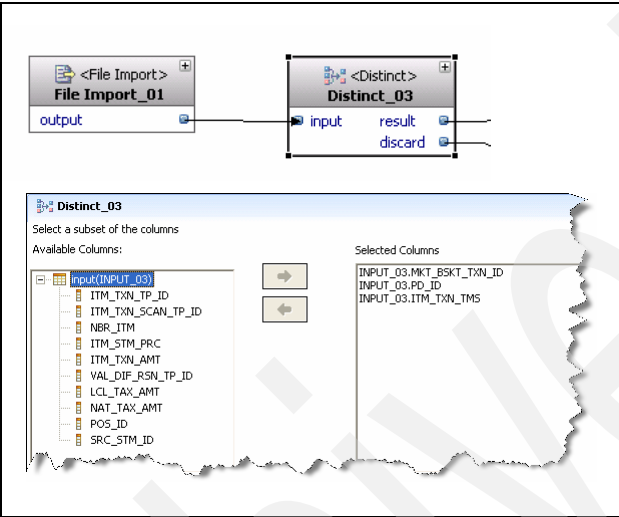


Figure 6-37 DISTINCT operator

WHERE CONDITION operator

The WHERE CONDITION operator is used to implement filtering of the input data based on a condition with matching rows flowing to the result port. There is no discard port on the WHERE CONDITION. The SPLITTER operator can be used when there is a need for multiple conditions and outputs.

As we see in Figure 6-38, the primary property is the filter condition, which can be built using the expression builder. All input rows that match this condition will be passed to the result port. There is no action taken for non-matching rows.

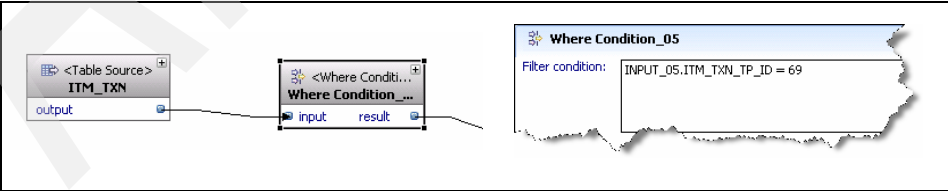


Figure 6-38 WHERE CONDITION operator

ORDER BY operator

The ORDER BY operator will sort the input data according to the values in one or more designated columns, passing all of the columns of the entire row to the result port.

Figure 6-39 shows an ORDER BY operator taking input from the output of a select list, sorting it, and passing it to a target table via the result port. The sort key ordering property specifies which input columns to use for sorting and the sort order of each column. When multiple columns are selected, they are sorted in the order listed.

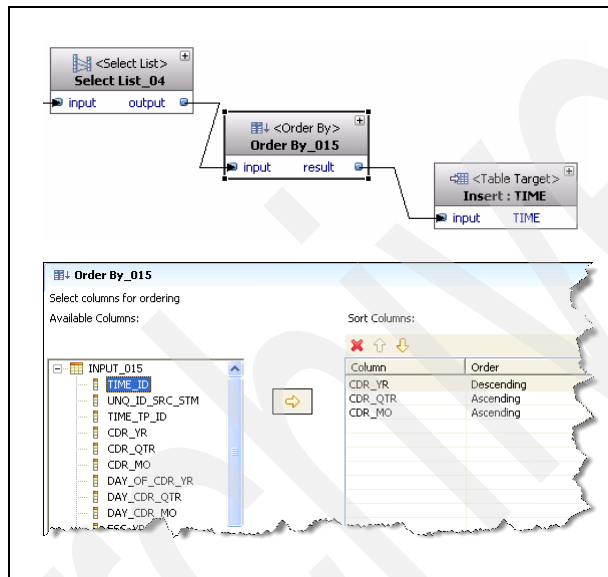


Figure 6-39 ORDER BY operator

UNION operator

The UNION operator performs the set operations of UNION, INTERSECTION, and DIFFERENCE. There are two inputs that are processed according to the selected set operation and passed to the result output port. All columns of each row are passed through.

The three types of set operators can be depicted using Venn diagrams, as seen in Figure 6-40. The set operator type is selected in the set details property tab.

- ▶ UNION unconditionally merges two sets of input rows into a single output data set.
- ▶ INTERSECTION merges two sets of input rows into a single output data set and retains only those rows that are common to both inputs.
- ▶ DIFFERENCE merges two sets of input rows into a single output data set and retains only those rows that exist in the first data set but not the second data set.

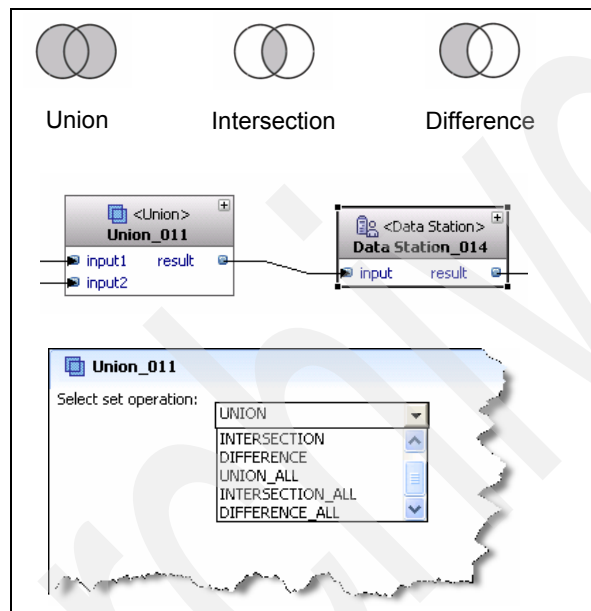


Figure 6-40 Union operator

When the set operator types are modified by the keyword ALL and duplicate rows occur in the input data set, the duplicates are retained in the output data set. If the ALL option is not used, the duplicate rows are discarded.

FACT KEY REPLACE operator

The FACT KEY REPLACE operator looks up surrogate keys from dimension tables or key mapping tables and uses them to replace corresponding natural keys in a fact table. The input consists of a data table, one or more lookup tables, and an output table. Natural and surrogate keys are identified for each lookup table and the natural keys of the lookup tables mapped to the corresponding column in the input table. Figure 6-41 shows a data flow that contains a FACT KEY REPLACE.

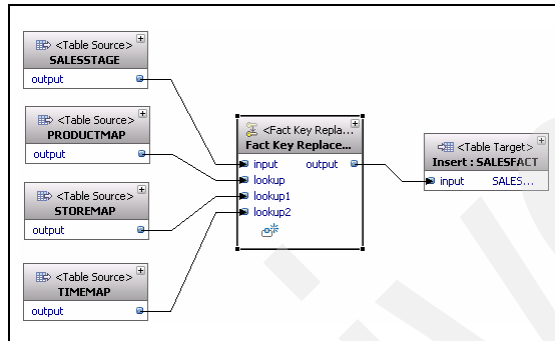


Figure 6-41 Fact key replace operator

The FACT KEY REPLACE operator differs from most other operators by requiring the properties of each lookup port to be set to indicate the surrogate key column and the natural key columns before completing the natural key map property page of the fact key replace operator. Figure 6-42 on page 186 depicts the properties for one of the lookup ports — the one for the PRODUCT lookup table. There is an additional property page, Keys Classification, where we indicate that PRODUCTID is the surrogate key and PRODUCT_NATKEY is the natural key.

Once the key classification properties of each lookup port have been defined, the natural key map of the fact key replace operator must be completed. The natural key of every lookup table is in the left pane and can be dragged to the appropriate columns of the input table in the right pane, as shown in Figure 6-42.

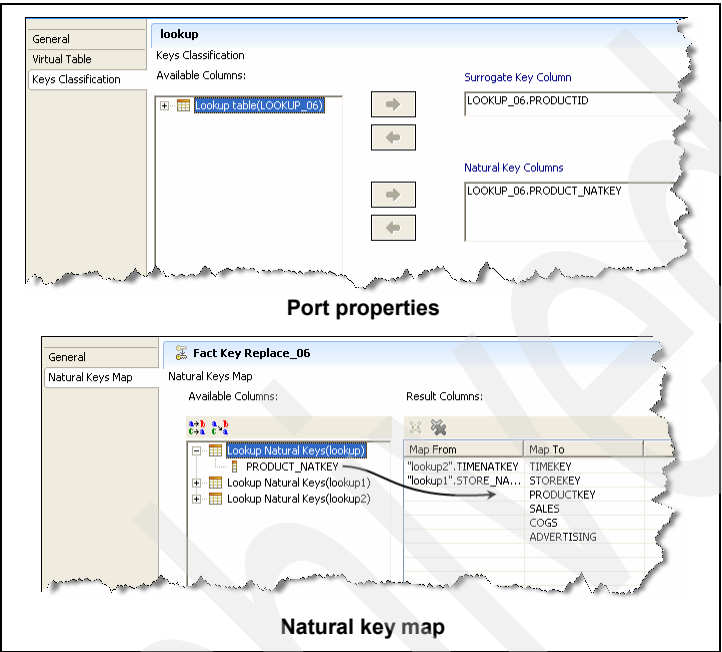


Figure 6-42 Fact key replace properties

PIVOT operator

The PIVOT operator groups data from several related columns into a single column by creating additional rows in the result. In other words, it turns column-oriented data into row-oriented data. An example is depicted in Figure 6-44 on page 188. Here we have data as it might typically appear in a spreadsheet, with one row for each store and quarter along with three columns of sales for each month in that quarter. This is typically not conducive to relational processing and querying. It is better to have one row in the table for each combination of store, quarter, and month, as shown in the result table in Figure 6-43.

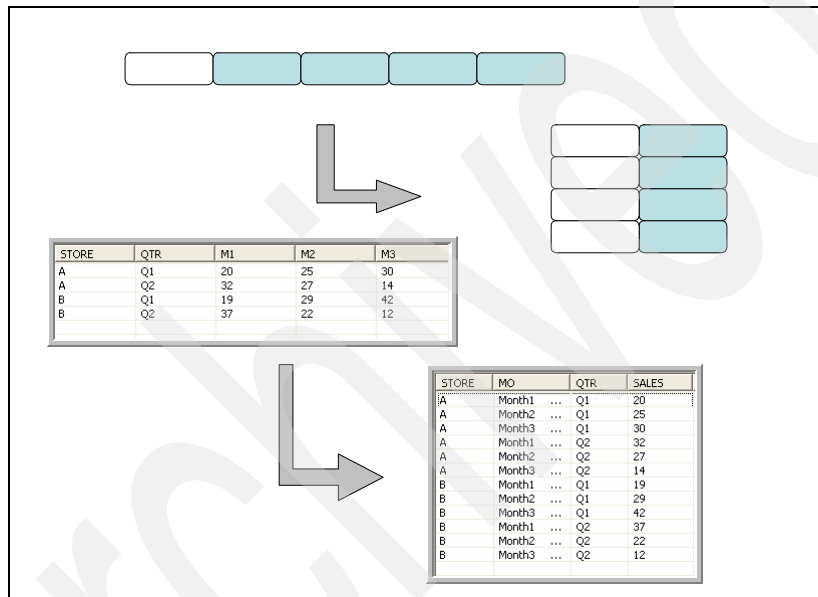


Figure 6-43 Pivoting data

The pivot operator has one input and one output, and operator properties that specify the carry-over columns, pivot columns, pivot groups, data group and type, and the pivot definition, as shown in Figure 6-44 on page 188. Brief descriptions of the operator properties in that example are below:

- ▶ *Carry over columns*: These are the columns, STORE and QTR, that move from the source to the target without change.
- ▶ *Pivot columns*: The columns M1, M2, and M3, whose values will pivot from columns to rows.
- ▶ *Pivot groups*: The number of columns the pivot operator creates. There is one in the example in Figure 6-43, which is named SALES.

- ▶ *Data group*: The name and data type of the new column that will contain the values that were the column names in the source data as defined in the pivot definition. In the example, the new column is MO with a data type of CHAR.
- ▶ *Pivot definition*: This maps from the column names of the pivoted columns from the source table to a value that will appear in the new data group column. In the example, the column names of M1, M2, and M3 are mapped to Month1, Month2, and Month3, which will appear in the MO column.

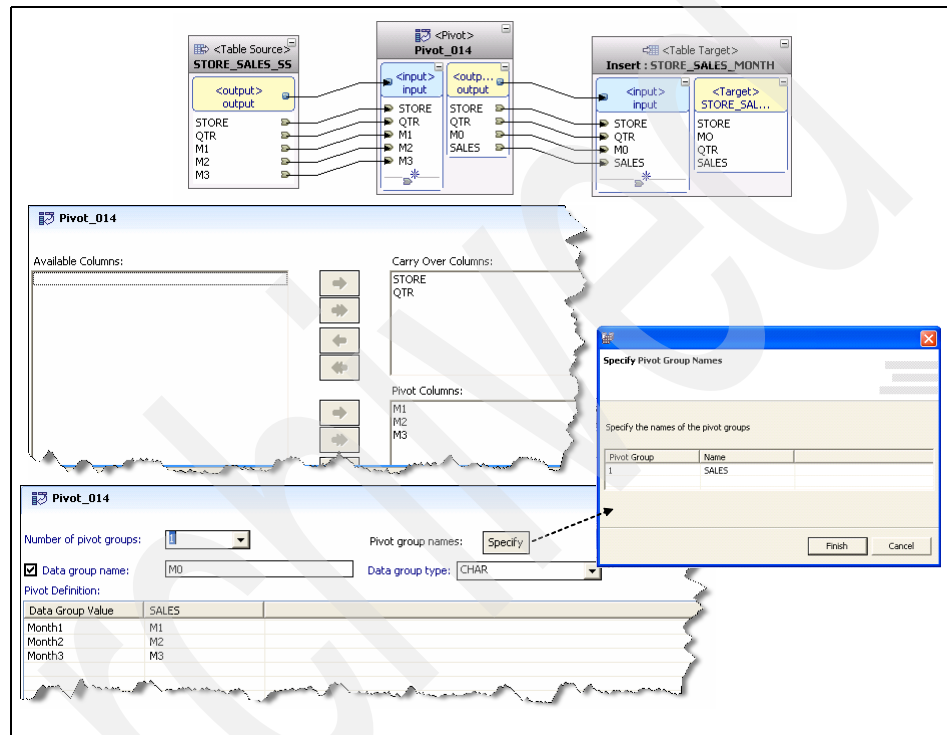


Figure 6-44 Pivot operator

Compare the properties in Figure 6-44 to the source and target values in Figure 6-43 on page 187.

UNPIVOT operator

The unpivot operator does just the opposite of the pivot operator by using repeating values (months or quarters, for example) in a single column, called the data group, as headings for a new set of columns. Values from a specified set of input columns, called unpivot columns, are arranged under the new column heading according to the pivot group value found in each row. Other columns,

called key columns, define the new set of rows. See Figure 6-45 for an example that does the exact opposite of what we described in “PIVOT operator” on page 187.

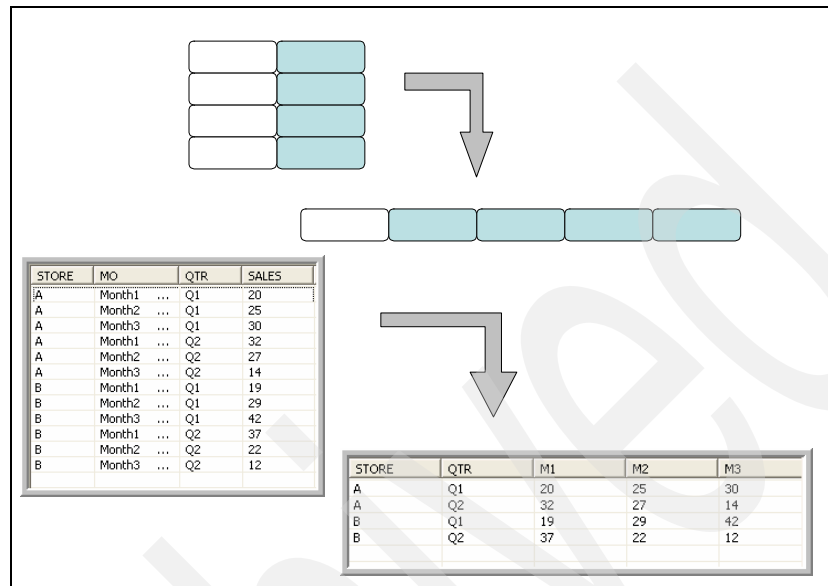


Figure 6-45 Unpivot operation

The unpivot operator has one input and one output, and the operator properties specify the data group column, unpivot columns, key columns, and the unpivot definition, as shown in “UNPIVOT operator” on page 188. We provide a brief description of those operators in the following list:

- ▶ **Data group:** This specifies the column that contains the repeated values that will become new column headings in the result table. In the example, column MO contains the values of the month that will become the column headings in the result table.
- ▶ **Unpivot columns:** This specifies the columns containing the data that will be arranged under the newly created column headings. In the example, this is the SALES column.
- ▶ **Key columns:** This specifies the columns that carry over from the source to the target. There will be one row in the result table for each unique combination of key columns values. In the example, the result will contain one row for each unique combination of STORE and QUARTER.
- ▶ **Unpivot definition:** This determines, based on the number of unpivot columns and the number of distinct values in the unpivot column, the number of output columns required and the mapping of source values to output columns. The

example has one unpivot column with three distinct values, which and requires three output columns. Any row containing the value of Month1 in the MO column will map to the result column M1. Values of Month2 and Month3 will map to the result columns of M2 and M3, respectively.

Compare the properties in Figure 6-46 to the source and target values in Figure 6-45 on page 189.

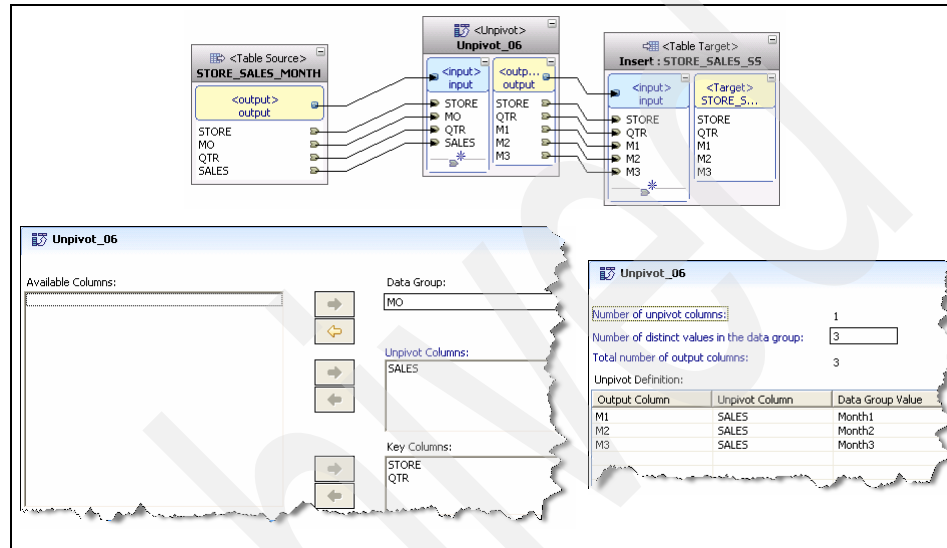


Figure 6-46 Unpivot operator

Sequence operator

The sequence operator generates values for a column, based on an existing DB2 sequence object. The DB2 sequence object must already exist in the physical data model. This sequence operator has no input ports and one output port that provides the next value in the sequence. The sequence operator may be used in many places. One major use is to provide the surrogate key for the slowly changing dimension operator.

Figure 6-47 shows the sequence operator providing the surrogate key for a slowly changing dimension operator. The property of the sequence operator is simply the name of the DB2 sequence as defined in the physical data model. Note that the output port of the sequence operator and the input port of the slowly changing dimension operator have to be expanded to make the connection from the NEXTVAL output port to the SEQUENCEVALUE input port.

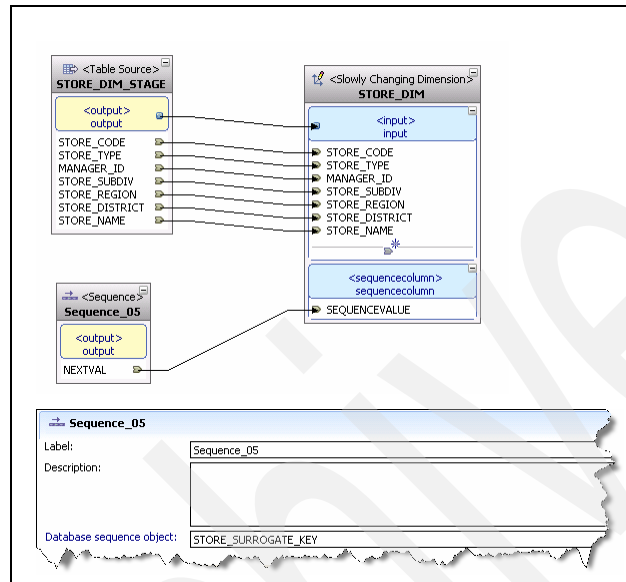


Figure 6-47 Sequence operator

Data station operator

The data station operator is used to define an explicit staging point in a data flow. Staging is done to store intermediate processed data for the purpose of tracking, debugging, or ease of data recovery. Staging may occur implicitly within a data flow as determined by the data flow code generator but does not persist after the data flow execution. However, you may want to have data put into a persistent store at various points during processing, perhaps as logical recovery points. The data station operator allows you to define when to put the data into a persistent store.

There are a number of reasons to persist intermediate data during processing. During development, you may want to view the data after certain operations to ensure that the processing was done correctly. You can add a data station in the middle of a data flow for debugging purposes. You may also need to keep track of the data at certain points in the processing for audit purposes or perhaps to provide a recovery point. A data station can be added at the appropriate points in the data flow.

There are four storage types for data stations. A *persistent table* will store the data in a permanent relational table existing in the data model. You can optionally specify to delete all data from the table after the data flow has executed. A *temporary table* will store the data in a temporary table object (global temporary table or regular table) during the data flow, but the data will not persist after the execution of the data flow. A *view* is useful to influence code generation to not implicitly persist data and is useful in data mining scenarios. A *flat file* will persist the data to a flat file, which may be useful in scenarios where you want to use a bulk loader to load data from a flat file. One flat file data station operator can be used instead of a file export target and a file import source.

The exact properties of the data station depend on the storage type, but basically provide the name of the object. A persistent table will need to already exist in the physical data mode. A temporary table will require a temporary tablespace to exist in the database. The names of temporary tables and views must be unique within the data flow. The path and file name of a flat file must be provided.

The pass-through property is valid for all data station storage types. If this is checked, then the data station operator is ignored by the code generator during execution. This is useful for testing execution scenarios and for debugging the data flow. When testing and debugging is finished, simply check the data station pass-through property to have the data station ignored.

Figure 6-48 shows a data station operator to explicitly persist data at a logical point in the data flow, after a series of operations to process imported flat files. The storage type is a persistent table with the name DWH.ITM_TXN_STAGE.

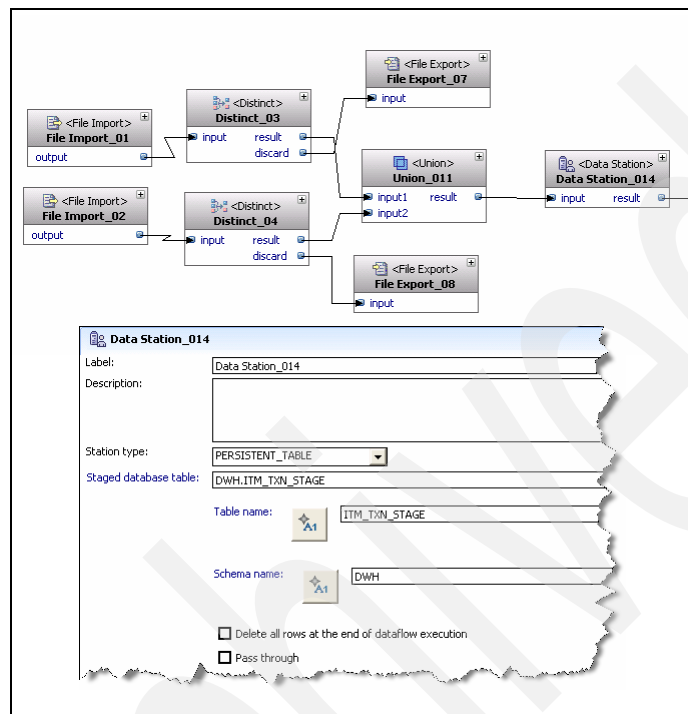


Figure 6-48 Data station operator

DB2 table function operator

A DB2 table function operator invokes predefined DB2 table functions. A table function is a logical database object that returns a table (one or more rows) based on a statement expressed in a specific language, such as SQL, C, or Java. Table functions can be used to define many types of SQL queries or programmatic requests for data from external sources. Table functions are powerful and flexible data transformation tools.

Customized table functions can be developed in any language supported by DB2 table functions. These custom developed functions can do whatever is supported by the underlying language. This allows access to just about anything and has it returned as a table to a dataflow.

In addition to developing customized table functions, there are a number of table function wizards provided by DB2 to access MQ queues, XML files, and OLE data providers. By providing some information to these wizards, the appropriate

table function will be defined. You then simply reverse engineer these functions into your data model and point to them with a DB2 table function operator.

Figure 6-49 shows two examples of using DB2 table function operators. The first one is an SQL-based custom table function that takes a list of time and store keys as input, invokes the table function, and returns a table that flows through to other operators. The second example invokes a DB2-provided C table function, which returns information from the history file that is associated with the currently connected database partition as a table that flows through to other operators (a file export operator in this simple example).

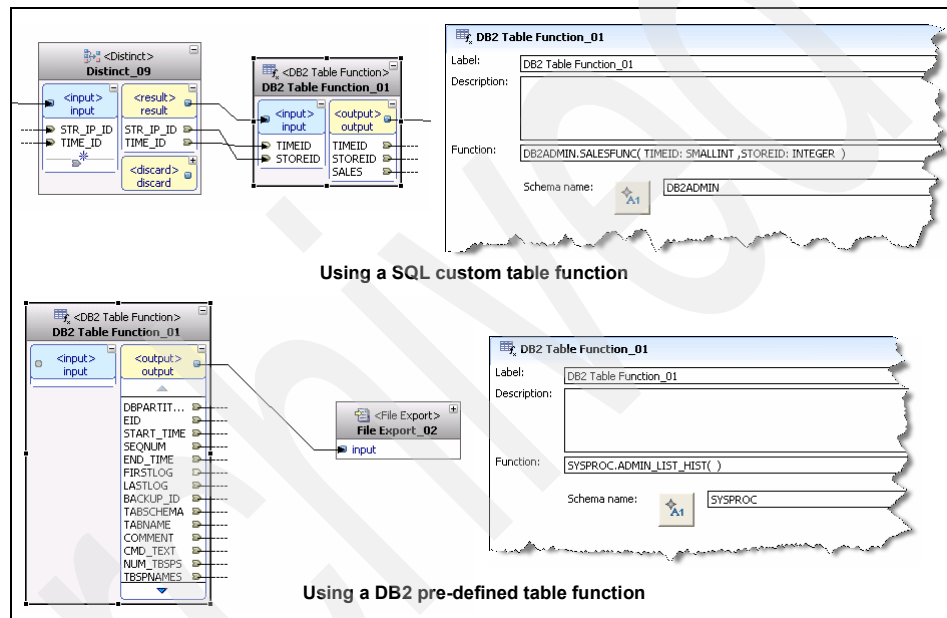


Figure 6-49 DB2 table function operator

Custom SQL operator

The custom SQL operator is used to add one or more embedded SQL statements that modifies the database in some way. The operator can have one or more input ports, but does not contain an output port. The input ports make available the input schemas available to the expression builder, but any table in the execution database may be referenced in the supplied SQL.

SQL statements are typed into the SQL code property, but the expression can be used as a helper. Any valid SQL statement may be used, and multiple SQL statements may be entered. These statements are not validated until execution time. See Figure 6-50 for an example of the custom SQL operator.

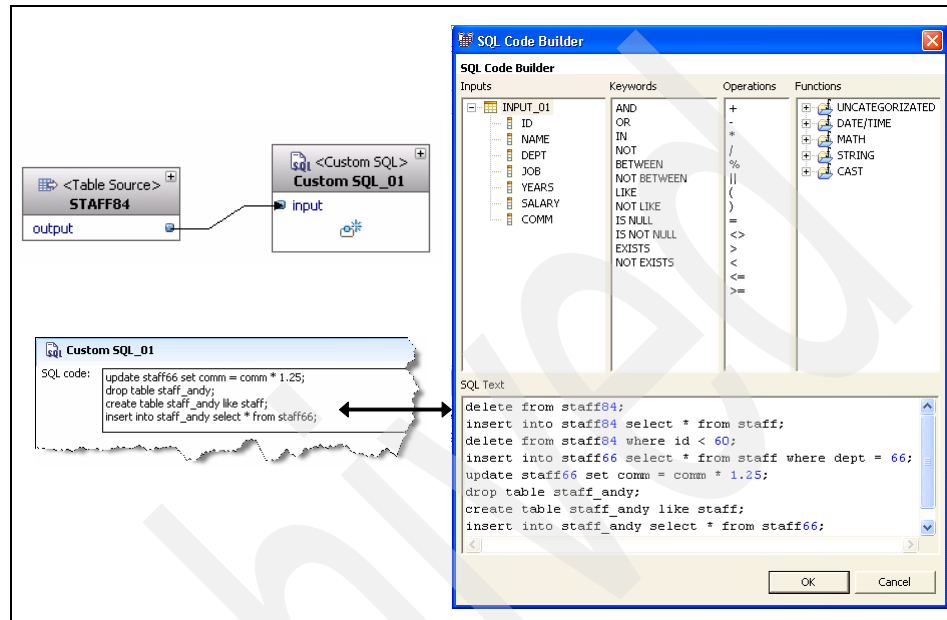


Figure 6-50 Custom SQL operator

Data-mining operators

For a detailed list of data-mining operators, refer to 7.5.4, “Data mining operators” on page 262.

6.2.4 Subflows

A subflow is a data flow that can be embedded inside another data flow. Subflows can also be embedded in another subflow, allowing the nesting of flows. The primary advantage of the subflow is that it is reusable across multiple data flows.

Subflows can be helpful when there are a series of operations that are the same across a number of data flows. One subflow can be created and connected inline into each data flow. Subflows can also be useful to simplify a complex data flow. A more complex flow can be segmented into several logical sections, with each implemented as a subflow. Then a more simple dataflow can be created by

referencing the subflows. This could help to give a much better overall understanding of what the data flow is doing.

Figure 6-51 shows three subflows, each with multiple operators, that implement some type of common business rules. Subflow A is used in both data flow 1 and data flow 2, while subflow B is used in data flow 1 and also nested in subflow C.

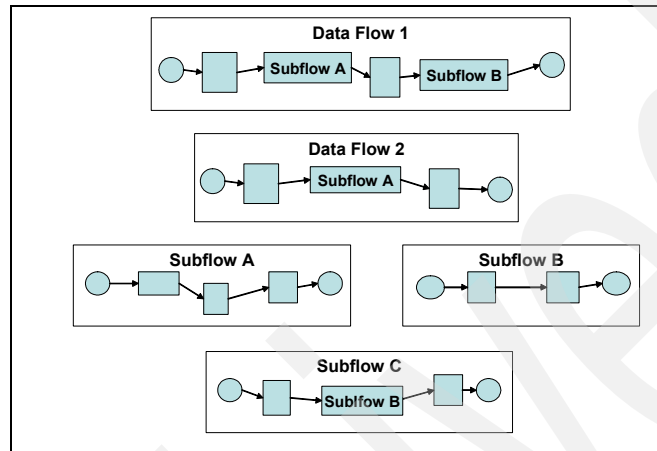


Figure 6-51 Usage of subflows

A subflow is embedded into a data flow by using the subflow operator, whose primary property is the subflow name. The input and output ports of the subflow operator will vary depending on the number of subflow input and subflow output operators defined within the subflow. Subflows may have one or more input and one or more output operators, or both. Figure 6-52 on page 197 depicts how the subflow input and output operators relate to the ports of the subflow operator that invokes the subflow. When adding a subflow input operator, a schema must be defined. It can be defined either by using the wizard manually or by loading an existing schema from a file or from the data model. The schema of a subflow output operator will flow normally from the connection of the input port. In addition to these special input and output operators, subflows can use any data flow operator.

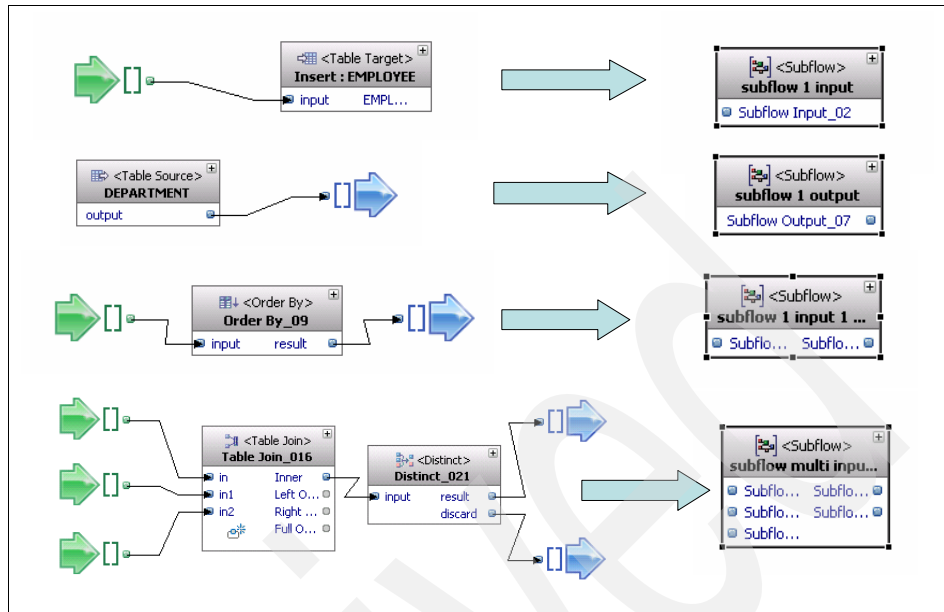


Figure 6-52 Subflow input and output operators become ports in the subflow operator

Tip: You can save a data flow as a subflow. This is useful if you are developing a data flow and want to be able to reuse it. Simply select the option **Save Data Flow as Subflow** from the Data Flow menu.

When a data flow goes through validation and code generation, the current version of a subflow will be used as the basis for generating the code as long as the input and output signatures do not change. If the input and output signatures do not match, or there are other validation errors in the subflow, then the entire data flow will fail validation.

6.2.5 Validation and code generation

A data flow represents what is to be done to the data — it is a model. Code is not being built in a data flow, but rather metadata definitions. However, at some point in time the data flow will be executed, which does require an executable, validation of the data flow, and code generation.

Validation and code generation can be explicitly invoked by using the Data Flow menu item in the menu bar. Any time a data flow is saved, validation is implicitly invoked. Code generation also invokes a validation.

Data flow validation

Validation is a DWE Design Studio function to examine the metadata of a data flow for correctness. If an issue is detected, it will be flagged and listed in the Problems view tab. If the problem is in an operator, there will be a visual notification icon in the upper left corner of each operator that has a problem.

Figure 6-53 shows how to explicitly invoke validation via the Data Flow menu item. The result of validation is shown in the data flow as well as in the Problems view. There are two operators that have errors, as indicated by the red X in the upper left-hand corner of the operator icon (pointed to by the arrows in the figure). Warnings are shown with a yellow triangle (indicated by an arrow and a circle around the symbol). If you hover the mouse over the error or warning symbol in an operator, a pop-up will appear with the text of the message. If you double-click the symbol, a more useful diagnostic dialog will open giving more information about the error. The Problems tab will contain errors and warnings from within your workspace. It is useful to click the resource heading to sort by resource name. As you can see highlighted in a shaded area, and pointed to by the blue arrow, all of the errors and warnings for our data flow are together.

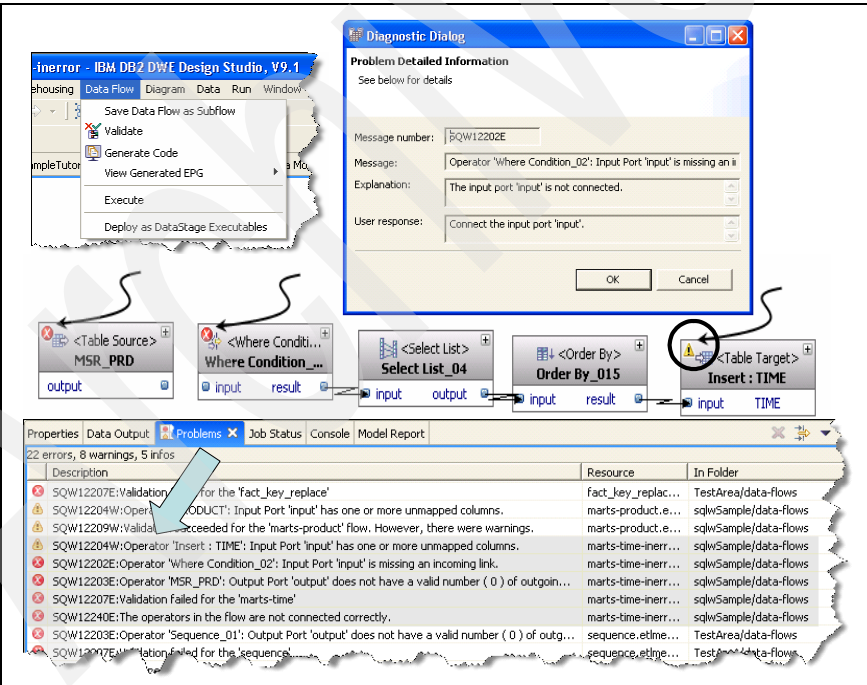


Figure 6-53 Validation

Using these methods to examine problems will eventually lead to a clean validation for the data flow. Remember that any subflows will also be validated

with this data flow. If there are warnings or errors in the subflow, they will also contain the appropriate symbol in the upper left corner of the operator icon.

Data flow code generation

Once the metadata has been validated and any problems corrected, code can be generated. You can explicitly invoke code generation via the Data Flow menu item. It is also implicitly invoked when you test execute a data flow. Code generation will examine the metadata of the data flow model, determine what is needed to execute, and generate the SQL and other calls to process the data flow.

The code is represented in a Execution Plan Graph (EPG) that can be viewed either textually or graphically. The EPG is a graph that describes the execution, transaction contexts, error handling, compensation, cleanup, and other semantics of the generated code.

This graph describes the sequence of code (called code units) to be executed as well as the type of such code (JDBC, DB2 SQL SCRIPT, Java method, command). The graph also has different block constructs, such as TXN - transaction block, TRY-CATCH-FINALLY block to implement compensation and cleanup functionality.

There are three types of execution plan graphs:

- ▶ *Deployment EPG* contains code that is run once (and only once) when the data warehouse application is deployed to the SQL Warehousing application-server-based runtime. This EPG is used to prepare the application environment for execution (for example, to register stored procedures).
- ▶ *Runtime EPG* contains execution time code that is executed every time a process is executed.
- ▶ *Undeployment EPG* contains code that is run when the data warehouse application is uninstalled from the SQL Warehousing runtime. It is the opposite of the deployment EPG. Hence, they are run only once.

When you explicitly generate code, the generated code will appear in a text window within the DWE Design Studio that you can browse. Depending on the data flow, this code may not be intuitively obvious, but you can see what will be executed. The EPG may also be viewed graphically. Figure 6-54 shows a data flow with a snippet of the generated code in text and graphics. The graphical viewer is used to step through and debug a data flow, as seen in 6.2.6, “Testing and debugging a data flow” on page 200.

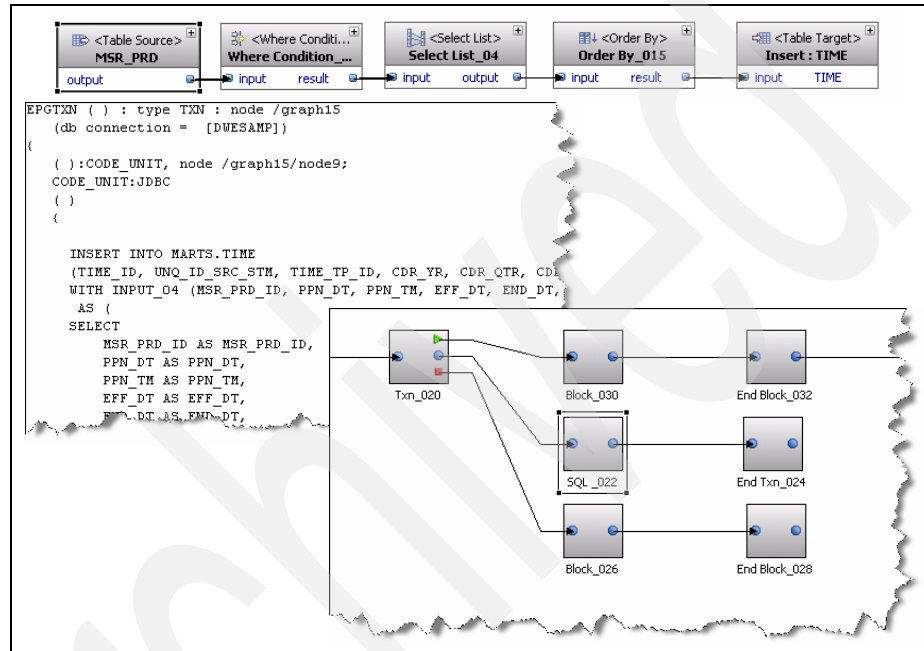


Figure 6-54 Generated code in text and graphical view

6.2.6 Testing and debugging a data flow

After successful validation, data flows can be tested with the DWE Design Studio without having to set up or deploy to a runtime environment. And, if needed, data flows can be debugged using the EPG graphical viewer.

To test execute a data flow, first establish a connection to the databases of interest in the Database Explorer. Make sure that the data flow is open and has the focus. Then select the **Execute** item from the Data Flow menu. You will be prompted with the Flow Execution dialog to provide information supporting the execution, such as the execution database, trace options, resource definitions, and set values for any variables. These values can be saved in a run profile for future use. Once the data flow has completed executing, there will be an

execution result, showing the status of the execution and any error messages that might have been received. An example is depicted in Figure 6-55.

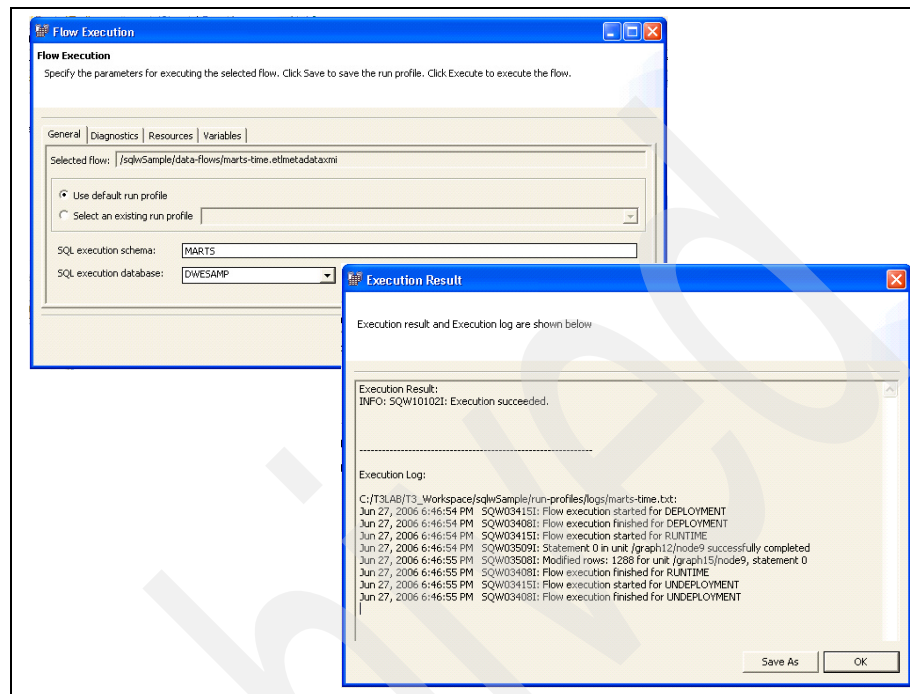


Figure 6-55 Test execution of a dataflow

Using the EPG graph viewer, you can debug the data flow by stepping through the various nodes in the graph. You can look at the code behind the nodes and, if it is SQL, even modify it. The modified code is not a permanent change and will be overwritten at the next code generation. To set up a debug session, you must first generate the code and then open then EPG viewer for one of the EPG graphs, typically the runtime EPG. Once the EPG viewer is open, you can start a debug session by clicking **Debug EPG** from the EPG menu on the menu bar. You will be prompted with a Flow Execution dialog for the debug flow session. Then simply provide the values for testing the data flow, and click the **Debug** button.

Tip: In the Diagnostics tab, selecting trace level content will give more information as you step execute through the data flow EPG.

When the debug session starts, the first node will be highlighted in orange (indicated with the circle in this example), as shown in Figure 6-56. The highlighted node is the next one to execute. There will also be a set of control buttons in the toolbar for step executing the highlighted node (next), for running the flow to the end (resume), or for quitting the debug session (cancel).

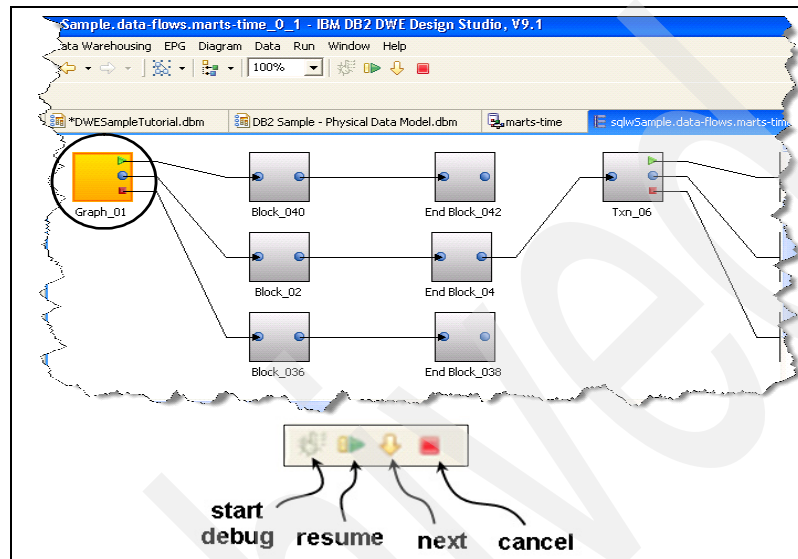


Figure 6-56 Debug session and debug controls

A number of the nodes in the graph represent code control structures, such as begin and end blocks, and begin and end transaction. They are not very interesting, and you can use the Next button to step through those. When there is something to execute, such as SQL, you will see a node with a name that indicates there is something to be executed, for example, the node named SQL_08, in Figure 6-57 on page 203, which is to be executed next. Then you can click the node and view the associated properties, which include a tab for Arguments and a tab for the SQL Code, which will allow you to see the SQL. You can see that SQL_08 will execute one statement to set the current schema. There may be many statements, and you may not even see the entire statement if it is long. If you click the statement in the Value cell, you can click the ellipsis button to see more of the statement. Then click the next button to execute this node. The results of the execution can be seen in the EPG Debut Output in the Console tab. By setting trace to content, you will see the SQL statement to be executed, as well as the result of the execution. Here you see that the set current schema executes successfully. Execution can continue in this step-wise fashion through the data flow EPG, you can resume execution, or you can cancel.

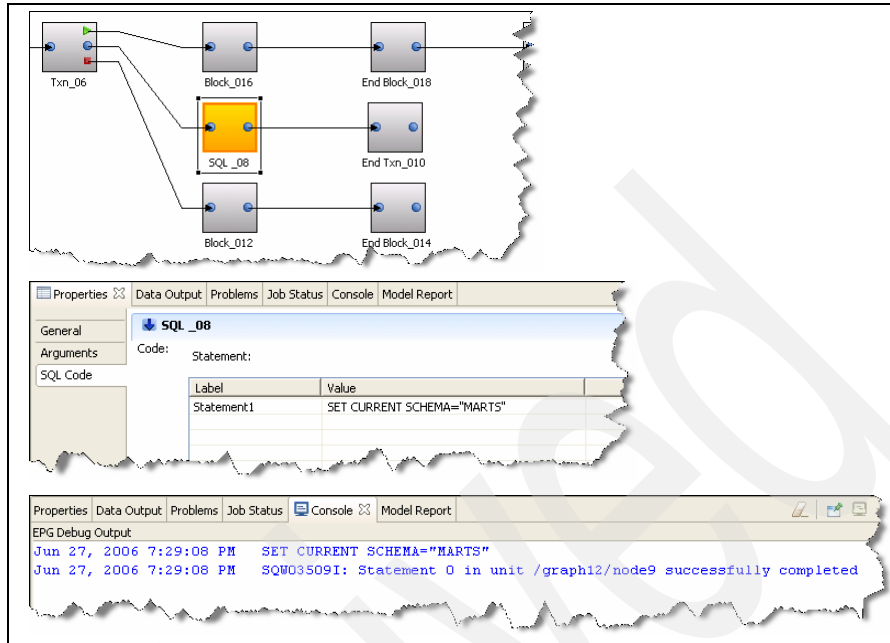


Figure 6-57 Stepping through a data flow EPG

6.2.7 A complete data flow

After the tour through the data flow components of the DWE Design Studio, let us pull it all together and see how a complete data flow looks. This data flow comes from lesson 4 of the DWE tutorial, and loads a data mart fact table from other data warehouse tables. The data flow name is marts-fact, and an overview graph is shown in Figure 6-58. It is at a 50% zoom level to show the overall flow. It is too small and detailed to read, but reading it is not necessary — it is just to help visualize an example of a complete data flow.

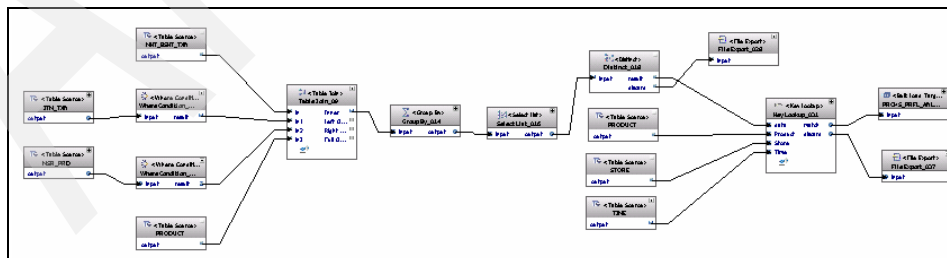


Figure 6-58 Marts-fact data flow from the DWE tutorial

The first part of marts-fact, depicted in Figure 6-59, joins four table sources from the lower, detailed level of the data warehouse. Two of the sources have filters applied.

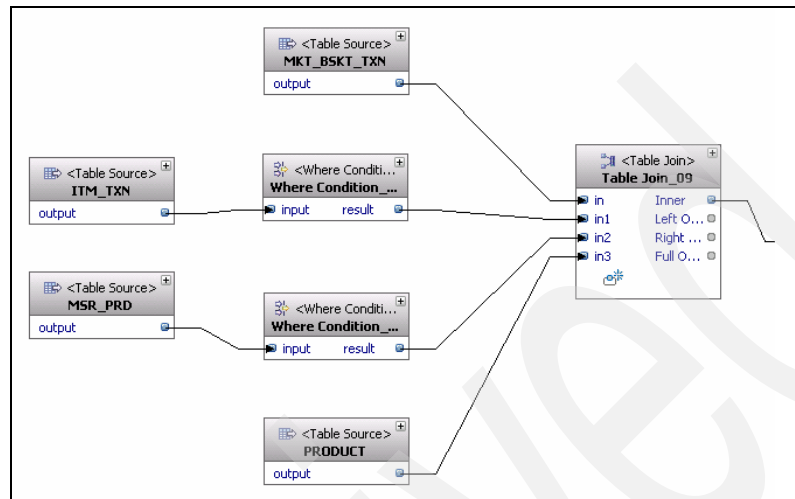


Figure 6-59 Marts-fact data flow part 1 - extract from data warehouse

In part two of the dataflow (Figure 6-60) the output of the join is aggregated with the *group by* operator, transformed by the *select list* operator, and the duplicates are removed with the *distinct* operator. Duplicates will be discarded to a file.

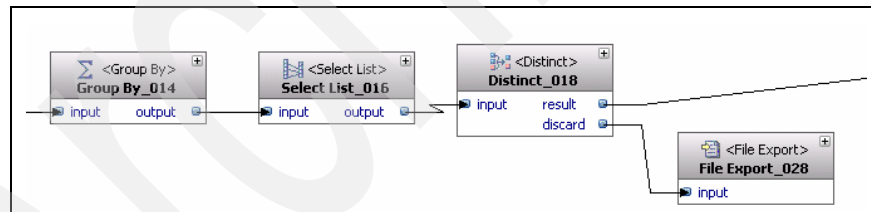


Figure 6-60 Marts-fact dataflow part 2 - process

The last part of the data flow (Figure 6-61) uses the *key lookup* operator to ensure that all of the data flowing into the data port has valid key matches in the dimension tables PRODUCT, STORE, and TIME. Rows that have valid entries in the dimension tables will be bulk loaded into the star-schema fact table, while rows that do not match will be placed in a file.

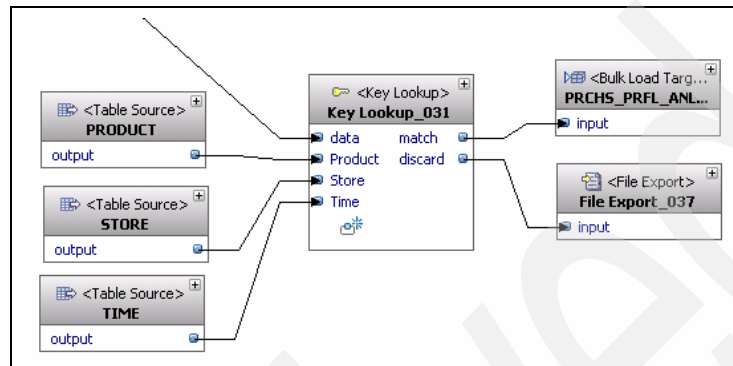


Figure 6-61 Marts-fact dataflow part 3 - load

6.3 Developing control flows

In this section we discuss in more detail the process of developing SQW control flows using the DWE Design Studio. We discuss the control flow operators, flow validation, code generation, testing, and debugging. For more information see the DB2 and DWE Information Center or the tutorial that comes with DB2 Data Warehouse Edition V9.1. The tutorial contains hands-on exercises to develop SQW flows.

6.3.1 Defining a control flow

A control flow model sequences one or more data flows or mining flows and integrates other kinds of data processing activities. Control flows form the basis of what is deployed to, and executed in, the DWE runtime environment. You cannot deploy data flows or mining flows directly. They have to be included in a control flow.

The DWE Design Studio provides a graphical editor with an intuitive capability to visualize and design control flows. Graphical operators model various SQW and data processing activities. By arranging these operators in a canvas work area, connecting them, and defining their properties you can create work flow models

that define the sequence of execution of the activities. Figure 6-62 depicts a simple control flow that sequences two data flows. When *Data Flow_02* finishes successfully, *Data Flow_03* will be executed. If either data flow fails, an e-mail operator will be executed to send a notification to an administrator.

A control flow, as in data flows, consists of operators, ports, and connectors. Figure 6-62 shows a data flow with six operators. There is the always present start operator, two data flow operators, two e-mail notification operators, and an end operator. Each operator has a set of properties that define the specific behavior of that operator.

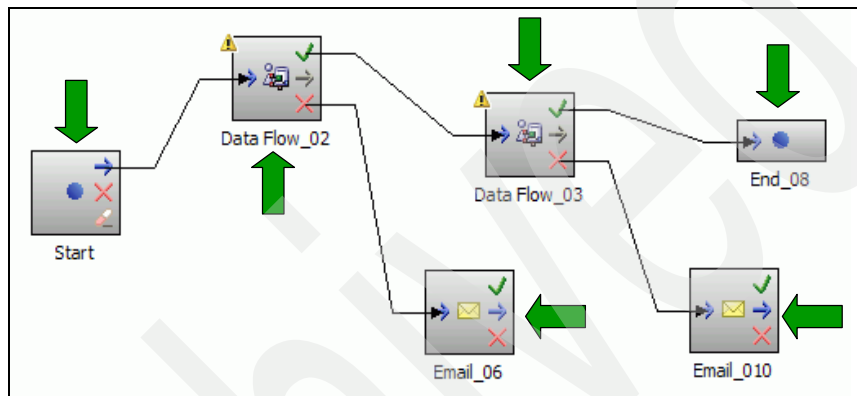


Figure 6-62 A simple control flow showing the operators

Control flow operators have ports that define the entry and exit points of the operator, which are circled in Figure 6-63. With the exception of the start and end operators, all control flow operators have one input port and three output ports. The completion status of the operator determines which output path is taken. Unlike the ports of a data flow operator, the ports of a control flow operator have no properties that can be set.

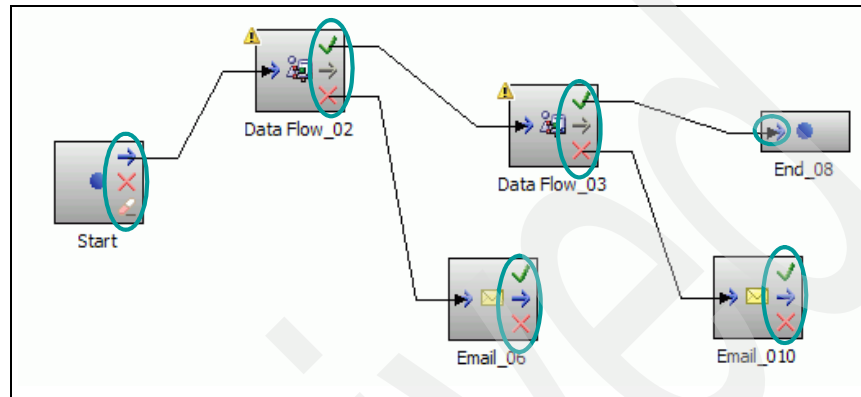


Figure 6-63 A simple control flow showing the ports

Figure 6-64 on page 208 shows the kinds of ports that are used for control flow operators. There is only one start operator and it has one input port and three output ports. The activity connected to the start process port will be the first activity to execute in the control flow. The process on-failure port branch will be taken if any operator completes unsuccessfully and after the completion of any activities connected to the on-failure branch of the failing operator. The cleanup process branch will be taken after any terminal point in the control flow is reached even if it is from any on-failure branch. The end operator represents the terminal point of any branch and therefore only has one input port. There may be many end operators but they are optional. Any operator that has no output ports connected has an implicit end operator.

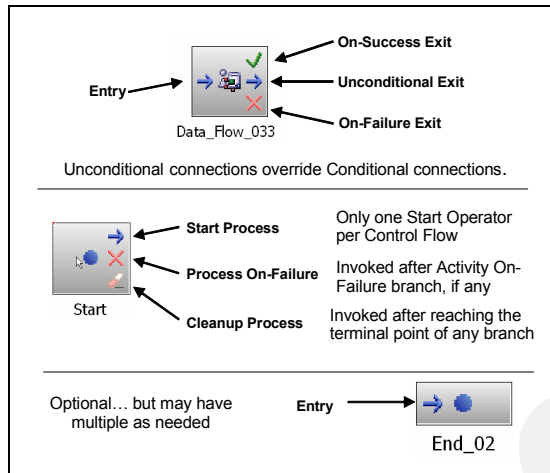


Figure 6-64 Control flow operator port details

Most control flow operators have one input port and three output ports. The input port represents the entry point. The output ports represent conditional branches that may be taken after the completion of the activity, and depends on the completion status. The on-success port will be taken if the activity completes successfully. The on-failure port will be taken if the activity does not complete successfully. The unconditional port will always be taken regardless of the completion status and, as such, overrides the on-success and on-failure ports.

Control flow operator ports are connected by connection arrows, as seen in Figure 6-65. These connections direct the flow of processing from the completion of one activity to the next activity. For example, the connection from the on-failure port of *Data Flow_02* input port of *Email_06* will be the flow of execution if *Data Flow_02* does not complete successfully.

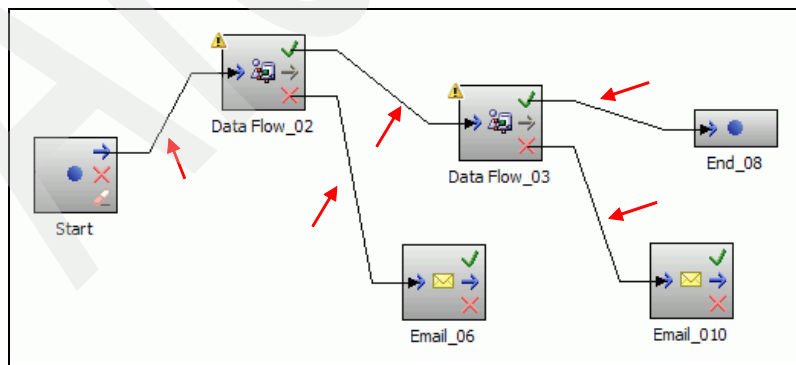


Figure 6-65 A simple control flow showing the connections

6.3.2 Control flow editor

Using the control flow editor is very similar to using the data flow editor. There is a graphical editor specifically for control flows. It has a canvas the control flow is modeled on using the operators that are in the palette. The process is to drag an operator onto the canvas, draw a connection from an output port of the previous operator to the input port, and define the properties of the operator.

The standard components of the DWE Design Studio will be used, just the same as for developing data flows. See 6.2.2, “Data flow editor” on page 151, for more information.

Keep the following in mind as you develop control flows:

- ▶ Be familiar with the common functions in the DWE Design Studio:
 - Working with perspectives
 - Working with views
 - Using the Data Project Explore
 - Using the Database Explorer
 - Dragging and dropping from the Data Project Explore and palette to the canvas
- ▶ Be familiar with developing data flows.
- ▶ Orient the control flow from left to right. This makes a neater diagram, as the output ports are on the right side of an operator and the input ports are on the left side.
- ▶ Work with only a few operators at a time.
- ▶ Operators have properties that must be defined.
- ▶ Operators have on-success and on-failure ports for connection purposes.
- ▶ Control flows use the same validation features as data flows.
- ▶ Think of simple data processing logic rules for success and failure conditional paths.
- ▶ Validate, generate code, and test each data flow before testing in a control flow.

6.3.3 Control flow operators

Control flow operators represent some type of data-processing activity to be executed in the sequence of the control flow. Operators are graphical objects that are dragged from the palette and dropped onto the editor canvas and form the nodes of the control flow sequence. An example of the palette is depicted in

Figure 6-66 on page 211. Each operator represents a specific type of activity and has properties to define that activity:

- ▶ SQW flow operators
 - Data flow operator
 - Mining flow operator
- ▶ Command operators
 - DB2 shell scripts
 - DB2 scripts
 - FTP
 - Executables
- ▶ Control operators
 - Start
 - End
 - File wait
 - Iterator
- ▶ Notification operators
 - E-mail notification
- ▶ DataStage operators
 - DataStage parallel job
 - DataStage job sequence

DataStage operators are discussed in 6.6, “Integrating with IBM WebSphere DataStage” on page 229.

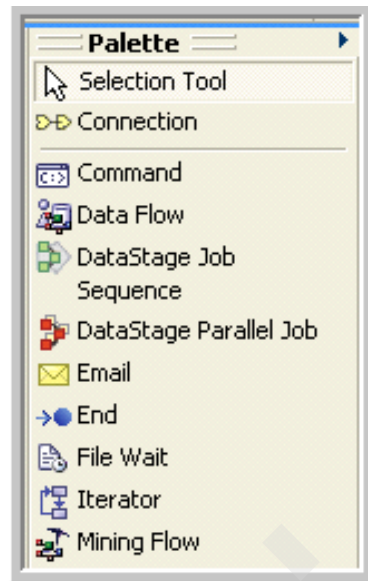


Figure 6-66 Control flow palette

A control is typically developed after the data flows and mining flows. However, a model of the control flow can be developed to document the expected overall flow by adding the various expected operators to the canvas, but only specifying the label and description properties. When you save the control flow, it will have errors, but that is fine because you will use this only as a working document as you drill down into the development. See an example in Figure 6-67.

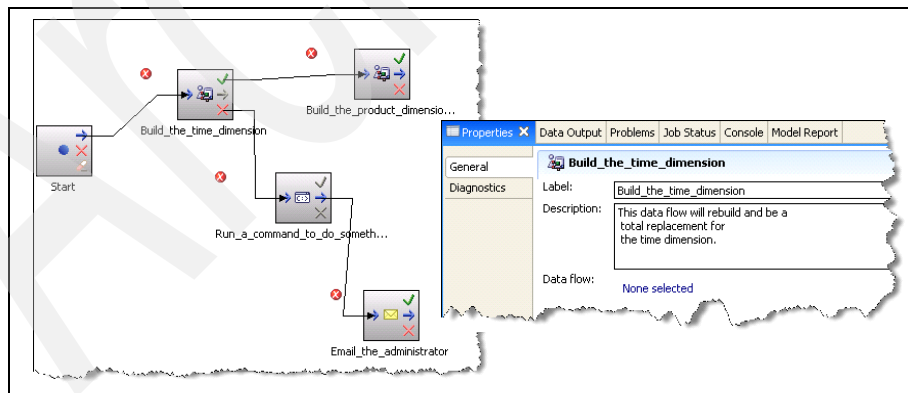


Figure 6-67 Using a control flow to document the overall flow design

SQW flow operators

The SQW flow operators represent the SQW objects, data flows, and mining flows that are developed within DWE SQW. These flows are developed using the appropriate DWE Design Studio editor and must be available in the same data warehouse project folder as the control flow.

Data flow operator

A data flow operator represents a data flow in the control flow sequence. The data flow must exist in the same data warehouse project. Drag the data flow operator onto the canvas, connect an output port from the previous operator in the sequence, and define the properties. The properties consist of a pointer to the data flow and logging/tracing information. An example is depicted in Figure 6-68.

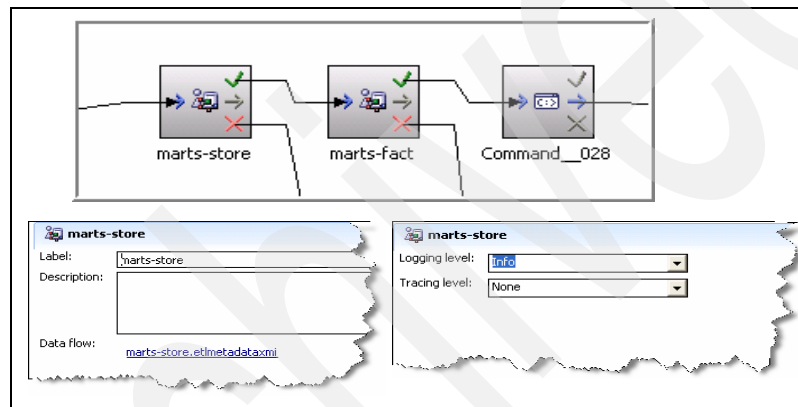


Figure 6-68 Data flow operator

Mining flow operator

A mining flow operator represents a mining flow in the control flow sequence. The mining flow must exist in the same data warehouse project. Drag the mining flow operator onto the canvas, connect an output port from the previous operator in the sequence, and define the properties. The properties consist of a pointer to the mining flow and logging/tracing information. An example is depicted in Figure 6-69.

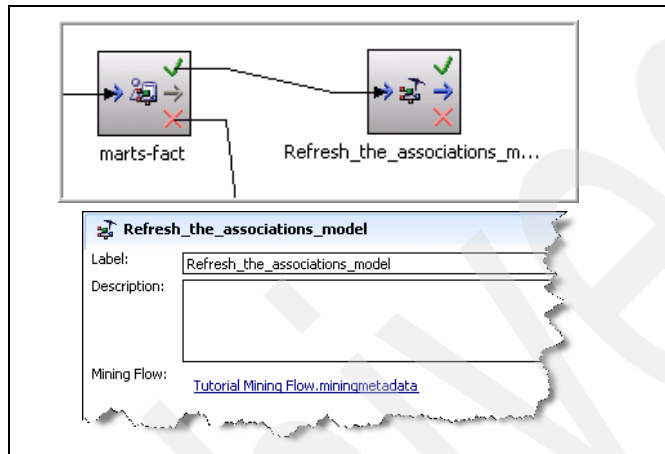


Figure 6-69 Mining flow operator

The command that is invoked with the command operator must end with an exit value. The command operator fails when a DB2 SQL script, a DB2 shell, or an executable is run and the result is an error level is greater than zero. Keep in mind that when the command operator fails, the next operator that runs is the one that is connected to either the on-failure or the unconditional port of the command operator. If this is not what you want, ensure that the script, shell, or executable handles error levels greater than zero and sets the exit value to zero. Setting the exit value to zero means that the command operator was successful.

Command operator

The command object is used to execute batch code that can be invoked via some type of command-line interface, but we do not recommend executing commands that prompt for user input. Commands can be operating system scripts, DB2 scripts, executable programs, or the FTP command, and all are supported by the same command operator. The type of command is a property of the command operator.

Figure 6-70 is a control flow that contains examples of the command operator to export data to a file, copy the file to a specific location, zip the file, and FTP it to a remote location. Refer to Figure 6-70 during the discussions in the following sections regarding the flow command operators.

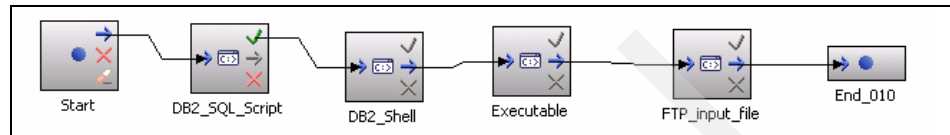


Figure 6-70 Control flow command operators

DB2 SQL script

You can use the command operator to execute a DB2 script that contains DB2 SQL and commands that the DB2 engine will execute. These would be customized scripts that you may have already developed to extend the capabilities of SQW, such as performing backups or executing RUNSTATS.

To define a command operator for DB2 SQL scripts, set the command type property to *DB2 SQL Script*. The property page changes to reflect the properties appropriate for this type, as seen in Figure 6-71. The location of the script and the database to execute the script must be set. In the DB2 SQL script property page we are executing a DB2 SQL script named `c:\temp\scripts\sqllexport.sql`, and the database connection to use is DB2SAMP.

The figure displays four property pages for control flow command operators:

- DB2_SQL_Script:**
 - Label: DB2_SQL_Script
 - Description: A simple DB2 script to export the STAFF table to a file.
 - Command type: DB2 SQL Script
 - SQL script location: C:\temp\scripts\sqllexport.sql
 - DB2 connection: DB2SAMP
- DB2_Shell:**
 - Label: DB2_Shell
 - Description: A simple windows bat file to delete old files from the ftp directory and copy the exported file to the ftp directory.
 - Command type: DB2 Shell
 - Executable location: C:\temp\scripts\copystaff.bat
 - Executable arguments: (empty)
- Executable:**
 - Label: Executable
 - Description: execute gzip to zip up the ftp file
 - Command type: Executable
 - Executable location: c:\gzip\gzip
 - Executable arguments: c:\temp\ftp\staff_ftp.txt
- FTP_input_file:**
 - Label: FTP_input_file
 - Description: ftp the zipped file
 - Command type: FTP
 - Remote host: Host_machine
 - Remote file: staff.txt.gz
 - Local file: C:\temp\ftp\staff_ftp.txt.gz
 - Transfer direction: Put
 - Mode: Binary

Figure 6-71 Control flow command operator properties

DB2 shell scripts

We can use the command operator to execute operating system scripts. These scripts can contain any command that can be entered at a DB2 command prompt. This is the way to use the multitude of operating system commands and utilities that are available.

To define a command operator for DB2 shell scripts, set the command type property to *DB2 Shell*. The property page changes to reflect the properties appropriate for this type, as seen in Figure 6-71. The location of the script and its arguments must be set. In the DB2 Shell property page, we are executing the Windows bat script called `c:\temp\scripts\copystaff.bat`, which has no arguments.

Executables

We can use the command operator to execute executable programs. These may be your own custom-developed programs or may be packaged utilities and programs. These programs should not prompt for any manual input.

To define a command operator for executables, set the command type property to *Executable*. The property page changes to reflect the properties appropriate for this type, as seen in Figure 6-71 on page 215. The location of the executable program and its arguments must be set. In the Executable property page, we execute the gzip utility from `c:\gzip\gzip.exe` with the argument for the file to compress, which is `c:\temp\ftp\staff_ftp.txt`.

FTP

We can use the command operator to use FTP to move files between the local system and an FTP server. We can get files from the FTP server or put files to the FTP server in both binary and ASCII modes.

To define a command operator for FTP, set the command type property to FTP. The property page changes to reflect the properties appropriate for this type, as seen in Figure 6-71 on page 215. The location of the FTP server is defined in, and selected from, a Remote Resource dialog. The resource definition includes the location of the FTP server and the credentials for login. Other properties include the location of the local and the remote files, the transfer direction, and the transfer mode. In the FTP property page, we transfer the local file, `c:\temp\staff_ftp_txt.gz`, in binary mode as `staff_txt.gz` to the host `Host_machine`.

Control operators

In addition to the conditional output ports of control flow operators, there are specific operators for altering the flow of processing. We have already discussed the start and end operators. See Figure 6-64 on page 208 for a summary of the operator conditional ports, the start operator, and the end operator. In this section we discuss the file wait operator and the iterator operator.

File wait operator

The file wait operator can pause the control flow execution for a period of time while the system checks for the existence or non-existence of a specified file. You can specify the amount of time to allocate to this operator.

In addition to the obvious use of waiting for a file to arrive to be processed, the file wait operator is very useful for coordinating activities between executing processes or organizations by using small files as flags or semaphores. Another organization or another process could place a file in a certain directory to reflect a certain status. A control flow can check the existence of this file to determine what it should do.

In the example shown in Figure 6-72 we have a file wait operator that will wait for the c:\temp\myfile to appear. The operator will check every five minutes for up to two hours. If the file is found during that time the on-success branch will be taken. Otherwise, after two hours, the operator will fail and take the on-failure branch.

Attention: The two times are specified in seconds. If the *check for* property is set to zero, then there will be just a one-time check for the file.

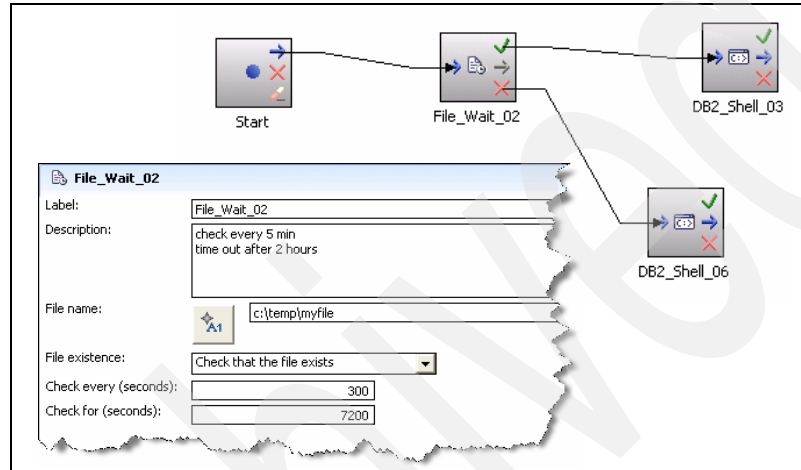


Figure 6-72 File wait operator

Iterator operator

The iterator operator is used for looping over a set of control flow operators, causing the operators to be repeated until certain conditions are met. There are three types of iteration supported:

- ▶ Repeat loop a fixed number of times.
This number of times to execute the loop is based on integer values for the starting value, the step increment, and the end value. The actual value will be available in the defined iteration variable.
- ▶ Repeat loop for each delimited data item in a file.
This will read a delimited data file and will loop once for each value provided between the defined delimiter, including whitespace. A comma-delimited file with the values 1, 6, 18, 22 will loop four times and in order, passing the current value via the defined iteration variable.

- Repeat loop for each file in a directory.

This loop technique will read the names of all the files in a directory and loop once for each, making the file name available in the defined iteration variable.

The iterator uses the iteration variable for holding the actual value of the specific iterator type. This variable can be used in any variable field in the underlying set of operators, even referenced in an underlying data flow. See 6.4, “Variables in data flows and control flows” on page 222, for more information about using variables in data flows and control flows.

When adding an iterator operator onto the canvas, you will actually get two operators on the canvas, the iterator operator itself and an end iterator operator, as shown in Figure 6-73. Within the loop there is one data flow operator. Of course, you can have as many operators within the loop as needed. The properties of the iterator operator control the looping. The iterator will read file names from a comma-delimited file, and loop once for each file name executing the data flow each time. The file name value will also be available via the iteration variable, `${dwh_update_files/dwh_itm_txn_update_file}`. The data flow uses this file name to load the proper file.

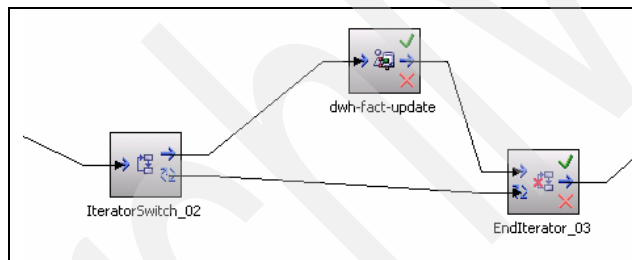


Figure 6-73 Control flow iterator operator

Notification operators

Notification operators are used as an alert for some type of important event that has taken place in the control flow. This is typically used for notifying an administrator that some error has occurred, but could be an event such as the successful completion of the control flow. The control flow uses an e-mail operator to accomplish notification.

Email operator

The email operator will simply send an e-mail to a specified e-mail address with the provided message. Figure 6-74 shows a control flow with several email operators attached to the on-failure ports of data flow operators, which will send an e-mail whenever a data flow encounters an error. There are properties for the sender e-mail address, the recipient e-mail address, the subject text, and the message text. The smtp server is not specified in the operator, but is defined in the runtime environment as a system resource. E-mail will not be sent when executing a control flow test within the DWE Design Studio.

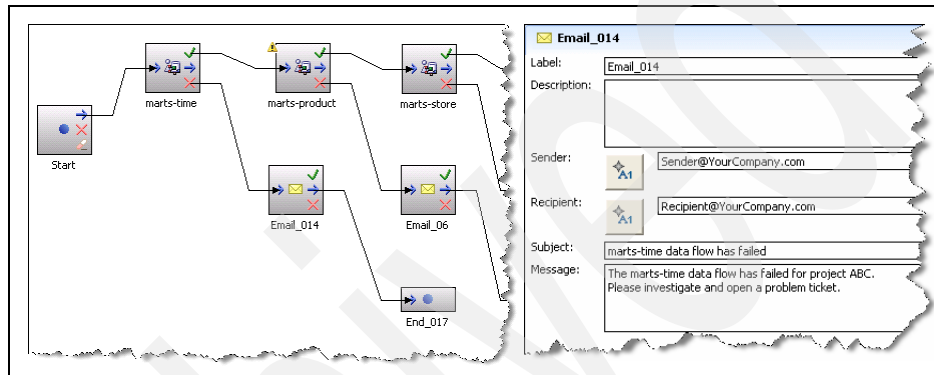


Figure 6-74 Control flow email operator

DataStage operators

Jobs for the IBM WebSphere DataStage product can be integrated into, and invoked from, a DWE SQW control flow. There is one operator for invoking a DataStage parallel job and one for invoking a DataStage job sequence. These jobs will execute at the DataStage server. See 6.6, “Integrating with IBM WebSphere DataStage” on page 229, for more information.

6.3.4 Validation and code generation

A control flow is simply a model that represents the sequence of the activities you want to perform. You are not building code in a control flow, but rather defining metadata. However, at some point in time you will want to execute the control flow, which does require something that can be executed, so you must validate the data flow and generate code.

Validation and code generation can be explicitly invoked via the Control Flow menu item in the menu bar. Any time a control flow is saved, validation is implicitly invoked. Invoking code generation also causes an implicit validation to occur.

Validation

Validation is the process that the DWE Design Studio uses to examine the metadata of a control flow to see if it is correct. If it detects a problem, it will flag the problem and list it in the Problems view tab. If the problem is in an operator, there will be a visual notification icon in the upper left corner of each operator that has a problem. In addition to the validation of the control flow metadata, validation will also individually validate all of the data flows and subflows in a hierarchical manner. If you have a large control flow containing many data flows, expect that the validation will take a bit longer.

Figure 6-75 shows how to explicitly invoke a control flow validation from the Control Flow menu, a control flow with a data flow operator that has a warning, and a DB2 shell operator with an error as indicated by the symbols in the upper left corner of the operator icon. See 6.2.5, “Validation and code generation” on page 197, for more information. The diagnostic dialog shows that the DB2 shell operator is missing a file name in its properties. The warning for the data flow is within the data flow itself. You will have to open the data flow to determine this error. The properties page for the data flow operator has a link to the underlying dataflow.

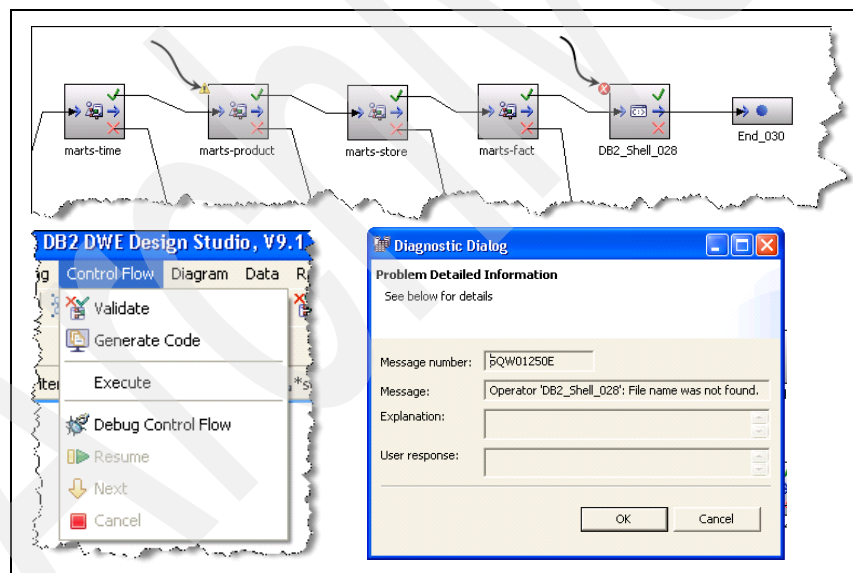


Figure 6-75 Control flow validation

We recommend that you validate and test all data flows and mining flows individually before using them in a control flow. This will make finding validation errors much easier.

Code generation

Once you have validated the metadata and corrected any problems, you can generate the code. You can explicitly invoke code generation via the Control Flow menu item. It is also implicitly invoked when you test execute a control flow. Code generation will examine the metadata of the control flow model, determine what is needed to execute, and generate the structured code that is needed to execute the sequence of operators. For data flows and mining flows, it will also generate the code for those individual flows that will be included in the overall code. Therefore, expect that code generation for a large control flow will take a bit of time. When you explicitly invoke code generation, the generated code will be displayed in a DWE Design Studio text editor.

As with the data flow, the code is represented in an Execution Plan Graph (EPG). However, there is no special EPG viewer needed for a control flow because the control flow itself is a type of graphical EPG. See 6.2.5, “Validation and code generation” on page 197, for more details on code generation.

6.3.5 Testing and debugging a control flow

After a successful validation, control flows can be tested from within the DWE Design Studio without having to set up or deploy to a runtime environment. And, if needed, control flows can be debugged using the control flow debugger.

To test execute a control flow, first establish a connection to the databases of interest in the Database Explorer. Make sure that the control flow is open and has the focus. Then select the **Execute** item from the Data Flow menu. You will be prompted with the Flow Execution dialog, which is where you will provide information supporting the execution, such as the trace options, resource definitions, and set values for any variables. You can save these values in a run profile for future use. Be aware that a test execution will also implicitly invoke both validation and code generation, so it may take longer than you expect for the execution to start. Once the control flow has completed executing, you will be presented with the execution result showing the status of the execution and any error messages that may have been received.

To debug a control, select the **Debug Control Flow** item from the Control Flow menu and complete the Flow Execution dialog. Be aware that a test execution will also implicitly invoke both validation and code generation, so it may take longer than you expect for the debug session to start. The debug session will take place directly in the control flow editor instead of a separate EPG viewer, but operates just like the data flow operator. Figure 6-76 shows a control flow debug session. See 6.3.5, “Testing and debugging a control flow” on page 221, for more details on using and operating the debugger.

Attention: The control flow debugger will execute data flows as a black box, and will not step into the data flow EPG. It is expected that you will have already debugged the data flows individually.

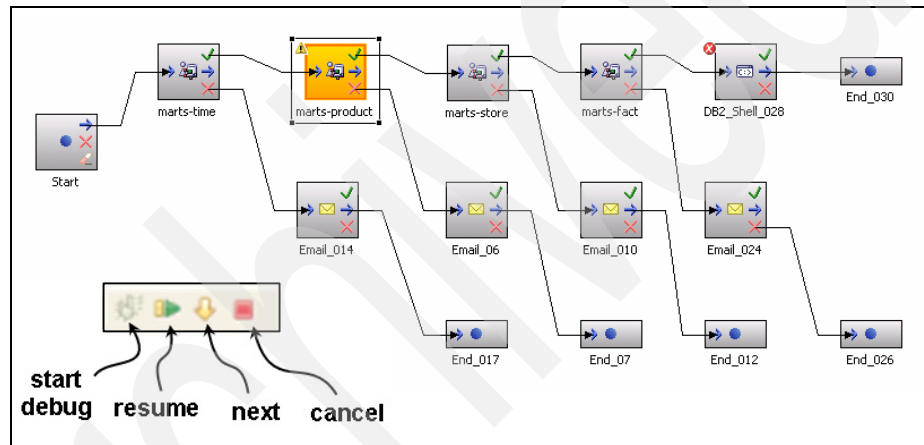


Figure 6-76 Control debugging session

6.4 Variables in data flows and control flows

A variable is a user-defined name that allows you to defer the definition of critical properties until a later phase in the life cycle of the application. Using variables provides you with an increased level of flexibility. Variables can be used for most properties of objects in data flows, mining flows, and control flows.

Variables are useful in many different data warehousing scenarios. For example:

- ▶ The designer does not know the names of specific database schemas or tables that will be used at runtime.
- ▶ The designer knows the file format that is required for an import operation but does not know the names of specific files that will be used.

- A data flow needs to be run against two different target databases.

Variable names are enclosed in parentheses and preceded by a dollar sign, for example, \$(variablename). Variables may also be concatenated with a constant string. For example, you may know the file name, but the directory may not be known at design time. You can use a variable for the directory concatenated with the file name, as in \$(myfiledirectory)myfile.txt.

The variables manager is used to manage, define, and select variables. The variables manager can be opened in two ways, as depicted in Figure 6-77. It can be opened from the Project Explorer by right-clicking the **Variables** folder, which will bring up the context menu, and selecting **Manage Variables**. It can also be opened from a property page. Every property that is eligible to be a variable will have an icon preceding the property field. Clicking the icon will bring up a selection menu that will allow you to select whether this property is to be a constant value or a variable. If you select variable, then a set of ellipses will appear at the end of the field. Selecting the ellipsis button will open the variables manager. Highlight the variable group and the variable that you want, then click the **Select** button to insert the variable name in the property field.

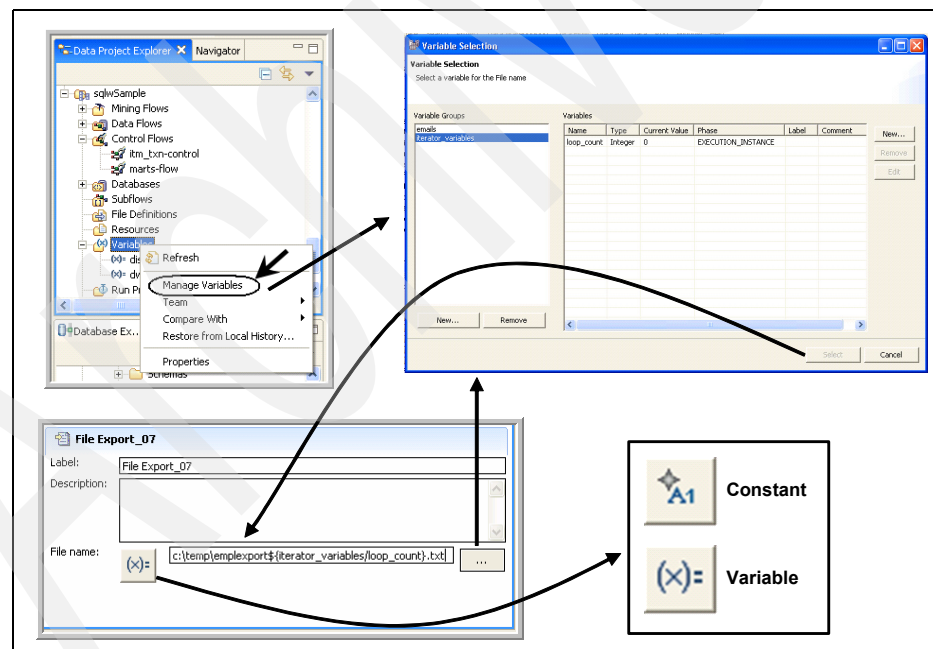


Figure 6-77 Variables manager

The variables manger is used to define, edit, remove, and select variables and variable groups. Variables are defined into user-defined groups, allowing a logical organization of the variables.

To define a variable using the variables manager, define a new group, or select an existing group, select the **New** button and use the variables information to define the variable name, variable type, initial value, and phase. This is depicted in Figure 6-78.

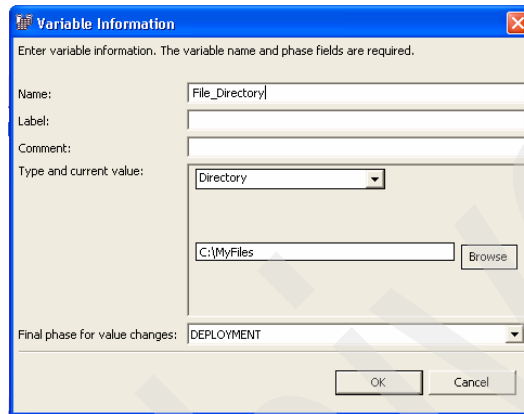
The image shows a 'Variable Information' dialog box with a blue title bar and a close button. The main area is light beige. At the top, it says 'Enter variable information. The variable name and phase fields are required.' Below this are several input fields: 'Name:' with the text 'File_Directory', 'Label:', 'Comment:', and 'Type and current value:' with a dropdown menu showing 'Directory'. Below the dropdown is a text box containing 'C:\MyFiles' and a 'Browse' button. At the bottom, there is a dropdown for 'Final phase for value changes:' set to 'DEPLOYMENT', and 'OK' and 'Cancel' buttons.

Figure 6-78 Variable definition

The allowed variable types, and how to set the initial value, are:

- ▶ *String*: Type a series of characters up to a maximum of 255.
- ▶ *Integer*: Type a whole number.
- ▶ *Boolean*: Check the check box for true or leave it unchecked for false.
- ▶ *File*: Type the name of a file or click **Browse** to select a file.
- ▶ *Directory*: Type the name of a directory or click **Browse** to select a directory.
- ▶ *SchemaName*: Type the schema name, which is case sensitive.
- ▶ *TableName*: Type the DB2 table name, which is case sensitive.
- ▶ *DBConnection*: Select a database connection from the drop-down list.
- ▶ *MachineResource*: Select a machine resource from the drop-down list.
- ▶ *DataStageServer*: Select the name of the DataStage server from the drop-down list.

There are a number of points in the data warehouse application life cycle at which a value can be set for a variable. Setting the final phase for the value

changes property of a variable defines the latest point in the life cycle that the value can be set at, after which the value can no longer be changed.

The phases that can be defined are:

- ▶ **DESIGN_TIME**: Values are set during design time and cannot be changed.
- ▶ **DEPLOYMENT_PREP**: This is the latest phase that applies to the DWE Design Studio. Values are set by the designer as the application is prepared for deployment and cannot be changed.
- ▶ **DEPLOYMENT**: This allows values to be set at the time an application is deployed to a runtime environment and cannot be changed.
- ▶ **RUNTIME**: This phase can be used for values that can change after deployment, but not for every execution. Once set, the value persists for every execution until modified.
- ▶ **EXECUTION_INSTANCE**: The value can be set for every execution. If manually starting the process, there will be a prompt for the value. If a scheduled process contains EXECUTION_INSTANCE variables, the values must be provided by a process configuration profile.

When performing a test execution of a data flow or a control flow in the DWE Design Studio, variables can be set or modified during the Flow Execution dialog, as in Figure 6-79. A run profile can also be set and reused by the Flow Execution dialog. In the runtime environment, the DWE Administration Console is used to manage the variables.

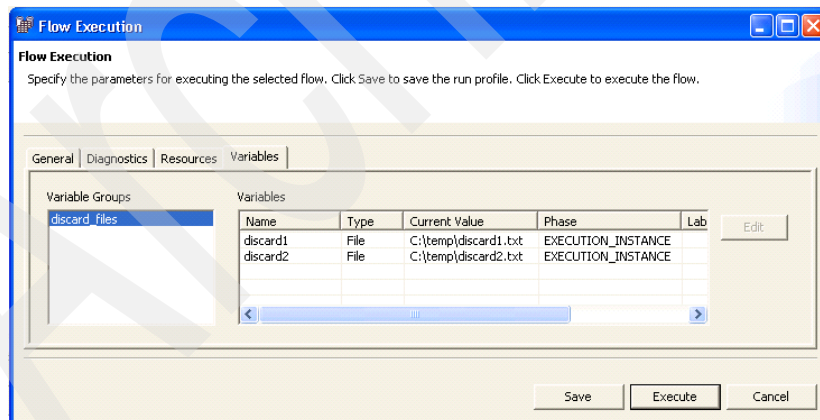


Figure 6-79 Setting variables for a test execution

6.5 Preparing for deployment

The DWE SQW warehouse application, as with any application, follows the usual development, test, and production cycle. So far in this chapter we have focused on the development of data flow and control flows and have tested them in an isolated development environment. In this section we look at the final step the designer will perform in the DWE Design Studio to prepare an application for deployment to a runtime environment.

The ultimate goal is to execute these warehousing applications in a production system. Before going to production, you want to be sure that the applications will work well with other applications and production-like data, and within the system infrastructure. This may be called system testing, quality assurance, or user acceptance testing. We want to be sure that, when the application is installed in the production system, problems will have already been corrected.

Moving the warehousing application from a developer's environment to some type of runtime environment, whether test or production, is called deployment. DWE SQW deployment is performed in two steps. First, the developer uses the DWE Design Studio to prepare and package the application for deployment resulting in a deployment file. This deployment file is then deployed, or installed, into a runtime environment for system testing. If there are problems, then the data flow or control flow has to be modified in the development environment, the DWE Design Studio, and then redeployed to the test runtime. Once successfully tested, the same deployment package can be installed into the production environment.

This section describes how to prepare a set of data flows and control flows for deployment, resulting in a deployment package. The actual installation, or deployment, into a runtime environment is discussed in Chapter 9, "Deploying and managing DWE solutions" on page 365.

The process of preparing an application for deployment uses wizards to define a warehouse application, define application profiles, generate code, and create a zip file containing the actual deployment package.

6.5.1 Defining warehouse applications

The term *warehouse application* has been used several times so far. As depicted in Figure 6-80, a warehouse application is simply a collection of control flows, which in turn are sequenced collections of data flows. This set of control flows will be packaged into a zip file called the deployment package. The contents of the entire deployment package are installed into a runtime environment as a unit.

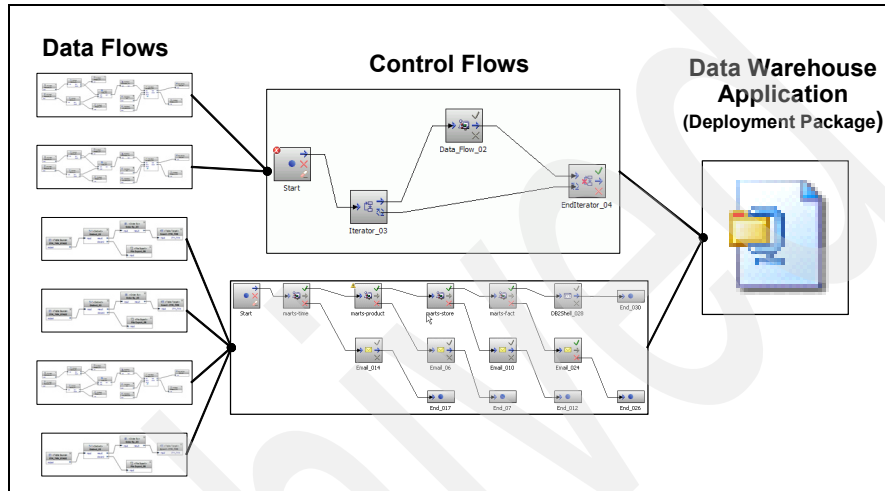


Figure 6-80 Relationship of data flows, control flows, and warehouse applications

6.5.2 Defining application profiles

A warehouse application is defined by creating a warehouse profile that contains configuration information about a deployable data warehouse application. The application profile is created by using the Application Profile Wizard.

The Application Profile wizard, depicted in Figure 6-81, guides you through selecting the control flows to include, mapping resource definitions, and setting the values of variables. The application profile is saved in the application profiles folder of the warehouse project. There may be multiple application profiles, hence multiple warehouse applications, for one warehouse project.

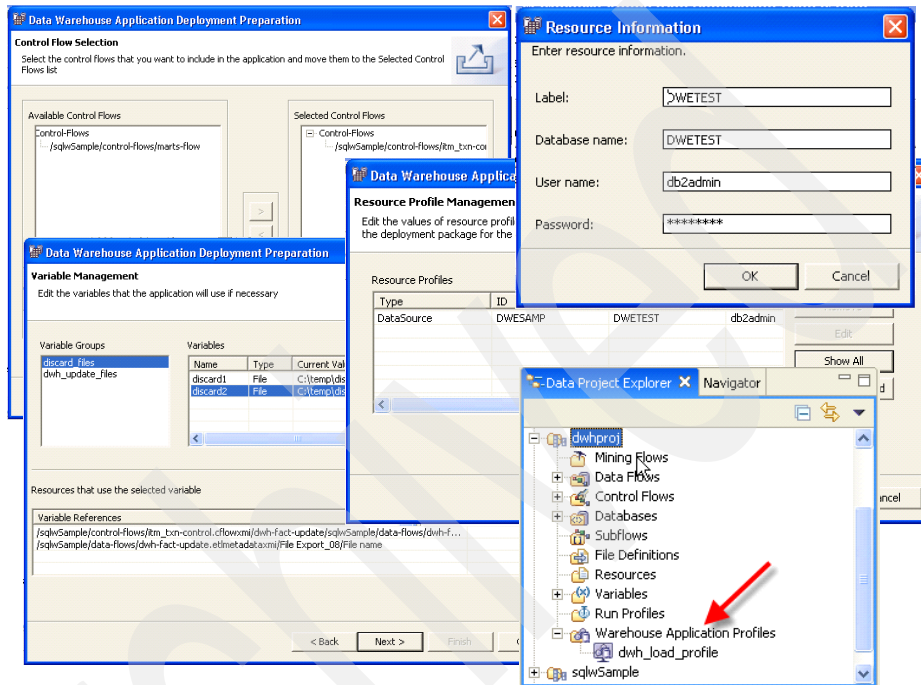


Figure 6-81 Application profile wizard

Resource mapping can also be set or modified at deployment time and by using the DWE Administration Console applied to database, FTP, and DataStage servers. As you develop in the DWE Design Studio, you might use resource names that are different from the runtime environments. For example, in development, you may use a database connection called DWESAMP, which is the development database. However, this might be named DWETEST and DWEPROD in test and production environments, respectively. You do not have to worry about the connection name, as you can map the database resource DWESAMP to DWETEST when deploying to the test environment, and similarly for deployment to production.

6.5.3 Generating code and packaging for deployment

The final function is to generate the code for the control flows included in the warehouse application as defined by the application profile. Code generation may be invoked as part of the application profile wizard or may be invoked separately via the application profile context menu.

Code will be generated for all of the control flows and data flows in the warehouse application, as depicted in Figure 6-82. This results in a number of XML-based files that contain the execution plan graphs (EPGs) and other metadata that are zipped into the deployment package and saved in the user-defined location. Having a large number of control flows and data flows included will affect the length of time that it takes to generate and package the code.

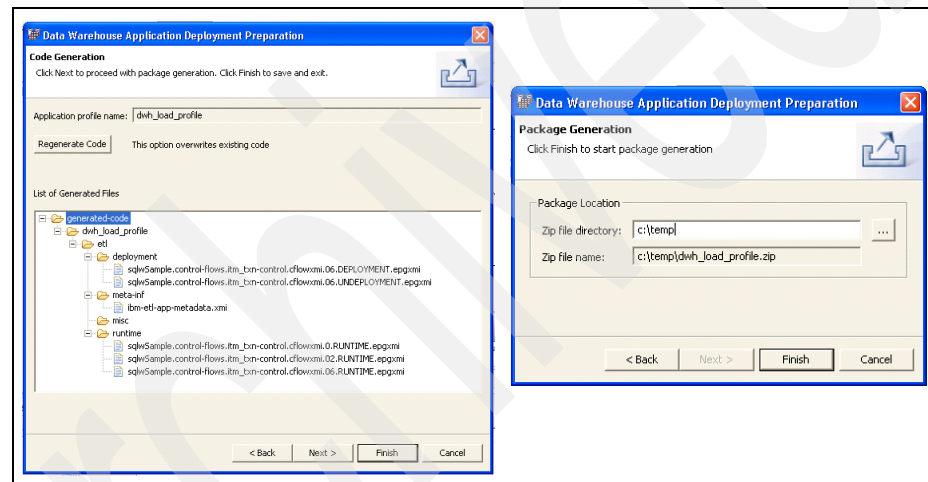


Figure 6-82 Code and package generation

The resulting deployment zip file is given to the administrator of the runtime environment for installation into that environment.

6.6 Integrating with IBM WebSphere DataStage

When it comes to data movement and transformation tasks, there are numerous ways to accomplish the same goal. Each client has different requirements, so the solution to a problem might be a single tool, a set of tools that work together in some way, or even something unique and custom-built. The adage *the right tool for the job* definitely applies to this topic.

So far in this chapter we have focused on one such tool that is provided in the DB2 Data Warehouse Edition, the SQL Warehousing Tool (SQW). In this section we take a look at the IBM enterprise ETL tool, IBM WebSphere DataStage, to understand:

- ▶ When, where, and why to use each type of tool
- ▶ How SQW and enterprise ETL tools complement each other
- ▶ How and why to integrate DataStage functionality into SQW

SQW is a strategic design and deployment component within DWE that leverages the SQL processing of DB2 to perform data movement and transformation. In contrast, IBM WebSphere DataStage uses its own high-performance parallel and scalable engine that can operate independently of the database vendor. It can produce limited database-specific SQL for processing that is completed within the database.

DataStage is optimized for integrating information from myriad sources inside and outside the enterprise while processing high volumes of data resulting in the population of the detail layer of the warehouse. DataStage has a rich library of pre-built transformations to make the task faster and easier. SQW is optimized exclusively for DWE, using a set of SQL operations to create and maintain data structures, based primarily on the data in the detail layer. These structures are typically used for analytic applications found in BI, and comprise part of the business intelligence framework, as discussed in Chapter 1, “Information warehousing for business insight” on page 1.

Despite their differences, DataStage and SQW functionality sometimes overlap and clients might find that one or both tools suit their requirements. In fact, these two products can complement each other in an enterprise data warehousing environment based on IBM DB2.

6.6.1 Overview of IBM WebSphere DataStage

IBM acquired Ascential Software in 2005. Their DataStage product is now the strategic enterprise ETL component for IBM WebSphere Information Integration. Embracing the concepts of service-oriented architecture (SOA), WebSphere Information Integration delivers multiple discrete services that hide the complexities of distributed configurations.

In this way, services can focus on the functionality while the individual WebSphere components can be used to compose intricate tasks without custom programming. This design supports the configuration of integration jobs that match a wide variety of client environments and tiered architectures. For example, WebSphere DataStage supports the collection, transformation, and distribution of large volumes of data, with data structures ranging from simple to highly complex.

The system manages data arriving in real time as well as data received on a periodic or scheduled basis. Companies can solve large-scale business problems through high-performance processing of massive data volumes. By leveraging the parallel processing capabilities of multiprocessor hardware platforms, DataStage can scale to satisfy the demands of ever-growing data volumes, stringent real-time requirements, and ever-shrinking batch windows.

WebSphere DataStage has the functionality, flexibility, and scalability required to perform the following demanding data integration tasks:

- ▶ Integrate data from the widest range of enterprise and external data sources.
- ▶ Incorporate data validation rules.
- ▶ Process and transform large volumes of data using scalable parallel processing.
- ▶ Handle very complex transformations.
- ▶ Manage multiple integration processes.
- ▶ Provide direct connectivity to the enterprise application as sources or targets.
- ▶ Leverage metadata for analysis and maintenance.
- ▶ Operate in batch, real time, or as a Web service.

In its simplest form, DataStage performs data movement and transformation from source systems to target systems in batch and in real time. The data source may include indexed files, sequential files, relational database, archives, external data sources, enterprise application, and message queues. The transformations may include:

- ▶ String and numeric formatting and data type conversions from source to target systems.
- ▶ Business derivation and calculations performed by applying business rules and algorithms to the data. Examples range from straightforward currency conversions to more complex profit calculations.
- ▶ Reference data checks and enforcement to validate client or product identifiers. This technique is used in building a normalized data warehouse.
- ▶ Conversion of reference data from disparate sources to a common reference set, creating consistency across these systems. This technique is used to create a master data set (or conformed dimensions) for data involving products, clients, suppliers, and employees.
- ▶ Aggregations for reporting and analytics.
- ▶ Creation of analytical or reporting databases, such as data marts or multidimensional cubes. This process involves denormalizing data into

structures such as star or snowflake schemas to improve performance and ease of use for business users.

These transformations require the movement of data from source systems into a data warehouse. However, more sophisticated data architectures may include building data marts from the data warehouse. In the latter case, WebSphere DataStage would treat the data warehouse as the source system and transform that data into a data mart as the target system — usually with localized, subset data such as clients, products, and geographic territories.

DataStage delivers core capabilities, all of which are necessary for successful data transformation within any enterprise data integration project, such as:

- ▶ Connectivity to a wide range of mainframe and enterprise applications, databases, and external information sources enables every critical enterprise data asset to be leveraged. A comprehensive, intrinsic, pre-built library of over 300 functions reduces development time and learning curves, increases accuracy and reliability, and provides reliable documentation that lowers maintenance costs.
- ▶ Maximum throughput from any hardware investment allows the completion of bulk tasks within the smallest batch windows and the highest volumes of continuous, event-based transformations using a parallel, high-performance processing architecture.
- ▶ Enterprise-class capabilities for development, deployment, maintenance, and high availability reduce on-going administration and implementation risk and deliver results more quickly than hand-coded applications.

For more details about WebSphere DataStage features, see the Web site:

<http://ibm.ascential.com/products/datastage.html>

6.6.2 Key differences between SQW and DataStage

It is clear that SQW and DataStage are different in some key ways. These are briefly summarized in Table 6-2.

Table 6-2 DataStage and SQW

| Characteristic feature | SQL Warehousing Tool | DataStage Enterprise Edition |
|------------------------|----------------------|--|
| Target market | DB2 clients. | Data integration (not necessarily in a BI or warehousing context). |

| Characteristic feature | SQL Warehousing Tool | DataStage Enterprise Edition |
|-------------------------------|--|---|
| Product bundling | Component of DWE, an integrated warehousing platform based on DB2. | Independent product; no RDBMS dependency; also sold as part of a larger Data Integration Suite. |
| Supported sources and targets | DB2 objects in the data warehouse. Secondly, limited support for JDBC data sources (relational tables in DB2 and other databases), SQL replication, flat files. | Very extensive set of database and file types supported. See http://ibm.ascential.com for more details. |
| Data transformations | SQL-based operators, with specialized data warehousing functions and integrated mining steps. | Extensible library of over 100 pre-built ETL components. Users can create new components quickly and easily. |
| Code generation and execution | Generated SQL optimized for DB2; native DB2 utility loads. | Can generate and execute independently of the database using the scalable parallel engine. Also generates limited database-specific SQL code. |
| Scheduling and administration | Deployment to WebSphere Application Server and subsequent control via a Web client console. | DataStage client tools support scheduling and administration tasks via a DataStage server and include a graphical sequencing and scheduling tool, which can also be linked to a third-party scheduling tool. Services can be shared and called from any application using Web Service, Java Messaging Service (JMS), or Enterprise Java Beans (EJB™). |
| Performance | Native code generation provides good throughput for most transformations. Parallelism depends on the DB2 DPF features or native DB2 parallelism; no knowledge of parallelism needed during design. | Parallel engine provides almost linear performance scalability. You can design and deploy jobs in parallel and change the degree of parallelism dynamically. |

| Characteristic feature | SQL Warehousing Tool | DataStage Enterprise Edition |
|---|---|--|
| Target data currency | Real-time support provided by native DB2 table functions over MQ Series queues (read/write queue messages with SQL). | |
| Integration with other BI modeling tasks | Rational Data Architect data models, OLAP metadata, and data mining flows all accessible within same interface; live JDBC database connections for reverse engineering and data sampling. Integration with versioning and teaming tools such as PVS and Rational Clearcase. | Full integration with a variety of modeling tools using bridges (called MetaBrokers) to the MetaStage repository. |
| Support for data profiling, cleansing, and analysis tools | Not supported. | Suite of compatible DataStage tools, including QualityStage and ProfileStage, for data profiling, cleansing, and analysis. |

To summarize, SQW is optimized for heavy workloads inside a DB2 data warehouse, especially work that occurs above the detail data layer and work that requires specialized functionality for maintaining warehousing, data mining, and OLAP data structures. As an enterprise ETL system, DataStage is optimized for heavy workloads outside the context of DB2 (or any database), especially work that builds the detail data layer by extracting large data sets from disparate data sources. DataStage does most of the work *outside* of the database, using a highly scalable parallel engine and file system. SQW leverages the DB2 engine to do all of the work *inside* the database.

A common scenario for DB2 clients, particularly large clients, is to use a standard ETL tool such as DataStage to handle the processing required to extract, transform, and load the atomic level data into the data warehouse. Then they use SQL-based processing to build and maintain analytics in the warehouse, such as aggregates, data mining data sets, and so on. The SQL-based processing takes advantage of the DB2 in-database processing power rather than extracting from the database, processing, and loading back to the database.

Developing SQL scripts and stored procedures is an exercise in hand-coding. SQW brings the same type of productivity benefits to developing *in-database*

data movement and transformation routines as standard ETL tools do for *outside-database* routines.

If any of your data warehouse processing relies heavily on hand-coded SQL scripts, procedures, or applications, SQW would be a good fit and could replace the custom code with data flows or have a combination of SQW data flows and custom code integrated into control flows. Also, smaller DB2 shops that cannot yet justify the cost of a full-scale ETL product could use SQW for their data movement and transformation flows.

6.6.3 Integrating DataStage and SQW

For those situations where it is desired to use DataStage and SQW together, SQW has built-in integration points. Although DataStage and SQW are sold separately and have their own development interface, the resulting jobs of both tools can be integrated and managed together from either DataStage or DWE runtime components.

Integrating DataStage jobs into SQW

Integrating DataStage jobs into SQW allows you to use the capabilities of the DWE runtime environment to manage and schedule the execution of both DataStage jobs and SQW flows. DataStage jobs will be submitted by the DWE runtime code to a DataStage server for execution while the SQW flows are submitted to the DB2 execution database for processing.

There are two ways to integrate DataStage jobs into SQW:

- ▶ Embed a DataStage parallel job into a data flow as a subflow. This is accomplished by first exporting a DataStage job in XML format and then importing it into the DWE Design Studio. This subflow can then be used as an operator in one or more SQW data flows and connected directly to SQL operators, thereby becoming part of the data flow.
- ▶ Embed DataStage jobs into control flows, thereby sequencing DataStage jobs along with data flows, mining flows, and other operators supported by SQW control flows. SQW control flows have operators for DataStage parallel jobs and DataStage job sequences, which are put onto the canvas, and then properties that point to the DataStage server and the particular job are defined.

To support the integration of DataStage jobs, SQW provides a DataStage Server view and a Job Status view. These views provide information about the DataStage servers that are available and the particular jobs they are running.

Integrating SQW flows into DataStage

Integrating SQW flows into DataStage jobs allows you to use the capabilities of DataStage to manage and schedule the execution of both DataStage jobs and SQW flows. In this case, control flows and the DWE runtime environment are not used. Instead, data flows are transformed into SQL scripts that are embedded as DataStage command stages in a DataStage job. The resulting job can then be used in DataStage job sequences.

6.6.4 Conclusion

DWE and WebSphere DataStage have separate and strong value propositions for data warehousing and analytics, and data integration. While on the surface there seems to be a bit of overlap in functionality, each tool has its particular strengths. DataStage is extremely strong in the standard, outside the database, ETL processing, while SQW excels in leveraging the DB2 relational engine for inside the database processing.

Many clients are using a combination of ETL and SQL-based processing for a complete end-to-end data movement and transformation solution. Having the ability to integrate DataStage and SQW processing allows these two products to be used in a very complementary way, while allowing each tool to do what it is best at — as earlier stated, the *right tool for the job*.

WebSphere Information Integration provides the strategic direction for enterprise data integration and, in particular, all enterprise clients can benefit from WebSphere DataStage. DB2 clients can also benefit from using DWE SQW along with DataStage. However, in those cases where DWE is deployed without DataStage, or any other enterprise ETL system, clients should leverage SQW for all of the value it offers. In addition to the core strengths outlined in this chapter, to maintain the business access and performance layers of the data warehouse, clients can also exploit the external connectivity of the tool, perhaps along with the IBM data federation capability found in WebSphere Information Integrator, to help populate the atomic data assets layer.

DWE and data mining

In this chapter we discuss data mining and its implementation in the DWE Design Studio. This includes a description of the data mining process and the concept of embedded data mining for deploying mining-based BI solutions to large numbers of business users. We also discuss the DB2-based components of DWE Data Mining for building, applying, and visualizing models in the database. And we show how mining flows are developed and used in conjunction with other flows and functions in the DWE Design Studio to support advanced analytics in BI applications for solving high-value business problems.

7.1 Data mining overview

Data mining means different things to different people. Traditional statistics, OLAP, and database query are sometimes referred to as data mining in the sense that they are tools or techniques for data analysis. These methods, however, are *hypothesis driven*, implying that we understand the system well enough to formulate precise queries or specify and test explicit hypotheses about relationships in the data.

In the context of the discussion here, we begin with the premise that we do not know what patterns or relationships exist in the data. Rather, we may ask broad questions such as *What do my clients look like?* and then leave it up to the data mining algorithms to tell us what the patterns are.

Thus, we define data mining as the process of discovering and modeling non-trivial, potentially valuable patterns and relationships hidden in data. (For example, see Frawley *et al.* in “Related publications” on page 557.) Data mining is *discovery driven*, meaning that these techniques can find and characterize relationships that are unknown and therefore cannot be expressed explicitly.

Data mining techniques can be divided into two broad groups called *discovery* and *predictive* techniques. For more information, see the IBM Redbook *Mining Your Own Business in Health Care Using DB2 Intelligent Miner for Data*, SG24-6274.

7.1.1 Discovery data mining

Discovery data mining refers to techniques that find patterns without any prior knowledge of what patterns exist in the data. These techniques fall into the realm of *unsupervised learning* in which a mathematical representation of the data, which we call a *data mining model*, is fit to the data without using *training data* (paired inputs and outputs that represent prior knowledge of relationships in the data, such as a response that depends on a set of behavioral or characteristic attributes). Discovery techniques include:

- ▶ Clustering: techniques that group data records on the basis of how similar they are. An example is client segmentation to find distinct profiles of clients with similar behavioral and demographic attributes.
- ▶ Link analysis: techniques that find links or associations among data records within a single transaction or across sequential transactions. An example is the *associations* technique (also known as *market basket analysis*) to find which items tend to be purchased together in a single transaction, for example, drinks and chips. Another example is the *sequential patterns* (or

sequences) technique to find which items tend to be purchased over a series of transactions, for example, flowers → wedding ring → diapers.

7.1.2 Predictive data mining

Predictive data mining refers to techniques that find relationships between a specific *target variable* (called the *class variable* or *dependent variable*) and the other variables (columns or fields in a database table) in the data. Predictive techniques that classify data records into categories of a class variable or predict continuous numeric values of a dependent variable are forms of *supervised learning* in which a model is fit to the training data and then validated using *testing data* to verify the quality of the model by comparing the predicted values of the target field with the known values. Predictive techniques include:

- ▶ Classification: techniques that assign or classify records into categories of a class variable. Decision trees are examples of classification techniques. An example is client churn (attrition) analysis to predict a client's response (stay, leave) based on age, length of time as a client, and prior products and services purchased.
- ▶ Value prediction: techniques that predict the value of a numeric variable. Linear, polynomial, and other regression methods are examples of value prediction techniques. An example is predicting a client's annual expenditure based on age, gender, income, and previous spending behavior. Another example is predicting the likelihood that a cardiac patient will be readmitted to the hospital based on age, gender, and prior medical history.

7.2 The data-mining process

Data mining is an iterative process. As we mine the data, visualize the model, and interpret the results in light of the business problem, we learn about the data and the hidden patterns and relationships revealed by the mining. These insights guide us to correct invalid values, restructure certain fields to represent behaviors more meaningfully, collect additional data, or adjust the model parameters and then mine again. Frequently, we try various mining techniques or a combination of techniques to gain additional insights or to model the behavioral system better. For more information about the data-mining process, see the IBM Redbook *Mining Your Own Business in Health Care Using DB2 Intelligent Miner for Data*, SG24-6274, or the *CRoss Industry Standard Process for Data Mining (CRISP-DM)* listed in "Related publications" on page 557.

7.2.1 Understanding the business problem

The data mining process begins not with data but with a problem to be solved. Since the purpose of this process is to generate results (discoveries, insights, information, models) that will help to solve the business problem, you must begin by clearly stating the business problem, translating it into one or more questions that data mining can address, and understanding how the data mining results will be used in a BI solution to improve the business.

Data mining has many high-value uses, but it is not always the right tool for a particular purpose. For example, if the business need is to make the sales force more productive by reducing the time required to gather certain information, then the right solution might be a new reporting system based on real-time, multidimensional queries. On the other hand, if the need is to better understand client behaviors and preferences to improve targeting for promotional campaigns, then the right solution might be based on client segmentation and link analysis (associations and sequences). Understanding the business problem and how the data mining results will be deployed into the business enable us to determine the best approach to solving the problem.

Stating the business problem clearly and concisely forces us to focus on the root issue of what needs to change to enable us to accomplish a specific goal. Once you identify the root issue and determine that data mining is appropriate, then you can formulate one or more specific questions to address that issue. For example, to design a new targeted promotion, you might ask: What are the distinct profiles of behavioral and demographic attributes of our clients? Which of these client segments looks best to target for the product that you want to promote? For the target segment, what are the right client attributes to use in designing an appealing promotion?

Data mining is used successfully in many different business and scientific fields including retail, banking, insurance, telecommunications, manufacturing, healthcare, and pharmaceuticals. For example, in the retail, financial, and telecom industries, well-known uses of data mining include client segmentation for targeted marketing and fraud detection, store profiling for category and inventory management, associations for cross-selling and upselling, and predictive techniques for client retention and risk management. In manufacturing, data mining applications include profiling and predictive methods for quality assurance, warranty claims mitigation, and risk assessment. In healthcare and pharmaceutical research, high-value uses include associations, sequences, and predictive techniques for disease management, associations and prediction for cost management, and patient segmentation for clinical trials. In every case, the data mining results must be conveyed in such a way that someone can make a better decision or formulate an action to solve the business problem.

7.2.2 Understanding the data model

Understanding the data resources that are available to support data mining is integral to identifying the right analytical approach to solve a business problem. In most enterprises, vast amounts of data are collected and stored, typically in data warehouses and often in dedicated data marts, to support various applications. As a central repository of data resources, the data warehouse can greatly improve the efficiency of extracting and preparing data for data mining. Additional demographic data may be available from third-party or governmental sources to supplement the existing data and enrich the data mining analysis.

For any application, the data warehouse or specific data mart contains information (metadata) about the data, how it was derived, cleansed, and transformed, how it was formatted, and so on. The data and metadata together form a *data model* that supports the application and typically defines the data sources, types, content, description, usage, validity, and transformations.

Like any other application, data mining requires a data model to specify the data structure for each data mining technique. In most cases, the data model calls for a single *denormalized* table as the primary data table with perhaps one or more relational tables containing additional information such as name mappings or hierarchies. The denormalized table may be either a physical table or a view of joined tables. The most common data types used in data mining are:

- ▶ Transactional data - operational data generated each time an interaction occurs with the individual (*target*). This typically contains a time stamp, target identifier, and item identifier (where an *item* may represent a product, event, or some other attribute).
- ▶ Behavioral data - data about individuals' behaviors or relationship characteristics, such as quantities purchased, amounts spent, balances, and number or rate of interactions.
- ▶ Demographic data - information about individuals' characteristics such as age, gender, and location. This is sometimes obtained by third-party data providers.

Some mining techniques, such as clustering, classification, and regression, require a single record per individual (client, patient, store, event, as examples) with multiple columns containing the behavioral and demographic attributes of each individual. This format is referred to as the *demographic format*.

The associations technique requires a transactions table with multiple records per transaction, where each record contains a transaction identifier and one of the items in that transaction. For analyzing sequences of transactions, a sequence group identifier is needed as well. This is referred to as the *transactional format*. Typically, a relational table containing a mapping of item

numbers to their respective descriptions is also needed, as well as additional relational tables describing a hierarchy such as item → group → department and the mappings of each group or department identifier to its respective description. The data model specifies how each of these tables must be structured to meet the needs of each data mining technique.

7.2.3 Identifying the data mining approach

Understanding the business problem and how the data mining results will be used guides us in identifying the data-mining approach and techniques that are most suitable to address the problem. In some cases, a combination of data-mining techniques is most appropriate. For example, if the business issue is to improve the effectiveness of a marketing campaign for an upcoming back-to-school promotion, then we may decide to perform client segmentation using the clustering technique to identify a target group of clients, followed by a market basket analysis using the association's technique with the transactions only for the target group to find item affinities on which to base the promotion.

In practice, the choice of data mining approach may be driven in part by the data available or the cost of transforming the available data to a suitable format. For example, if the data are available in transactional format and a client segmentation to identify cross-selling opportunities is the first choice of technique, then the transactional data must be aggregated and transformed to the client level (one record per client in demographic format). If this data preparation procedure is not practical within the scope or budget of the data mining project, then a mining technique that uses transactional data (associations or sequences) might be selected instead. In this sense, the steps of identifying the data mining approach and identifying and understanding the data are closely interrelated.

7.2.4 Extracting and preparing the data

In conjunction with the data model, the data mining approach determines the data to be extracted and prepared for mining to answer the business question. Typically, most of the overall time to perform an analysis is spent preparing the data. Proper data preparation is essential to a successful analysis, and shortcuts should not be taken here. The data warehouse and its specific data structures yield great value by maintaining and making most or all of the data that will be used. By automatically handling certain data preparation steps to transform the basic data into the required structure, domain-specific or project-specific data marts can reduce data preparation time and therefore the overall project duration.

As a first step in data preparation, the raw data needed for the mining analysis are extracted from the data warehouse. Table joins are commonly done to collect columns from multiple tables into one or more initial tables or views having the correct data type format needed for the analysis.

Despite the efficiencies of data warehouses and project-specific or domain-specific data marts, some amount of additional data preparation is generally required to prepare the data to support a particular data-mining analysis. Having the data ready for data mining usually is not the same as having data that are clean from a data warehousing perspective. Additional transformations or derived variables not already created, as part of a specific data mart may have to be done. For example, market basket analysis requires data to be in a transactional format, while client segmentation requires data to be aggregated to the client level. Furthermore, some data fields may be mutually inconsistent, requiring modification or redefinition. Many other transformations and derivations (such as conversion of birth date to age) may be required, depending on the data needs of the analysis at hand. Perhaps the least amount of data preparation is required for market basket analysis, which uses transactions files (sometimes referred to as *t-logs*) that may already be in the proper format, leaving only name mappings and a hierarchy (*taxonomy*) to be prepared.

Once the data have been extracted and prepared, the distributions of the variables should be visually inspected to assess whether data distributions and ranges are plausible and what to do if they are not, to identify variables comprised of a single value or a large proportion of missing values and how to deal with them, and to find obvious inconsistencies across variables. DWE offers several functions to facilitate data exploration as part of data quality assessment and data preparation, including the visualization of univariate, bivariate, and multivariate distributions. The need for additional data preparation may be revealed after the initial data mining runs have been made.

7.2.5 Building the data mining model

After the data have been prepared, the data mining model is created. The model is built to execute the desired technique with appropriate parameters. Modeling is an interactive and iterative process as the initial results are reviewed, model parameters are adjusted to produce a better model, and any additional data preparation is performed.

For a predictive technique, training and testing sets are used for model building and validation, respectively. Overfitting is a potential problem when building a predictive model. A high degree of overfitting means that the model closely fits the training data but is not very accurate when applied to new data. Since you want to use the model to predict outcomes that have not happened yet, there is

particular interest in how accurate the model will be when applied to new data. Thus, you can train the model with a training data set and then validate the model using a testing data set that is statistically the same as the training data. Both the training and testing sets consist of historical data, meaning that they both must contain the outcome (target field) to be able to construct and validate a predictive model. DWE has a function to randomly split the input data into training and testing sets and to perform the modeling and validation steps automatically. In DWE, the mining algorithms are implemented with internal checks to avoid overfitting a model during training.

The modeling process produces a data mining model that is stored in PMML format in a designated DB2 table. PMML stands for Predictive Model Markup Language, an XML-based language for defining statistical and data-mining models so that they can be shared and used by PMML-compliant applications. For more information about PMML, refer to “The Data Mining Group” on page 558 listed in “Related publications” on page 557.

Each DB2 database used for data mining must be *enabled* for mining. To enable a database for mining means that the DB2 extenders for data mining are activated for the database and the necessary tables to support the mining are created. When a database is enabled for data mining, a set of tables under the schema IDMMX is created to contain information needed by DB2 to execute the data mining procedures. Four of these tables contain the PMML models by type. The PMML models in these tables are available to DWE for deployment, application, or visualization. They also can be accessed by PMML-compliant applications such as an Alphablox application or a third-party application.

7.2.6 Interpreting and evaluating the data mining results

DWE allows a model to be visualized for interpretation and evaluation. Tailored to each type of data mining model, the visualizations present information about model quality, specific results such as association rules or clusters, and other information about the data and results pertinent to the particular model. This information enables the data-mining analyst to assess model quality and determine whether the model fulfills its business purpose. Improvements to the input data, model parameters, and modeling technique can then be made as needed to obtain a good model that meets the business objective.

For example, a clustering model intended to find high-potential clients to target for a new family plan cellular phone offer has one large cluster that contains 90% of all the clients with the remainder distributed across several very small clusters. Because it provides little distinctive information about the clients, the model is of poor quality and does not meet the objective of identifying high-potential clients with distinct behavioral and demographic profiles to support designing a promotional offer. Adjusting certain parameters and re-executing the mining run

yields a model with clusters ranging in size from 30% to 1%, providing a set of distinct profiles, each with enough clients to be meaningful in terms of the business objective.

As another example, a classification model intended to screen a hospital's patient database to identify those at high risk of developing a certain disease fits the training data extremely well and is highly predictive of the propensity to develop the disease. But one of the key predictors in the model is a costly test not covered by insurance except for patients already diagnosed with the disease. Thus, this variable is unknown for most patients in the database. Although this model is of high quality, it does not meet the objective of an early warning indicator for patients who have not developed the disease. Eliminating that predictor from the model yields a less accurate but more useful model that can be widely deployed to screen patients and identify those who would benefit from preventive treatment before they develop the disease.

As a third example, a regression model intended to predict the likelihood that a client will default on a new bank loan fits the training data very well, is highly accurate with the testing data, and contains only those variables that are readily attainable at the time of loan application. But when the model predicts that a loan applicant has a high propensity to default, the loan officer is unable to state precisely why the loan application must be declined. Although this model is accurate and useful, the bank's legal department points out that the model does not meet the business need to be able to cite specific reasons for declining a loan. Building a decision tree classification model using the same data and variables yields a model of slightly lower accuracy, but has the advantage of explicit decision paths for predicting the likelihood that a loan applicant will default, thereby enabling the loan officer to meet the legal requirement of stating exactly why a loan application is declined.

7.2.7 Deploying the data mining results

The final step in the data mining process is to deploy the data mining results in the business as part of a BI solution. Deployment may be the most important step in the data mining process because how and where the results are deployed are crucial to realizing the maximum value from the data mining. Data mining results can be deployed in various ways to diverse business processes or systems, depending on the business needs:

- ▶ *Ad hoc* insight - Use data mining on an *ad hoc* basis to address a specific, nonrecurring question. For example, a pharmaceutical researcher may use data mining techniques to discover a relationship between gene counts and disease state for a cancer research project.
- ▶ Interactive insights - Incorporate data mining into a BI application for interactive analysis. For example, a business analyst may regularly use a

targeted marketing application to segment the client base, select a segment suitable for the next promotion, and perform market basket analysis specific to that segment. The item affinities and client profile then feed analytical/reporting functions that identify the highest-value item relationships for the promotion and the best promotional channel given the characteristics of the target group.

- ▶ Scoring - Apply a data mining model to generate some sort of prediction for each record, depending on the type of model. For a clustering model, the score is the best-fit cluster for each individual. For an associations model, the score is the highest-affinity item, given other items. For a sequences model, the score is the most likely action to occur next. For a predictive model, the score is the predicted value or response.
 - Batch scoring - Embed a scoring function in a BI application for periodic scoring of a database using a data-mining model and generating an action or alert when required. An application might be set up to periodically score a database of clients and proactively trigger an action in response to a change in an individual's score. For example, when an insurance client's life situation changes, such as having a child, the application registers a change in that client's scored propensity to purchase life insurance. An informational letter is automatically sent to the client, and the agent is notified to follow up with an offer for a new or upgraded life insurance policy.
 - Real-time scoring - Embed a scoring function in a BI application for real-time, on-demand scoring of an individual during an interaction and immediately generate a recommended offer or action. For example, a call center client service representative may call up a client's record during a phone conversation, enter the current order, and receive a recommendation for a suitable item to offer the client to complement the item just ordered.

7.3 Embedded data mining

Leading-edge companies have profited greatly from employing data mining to help them make better business decisions. But this technology is complex and has traditionally been the domain of expert statisticians working in an environment separate from the central data repository and BI infrastructure.

Businesses today are looking for ways to make data mining accessible to large numbers of line-of-business analysts and decision makers throughout the enterprise, or *bring data mining to the masses*. *Embedded data mining* is the concept of delivering data mining to business analysts through portals, reporting tools, and customized applications tailored to specific business areas, all within

the database environment. With embedded data mining components implemented as DB2 extenders for model building and application, DWE facilitates the development and enterprise-wide deployment of data mining-based solutions by enabling data mining functions to be integrated into BI tools. Because data mining capabilities are implemented as DB2 extenders, any reporting or query tool that can export SQL can become a data mining platform. Business analysts who are not accomplished statisticians can readily bring the power of data mining to bear on many different business problems.

Embedded mining is based primarily on discovery data mining methods (associations, sequential patterns, and clustering) that are relatively easy to execute and interpret and so lend themselves well to incorporation into business applications for business analysts. Predictive data mining methods (classification, value prediction) also can be implemented via embedded mining, but these methods are more complex to execute (particularly regarding parameter specifications) and interpret. For predictive methods, embedded mining may be most useful to business analysts for scoring using models created by statisticians. For more information about embedded data mining, see the articles “Accessible Insight: Embedded Data Mining” and “Embedded Data Mining: Steps to Success,” listed in “Related publications” on page 557.

For embedded data mining, the data warehouse is the single, centralized, and comprehensive data source for integrated BI analytical and related functionality. Through centralized management of data resources, many of the problems, errors, and costs inherent in moving and duplicating data can be avoided. Working directly in the database also enables the use of high-throughput, scalable data warehouse platforms for BI analytics.

BI analytics can be effectively supported by domain-specific or project-specific data structures within the overall data warehouse, reducing data manipulation costs. For example, to support an ongoing client segmentation project, a data mart could be established to regularly perform the required data transformations to populate and refresh a table containing the client data in the correct structure for the analysis. Another data mart could likewise support market basket analysis, which requires a different data structure than client segmentation. By maintaining and refreshing the data for different analytical uses, data marts can greatly reduce the overall analysis time.

The need to integrate mining into the database, with both embedded data mining and traditional data mining (working outside the central database environment) has led to the creation of standards establishing the Predictive Model Markup Language (PMML) and a standard SQL interface to DB2's data mining capabilities. PMML provides a way to express a data-mining model as an XML object that can be ported between analytical environments without having to rebuild or recode the model. This capability enables analysts to create

data-mining models and deploy them in their native forms in the data warehouse where they can be used by applications. PMML also facilitates model refreshing without recoding and retesting, greatly reducing the time to implement refreshed models and ensuring that business analysts are able to use up-to-date models in their decision making. Many data-mining vendors have embedded DB2-based mining in their analytical and reporting tools. For more information about PMML refer to The Data Mining Group listed in “Related publications” on page 557.

7.4 DWE data-mining components

DWE has three components for DB2-based data mining. These are called modeling, visualization, and scoring. For more information about these components see *IBM DB2 Data Warehouse Edition Administration and Programming for Intelligent Miner Modeling, Version 9.1*, SH12-6838, or the IBM Redbook *Enhance Your Business Applications: Simple Integration of Advanced Data Mining Functions*, SG24-6879, listed in “Related publications” on page 557. Also see the DB2 Business Intelligence section of the DB2 Information Center installed as part of DWE.

7.4.1 Modeling

Modeling is a DB2 SQL application programming interface (API) implemented as a DB2 extender. Modeling is accessed graphically through the DWE Design Studio to build data-mining models from information residing in DB2 databases.

Modeling offers five data-mining methods. The clustering and regression methods have multiple algorithms (two and three, respectively) to facilitate model development. The data mining methods are:

- ▶ Associations: Which item affinities exist in the data?
- ▶ Sequences: Which sequential patterns exist in the data?
- ▶ Clustering: Which groups or segments exist in the data?
- ▶ Decision tree classification: How do you predict categorical values?
- ▶ Regression: How do you predict numeric values?

DWE Modeling architecture is illustrated in Figure 7-1 on page 249. As a DB2 extender, modeling provides a set of SQL stored procedures and user-defined functions to build a model and store it in a DB2 table. These procedures and functions are collectively referred to as *easy mining procedures*. As the model is set up via graphical wizards in the Design Studio, the easy mining procedures use the wizard inputs automatically to create mining tasks that specify the type of model to be built, the parameter settings, the data location, and data settings (for example, which columns to use in the model) and then to call the appropriate mining kernel in DB2. A DB2 table containing columns representing the

behaviors and other attributes of the records (such as clients, stores, accounts, and machines), including the response or outcome, if any, is used as the data source for building (training) the model and, for predictive mining, validating (testing) the model as well. As described in 7.2.5 “Building the data mining model” on page 243, the new model is stored in PMML format in a DB2 table where it is accessible for deployment.

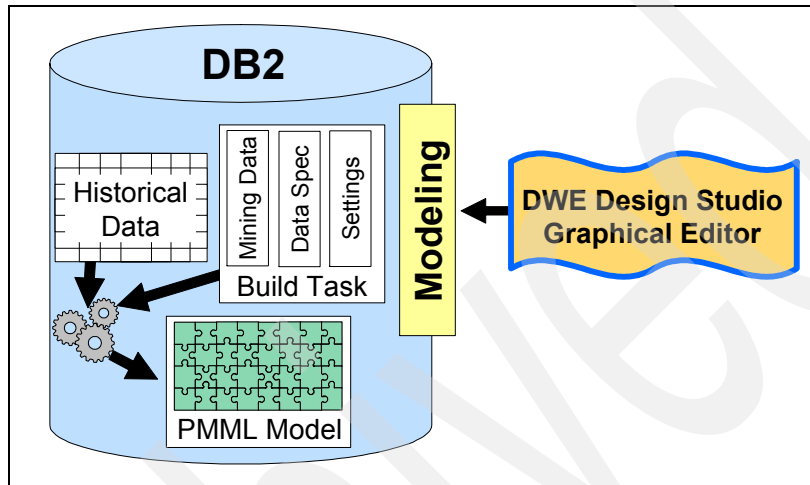


Figure 7-1 DWE Modeling architecture

7.4.2 Visualization

After a data-mining model has been created, the analyst can explore the model using visualization. Visualization is a Java application that uses SQL to call and graphically display PMML models, enabling the analyst to assess a model's quality, decide how to improve the model by adjusting model content or parameters, and interpret the final model results for business value. DWE visualization architecture is illustrated in Figure 7-2.

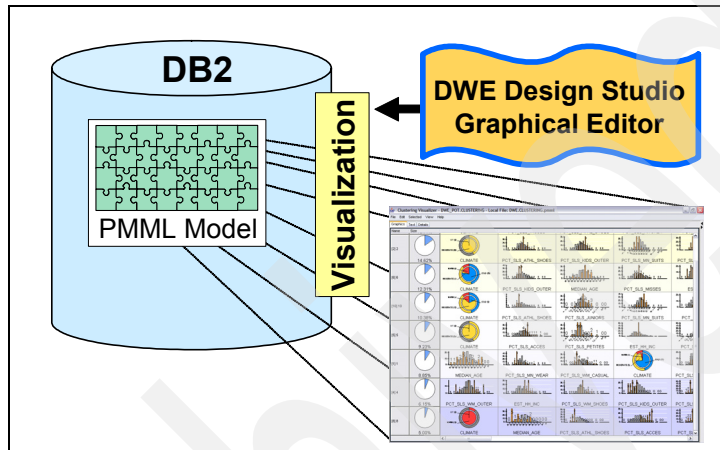


Figure 7-2 DWE visualization architecture

Visualization has visualizers for all five mining methods in the modeling. The visualizers are tailored to each mining method and provide a variety of graphical and tabular information for model quality assessment and interpretation in light of the business problem.

Visualization can also display PMML models generated by other tools if the models contain appropriate visualization extensions such as quality information or distribution statistics as produced by modeling. But some of these model types do not contain much extended information and hence do not present very well in visualization.

7.4.3 Scoring

Like modeling, scoring is implemented as a DB2 extender. It enables application programs using the SQL API to apply PMML models to large databases, subsets of databases, or single records. Since the focus of the PMML standard is

interoperability for scoring, scoring supports all the model types created by modeling as well as selected model types generated by other applications (as examples, SAS and SPSS) that support PMML models:

- ▶ Associations
- ▶ Sequences
- ▶ Clustering (distribution-based, center-based)
- ▶ Classification (decision tree, neural, logistic regression)
- ▶ Regression (linear, polynomial, transform, neural)

Scoring also supports the radial basis function (RBF) prediction technique, which is not yet part of PMML. RBF models can be expressed in XML format, enabling them to be used with scoring.

Scoring includes scoring Java Beans, enabling the scoring of a single data record in a Java application. This capability can be used to integrate scoring into client-facing or e-business applications, for example.

DWE Scoring architecture is illustrated in Figure 7-3 on page 252. A PMML model is either created and automatically stored by Modeling or created by another application and imported into DB2 using Scoring's SQL import function. Accessed through the Design Studio, Scoring applies the PMML model to new data. The *score* assigned to each record depends on the type of mining model. For example, with a clustering model, the score is the best-fit cluster for a given record. The results are written to a new DB2 table or view, where they are available to other applications for reporting or further analysis such as OLAP. Because it exploits DB2's parallel processing, Scoring is faster than single-threaded scoring routines.

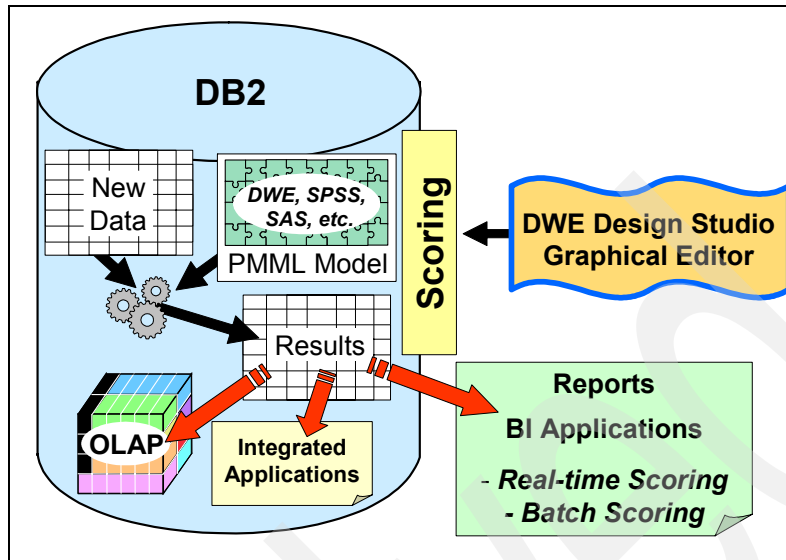


Figure 7-3 DWE Scoring architecture

7.5 Data mining in the DWE Design Studio

The Design Studio is the interface to data exploration and data mining in DWE. In this section we discuss the process and functions for developing *mining flows* in the Design Studio. We also discuss useful functionality for data exploration as a way to better understand the data as it is prepared for mining. For more information see the IBM DB2 Data Warehouse Enterprise Edition Tutorial V9.1.

7.5.1 Mining flows

A mining flow is one of the types of SQW flows in the Design Studio. Like data flows and control flows, mining flows establish SQL-based movement and transformation of data for data mining model construction and deployment in a database. Mining flows also enable data preparation and model visualization as well as SQL code generation for use in Alphablox pages or a suitable BI application to provide embedded analytics. Mining flows use the same graphical editor and some of the same graphical operators as other SQW flows, as explained in Chapter 6, “Data movement and transformation” on page 137, along with many mining-specific operators. A mining flow can be executed alone or incorporated as an element in a control flow.

Unlike data flows, mining flows are developed interactively with a live connection to a database. This means that individual branches of the mining flow can be executed and validated separately as the mining flow is being built.

Although the Design Studio allows the user to connect to more than one database, a particular mining flow works with only one database. This is because DB2-based data mining uses stored procedures and user-defined functions that operate only in the database where the mined data reside. To allow a table residing in a different database to be used in a mining flow, an alias referring to that table and database would have to be created in the database used by the mining flow.

7.5.2 Database enablement

As explained in 7.2.5 “Building the data mining model” on page 243, each database used for data mining must be enabled for mining, which is easily done in the Design Studio. To enable a database, locate the database in the Connections folder in the Database Explorer on the Design Studio screen and connect (or reconnect) to the database. As illustrated in Figure 7-4, expand the database folder to reveal the blue icon representing the database, right-click the blue icon, and choose the selection to enable the database for data mining.

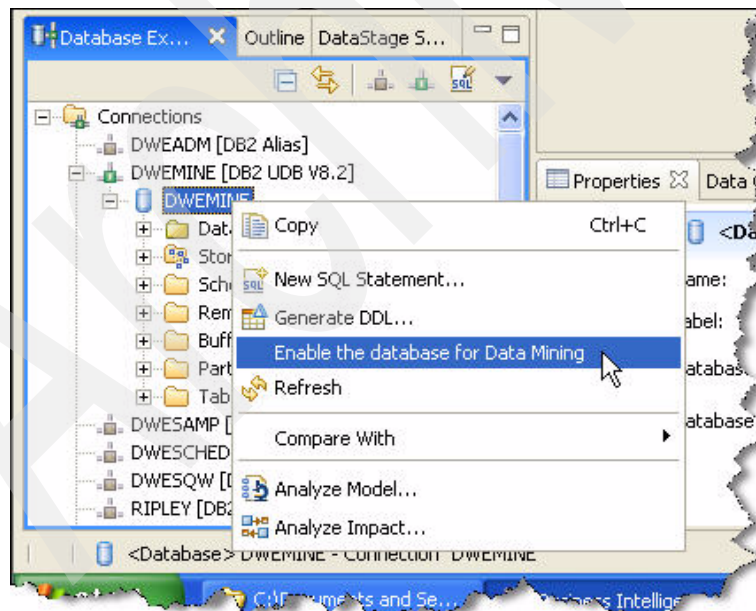


Figure 7-4 Enabling a database in the Database Explorer

When a database is enabled, the schema IDMMX and a set of tables used by the mining functions is automatically created, as illustrated in Figure 7-5. Most of these tables are repositories for settings and specifications used by modeling and scoring. Four of the tables are used to store PMML models:

- ▶ IDMMX.RULEMODELS - associations and sequences models
- ▶ IDMMX.CLUSTERMODELS - clustering models
- ▶ IDMMX.CLASSIFMODELS - classification models
- ▶ IDMMX.REGRESSIONMODELS - prediction models

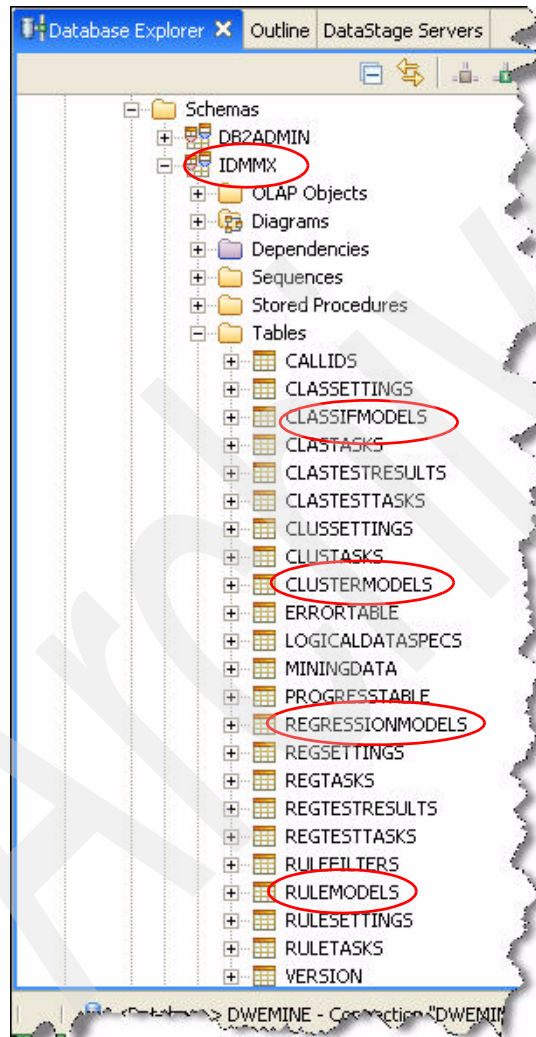


Figure 7-5 Tables created under the schema IDMMX

7.5.3 Data exploration

As discussed in 7.2.4 “Extracting and preparing the data” on page 242, data preparation is a key step in data mining. The Design Studio offers several data exploration functions to facilitate data preparation. In particular, four functions that allow the analyst to inspect data values and distributions quickly and easily are available in the Database Explorer:

- ▶ Table sampling
- ▶ Univariate distributions
- ▶ Bivariate distributions
- ▶ Multivariate distributions

Sampling the contents of a table in the Database Explorer is illustrated in Figure 7-6. Right-clicking the desired table brings up a menu for selecting **Data** → **Sample Contents**.

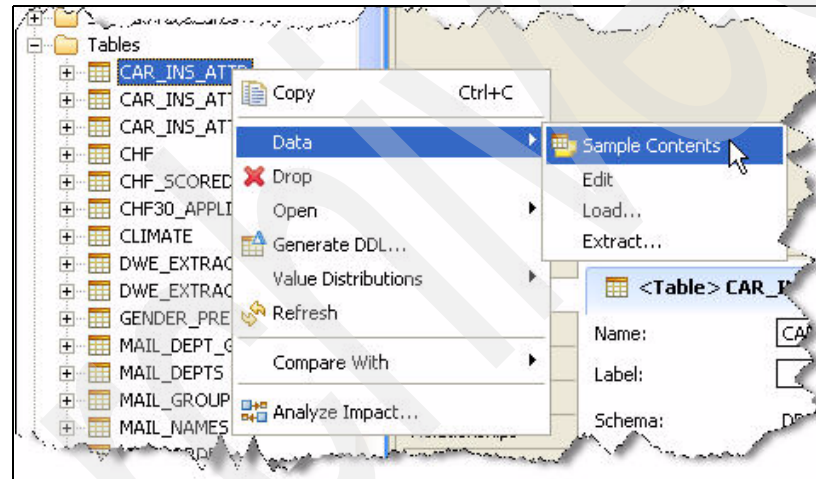
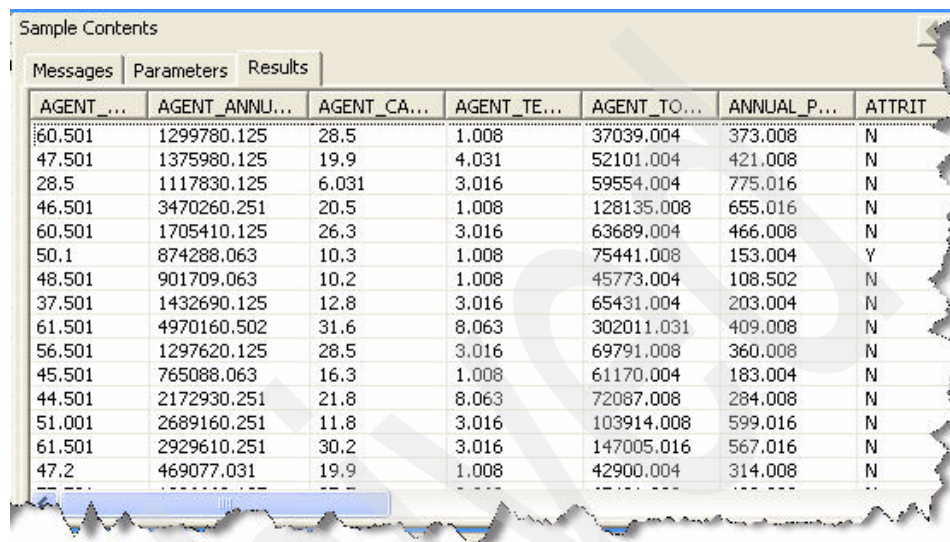


Figure 7-6 Table sampling

This selection produces a table of all the columns and the first 50 rows of the table. An example of the sampled contents is shown in Figure 7-7. This view appears in the lower right corner of the Design Studio screen. The analyst can inspect the sampled rows to get an initial feel for what the data look like.



| AGENT_... | AGENT_ANNU... | AGENT_CA... | AGENT_TE... | AGENT_TO... | ANNUAL_P... | ATTRIT |
|-----------|---------------|-------------|-------------|-------------|-------------|--------|
| 60.501 | 1299780.125 | 28.5 | 1.008 | 37039.004 | 373.008 | N |
| 47.501 | 1375980.125 | 19.9 | 4.031 | 52101.004 | 421.008 | N |
| 28.5 | 1117830.125 | 6.031 | 3.016 | 59554.004 | 775.016 | N |
| 46.501 | 3470260.251 | 20.5 | 1.008 | 128135.008 | 655.016 | N |
| 60.501 | 1705410.125 | 26.3 | 3.016 | 63689.004 | 466.008 | N |
| 50.1 | 874288.063 | 10.3 | 1.008 | 75441.008 | 153.004 | Y |
| 48.501 | 901709.063 | 10.2 | 1.008 | 45773.004 | 108.502 | N |
| 37.501 | 1432690.125 | 12.8 | 3.016 | 65431.004 | 203.004 | N |
| 61.501 | 4970160.502 | 31.6 | 8.063 | 302011.031 | 409.008 | N |
| 56.501 | 1297620.125 | 28.5 | 3.016 | 69791.008 | 360.008 | N |
| 45.501 | 765088.063 | 16.3 | 1.008 | 61170.004 | 183.004 | N |
| 44.501 | 2172930.251 | 21.8 | 8.063 | 72087.008 | 284.008 | N |
| 51.001 | 2689160.251 | 11.8 | 3.016 | 103914.008 | 599.016 | N |
| 61.501 | 2929610.251 | 30.2 | 3.016 | 147005.016 | 567.016 | N |
| 47.2 | 469077.031 | 19.9 | 1.008 | 42900.004 | 314.008 | N |

Figure 7-7 Sample contents

More illuminating insights about the data can be gained by inspecting the distributions of the columns in the table. The function to select univariate, bivariate, or multivariate distributions is illustrated in Figure 7-8. Right-clicking the desired table brings up a menu for selecting **Value Distributions** and the desired view (Univariate, Bivariate, Multivariate) of the data distributions.

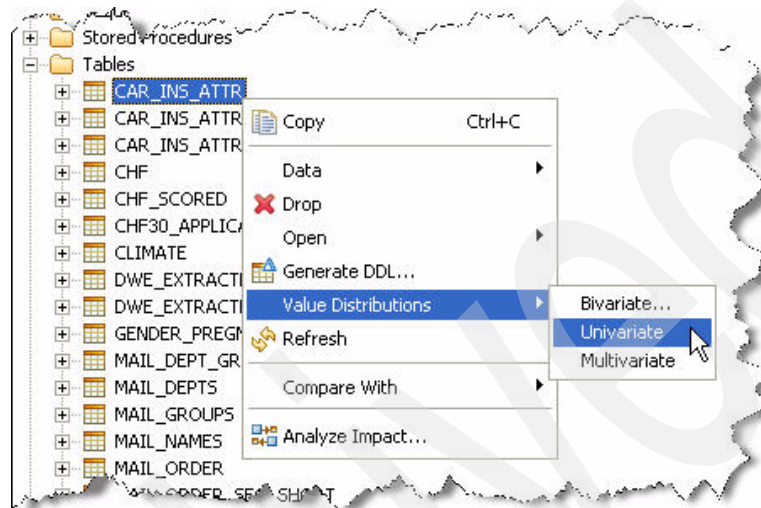


Figure 7-8 Selecting distributions for viewing

Selecting Univariate generates a set of histograms representing the probability distribution of each column in the table individually. The distributions are displayed in graphical form, as illustrated in Figure 7-9 on page 258.

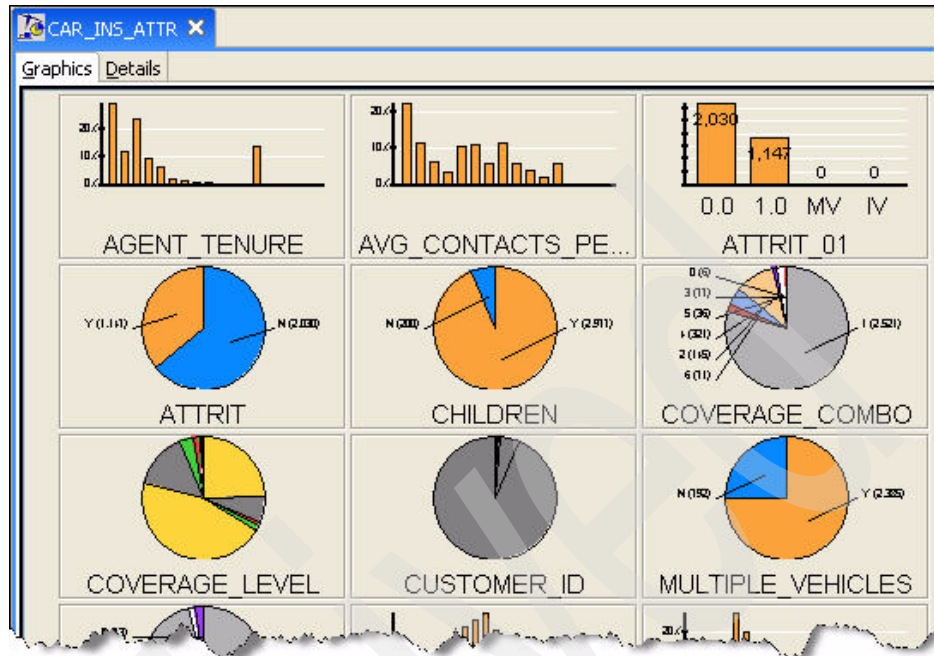


Figure 7-9 Univariate distributions example

They are also displayed in tabular form, as depicted in Figure 7-10 on page 259. Inspecting the univariate distributions helps the analyst assess data quality by showing out-of-range values or other potential data problems that the analyst should investigate further before proceeding with any analysis. (Note that extreme values may indicate bad data, for example, inadvertently including a few business accounts when extracting personal account information, or they may represent the most interesting behavior in the entire data set.)

| Field | Type | Modal Value | Modal Freq. | Chi Squared | Homogeneity | Min. | Max. | Mean |
|-------------------|---------------------|-----------------|-------------|-------------|-------------|------|-----------|-------|
| AGENT_TENURE | Numeric, discrete | 1.0 | 943 | N/A | 0.47 | 1 | 11.063 | 2.6 |
| AVG_CONTACT... | Numeric, discrete | 0.0 | 717 | N/A | 0.332 | 0 | 8.063 | 2.265 |
| ATTRIT_01 | Numeric, discrete | 0.0 | 2,030 | N/A | 0.539 | 0 | 1 | 0.38 |
| ATTRIT | Categorical | N | 2,030 | N/A | 0.539 | N/A | N/A | N/A |
| CHILDREN | Categorical | Y | 2,977 | N/A | 0.882 | N/A | N/A | N/A |
| COVERAGE_CO... | Categorical | 1 | 2,521 | N/A | 0.643 | N/A | N/A | N/A |
| COVERAGE_LEV... | Categorical | 100 | 1,436 | N/A | 0.292 | N/A | N/A | N/A |
| CUSTOMER_ID | Categorical | 2035 | 2 | N/A | 0.01 | N/A | N/A | N/A |
| MULTIPLE_VEHIC... | Categorical | Y | 2,385 | N/A | 0.626 | N/A | N/A | N/A |
| VEHICLE_TYPE | Categorical | 1 | 3,055 | N/A | 0.925 | N/A | N/A | N/A |
| AGENT_AGE | Numeric, continu... | 50 - 55 | 598 | N/A | 0.275 | 24.3 | 85.8 | 49.37 |
| AGENT_ANNUAL... | Numeric, continu... | 1,000,000 - ... | 841 | N/A | 0.379 | 0 | 8,805,... | 1,995 |
| AGENT_CAREER | Numeric, continu... | 20 - 25 | 718 | N/A | 0.279 | 1.1 | 44.3 | 19 |

Figure 7-10 Univariate statistics example

Selecting Bivariate generates a set of histograms showing how data values are distributed between two fields (columns in the table). Unlike univariate distributions, bivariate distributions can reveal data inconsistencies between two fields, as illustrated in Example 7-1, Figure 7-11 on page 260, and Figure 7-12 on page 260. In this example, both fields individually appear valid based on univariate distributions, but the bivariate distributions visualization clearly shows that some records have an inconsistent combination of values, such as, some people are coded as being both male and pregnant. The analyst must examine these records and resolve the data quality problems before proceeding with the analysis.

Example 7-1 Data quality problem

A table contains two fields called GENDER with values of [M, F, U] for Male, Female, and Unknown and another field called PREGNANCY with values of [Y, N] for Yes and No. From a data warehousing perspective, each field is considered clean because there are no invalid values within an individual field. But an inherent inconsistency is revealed when values of GENDER=M and PREGNANCY=Y occur together for a particular record.

Figure 7-11 depicts a univariate distribution example.

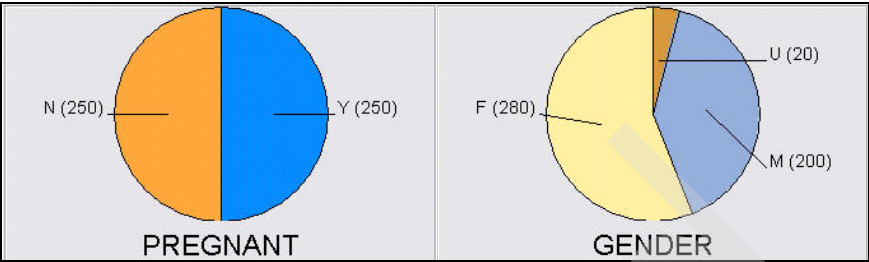


Figure 7-11 Univariate distributions example

Figure 7-12 depicts a bivariate distributions example.

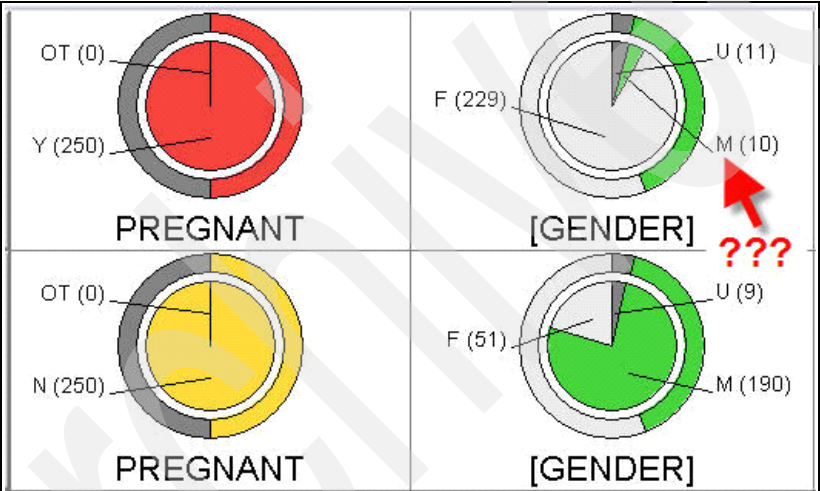


Figure 7-12 Bivariate distributions example

Selecting Multivariate generates a set of histograms showing how data values are distributed among multiple fields. The multivariate view has the capability to highlight how groups of records are distributed across all fields, as illustrated in Figure 7-13 on page 261. (Note that the distributions are intended only for data exploration and thus, in the interest of execution time to calculate the table sample statistics, are based on a limited number of records.) In this example we are interested in understanding the characteristics of insurance agents serving clients who cancel their policies. Each record in this data table represents a client account. Clicking the **Y** value for the field **ATTRIT** highlights that histogram bar and displays where those clients (*attriters* with **ATTRIT=Y**) fall in the distributions of the other fields. For example, we see from the **ATTRIT** distribution that 33% of clients attrit (407 attriters out of 1,235 clients in the sample). For clients served by

highly productive agents, say, those with annual sales of three to four million dollars (the fourth bar in the AGENT_ANNUAL_SALES histogram), the client attrition rate is only about 24% (23 out of 96 in this subgroup). For clients served by inexperienced agents (less than 1 year of experience, represented by the first bar of the AGENT_TENURE histogram), the attrition rate is about 61% (201 out of 329 in this subgroup).

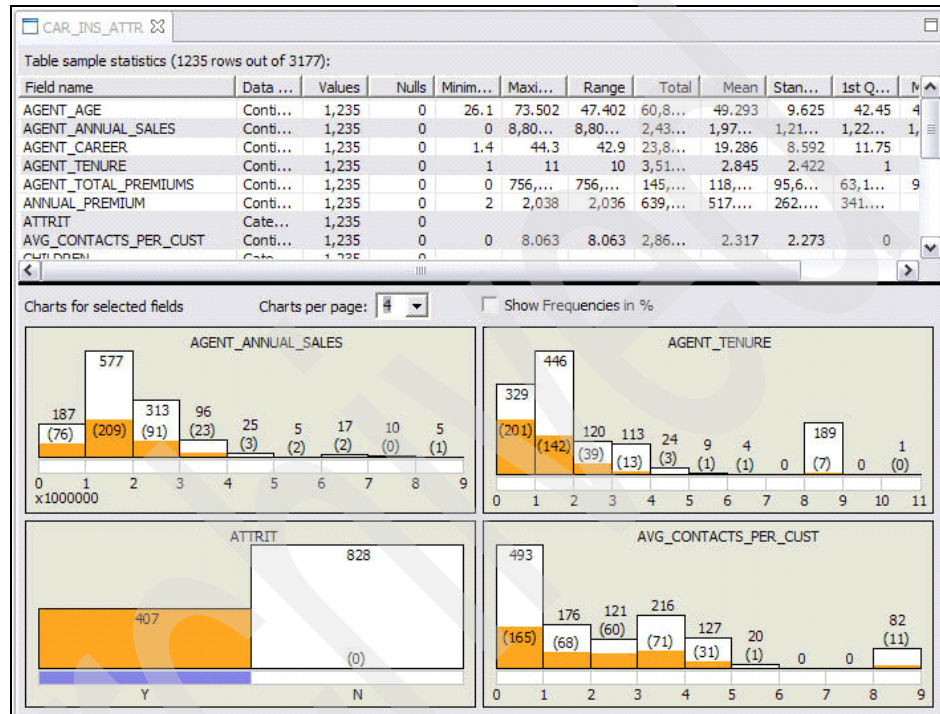


Figure 7-13 Multivariate distributions example

More detailed exploration can be done by holding down the Ctrl key and selecting multiple histogram bars. As illustrated in Figure 7-14, clicking the **Y** value for ATTRIT and the **[0-1]** value for AGENT_TENURE shows how the records having both these values are distributed across all fields.

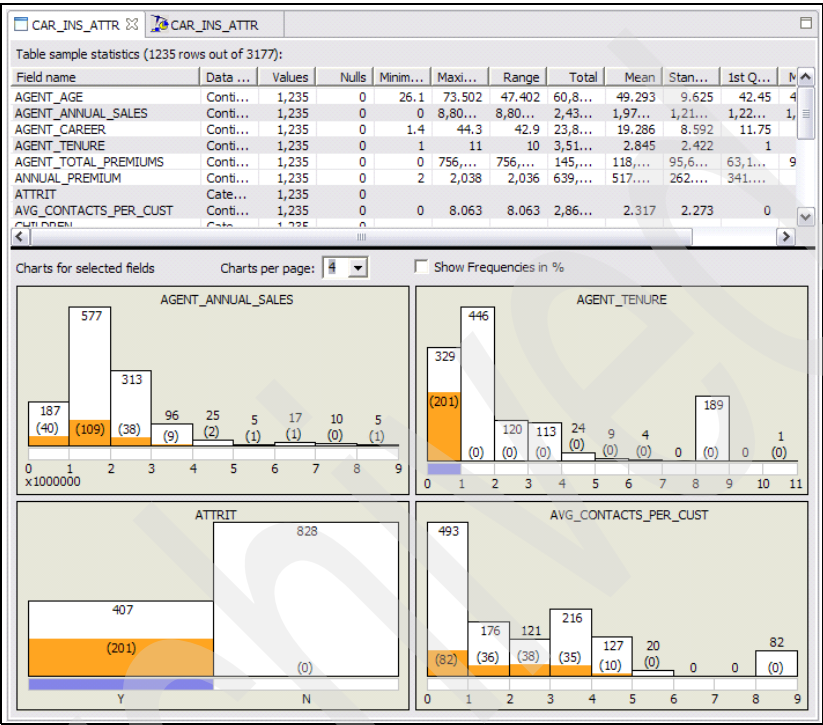


Figure 7-14 Multivariate distributions example

7.5.4 Data mining operators

Mining flows use many of the same operators used in SQW flows, which are discussed in Chapter 6, “Data movement and transformation” on page 137. Mining flows also use several operators that are specific to data mining. Operators are selected from the palette and dropped onto the mining flow canvas. Additional information about operators is available under the Help tab in the Design Studio menu.

Preprocessing operators

In addition to other data preprocessing operators, mining flows use three preprocessing operators specific to data mining. These are the *sampler*, *random split*, and *discretize* operators, shown in Figure 7-15.

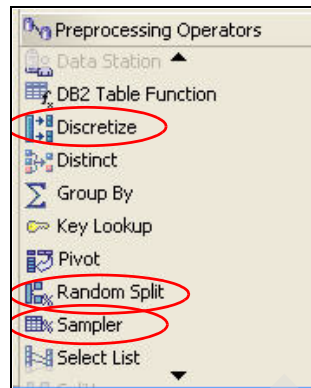


Figure 7-15 Preprocessing operators in the Design Studio palette

The *sampler* operator is a filter that creates a random sample of an incoming table represented by another operator. For very large input tables, processing times to build mining models may be long. One solution to reduce processing time is to create a mining model using a random sample of the input data during the iterative development phase (when different parameters and settings are tested to generate the best model) and then run the final model with the entire data set.

When sampling a large table source, however, the random sample can be created more efficiently by specifying a sampling rate on the Table Source operator, which exploits DB2's native sampling with very good performance. An example of specifying the sampling rate (entered as a percentage) in the Table Source properties is illustrated in Figure 7-16.

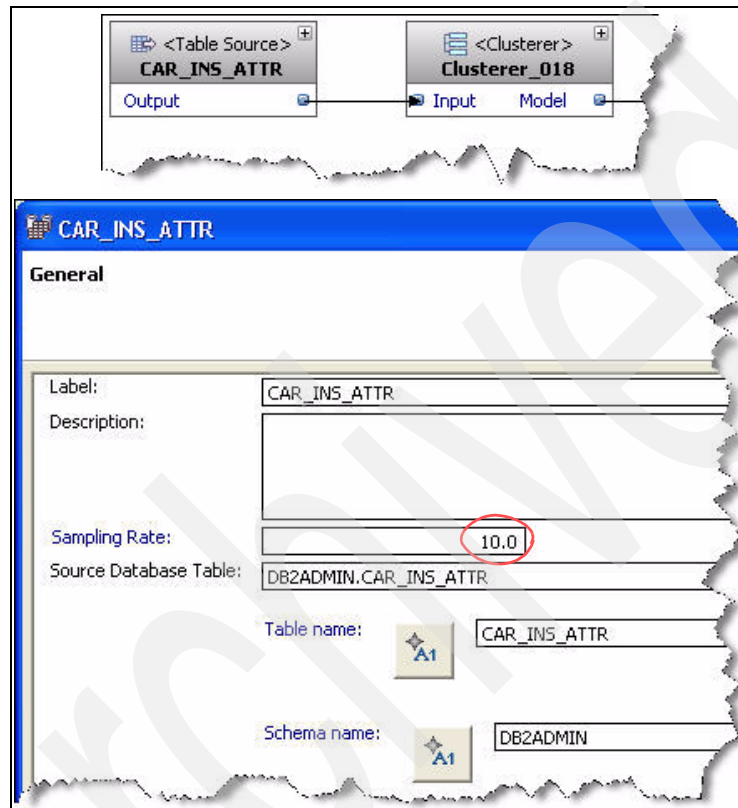


Figure 7-16 Example of specifying a sampling rate in a table source operator

The *random split* operator is used to split source data into training and testing data sets for building and validating predictive models, respectively. Unlike the splitter operator, which splits the rows of a table into multiple tables based on user-specified conditions, the random split operator splits the input data randomly to ensure that the training and testing sets properly represent the input data.

As illustrated in Figure 7-17, the random Split operator is positioned between the input data and the model building and validation operators. The random split operator randomly selects a testing data set and creates a training data set, which is the complement of the testing data unless stratified sampling is performed. The training data becomes the input data for building the data model, and the testing data becomes the input data for testing or validating the model.

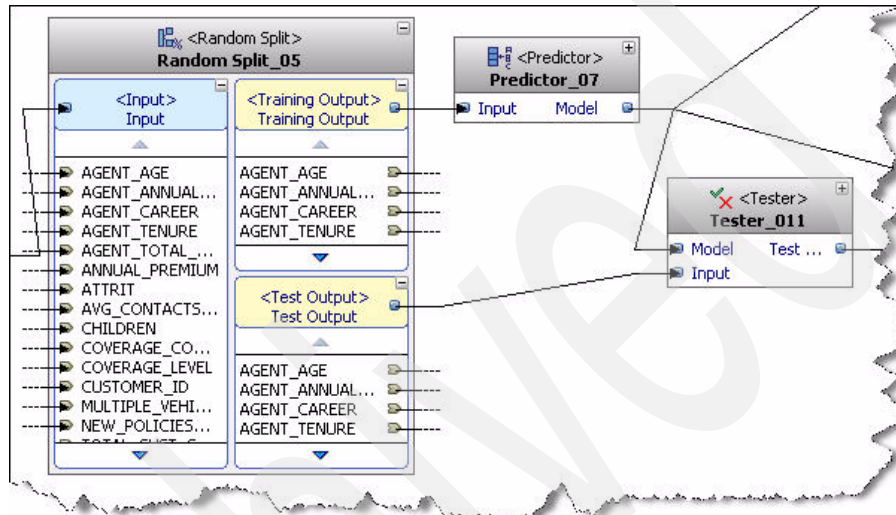


Figure 7-17 Example of training and testing data sets created by Random Split operator

The properties of the random split operator are illustrated in Figure 7-18 on page 266. The analyst specifies the percentage of the source data to be randomly selected for the testing data set. Typically, stratified sampling is not done, but it can be useful to an experienced analyst who wants to create an *enriched sample* for building a classification model when the response or target value of interest occurs for a small percentage of the records. For example, when building a model to predict adverse patient reaction to a medical treatment protocol, the target value NO might occur with a frequency of 99% and the target value YES with a frequency of only 1%. A highly accurate classification model would be simply to predict that a patient will never have an adverse reaction (NO). Although it would be 99% accurate, such a model would be trivial because it would tell us nothing about those who will react adversely (YES). Using stratified sampling to construct an enriched training data set having approximately equal percentages of YES and NO records often yields a better model, which is then validated using a testing data set with the actual percentages of YES and NO. If stratified sampling is not specified, then the training data set consists simply of the records of the complement of the testing data set.

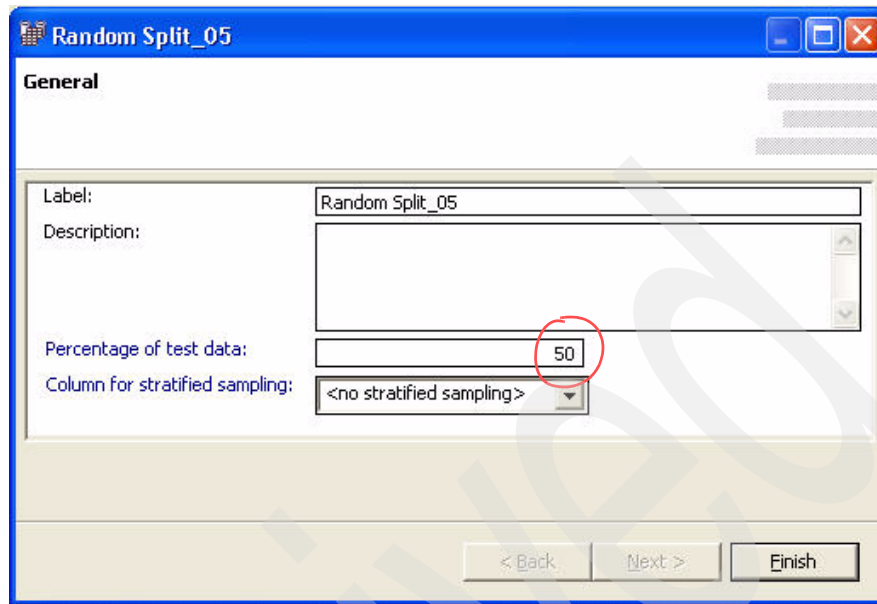


Figure 7-18 Example of random split operator properties

The discretize operator converts the values of a continuous input field (variable) into a defined set of categorical or numeric buckets. For example, the variable INCOME ranges from \$0 to \$10 million with some records having missing (null) values. The discretize operator can be used to transform the numeric values into categories of LOW, MEDIUM, HIGH, and UNKNOWN. Although discretizing a variable reduces the amount of information contained in the variable, it may be suitable in some cases, depending on the objectives of the analysis or issues such as poor data quality (inaccurate or missing values in many of the records). As illustrated in Figure 7-19, the discretize operator is positioned after the input data setup.

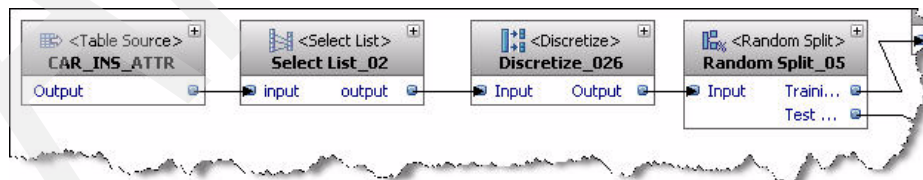


Figure 7-19 Example of the discretize operator in a mining flow

The properties of the discretize operator are illustrated in Figure 7-20. The analyst selects the variable (column) to be discretized and enters the name and type of the new discretized column to be created.

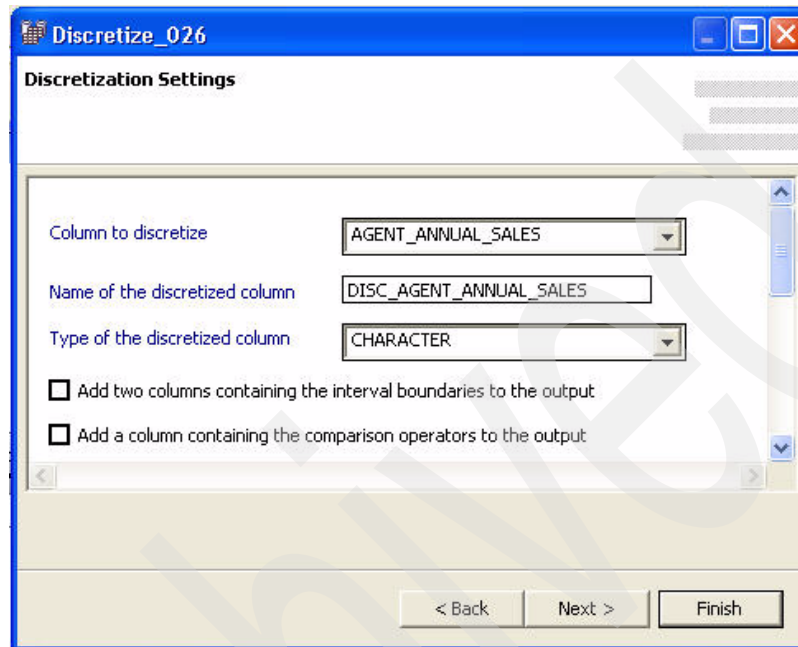


Figure 7-20 Example of discretize operator properties

As illustrated in Figure 7-21 on page 268, any of three methods can be selected to create the discrete interval boundaries for the discretized variable. These methods are:

- ▶ Define the intervals manually.
The analyst creates each interval separately by specifying the lower limit, upper limit, and the new discretized value for each interval.
- ▶ Calculate the intervals from specified values.
The analyst specifies the desired number of intervals, the lower and upper limits of the overall value range, and a prefix used in generating the new discretized values. The system then calculates intervals of equal width plus one interval for all values below the lower limit and one interval for all values above the upper limit. For each interval, the new discretized value is generated by appending a consecutive number to the prefix.

- Calculate the intervals automatically calculated based on statistical analysis.
The analyst specifies the approximate number of intervals to be generated and the prefix for the new discretized values. Based on the distribution statistics, the system generates the exact number of intervals, the lower and upper value range limits, and the interval width, creating intervals with smooth boundaries and equal widths. This method is especially useful if the range of field values is unknown or if the interval generation is to be based on data located in a different column or table.

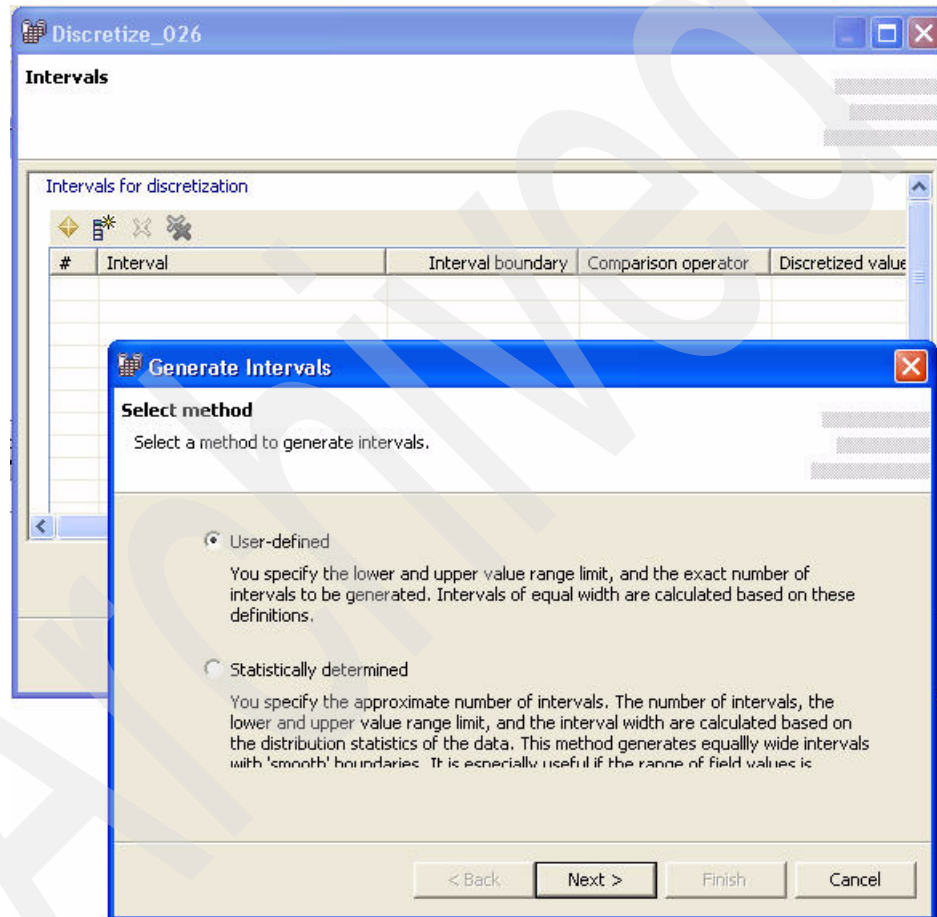


Figure 7-21 Example of selecting interval method for discretize operator

Mining operators

In addition to the table source and table target operators, mining flows use several specific mining operators. These operators perform functions for model visualization, creation, validation, scoring, and results extraction.

The *visualizer* operator displays a data mining model using the appropriate visualizer. Each visualizer provides graphical and tabular information about the model results and model quality, depending on the mining method.

The *model source* operator represents a data-mining model stored in the database. With this operator, an existing model can be placed in a mining flow and used for scoring. This operator is particularly useful when incorporating a third-party PMML model, such as one generated by a SAS tool, into a mining flow.

The *find deviations* operator finds records in the input table that deviate from the majority of records as determined by combinations of characteristics that exist only for very few records. The procedure is based on clustering and uses data in demographic format. Examples of useful deviations that can be discovered in the data include clients who deserve special attention because they buy expensive products of high quality, possibly fraudulent behavior indicated by clients who consistently have high discounts or high numbers of accident claims, or data quality problems indicated by unusual combinations of characteristics. This is a high-level operator that needs only minimal parameter specifications and does not use a visualizer operator. It creates a clustering model that can be viewed by opening it in the Database Explorer, as shown in Figure 7-22.

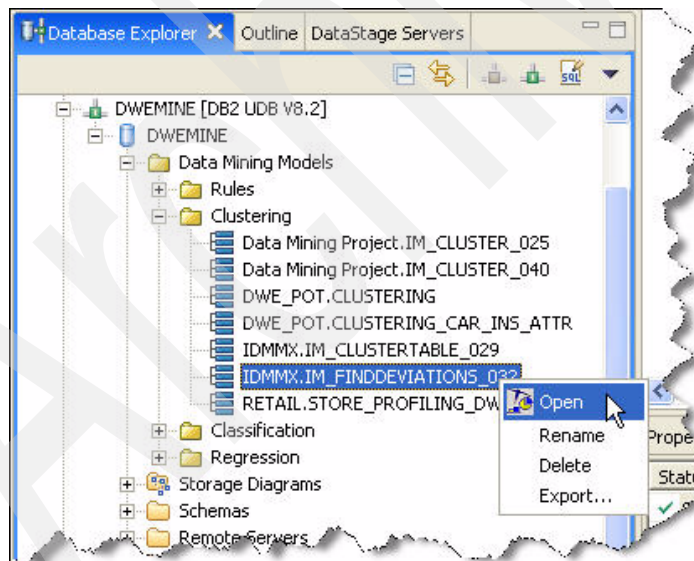


Figure 7-22 Opening a data mining model from the Database Explorer

The *find rules* operator finds relationships in the input table and creates a set of rules describing those relationships. This operator accepts data in either transactional or demographic format. The results consist of a model and a set of

associations rules describing item affinities or relationships among columns. This is a high-level operator that uses minimal parameter specifications and does not use a visualizer operator. The rule model can be viewed by opening it in the Database Explorer.

The *cluster table* operator finds groups with similar characteristics in the input table, using clustering to group similar records. Using data in demographic format, the cluster table operator creates a clustering model and a view of the input records with their respective cluster assignments. This is a high-level operator that uses minimal parameter specifications and does not use a visualizer operator. The clustering model can be viewed by opening it in the Database Explorer.

The *predict column* operator predicts the values of a target field by building a model from the columns in the input table. This operator uses data in demographic format. Depending on whether the target field is categorical or numeric, the resulting model type will be either classification or regression, respectively. This is a high-level operator that uses minimal parameter specifications and does not use a visualizer operator. The classification or regression model can be viewed by opening it in the Database Explorer.

The *associations* operator builds a rule model containing association rules describing relationships among the items (which may be, for example, products, events, and characteristics) occurring together in a transaction. The operator uses an input table in transactional format with a transaction identifier column and an item column. The analyst has extensive control over parameters. The output port of the associations operator flows to the input port of a Visualizer operator, and the model visualization is displayed automatically upon completion of the mining run. The rule model also can be viewed by opening it in the Database Explorer or from the visualizer operator in the mining flow.

The *sequences* operator builds a rule model containing sequence rules describing relationships among items (such as products and events) occurring together across sequential transactions. The operator uses an input table in transactional format with a sequence identifier column (for example, client) in addition to the transaction identifier and item identifier columns. The analyst has extensive control over parameters. The sequences operator pipes the model to a visualizer operator to display the model visualization automatically when the mining run has finished. The rule model can also be viewed in the Database Explorer or from the Visualizer operator in the mining flow.

The *clusterer* operator builds a clustering model that contains groups of similar records. The operator uses an input table in demographic format and allows extensive control over parameters. The clustering model flows to a visualizer operator to display the model visualization automatically when the mining run

has finished. The clustering model can also be opened in the Database Explorer or from the visualizer operator in the mining flow.

The *predictor* operator creates a mining model that predicts the value of a target field (called the class variable in a classification model or the dependent variable in a regression model). Depending on whether the target field is categorical or numeric, the resulting model type will be either classification or regression, respectively. The predictor uses data in demographic format and allows extensive control over parameters. As illustrated in Figure 7-23, a random split operator is used to generate training and testing data sets, passing the training data to the predictor operator to build the model.

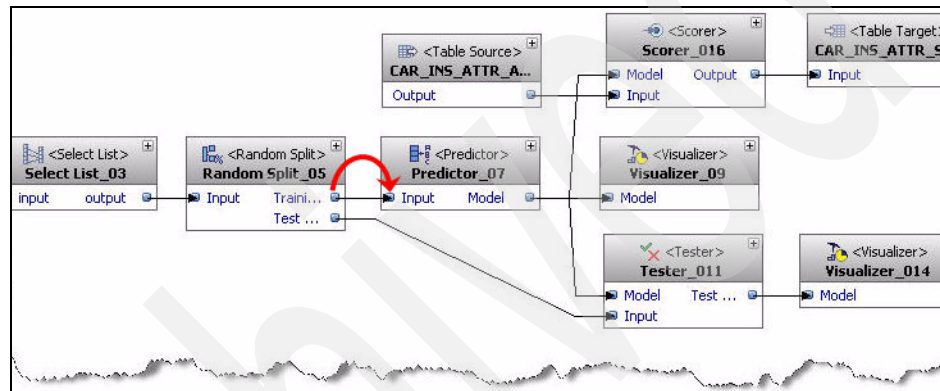


Figure 7-23 Example of predictor operator in a mining flow

The *tester* operator tests a classification or regression model by applying the model to the testing data set and computing test results containing information about model quality (prediction accuracy, ability to rank records correctly, and reliability on new data). As illustrated in Figure 7-24, the tester applies the trained model created by the predictor and applies it to the testing data flowing from the random split operator.

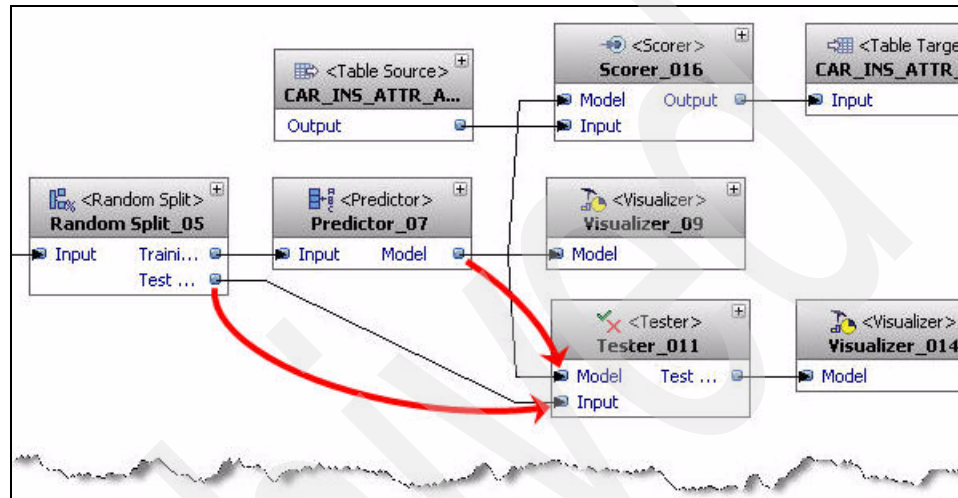


Figure 7-24 Example of tester operator in a mining flow

As shown in Figure 7-24, two visualizer operators are used to display, respectively, the trained model from the predictor and the testing results from the tester. When the mining flow is executed, visualizations for both the training and the testing are generated.

The *scorer* operator applies a mining model to an input table to compute scores for new records. The scorer operator reads a data-mining model from a DB2 table and applies it to the new input data. The scorer is not algorithm-specific, but works with many types of data mining models, as described in 7.4.3 “Scoring” on page 250. Depending on the model type, the scorer creates different scoring results in the output table. The scorer operator is positioned in a mining flow, as illustrated in Figure 7-25. The scorer receives the mining model from the predictor and applies it to the table source representing the *application data* to be scored. The scoring results then are piped to a new table target operator.

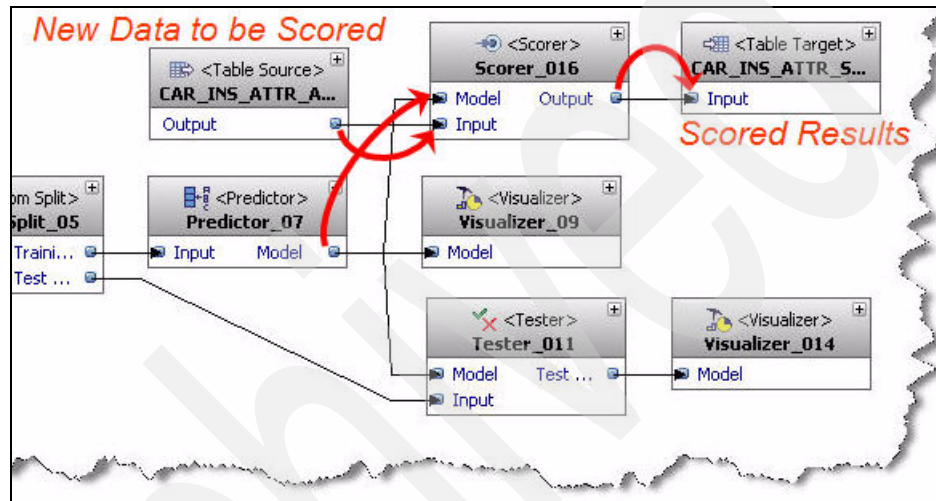


Figure 7-25 Example of scorer operator in a mining flow

The *associations extractor* operator extracts information from an association rules model and stores the extracted rules and related information in an output table using a table target operator. Associations rules stored in relational tables can, for example, be accessed by a reporting tool to generate reports. The associations extractor operator is illustrated in Figure 7-26 on page 274.

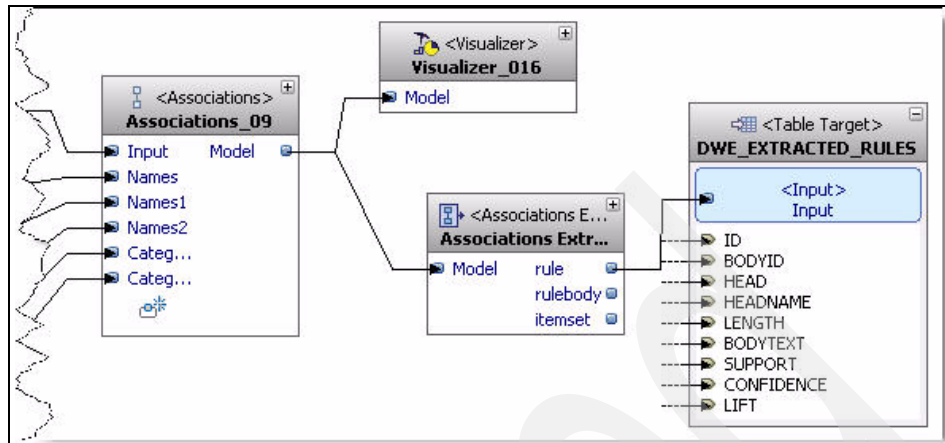


Figure 7-26 Example of associations extractor operator in a mining flow

The *sequences extractor* operator extracts information from a sequences rules model and stores the extracted rules and related information in an output table. Each row in the output table represents one sequence and contains information about the calculated rules, details about each sequence, and the item sets used in the rules. The sequences extractor is positioned in a mining flow similarly to the extractor illustrated in Figure 7-26, although the information contained in the output table is specific to the mining model type.

The *cluster extractor* operator extracts information from a clustering model and stores the extracted information in an output table. Each row in the output table represents one cluster and contains information about each cluster's size, homogeneity of the records in the cluster, and a description of the attributes of the cluster. The cluster extractor is positioned in a mining flow similarly to the extractor illustrated in Figure 7-26.

The *correlations extractor* operator extracts correlations (Pearson's correlation coefficient, cf. listed in "Related publications" on page 557) from a clustering, classification, or regression model and stores them in an output table. Correlation coefficients are calculated for each pair of columns (variables) in the input data used to train the model.

The *fields extractor* operator extracts field information from a mining model and stores the information in an output table. The extracted information includes the set of input data fields used in the model's internal equations and the importance (relative contribution) of each field in the model.

The *tree rules* extractor operator extracts rules (decision paths) from a decision tree classification model and stores the information in an output table. The

extracted information includes the node identifier, scored value and confidence, size, depth, and decision path for each node.

The *gains extractor* operator extracts a table containing gains chart information from a classification or regression model. If the model is a decision tree, then the desired value of the class variable is specified in the gains extractor operator. A gains chart is useful in assessing the quality of a model or in comparing the relative performance of multiple models in predicting the class variable or dependent variable.

The *quality extractor* operator extracts model quality information from a data mining model or a test result and stores the result in an output table. The extracted information is specific to the type of data mining model.

7.5.5 Building a mining flow

In this section we demonstrate how a typical mining flow is built. The assumptions are that one or more data tables have already been created and prepared by a data flow, as part of a domain-specific or project-specific data mart or by some other process, and that these tables are in the correct format (transactional or demographic) required for the mining.

In the example used to illustrate the creation of a mining flow, the business question has to do with identifying insurance clients who are at high risk of attriting (voluntarily terminating or not renewing their policies), so each data record should represent a client with columns representing the client's behavioral and demographic attributes. This means that the input data should be in demographic format. The input table contains a categorical column called ATTRIT with values of Y or N to indicate whether a client has attrited. Given the business question, a decision tree classification model is best suited to predicting attrition.

The first step in building a mining flow is to create a new data warehousing project in the data Project Explorer by selecting **File** → **New** → **Data Warehouse Project** from the menu and stepping through the wizard, as illustrated in Figure 7-27. This new project will contain all of the mining flows, data flows, control flows, and other objects created under it.

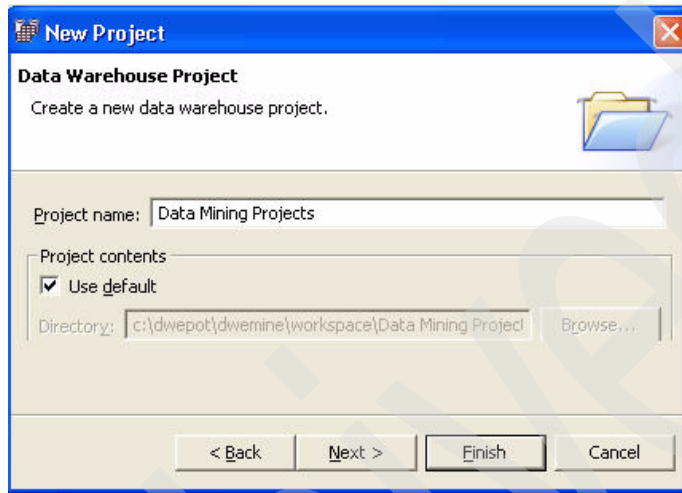


Figure 7-27 Creating a new data warehouse project for mining flows

The next step is to create a new mining flow within the new project, as shown in Figure 7-28. The new mining flow is given a suitable name, and the appropriate database connection is specified. Unlike data flows, mining flows are created and executed with a live connection to a database. Each mining flow can have only one database connection. If a table in a second database must be used, then an alias pointing to the desired table in the second database can be created in the first database.

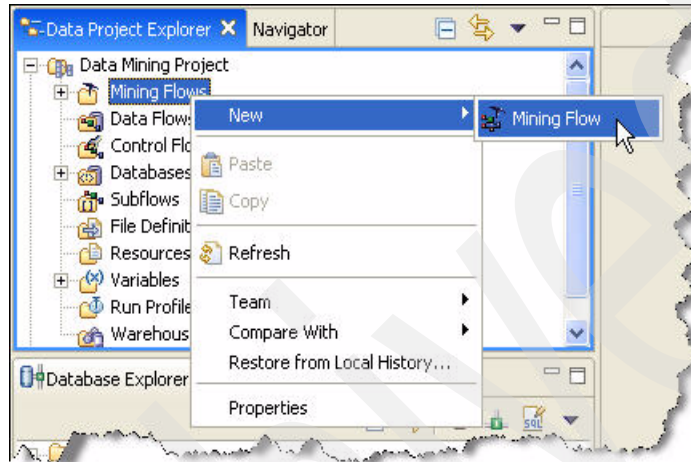


Figure 7-28 Creating a new mining flow

All of the mining flows created under the project are listed under the Mining Flows folder, where each mining flow can be opened by double-clicking it, as shown in Figure 7-29. The mining flow canvas appears in the upper right quadrant of the Design Studio. The canvas initially is blank and ready for the mining flow itself to be created.

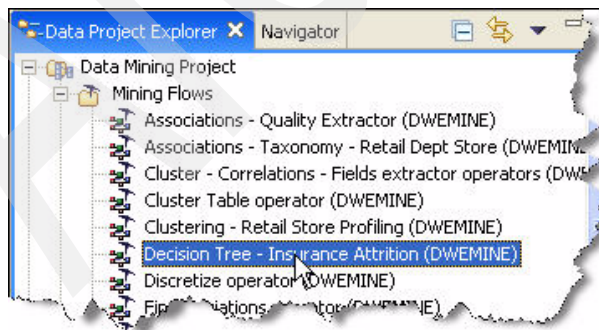


Figure 7-29 Opening a mining flow

Now the construction of the mining flow itself begins. A typical mining flow is depicted in Figure 7-30. Although only some of the many available operators are used, this example illustrates the basics of input data, data preparation, model creation, visualization, and model application. In the rest of this section we discuss each operator in the example mining flow in Figure 7-30. Although every mining flow will vary according to the model type and other requirements driven by the business problem, this example highlights some of the most commonly used mining operators and shows how they are interrelated in the flow.

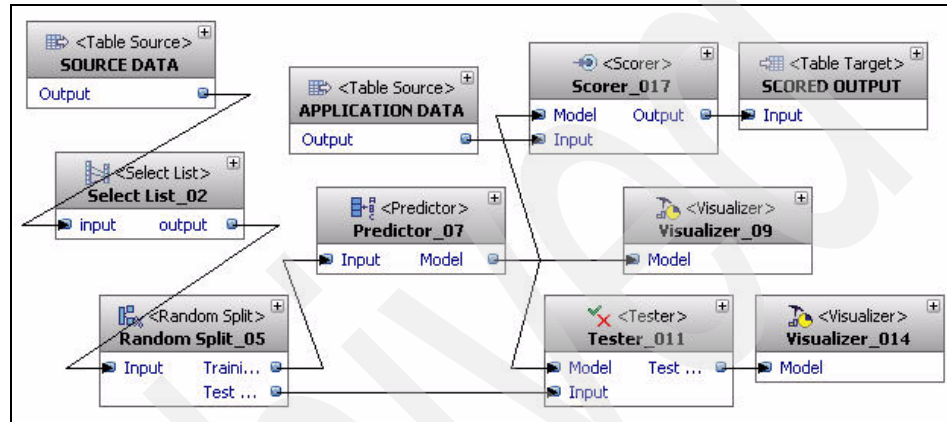


Figure 7-30 Typical mining flow in the DWE Design Studio

Referring to Figure 7-30 on page 278, we begin by creating a table source to represent each input data table. In this example only one table is used as input, but mining flows often use multiple input tables. Here the table source operator is labeled SOURCE DATA. This operator represents the table selected, as shown in Figure 7-31.

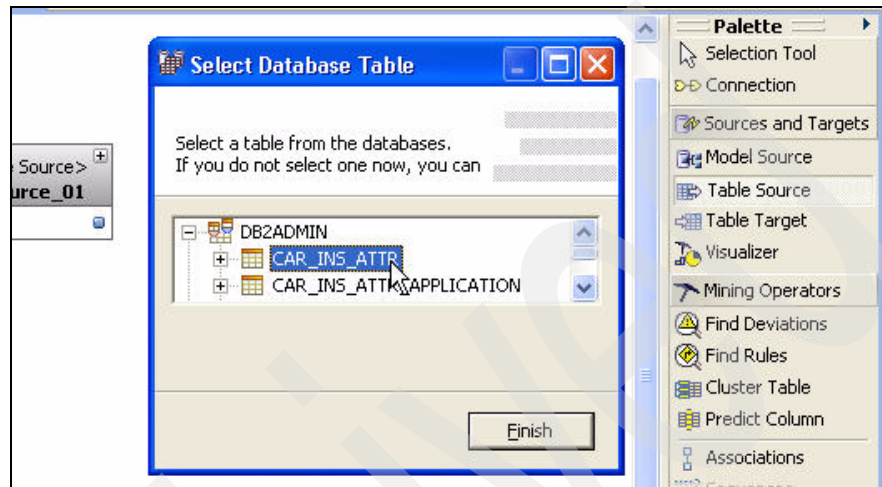


Figure 7-31 Selecting the input table for a new table source

Since the input data table has already been prepared in demographic format, no further preprocessing is needed. Because some of the columns in the input table will not be included as variables in the data mining model, however, a select list operator (labeled Select List_02 in Figure 7-30 on page 278) is used to select the desired subset of columns. In this case, we remove one column (ATTRIT_01) because it is a numeric version of the target field ATTRIT (the class variable in the decision tree model) and is not to be used in the model. As shown in Figure 7-32, unwanted columns are selected and removed by clicking the red X in the select list operator.

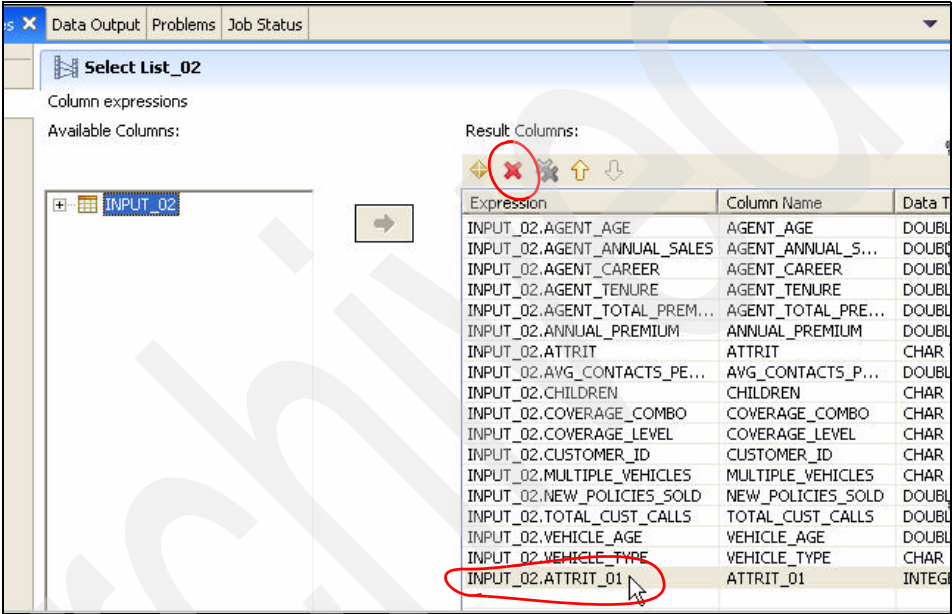
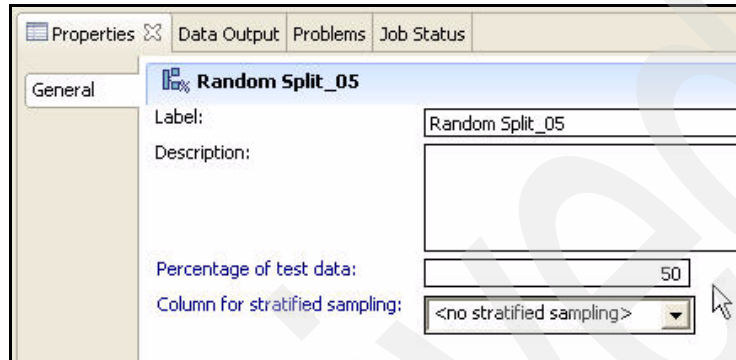


Figure 7-32 Selecting variables for the model in the Select List operator

Because the decision tree is a predictive model, the virtual input table resulting from the select list operator must be randomly split into two virtual tables for training and testing the model. In Figure 7-30 on page 278, the select lister operator flows to a random split operator labeled Random Split_05. In the random split operator, we accept the default parameters of a 50% testing set size (with the complement becoming the training set) and no stratified sampling, as shown in Figure 7-33.

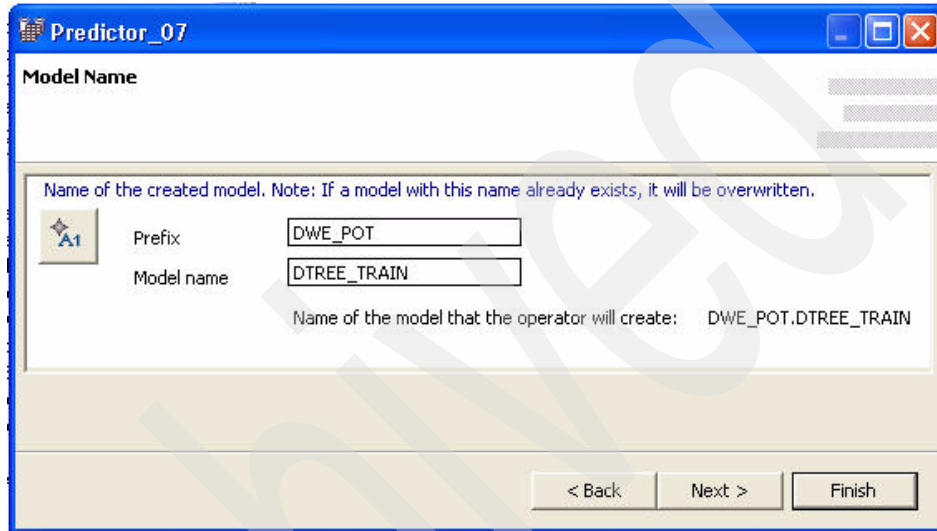


The screenshot shows a software interface for configuring a 'Random Split_05' operator. The interface includes a top navigation bar with tabs: 'Properties', 'Data Output', 'Problems', and 'Job Status'. Below this, a left sidebar contains a 'General' tab. The main area displays the following configuration options:

- Label:** A text field containing 'Random Split_05'.
- Description:** An empty text area.
- Percentage of test data:** A numeric input field set to '50'.
- Column for stratified sampling:** A dropdown menu currently showing '<no stratified sampling>'.

Figure 7-33 Parameter specifications for the random split operator

Next we set up the mining operator to train the model. Since the model is to be a decision tree classification model, we use the predictor operator labeled Predictor_07 in Figure 7-30 on page 278. The training set from the Random Split operator goes to the Predictor operator. The predictor automatically recognizes the target field ATTRIT as categorical and uses the classification kernel to compute the model. The predictor operator is configured with a two-part name of the mining model to be created, as depicted in Figure 7-34.



The screenshot shows a window titled "Predictor_07" with a blue title bar. Below the title bar, the text "Model Name" is displayed. A large text box contains the instruction: "Name of the created model. Note: If a model with this name already exists, it will be overwritten." Below this, there are two input fields: "Prefix" with the value "DWE_POT" and "Model name" with the value "DTREE_TRAIN". To the left of these fields is a small icon with the letter "A1". Below the input fields, the text "Name of the model that the operator will create:" is followed by the value "DWE_POT.DTREE_TRAIN". At the bottom of the window, there are three buttons: "< Back", "Next >", and "Finish".

Figure 7-34 Specifying the model name in the predictor operator

There are also other parameters, including the target field, as depicted in Figure 7-35.

The image shows a software window titled "Predictor_07" with a "Mining Settings" tab. Inside the window, there are several configuration options. The "Target column" is a dropdown menu currently showing "ATTRIT". Below it is an "Optional Parameters" section with an empty list box. Further down, under "Classification specific settings:", there are three input fields: "Maximum purity" with a value of 0, "Maximum depth" with a value of 0, and "Minimum number of records per leaf node" with a value of 0. At the bottom of the window are three buttons: "< Back", "Next >", and "Finish".

Figure 7-35 Specifying the model parameters in the predictor operator

In Figure 7-35, *purity* is the percentage of the modal value of the dependent variable in a decision tree node. When interpreting and applying the decision tree results, the purity of a node represents the likelihood of occurrence of the model value (for example, model value of Y with purity of 75 means a 75% probability that a client will attrit). The maximum purity parameter is a limit to stop further splitting of a node that has reached the specified purity value as the decision tree is being built. Valid values are percentages between 0 and 100. The default value of 0 means that the purity is not constrained. That is, it may go to 100%. *Depth* is the number of levels (nodes) that the decision tree has. The maximum depth parameter is a limit to stop the further splitting of nodes when the specified depth has been reached as the decision tree is being built. The default value of 0 means that the depth is unlimited. *Minimum number of records per leaf node* is the minimum number of records that must be in each leaf node of the generated decision tree. This is a limit to stop further splitting of a node that has reached the specified minimal size as the decision tree is being built. Valid values are positive integers. The default value of 0 means that the minimum value allowed is five records in a node.

The model output port of the predictor operator flows to a visualizer operator labeled Visualizer_09 in Figure 7-30 on page 278. After execution of the mining flow, the visualizer displays the trained model for interpretation and quality assessment.

To generate testing results for model validation and quality assessment for application, the model flows from the predictor to a tester operator labeled Tester_011 in Figure 7-30 on page 278. The testing data flows from the random split operator to the tester, which applies the model to the testing data. The tester operator is configured with a two-part name of the testing result, as illustrated in Figure 7-36.

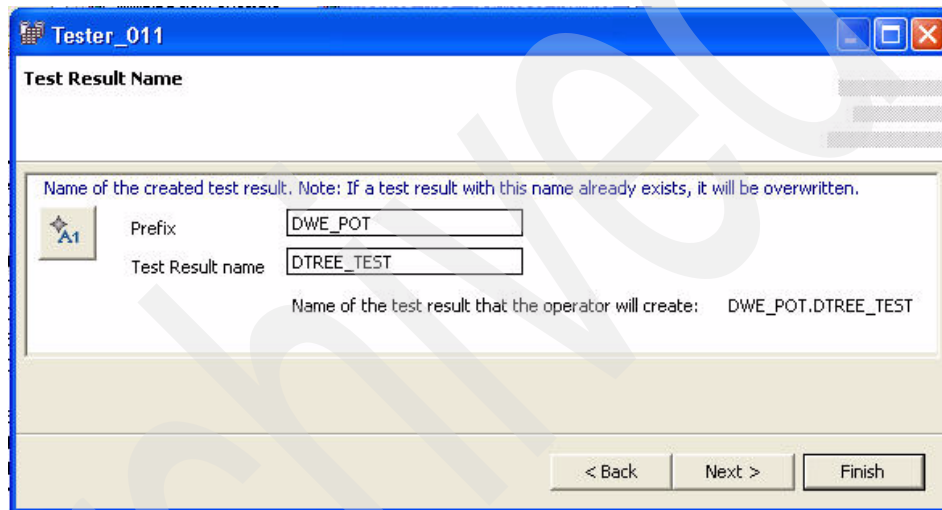


Figure 7-36 Specifying the test result name in the tester operator

The test result output port of the tester operator flows to another visualizer operator labeled Visualizer_14 (Figure 7-30 on page 278). After execution of the mining flow, the visualizer displays the testing result for model quality assessment including the model's reliability in predicting the target field for new records.

The next part of the mining flow is to apply the model to score new records. Another table source operator, labeled APPLICATION DATA in Figure 7-30 on page 278, represents the application data. This table must include all of the columns used as variables in the model with the exception of the target field ATTRIT. The records in this table represent clients who have not yet attrited. The objective is to predict each client's likelihood of attriting so that appropriate actions to improve client retention can be designed and implemented.

To score new records, the model flows from the predictor to a scorer operator labeled Scorer_017 (Figure 7-30 on page 278). The application data flows from the application data table source to the scorer, which applies the model to the application data and scores each record. An output table is needed to receive the scoring results, so a new table is created by right-clicking the output port of the scorer, as illustrated in Figure 7-37.

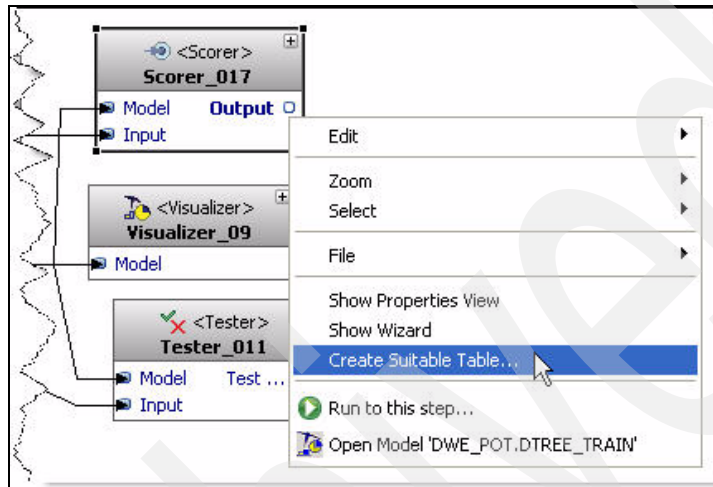


Figure 7-37 Creating a table target to receive scoring results

The new table contains columns for the scoring results and the input columns used in the model, as shown in Figure 7-38. The scoring results for this model include the predicted value of the target field ATTRIT and the confidence of the prediction, based on the modal value and purity of the node into which the record is classified.

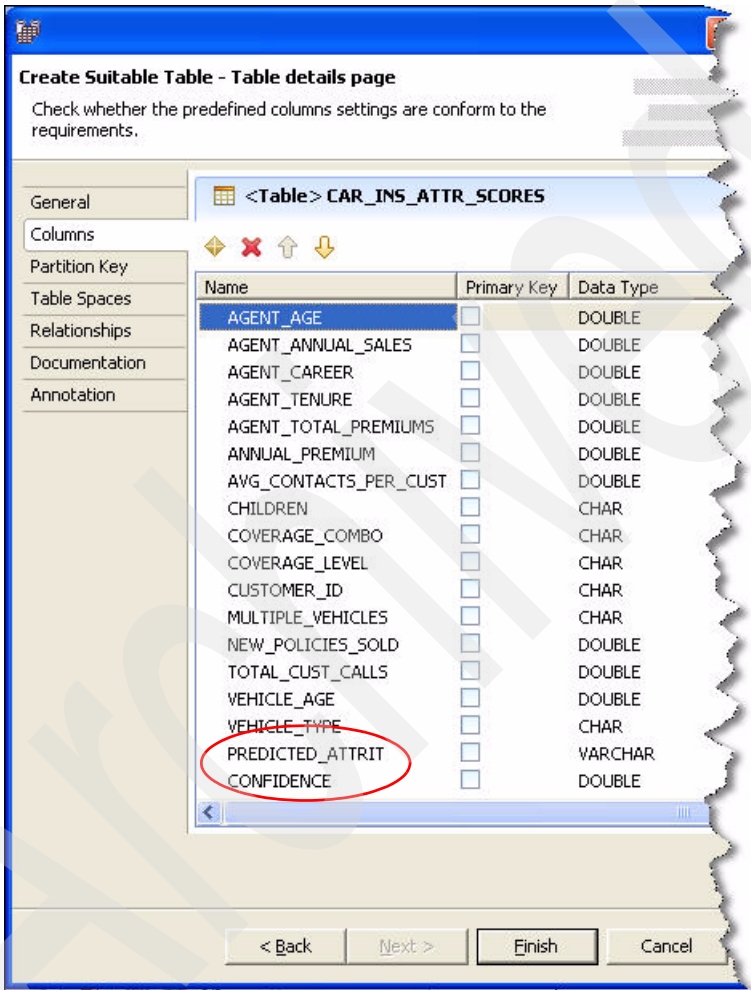


Figure 7-38 Columns created in the scoring output table

A table target operator, labeled SCORED OUTPUT in Figure 7-30 on page 278, is created to represent the new output table. The Table target operator is configured for the new table, as illustrated in Figure 7-39. The output port of the scorer is then connected to the table target operator to complete the mining flow.

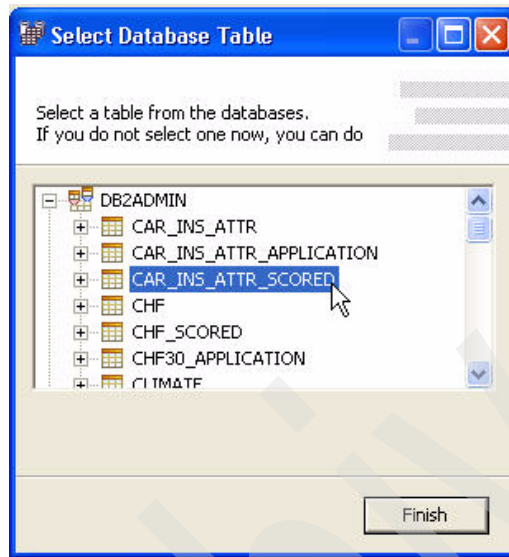


Figure 7-39 Selecting the table for the table target

7.5.6 Validating and executing a mining flow

When a mining flow is executed, all of the operations represented by the operators are executed in the order defined by the flow, as described in 7.5.5 “Building a mining flow” on page 275. The models generated in the flow are stored in PMML format in the appropriate DB2 tables under the schema IDMMX, as described in 7.5.2 “Database enablement” on page 253. Models are applied to new data for scoring, and scored results are written out to target tables. Visualizations of trained models and testing results are displayed to enable the analyst to understand, assess, and fine-tune the models.

Validating a mining flow

Prior to execution, the mining flow can be validated to ensure that operators are properly and completely connected. Although validation is not a prerequisite for execution, it can quickly identify some problems that would cause the execution to fail. A mining flow is validated either by selecting **Mining Flow** → **Validate Flow** from the menu or by clicking the Validate this mining flow icon on the toolbar. Successful validation is indicated by a pop-up message. If errors occur, then the problems can be resolved prior to executing the mining flow. An

incomplete mining flow may be validated during construction to verify the flow up to that point. In this case, some errors can be expected and can usually be ignored until the complete flow is validated.

Executing a mining flow

The completed mining flow is executed either by selecting **Mining Flow** → **Execute** from the menu or by clicking the Execute this mining flow in the database icon on the toolbar. An execution selection pop-up offers the choice of executing the complete mining flow (the default) or executing the mining flow step by step. The latter choice is particularly useful when debugging an execution error. The complete mining flow is executed by simply accepting the default execution method and clicking the **Execute** button, as shown in Figure 7-40.

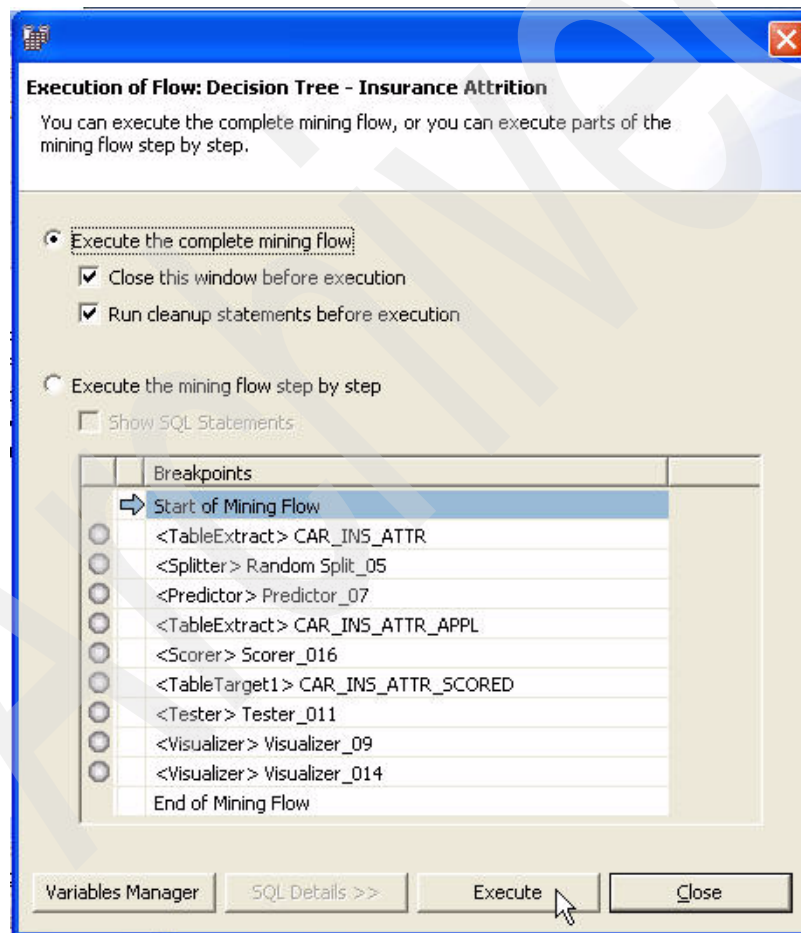


Figure 7-40 Executing a complete mining flow

The mining flow can be executed step-by-step by selecting that execution method and then clicking the radio buttons for the desired steps. For example, in Figure 7-41, three steps (random split, predictor, tester) have been selected. The mining flow will initially be executed up to the random split operator. If execution to that point is successful, then execution is resumed until the predictor is reached. If successful to that point, execution is resumed up to the tester, and then finally to the end of the mining flow. If an execution error occurs, this step-by-step execution process is helpful in isolating the point at which the error occurs. If desired as an additional aid in debugging the mining flow, the SQL for the mining run can be displayed by checking the Show SQL Statements box.

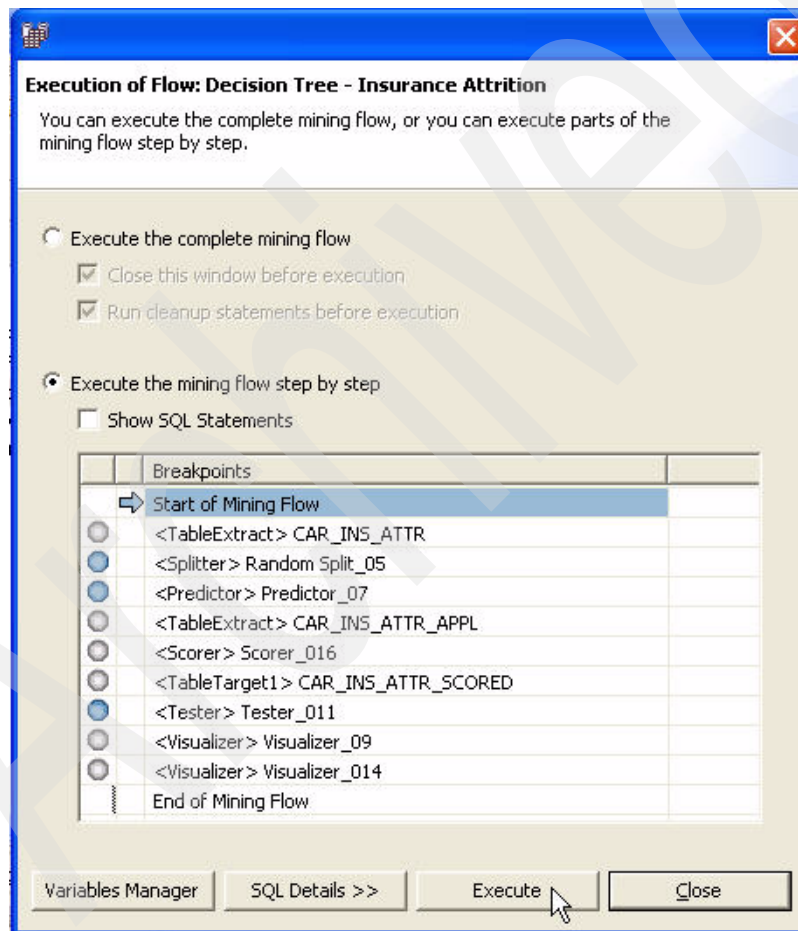


Figure 7-41 Executing a mining flow step-by-step

In addition to executing a complete mining flow, a selected portion of a mining flow can be executed without having to execute the entire flow. This feature is useful, for example, when verifying a portion of a mining flow during construction or rerunning a model after changing a parameter. As illustrated in Figure 7-42, a mining flow is executed up to a selected step by right-clicking the desired operator and selecting **Run to this step** from the pop-up. If the selected operator links to other operators not directly in the branch, then warnings may appear but can usually be ignored unless they indicate an error.

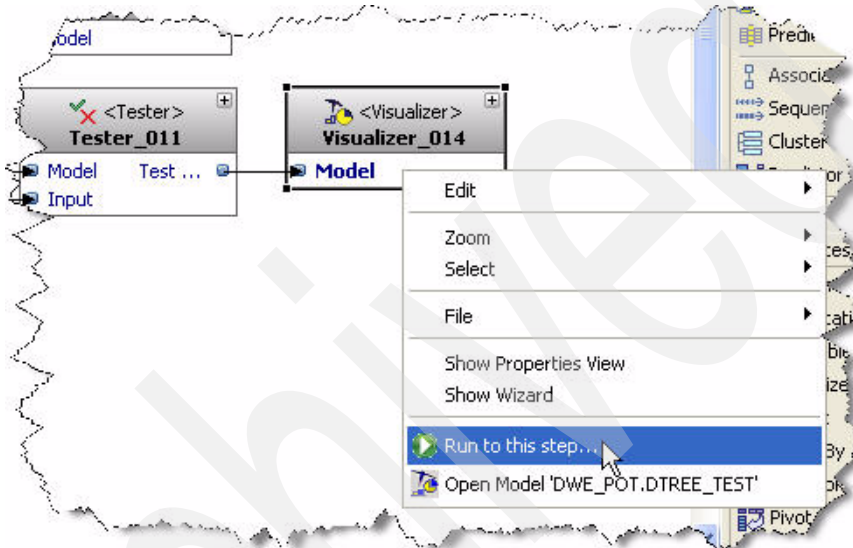


Figure 7-42 Executing a mining flow up to a selected step

Executing mining in other flows

Certain data mining modeling operators (Find Deviations, Find Rules, Cluster Table, Predict Column) are available in data flows. This enables some data-mining models to be created and applied within an operational data flow (for example, during nightly data loading operations, to refresh a set of models based on newly processed transactions and to apply those models to update clients' scores for use by client-facing applications).

A mining flow can be deployed by incorporating it into a control flow, thereby sequencing the mining operations with data flows and other data processing activities, as discussed in Chapter 6, “Data movement and transformation” on page 137. A simple example of a control flow with an embedded mining flow is shown in Figure 7-43. The mining flow is represented in the control flow sequence by a mining flow operator selected from the palette and placed onto the canvas. The mining flow is defined as a property of the mining flow operator. Execution of the control flow invokes the execution of the mining flow.

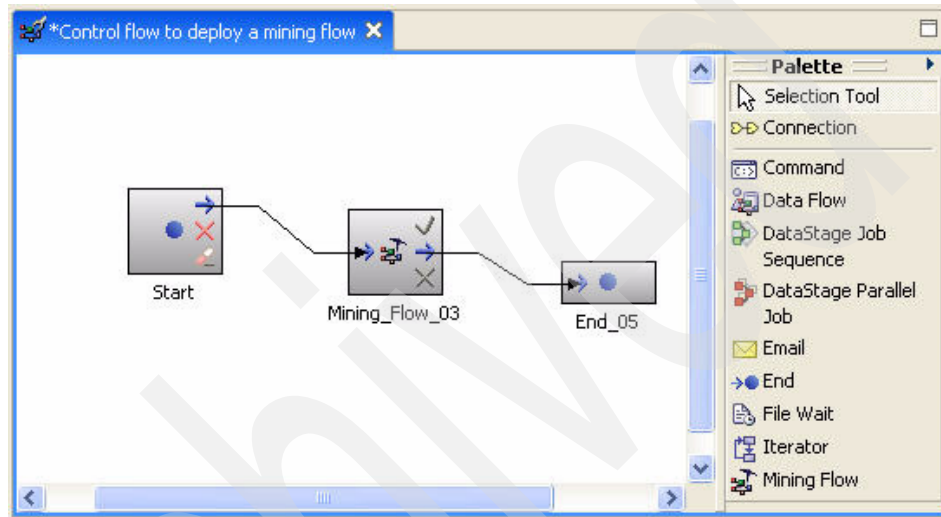


Figure 7-43 Control flow to deploy a mining flow

7.5.7 Visualizing models and test results

As discussed in 7.2.6 “Interpreting and evaluating the data mining results” on page 244, each visualizer presents technique-specific information about the model or testing result to enable the analyst to evaluate the model quality and performance. Visualizations generated by mining flows can be viewed in the Design Studio by any of the following three methods:

- ▶ When executing a mining flow
 - All visualizations generated during a mining flow are automatically displayed. For example, for a mining flow that includes training and testing functions, such as the flow illustrated in Figure 7-30 on page 278, the visualizations for the trained model and the test result are both displayed.

- From a visualizer operator in a previously executed mining flow

A visualization of a model or test result previously generated in a mining flow can be opened from its visualizer operator, as illustrated in Figure 7-44. The visualization is opened by right-clicking the visualizer operator and selecting **Open Model *name*** from the pop-up.

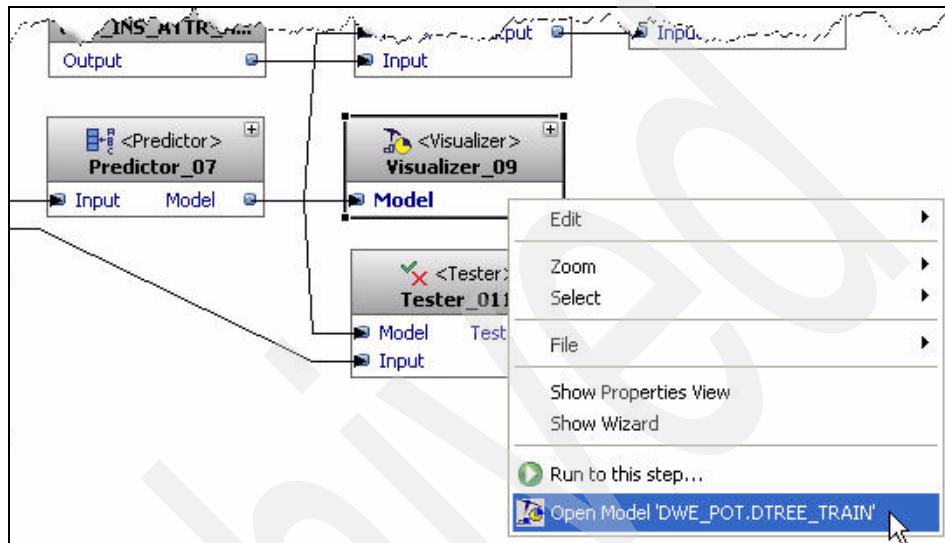


Figure 7-44 Opening a visualization from within a mining flow

► From the Database Explorer

A visualization of a model or test result previously generated in a mining flow can be opened from the Database Explorer, as illustrated in Figure 7-45. The visualization is opened by expanding the database's folder structure to the appropriate folder (rules, clustering, classification, regression) under the Data Mining Models folder and then either double-clicking the desired model or right-clicking and selecting **Open** from the pop-up.

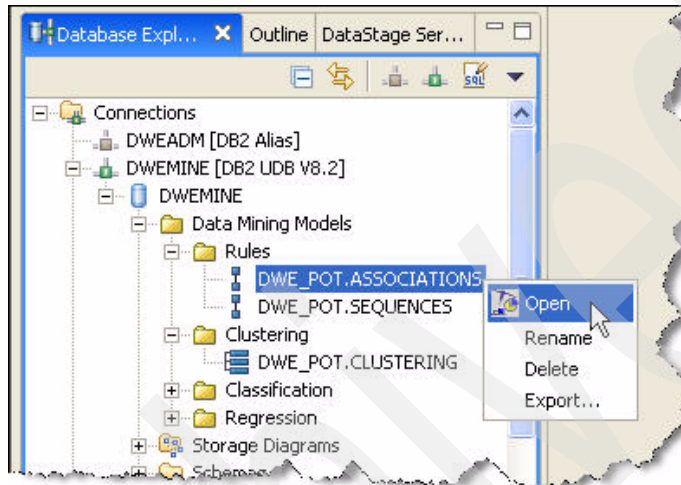


Figure 7-45 Opening a visualization from the Database Explorer

Each visualizer presents information specific to the mining method used. Thus, interpretation of the modeling or testing results also depends on the mining method. Explanations of each view in a visualization are available through the help function on the menu bar in the DWE Design Studio. Extensive documentation on all of the visualizers as well as modeling and scoring is provided in the DB2 Business Intelligence section of the DB2 Information Center installed as part of DWE.

Clustering visualization

To illustrate the process of interpreting a visualization, we present a clustering example. The business problem for this example is to design the layouts (proportion of floorspace allocated to each department) of several new stores that a nationwide retail department store chain is planning to build in the southern U.S. Specifically, the objective is to determine which departments should be emphasized or de-emphasized as a percentage of total store floorspace to best serve climate-dependent purchasing preferences of the clients in the target region. This problem is common in the retail industry and can be

approached as an application of clustering to perform an analysis called *store profiling*.

For store profiling analysis, we need variables representing each store's revenue by department as a percentage of that store's total revenue, thereby creating a sales performance measure for each department. For a given store, if the percentage sales for a particular department is greater (less) compared to all stores, then that store is considered to over perform (under perform) in that department.

The mining flow for the clustering model is shown in Figure 7-46. The mining flow points to the source table containing one record per store in the nationwide chain, with columns representing sales performance by department, other store information (climate for its geographic location and how long the store has been open), and demographic information about the client base for each store. These columns are used as variables in the clustering model. The visualizer generates a visualization of the model.

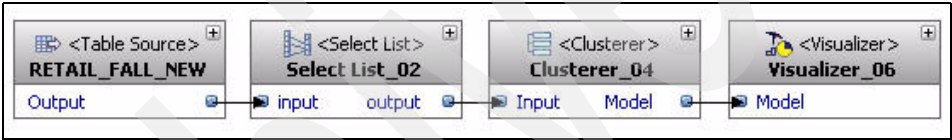


Figure 7-46 Clustering mining flow example

The main view of the visualization generated by the mining flow is shown in Figure 7-47. The clusters are displayed as rows sorted by size (percentage of records in each cluster). Within each cluster, the variables are sorted in descending order of importance (as measured by a chi-square statistic) in distinguishing the members of that cluster from all other stores. Because the objective of clustering is to segment individuals into homogeneous groups that are different from other groups, we interpret clusters by comparing each variable's distribution for a given cluster to its distribution for all stores.

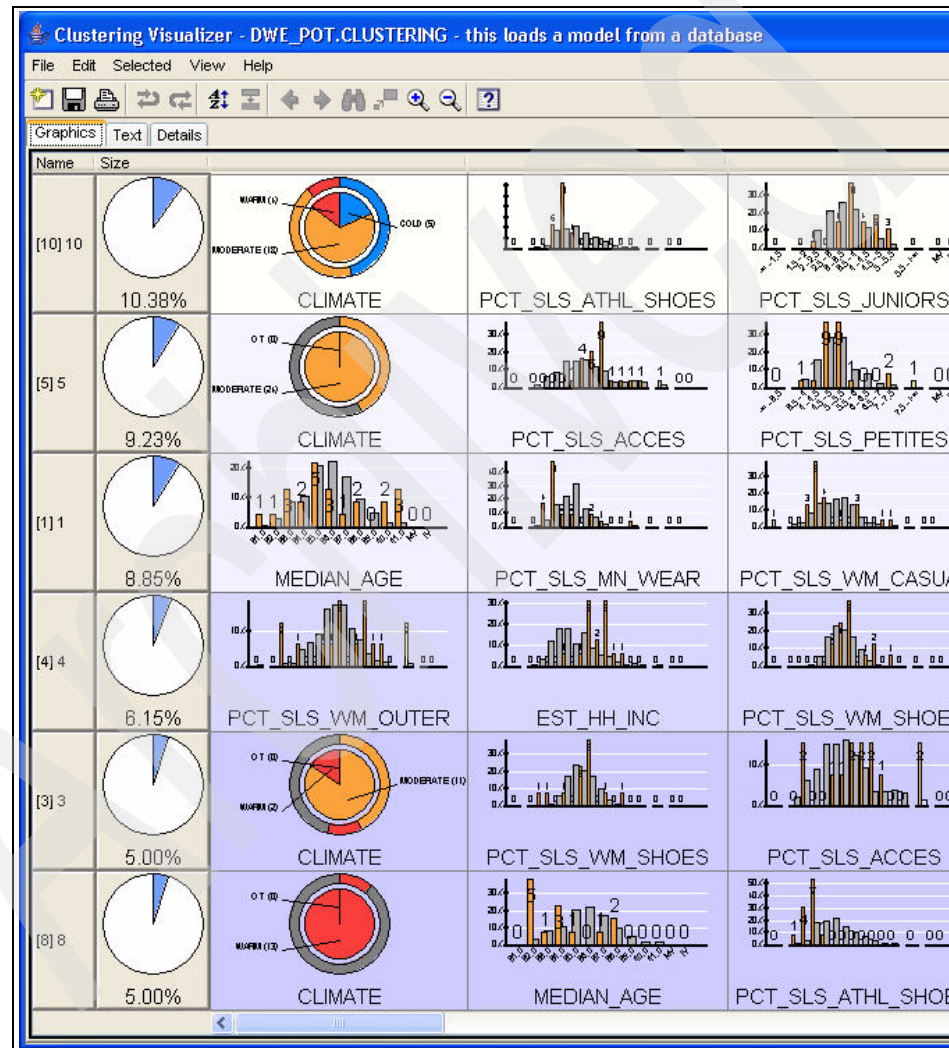


Figure 7-47 Clustering visualization example

For a categorical variable, the distributions are represented by concentric pie charts, as illustrated for a variable called CLIMATE in Figure 7-48. The inner circle represents a particular cluster, while the outer ring represents all stores as a whole. In this example, we see that 67% of the stores in this cluster are located in moderate-climate cities, compared to 43% of all stores. Fifteen percent of the stores in this cluster are located in warm-climate cities, compared with 11% of all stores. Only 18% are in cold-weather cities, compared to 46% of all stores. Thus, we interpret this cluster as being comprised primarily of stores in moderate and warm climates, in contrast to all stores in the nationwide chain.

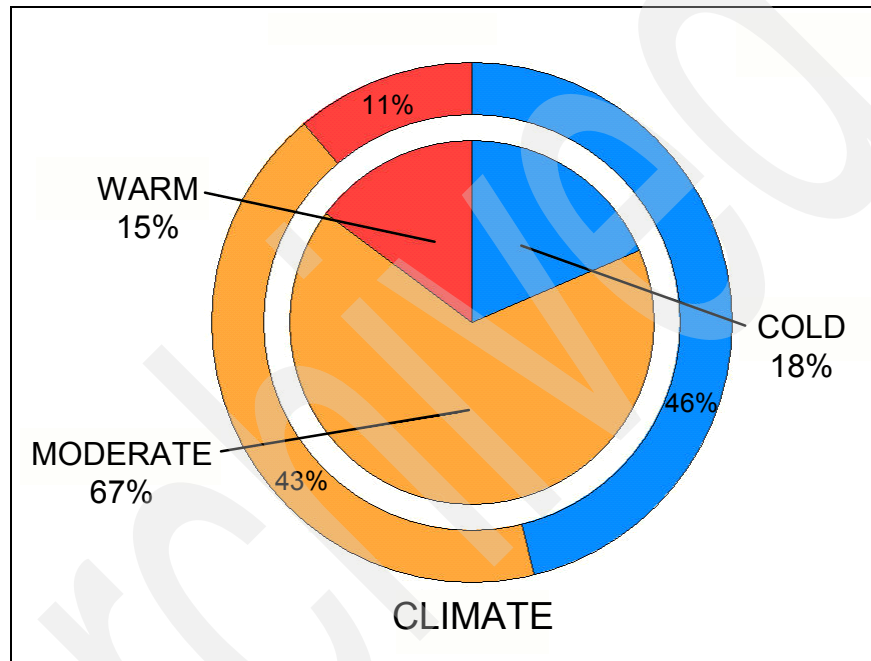


Figure 7-48 Representation of distributions of a categorical variable

For a numeric variable, the distributions are represented by percentage histograms, as illustrated for a variable called PCT SALES ATHL SHOES in Figure 7-49 on page 297. The foreground histogram (consisting of three bars labeled 22%, 63%, and 15%) represents a particular cluster, while the background histogram represents all stores as a whole. In this example we see that the stores in this cluster derive, on average, about 7–8% (that is, between 6% and 9%, as shown by the three bars) of their revenue from this department, compared to an average of around 8–9% for all stores (the approximate centroid of the background histogram). Thus, we interpret this cluster as being comprised of stores that, on average, derive less of their revenue from this department (that is, under perform), in contrast to all stores.

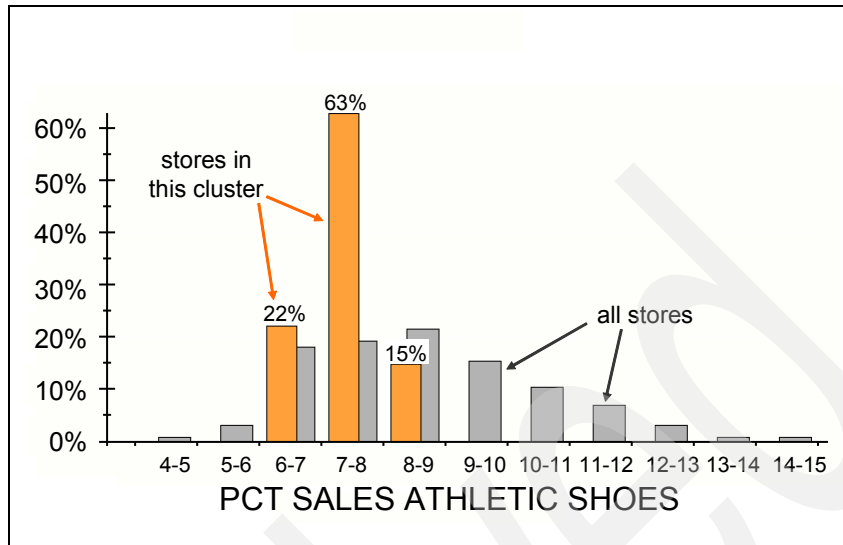


Figure 7-49 Representation of distributions of a numeric variable

When examining a clustering model, the differences among the clusters can readily be discerned by visually comparing the distributions of the most important (highest-ranked) variables across clusters. The relative sizes of the inner (cluster) and outer (all records) values for categorical variables and the relative positions of the foreground (cluster) and background (all records) histograms for numeric variables across the top several variables provide a readily interpretable overview of the distinct characteristic profiles represented by the clusters. In addition, the variables can be ordered in various ways, for example, alphabetically or by average values of a particular variable, or displayed as pie charts, histograms, or tables to assist with interpreting the model. Many other options such as colors and chart displays are available as well.

In this example we are interested in the expected purchasing patterns for the planned stores. We search the clusters to find the one that appears to best represent the new stores and hence can serve as a pattern for allocating floor space among departments. Retailers know that climate is a strong predictor of clients' purchasing behavior, so we look for a cluster having a predominance of warm-climate stores. In this case, cluster 8 (the last cluster in Figure 7-47 on page 295) meets this criterion as well as other relevant criteria such as relatively new stores, as shown in Figure 7-50 on page 298. (Note that labels have been added to assist with interpreting this cluster.) In terms of the business question, we see that stores in cluster 8 tend to under perform in certain departments (labeled Low Performance) and to over perform in other departments (labeled High performance) relative to other stores, as explained for Figure 7-49. These

characteristics indicate which departments to emphasize or de-emphasize in allocating floor space.

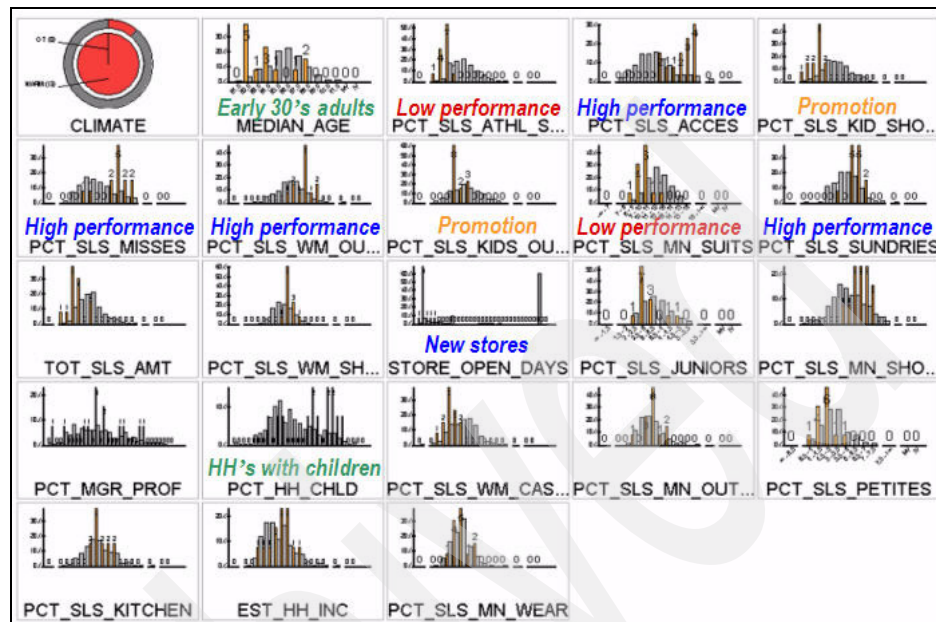


Figure 7-50 Clustering visualization example - drill down to one cluster

As is often the case with data mining, we may find other opportunities in addition to answering the primary business question. In this case, we see from the demographic attributes in Figure 7-50 that the client base for the stores in cluster 8 consists mostly of adults in their early 30s and households with children and relatively high household income. This finding suggests promotional opportunities in departments such as children's shoes and children's outer wear. Other clusters may also reveal valuable marketing and merchandising opportunities not directly related to the primary business question.

Associations visualization

The associations visualizer presents a set of rules describing affinities among items or higher levels of taxonomy that occur together in a transaction, as discussed in 7.1.1 "Discovery data mining" on page 238; 7.2.2 "Understanding the data model" on page 241; and 7.2.4 "Extracting and preparing the data" on page 242. The associations visualization includes tables of affinity rules, the item sets on which the rules are based, a graph of the rules, and a table of statistics about the transactions data, mining parameters, and rules. The graph and tables are interactive to aid the analyst in focusing on rules of interest for a particular business objective.

In this example, the rules table is illustrated in Figure 7-51. The rules can be sorted by any column in this table, and various filtering capabilities are available to enable the analyst to focus on a subset of item relationships of interest. When a taxonomy is specified in the associations model, rules relating entities across various levels of the taxonomy (for example, item → department) can be found.

Association Visualizer - DWE_POT.ASSOCIATIONS - this loads a model from a data

File Edit Selected View Help

Rules Item Sets Graph Statistics

Visible rules:

| Rule | Support | Confidence | Lift | All |
|---|---------|------------|--------|-----|
| [WWM WOVEN TOPS] ==> [MENS_WEAR] | 2.4711% | 40.2704% | 0.8668 | |
| [WTURTLENECKS] ==> [MENS_WEAR] | 2.3705% | 49.8414% | 1.0728 | |
| [WWM ACCESSORIES] ==> [WOMENS_WEAR] | 2.0865% | 60.2760% | 1.6125 | |
| [WSWEATERS] ==> [MENS_WEAR] | 1.6968% | 42.0299% | 0.9046 | |
| [MTURTLENECKS] ==> [WOMENS_WEAR] | 1.6466% | 40.0367% | 1.0711 | |
| [WWM ACCESSORIES] ==> [MENS_WEAR] | 1.4077% | 40.6681% | 0.8753 | |
| [MENS_WEAR]+[CHILDRENS_WEAR] ==> [WOMENS_WEAR] | 1.0784% | 50.7092% | 1.3566 | |
| [WOMENS_WEAR]+[CHILDRENS_WEAR] ==> [MENS_WEAR] | 1.0784% | 49.4810% | 1.0650 | |
| [WSLEEPWEAR] ==> [MENS_WEAR] | 1.0784% | 42.7717% | 0.9206 | |
| [MRUGBY] ==> [WOMENS_WEAR] | 0.9729% | 40.1452% | 1.0740 | |
| [MENS_WEAR]+[WWM ACCESSORIES] ==> [WOMENS_WEAR] | 0.9377% | 66.6071% | 1.7819 | |
| [WOMENS_WEAR]+[WWM ACCESSORIES] ==> [MENS_WEAR] | 0.9377% | 44.9398% | 0.9673 | |
| [0307385C1WM 50/50 COT/POLY T-NECK] ==> [MENS_WEAR] | 0.8321% | 52.3734% | 1.1273 | |
| [WACTIVE BOTTOMS] ==> [MENS_WEAR] | 0.7843% | 42.4490% | 0.9137 | |
| [MSLEEPWEAR] ==> [WOMENS_WEAR] | 0.7516% | 43.7135% | 1.1694 | |
| [WOMENS_WEAR]+[HOUSEWARES] ==> [MENS_WEAR] | 0.6988% | 51.4815% | 1.1084 | |

Figure 7-51 Associations visualization example - rules

The table of item sets used in constructing the rules is shown in Figure 7-52. This table is particularly useful for selecting a particular item set (containing one or more items) and displaying the rules and graph just for that item set. For example, if the analyst wants to focus on women's casual wear (the highlighted item set in the table in Figure 7-52) for a promotion, then he can select that item set and display only the rules containing women's casual wear.

Association Visualizer - DWE_POT.ASSOCIATIONS - this loads a model from a data

File Edit Selected View Help

Rules Item Sets Graph Statistics

Visible item sets:

| Item Set | Support | In Rules as Body | In Rules as Head | Item |
|-------------------------------------|----------|------------------|------------------|------|
| [MENS_WEAR] | 46.4605% | 0 | 420 | |
| [WOMENS_WEAR] | 37.3806% | 0 | 319 | |
| [MDRESS SHIRTS] | 10.7692% | 0 | 24 | |
| [MKNIT SHIRTS] | 8.1222% | 0 | 12 | |
| [WWM CASUAL WEAR] | 7.4208% | 0 | 22 | |
| [CHILDRENS_WEAR] | 7.2775% | 0 | 3 | |
| [WKNIT SHIRTS] | 6.4555% | 0 | 2 | |
| [MSPORTSHIRTS] | 6.1413% | 0 | 3 | |
| [WWM WOVEN TOPS] | 6.1362% | 1 | 4 | |
| [OLUGGAGE] | 5.2262% | 0 | 2 | |
| [MMN ACCESSORIES] | 5.2011% | 0 | 2 | |
| [WWM KNITS] | 5.0679% | 0 | 1 | |
| [MSWEATERS] | 4.9372% | 0 | 1 | |
| [WTURTLENECKS] | 4.7562% | 1 | 6 | |
| [MTURTLENECKS] | 4.1126% | 1 | 1 | |
| [WSWEATERS] | 4.0372% | 1 | 0 | |
| [BED_BATH] | 3.6199% | 0 | 1 | |
| [WDRESSES] | 3.5420% | 0 | 2 | |
| [WWM ACCESSORIES] | 3.4615% | 2 | 0 | |
| [WSLEEPWEAR] | 2.5214% | 1 | 0 | |
| [MRUGBY] | 2.4233% | 1 | 0 | |
| [WOMENS_WEAR]+[CHILDRENS_WEAR] | 2.1795% | 1 | 0 | |
| [MENS_WEAR]+[CHILDRENS_WEAR] | 2.1267% | 1 | 0 | |
| [WOMENS_WEAR]+[WWM ACCESSORIES] | 2.0865% | 1 | 0 | |
| [WWM CASUAL WEAR]+[WWM ACCESSORIES] | 1.9877% | 1 | 0 | |

Figure 7-52 Associations visualization example - item sets

A graphical representation of the rules table is shown in Figure 7-53. As the figure clearly illustrates, displaying all the rules at once can make for a complicated graph.

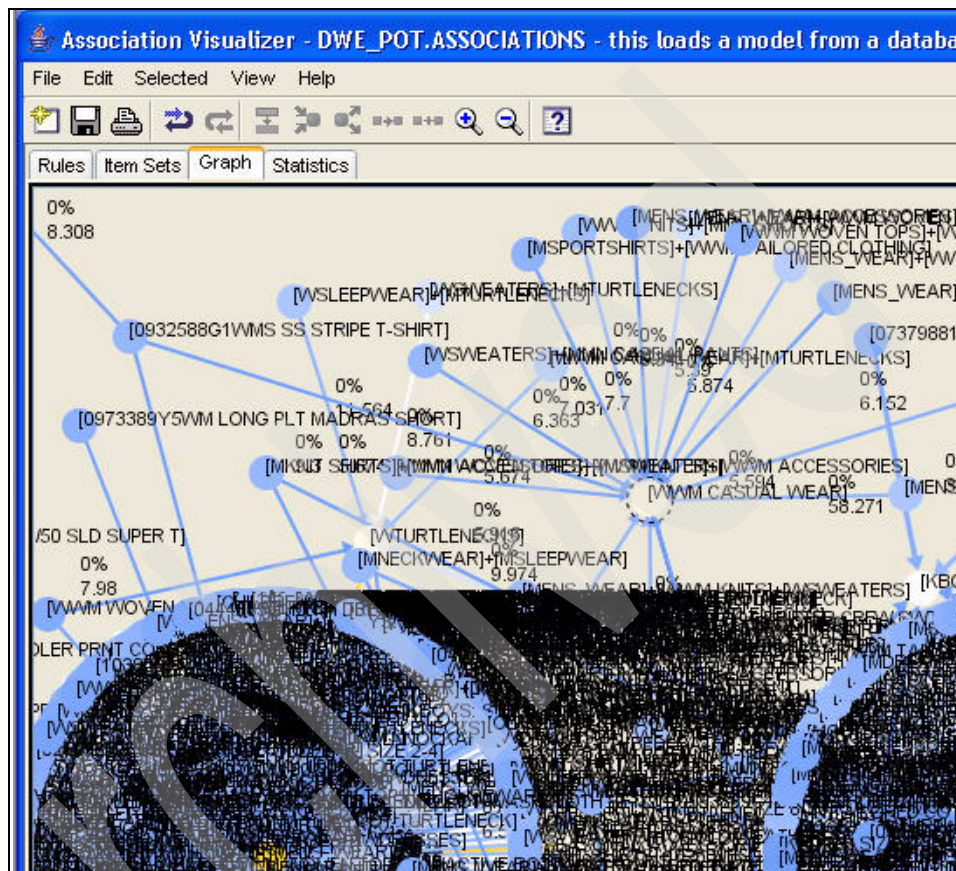


Figure 7-53 Associations visualization example - graph of all rules

But when a subset of rules of interest for a particular business objective is selected, the combination of the graph and rules table becomes much more useful. For example, if the objective is to identify a set of items associated with the women's casual wear subdepartment for an upcoming promotion, then the women's casual wear item set can be selected, as explained for Figure 7-52 on page 300. Then only the rules containing women's casual wear are displayed in the rules table and in the graph, as shown in Figure 7-54.

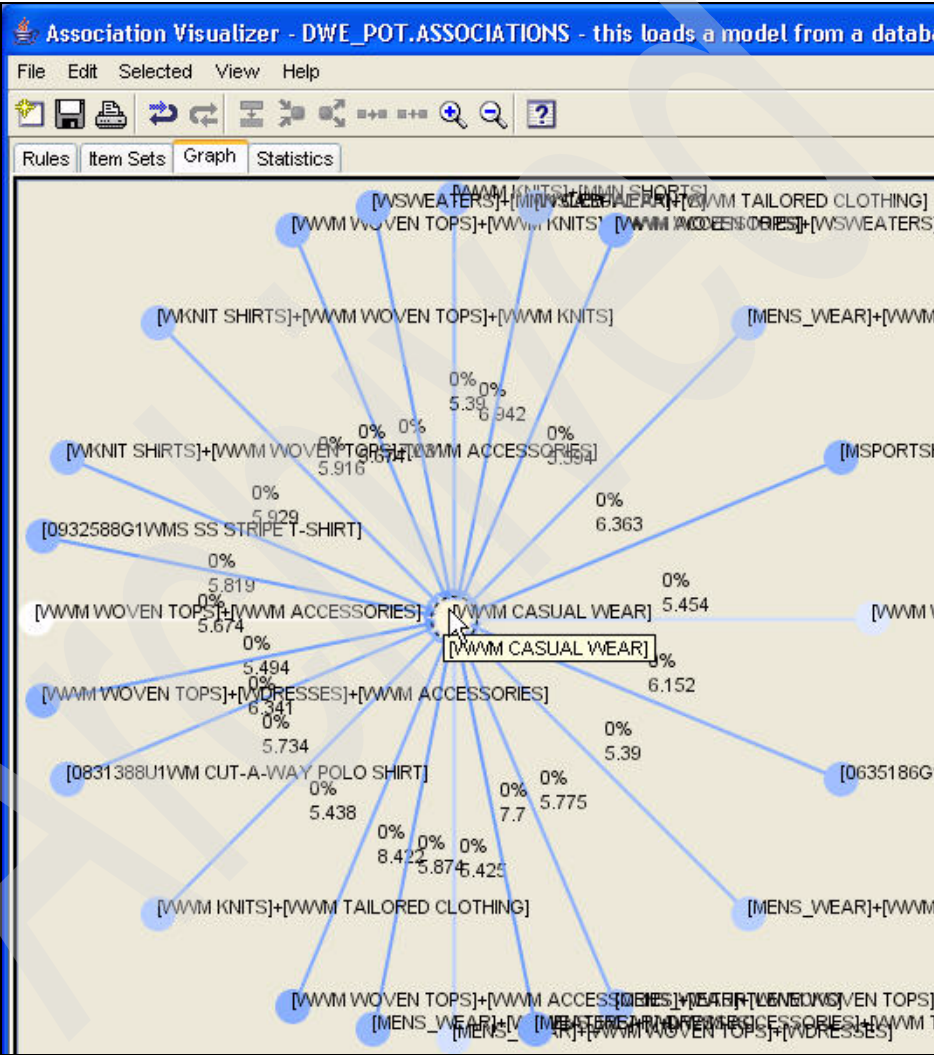


Figure 7-54 Associations visualization example - graph of rules for a selected item set

Once the relevant subset of rules has been found, the analyst can evaluate those rules in terms of the business objective. The rules involving women's casual wear are shown in Figure 7-55. The first rule shows that when the item *solid D-ring belt* is purchased, then an item in the women's casual wear subdepartment will also be purchased in 45% of those transactions (per the Confidence column). Thus, this item might be put on sale to draw clients who will then also purchase related items in women's casual wear.

| Rule | Support | Confidence | Lift |
|---|---------|------------|--------|
| [0635186G1VWM SOLID D-RING BELT] ==> [WWM CASUAL WEAR] | 0.0528% | 45.6522% | 6.1519 |
| [0831388U1VWM CUT-A-WAY POLO SHIRT] ==> [WWM CASUAL WEAR] | 0.0503% | 42.5532% | 5.7343 |
| [0932588G1VMS SS STRIPE T-SHIRT] ==> [WWM CASUAL WEAR] | 0.0402% | 42.1053% | 5.6739 |
| [WWM KNITS]+[MMN SHORTS] ==> [WWM CASUAL WEAR] | 0.0603% | 52.1739% | 7.0308 |
| [MSPORTSHIRTS]+[WWM TAILORED CLOTHING] ==> [WWM CASUAL WEAR] | 0.0427% | 47.2222% | 6.3635 |
| [WWSWEATERS]+[MMN CASUAL PANTS] ==> [WWM CASUAL WEAR] | 0.0402% | 42.1053% | 5.6739 |
| [WWM WOVEN TOPS]+[WWM ACCESSORIES] ==> [WWM CASUAL WEAR] | 0.2388% | 40.7725% | 5.4943 |
| [WWM WOVEN TOPS]+[WWM SWMMWEAR] ==> [WWM CASUAL WEAR] | 0.0855% | 40.4762% | 5.4544 |
| [WWM KNITS]+[WWM TAILORED CLOTHING] ==> [WWM CASUAL WEAR] | 0.0578% | 40.3509% | 5.4375 |
| [WWSLEEPWEAR]+[WWM TAILORED CLOTHING] ==> [WWM CASUAL WEAR] | 0.0402% | 40.0000% | 5.3902 |
| [WWM WOVEN TOPS]+[WWM ACCESSORIES]+[WTURTLENECKS] ==> [WWM ...] | 0.0503% | 62.5000% | 8.4223 |
| [MENS_WEAR]+[WWM ACCESSORIES]+[WWM TAILORED CLOTHING] ==> [W... | 0.0402% | 57.1429% | 7.7003 |
| [WWM WOVEN TOPS]+[WSWEATERS]+[WTURTLENECKS] ==> [WWM CASUA...] | 0.0427% | 51.5152% | 6.9420 |
| [WWM WOVEN TOPS]+[WDRESSES]+[WWM ACCESSORIES] ==> [WWM CASU...] | 0.0402% | 47.0588% | 6.3415 |
| [WWM KNITS]+[WWM WOVEN TOPS]+[WWM KNITS] ==> [WWM CASUAL WEAR] | 0.0000% | 0.0000% | 6.0000 |

Figure 7-55 Associations visualization example - rules for a selected item set

Sequences visualization

The sequences visualizer presents a set of rules describing a series of transactions that occur sequentially over time, as discussed in 7.1.1 “Discovery data mining” on page 238; 7.2.2 “Understanding the data model” on page 241; and 7.5.4 “Data mining operators” on page 262. The sequences visualization includes tables of sequence rules, sequences and item sets on which the rules are based, a graph of the rules, and a table of statistics about the rules. In this example, views of the sequence rules are depicted in Figure 7-56.

| Sequence Rules | | | Sequences | Item Sets | Statistics |
|------------------|--|--|-----------|-----------|------------|
| Table sequences: | | | | | |
| | Sequence | | | | Support |
| 00 | [083148833VWM SLVLESS BTTN TANK TOP] >>> [0307385C1VWM 50/50 COT/POLY T-NECK] | | | | 0.9690% |
| 40 | [030508511UNI COLLEGE STRIPE RUGBY] >>> [0025485A1MN LS 60/40 OXF SHIRT] | | | | 0.9690% |
| 56 | [0622987N1VWM T-SHIRT DRESS] >>> [0307385C1VWM 50/50 COT/POLY T-NECK] | | | | 0.9690% |
| 77 | [0307485C1MN 50/50 COT/POLY T-NECK] >>> [0307385C1VWM 50/50 COT/POLY T-NECK] >>> [030... | | | | 0.7750% |
| | [0307385C1VWM 50/50 COT/POLY T-NECK] >>> [0622887N1VWM TANK DRESS] >>> [0307385C1VWM... | | | | 0.9690% |
| 53 | [0534786A1MN CABLE SHETLAND SWTR] | | | | 0.9690% |
| 93 | [0295285A1MN 80/20 PP OXF SHIRT] + [0307385C1VWM 50/50 COT/POLY T-NECK] | | | | 0.9690% |
| 98 | [0295285A1MN 80/20 PP OXF SHIRT] >>> [0799988G1VWM RELAXED POLO SHIRT] | | | | 0.7750% |
| 46 | [030508511UNI COLLEGE STRIPE RUGBY] >>> [0799988G1VWM RELAXED POLO SHIRT] | | | | 0.7750% |
| 39 | [0271685K1VWM SERIOUS SWEAT PANTS] + [0210885K1VWM SERIOUS SWEAT CREWNECK] | | | | 0.9690% |
| 56 | [0416785C1UNI 2 1/4 PINSTRIPE RUGBY] >>> [0307385C1VWM 50/50 COT/POLY T-NECK] | | | | 0.9690% |
| 31 | [0257185C1VMS SS SLD INTERLOC SHIRT] >>> [0799988G1VWM RELAXED POLO SHIRT] | | | | 0.7750% |
| 96 | [0210885K1VWM SERIOUS SWEAT CREWNECK] | | | | 0.9690% |
| 98 | [0307085C1VWM 100% COT TURTLENECK] >>> [0307385C1VWM 50/50 COT/POLY T-NECK] >>> [030... | | | | 0.7750% |
| 91 | [0307485C1MN 50/50 COT/POLY T-NECK] >>> [0257185C1VMS SS SLD INTERLOC SHIRT] | | | | 0.9690% |
| 15 | [0307385C1VWM 50/50 COT/POLY T-NECK] >>> [051478641VWM WINDRUNNER PANTS] + [05148864... | | | | 0.9690% |
| 32 | [0307385C1VWM 50/50 COT/POLY T-NECK] >>> [0307085C1VWM 100% COT TURTLENECK] >>> [002... | | | | 0.7750% |
| | [0307385C1VWM 50/50 COT/POLY T-NECK] >>> [0307485C1MN 50/50 COT/POLY T-NECK] >>> [030... | | | | 0.9690% |

Figure 7-56 Sequences visualization example - sequences table

The graphical representation of those sequence rules is illustrated in Figure 7-57.

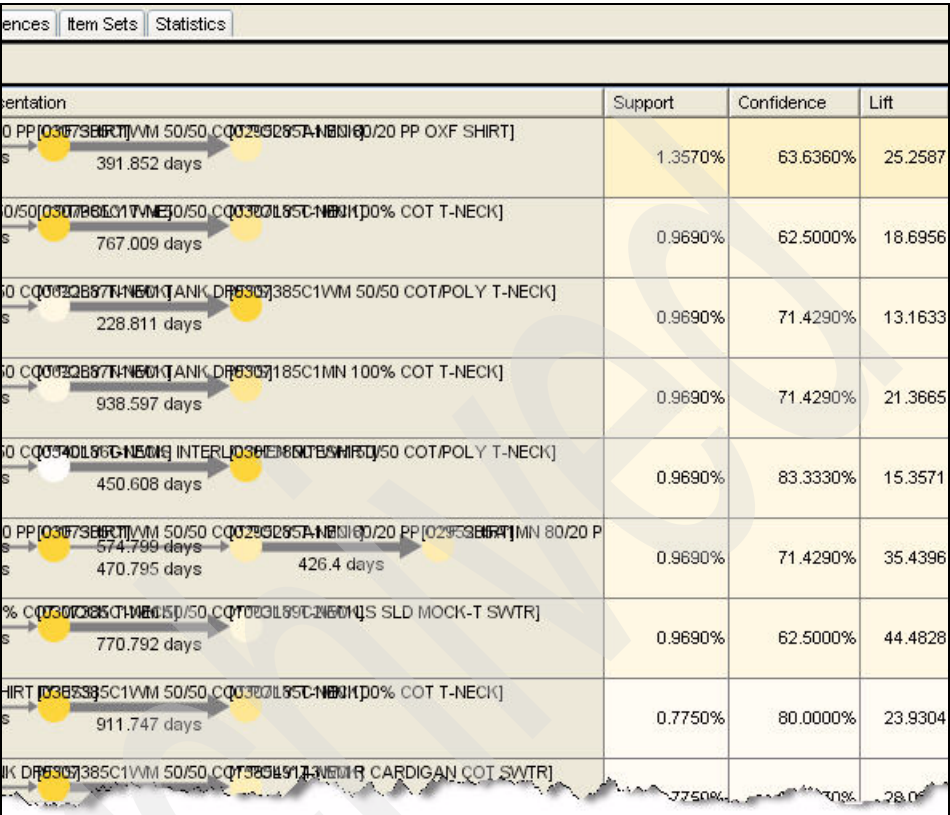


Figure 7-57 Sequences visualization example - sequence rules

Classification visualization

The classification visualizer presents information to evaluate a decision tree classification model and its testing result, as discussed in 7.1.2 “Predictive data mining” on page 239 and 7.5.5 “Building a mining flow” on page 275 and illustrated in Figure 7-30 on page 278. Here we consider an example of predicting the risk that a client will voluntarily close his account. That is, that he will attrit. The response variable ATTRIT has possible values of Y (yes) and N (no).

In a split screen view, the decision tree model is shown on the left in Figure 7-58. On the right, the decision path for the highlighted tree node is displayed. An individual whose attributes result in his being classified in this node receives a score of the dominant value (Y in this node) with the confidence indicated by the purity (84% for Y in this node). Thus, a client classified into this node is considered to have an 84% likelihood of attriting. This view of the decision tree model is taken from the visualization of the trained model.

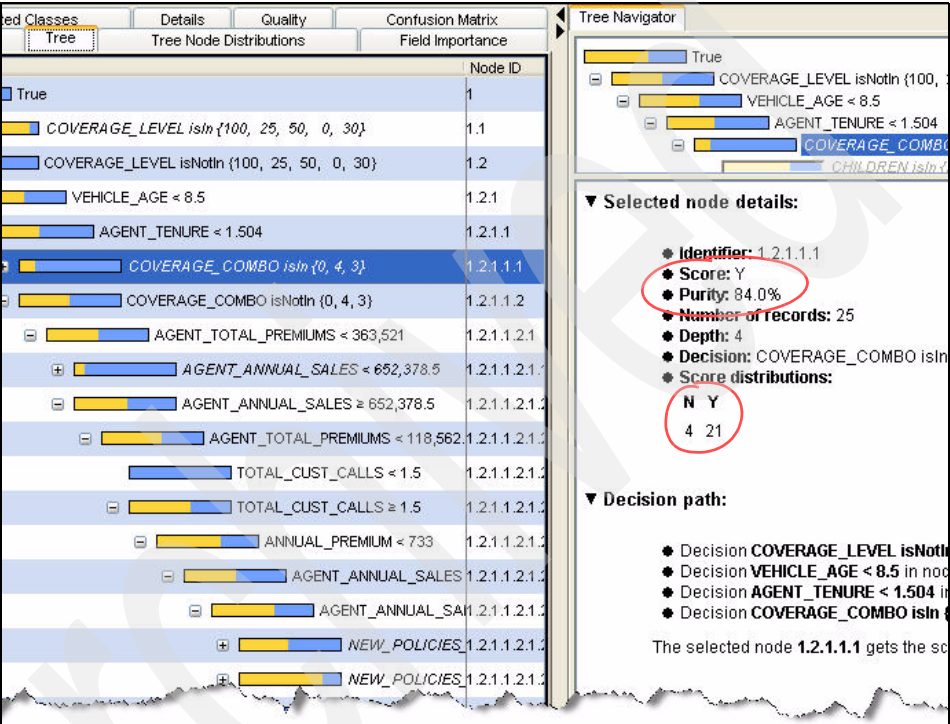


Figure 7-58 Classification visualization example - decision tree and decision path

Another useful graph from the visualization of the trained model is the field importance chart illustrated in Figure 7-59 on page 307. This graph shows the relative contribution of each of the explanatory variables used in the model to predict the ATTRIT response. In this case, the variable COVERAGE_LEVEL is the most important predictor of attrition, followed by VEHICLE_AGE, ANNUAL_PREMIUM, and so on. The variables AGENT_AGE, TOTAL_CUST_CALLS, and NEW_POLICIES_SOLD contribute relatively little to explaining attrition, given the other explanatory variables in the model. Any variable that is highly correlated with one of the variables used in the model will automatically be excluded and will not appear in the field importance chart.

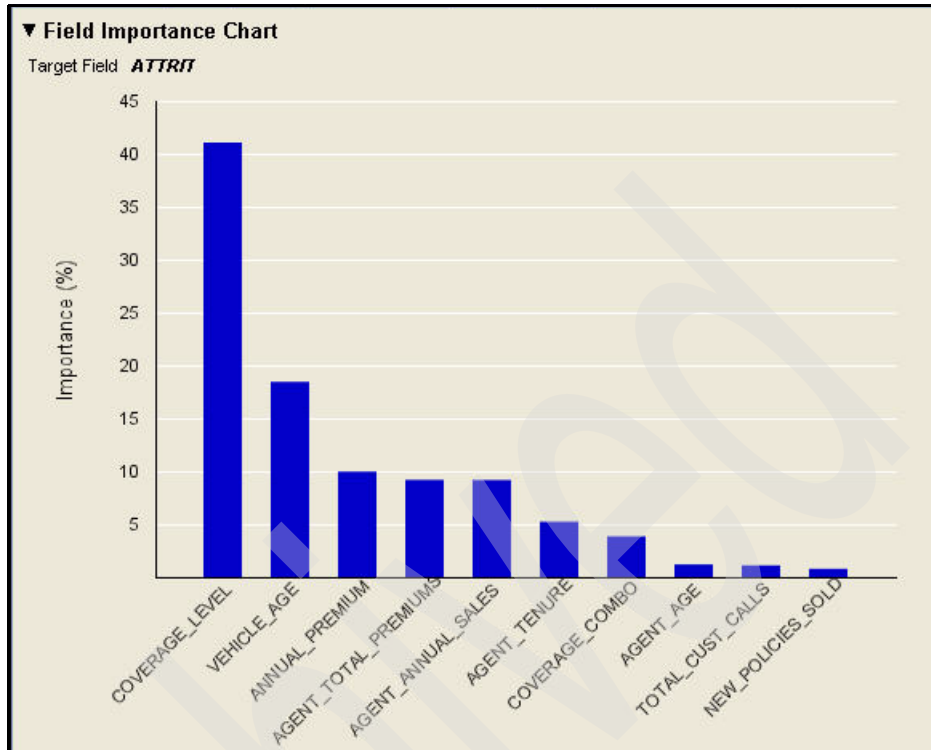


Figure 7-59 Classification visualization example - field importance chart

Model quality measures are provided in the visualizations of both the trained model and the testing results. The quality based on the testing results, illustrated in Figure 7-60, is more relevant for how the model can be expected to perform with new data.

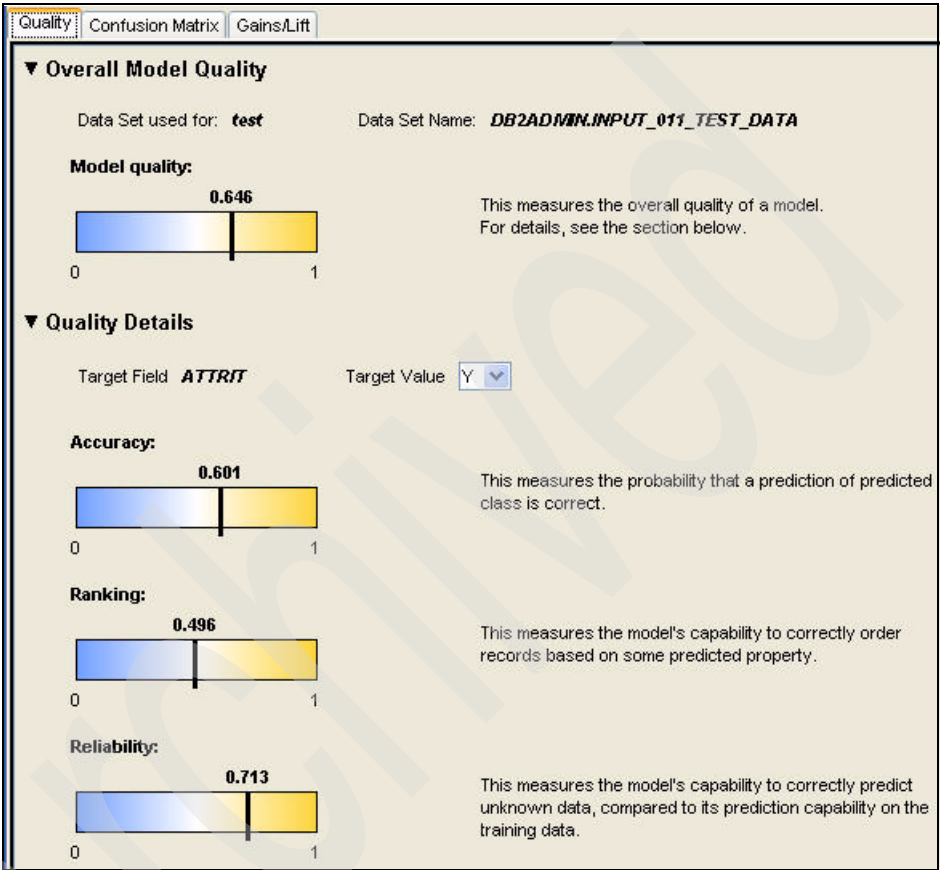


Figure 7-60 Classification visualization example - model quality

The gains curve is a graphical representation of the model's performance. The gains curve based on the testing data is represented by the solid curve in Figure 7-61. The greater the degree of curvature of the gains curve relative to the straight, dashed line below the gains curve, the better the model's performance in correctly identifying attriters. (The light dashed line above the gains curve represents the upper bound theoretically achievable by a model with perfect accuracy.)

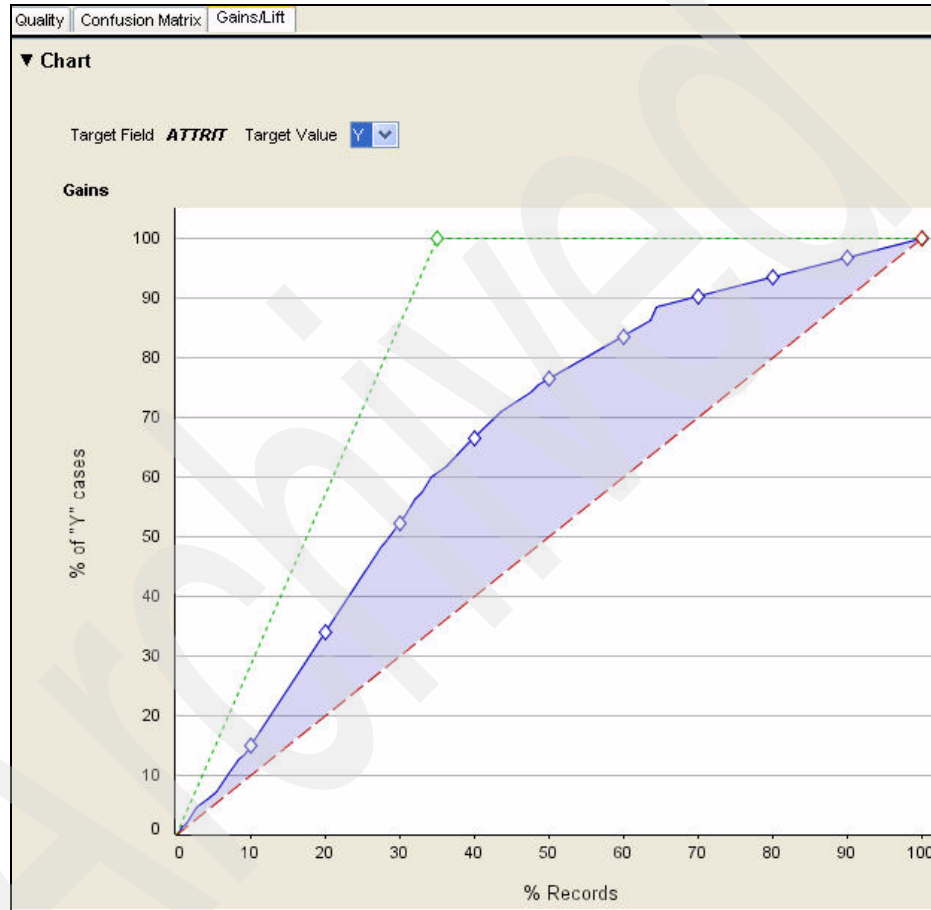


Figure 7-61 Classification visualization example - gains chart

Regression visualization

The regression visualizer presents information to evaluate a regression model and its testing result. Similar to the classification visualizer, the regression visualizer reports model quality, field importance, and a gains curve.

7.5.8 Scoring with other PMML models

Data mining models created by tools other than DWE Modeling can be used in mining flows to score new data. For example, an analyst may create a data mining model using a statistical tool capable of exporting data mining models in PMML format. This model can be exported from the statistical tool, imported into a DB2 table in the DWE Design Studio, and incorporated into a mining flow for scoring. This capability is useful for leveraging high-performance DWE Scoring with data mining models not generated by DWE Modeling.

Once a data mining model has been created and exported in PMML format to a file system, it is imported into a DB2 database using the Database Explorer. The folder for the appropriate model type is selected under the Data Mining Models folders. For example, a logistic regression model is imported into the Classification folder, as illustrated in Figure 7-62.

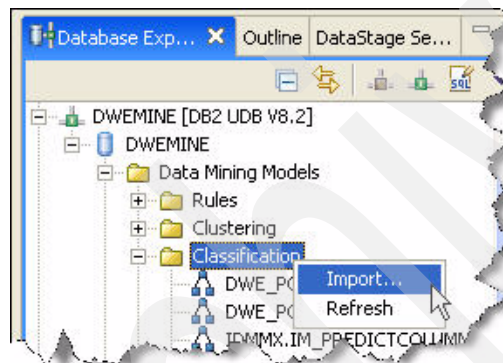


Figure 7-62 Importing a PMML model into the database

The PMML model is specified in a model source operator by creating a model source operator from the palette, as illustrated in Figure 7-63. The operator can also be created by dragging the PMML model onto the canvas from the Database Explorer.

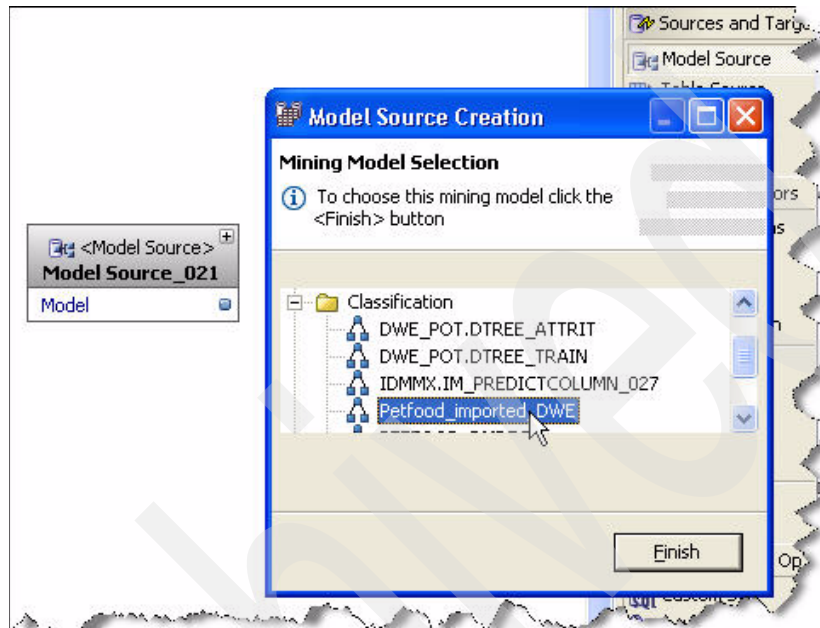


Figure 7-63 Selecting a PMML model in a Model Source operator

The mining flow containing the PMML model as the model source is illustrated in Figure 7-64. When the mining flow is executed, the scoring results are stored in the output table specified in the table target operator.

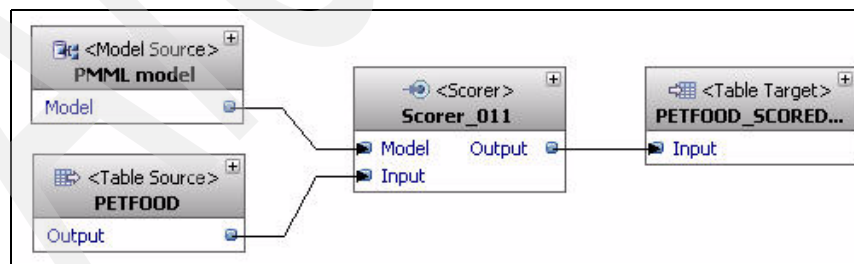


Figure 7-64 Mining flow with a model source operator for scoring

7.5.9 Managing data mining models

Data mining models residing in DB2 tables under the schema IDMMX can be opened for viewing, renamed, deleted, or exported in PMML format through the Database Explorer, as shown in Figure 7-65.

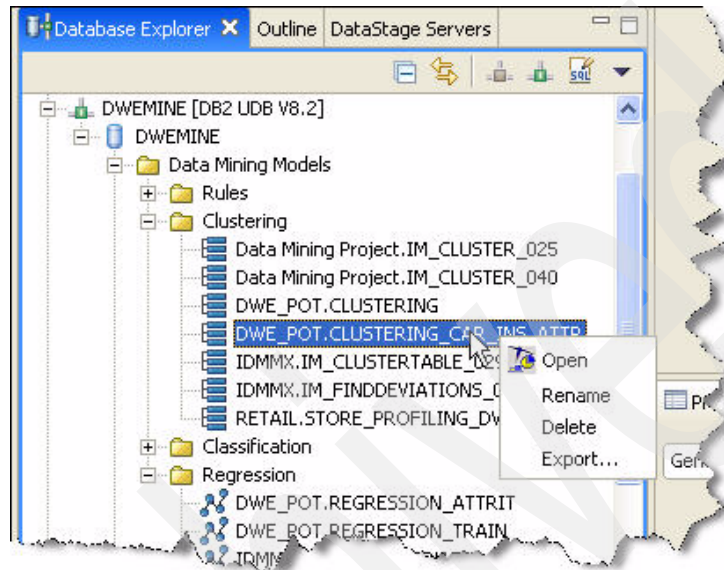


Figure 7-65 Managing data mining models

As illustrated in Figure 7-66, a model can be exported to another workspace or to the file system in PMML format. In addition, a model can be dragged from the Database Explorer and placed onto a mining flow canvas as a Model Source operator.

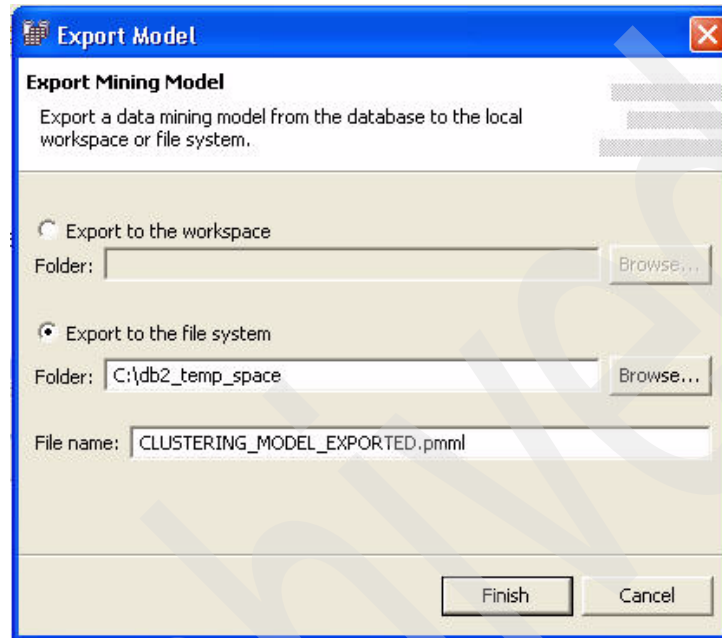


Figure 7-66 Exporting a data mining model

InLine Analytic Applications

A typical challenge that businesses face is that their business analysis tools and the business process tools are disconnected. Management is primarily looking at the operational data in a historical perspective and operating in a reactive mode. This needs to change. In addition, management seems to spend too much time searching through the data in the data warehouse to get the information they need to make business decisions, and then distribution of that data is typically tightly controlled and direct query capability quite complex. The result is two disjointed systems within the organization — one that is used to run the business in real-time and one that manages the business operation in an off-line batch mode. All too often there is little or no interactivity between those two systems.

Many of these common business problems can be mitigated by implementing InLine Analytic Applications. Inline Analytic Applications integrate analytics, reporting, and query into an easy-to-use interface that is customized to each user's role and context. It brings together the business analysis and business process into one platform so that the user can perform data analysis and decision making without ever leaving the application.

Some of the key features of Inline Analytic Applications are:

- ▶ Connection to all the data sources
- ▶ Broadly deployable
- ▶ Relevant to the way employees work
- ▶ Timely, rather than historical
- ▶ Personalized around roles and responsibilities

- ▶ Adaptable to ever-changing business processes
- ▶ Consistent with the IT infrastructure

In this chapter we introduce DB2 Alphablox and show how it can help to solve the common business problems by building applications with InLine Analytics, with relative ease.

8.1 DB2 Alphablox

DB2 Alphablox plays a special role within the DWE V9.1 suite in that it creates the interface through which the business looks at its enterprise data. It is used to develop and support information dashboards and enable interactive multidimensional data analysis.

DB2 Alphablox and all analytic applications enabled with DB2 Alphablox run as J2EE-compliant applications in an application server, and they are accessed by using a Web browser. Unlike traditional query and reporting tools that interact with application servers, DB2 Alphablox leverages the application services, portal services, and integration broker services provided by the application server. Moreover, DB2 Alphablox leverages the common foundation of J2EE architecture for developing, deploying, and maintaining distributed applications.

8.1.1 The architecture

DB2 Alphablox is comprised of the following elements:

- ▶ Platform
- ▶ Analytic-enabled solutions
- ▶ Administration application
- ▶ Application server adapters

The platform, the core component of DB2 Alphablox, runs within the business tier of the J2EE application server. While running as a J2EE application within the host application server, it provides a full array of functional services for interactive multidimensional analysis. For applications to fully leverage the analytic capabilities and services of DB2 Alphablox, the platform requires a separate installation of components and adapters that are not traditionally part of J2EE applications. These components are depicted in Figure 8-1.

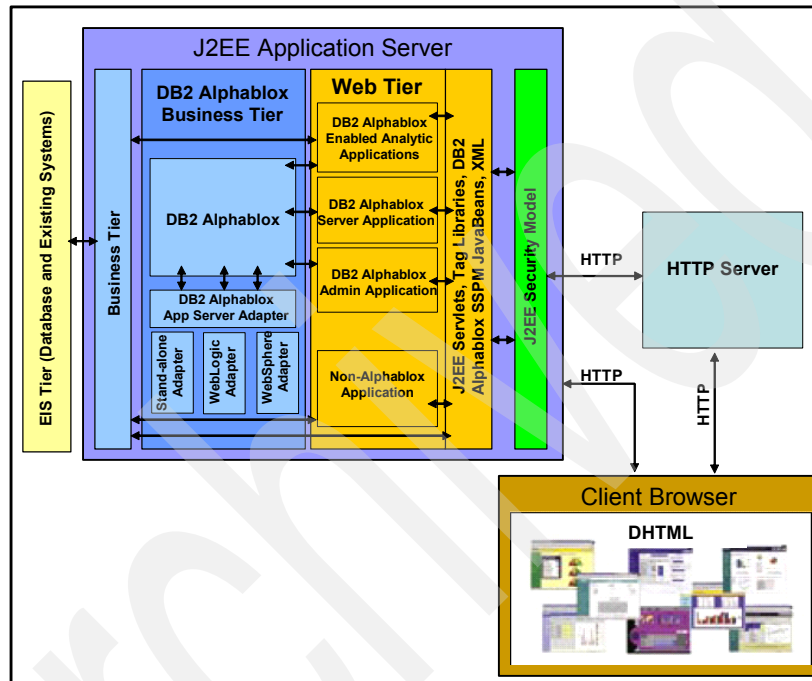


Figure 8-1 DB2 Alphablox architecture

The adapters allow DB2 Alphablox to communicate with each supported application server to perform administration functions. Many of these functions, such as defining applications, are set up differently on each application server.

Analytic applications enabled with DB2 Alphablox run on the application server within the Web tier. The applications, while interacting with DB2 Alphablox, are configured as separate and encapsulated J2EE applications. Updates to analytic applications enabled with DB2 Alphablox can be deployed, backed up, upgraded, and migrated independently of the DB2 Alphablox platform.

DB2 Alphablox also registers two J2EE applications within the Web tier of the application server. They are the DB2 Alphablox server application and the DB2 Alphablox administration application. The application server manages DB2

Alphablox in the same way in which it manages any other Web application. For example, it is auto-started by invoking a servlet. DB2 Alphablox is then suspended and resumed by the application server as needed, based on requests received by the application server and the management model.

DB2 Alphablox analytic components

DB2 Alphablox enables organizations to integrate analytics across functions and lines of business, and deploy analytic solutions for more informed decision making. The technology enables organizations to improve various areas within their business, including:

- ▶ Self-service reporting and analysis applications
- ▶ Operational analysis applications
- ▶ Financial reporting and analysis applications
- ▶ Planning applications
- ▶ Business performance and key performance indicators (KPIs) for interactive information dashboards

Data can be presented in several formats, including:

- ▶ Interactive grids, charts, and reports
- ▶ Informational dashboards
- ▶ Planning and modeling applications
- ▶ Information portals

DB2 Alphablox can access all enterprise information resources, including relational and multidimensional databases, transaction systems, and other external content feeds. This ensures that users have immediate access to all pertinent data, regardless of where or how it is stored. In DWE, DB2 Alphablox comes only with adapters that access DB2 data sources. To access data sources other than DB2, adapters can be separately purchased.

8.1.2 Enabled applications

Analytic applications enabled with DB2 Alphablox typically have the following characteristics, which may be implemented using various combinations of DB2 Alphablox features:

- ▶ Interactive and guided analysis
- ▶ Real-time data access, analysis, and alerts
- ▶ Personalization
- ▶ Sharing and collaboration
- ▶ Real-time planning through write-back

Interactive and guided analysis

Analytic applications enabled with DB2 Alphablox enable users to interact with real-time data via grids and charts, as well as other components such as drop-down lists.

These interactive analytic applications are served in dynamic HTML, based on Dynamic HTML (DHTML) technology, utilizing JavaScript™ and cascading style sheets (CSS). The DB2 Alphablox Dynamic HTML client provides the benefits of easy deployment with interaction. For example, a user can interact with a grid and have just that grid updated rather than having to refresh the entire page.

Users perform multidimensional analysis by manipulating the data displayed in the grid and chart, as depicted in Figure 8-2. Analysis actions such as drilling, pivoting, sorting, and selecting can be performed directly on the grid and chart, through toolbar buttons, through right-click menu options, or via the DB2 Alphablox form-based controls and components added by application developers.

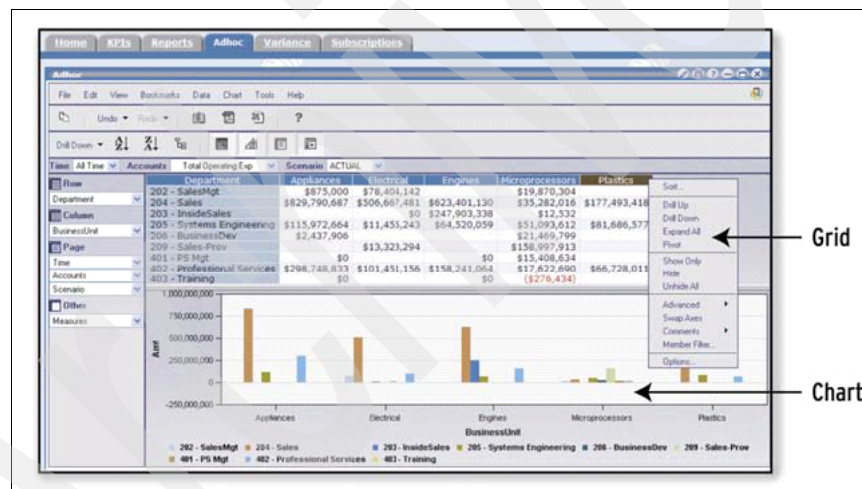


Figure 8-2 DB2 Alphablox grid and chart

Real-time data access and analysis

Analytic applications enabled with DB2 Alphablox can drive analysis of data from multiple data sources, both relational and multidimensional, including DWE Cube View cubes. Through the native ability to query a database, DB2 Alphablox exposes the analytic functionality in the database engine. Users can leverage capabilities such as ranking, derived calculations, ordering, filtering, percentiles, variances, standard deviations, correlations, trending, statistical functions, and other sophisticated calculations while performing analysis.

For example, a controller of a manufacturing company could choose to look at key performance indicators (KPIs) such as profit, bookings, billings, backlogs, trends, and comparisons of actuals to budget, as depicted in Figure 8-3. It is not necessary to read or understand the information in the figure, it is simply a depiction of the type of information that could be displayed. The data is displayed in real-time and the controller can choose to drill down on various items, such as total revenue, to get more detail.



Figure 8-3 DB2 Alphablox example: comparisons of actual to budget

The DHTML client in DB2 Alphablox is very flexible. Data can be presented the way users need to see it. For example, in Figure 8-4 a controller wanted to see a butterfly report in which the prior three months of actual figures are shown to the left of the accounts row headers and the current month details are shown to the right of those headers.

| Comp Time | | Business Unit | | Department | | Base Time | | |
|---------------|------------|-------------------|--|--------------------|------------|---------------|------------|-------|
| Last 3 Months | | Total Corporation | | All Departments | | Current Month | | |
| Feb | Mar | Apr | | | May | | | |
| ACTUAL | ACTUAL | ACTUAL | | Accounts | ACTUAL | BUDGET | Variance | Varia |
| 275,522 K | 202,728 K | 195,090 K | | Payroll & Benefits | 230,320 K | 298,227 K | 67,907 K | |
| 324 K | 1,125 K | 503 K | | Employee Devel | 862 K | 3,237 K | 2,375 K | |
| 73 K | 0 K | 2 K | | Recruiting | | 2,314 K | 2,314 K | |
| 491 K | 195 K | 197 K | | Admin Fees | 102 K | 265 K | 163 K | |
| 23,284 K | 11,737 K | (4,285 K) | | Professional Fees | 6,185 K | 6,227 K | 42 K | |
| 33,492 K | 25,001 K | 20,038 K | | T & E | 21,143 K | 20,022 K | (1,121 K) | |
| 6,290 K | 8,242 K | 10,954 K | | Marketing | 5,428 K | 9,921 K | 4,493 K | |
| 539 K | 196 K | 665 K | | Events | 187 K | 971 K | 783 K | |
| 5,176 K | 1,637 K | 1,186 K | | Equipment Cost | 1,644 K | 4,053 K | 2,409 K | |
| 15,109 K | 14,974 K | 16,754 K | | Depreciation Exp | 14,009 K | 16,913 K | 2,904 K | |
| 22,311 K | 71,743 K | 39,362 K | | Occupancy Exp | 28,629 K | 40,886 K | 12,258 K | |
| 10,898 K | 7,250 K | 7,120 K | | Office Exp | 8,089 K | 12,667 K | 4,577 K | |
| (33,506 K) | (55,078 K) | (48,367 K) | | Allocations | (37,837 K) | (57,547 K) | (19,710 K) | |
| 31 K | 83 K | | | Other | | | | |

Figure 8-4 DB2 Alphablox butterfly report

Personalization

Users have different data and business needs. Therefore, analytic applications enabled with DB2 Alphablox can be personalized to meet the needs of each individual user. For example, the first log-on screen that users see can be customized according to their role in the organization. Users in the sales department may see the top five best-selling products or the most profitable regions for the month-to-date. Users in finance may be more interested in monthly summary figures for sales, cost of goods, marketing, payroll, and profit, as shown in Figure 8-5.

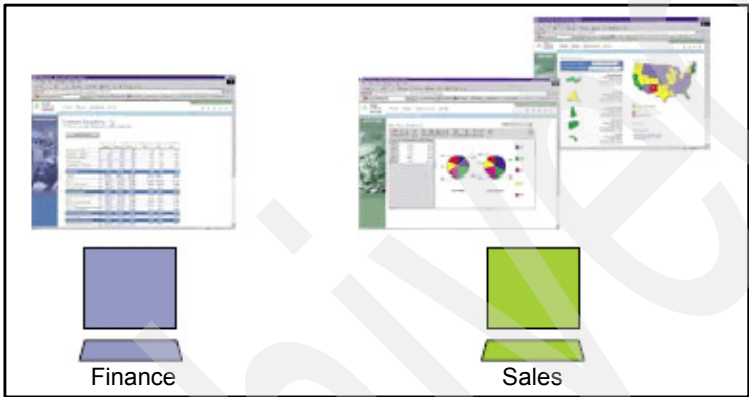


Figure 8-5 DB2 Alphablox customization example

In addition, each analytic applications enabled with DB2 Alphablox may contain custom user preference screens that enable users to personalize the solution to their needs, as depicted in Figure 8-6. In this example, the user can choose the business units and accounts that are displayed in the dial gauges.

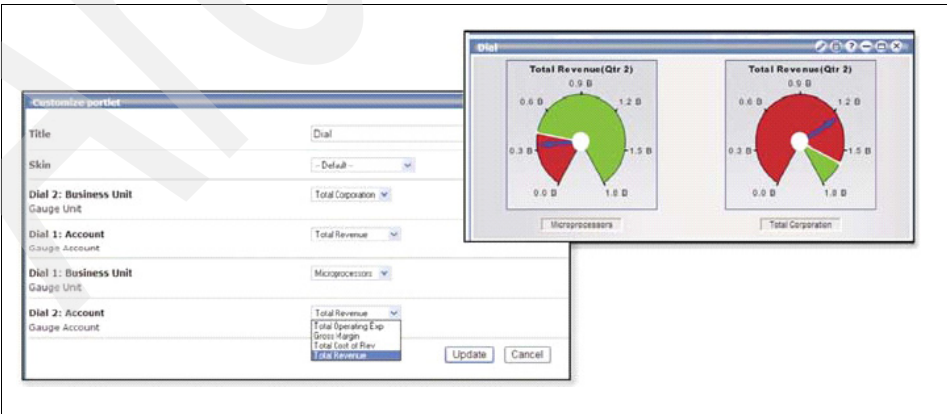


Figure 8-6 DB2 Alphablox personalization example

Sharing and collaboration

Analytic applications enabled with DB2 Alphablox support collaboration, enabling users to leverage existing messaging and workflow systems like WebSphere MQ Workflow to save and share application views once the analysis is performed, as depicted in Figure 8-7. In addition, DB2 Alphablox supports collaboration features such as bookmarking, e-mail, and PDF generation.

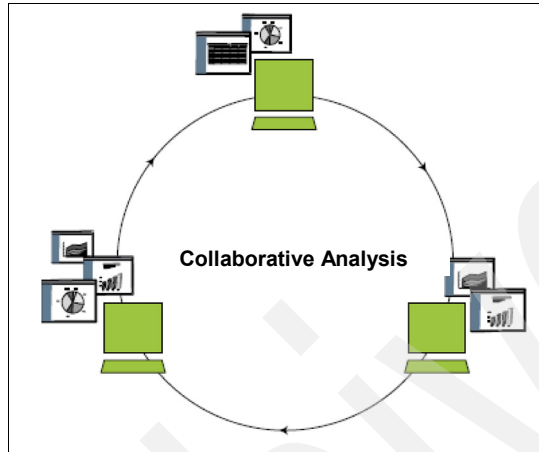


Figure 8-7 Alphablox collaboration flow

Analytic applications enabled with DB2 Alphablox support collaboration, enabling users to leverage existing messaging and workflow systems to save and share application views once analysis is performed.

Real-time planning through write-back

Analytic applications may range from historical analysis to forward-looking forecasting and proactive resource allocation. The DB2 Alphablox data write-back capability enables developers to build real-time planning applications such as budgeting, sales forecasting, what-if modeling, and collaborative demand planning, as seen in Figure 8-8. The write-back feature allows the user to modify the result set and update the underlying data source.

Microprocessor Expense

Business Unit

Microprocessors

Department

110 - Facilities

SAVE CHANGES

| ACTUAL | | | | BUDGET | | |
|---------|--------|----------|-----------------------------|--------|--------|--------|
| Feb | Mar | Apr | Accounts | May | Jun | Jul |
| 13509 | 10213 | (109948) | Outside Computer Services | | | |
| | | | Non-Capital Equip < \$1,000 | 80000 | 80000 | |
| | | | Software Licenses | | | |
| 1132378 | 291451 | (220) | Equipment Rental | 750000 | 750000 | 145000 |
| | | | Equipment Repair | 10000 | 10000 | 0 |
| 41000 | 26500 | 26500 | Equipment Maintenance | 40055 | 40000 | 30000 |
| 1186887 | 328164 | (83668) | Equipment Cost | 880055 | 880000 | 175000 |

Figure 8-8 DB2 Alphablox what-if modeling example

Note: The write-back feature of DB2 Alphablox is only supported with Hyperion Essbase and MSAS as data sources, and they are not part of the DWE package. To use these features you must buy adapters (sold separately) for those data sources.

DB2 Alphablox and the application server

An enterprise can gain competitive advantage by quickly developing and deploying custom applications that provide unique business value.

The J2EE standard provides an opportunity for analytic solutions to undergo a true paradigm shift. Prior to J2EE, there was not a standard, cross-platform architecture that would enable truly distributed computing in a Web environment. J2EE simplifies enterprise application development and deployment in several ways:

- ▶ Development environment based on standardized, modular components
- ▶ A complete set of services to application components
- ▶ The ability to extend existing services and add new services that provide complete interoperability with standard services

- Handle the details of application behavior automatically without complex programming

The DB2 Alphablox architecture capitalizes on this standard, cross-platform environment to deliver analytic solutions. DB2 Alphablox draws on Java technologies to implement a Web-based, N-tier architecture for delivery of analytic solutions. J2EE provides the framework for distributed, multi-tiered applications. Application logic is divided into components according to function. The most common configuration is a three-tier configuration, as depicted in Figure 8-9, which consists of the following:

- The Enterprise Information Systems (EIS) tier, also known as the Database tier, runs on database servers. Data resides on these servers and is retrieved from relational or multidimensional data servers.
- The J2EE application server is host to the business and the Web tiers. The business tier is the code that implements the functionality of an application, and the Web tier supports client services through Web containers.
- The Client tier is where client applications are presented. For a Web-based J2EE application such as DB2 Alphablox, the Web page display and user interface presentation occur on a client machine through a Web browser. The Web browser downloads Web pages and applets to the client machine.

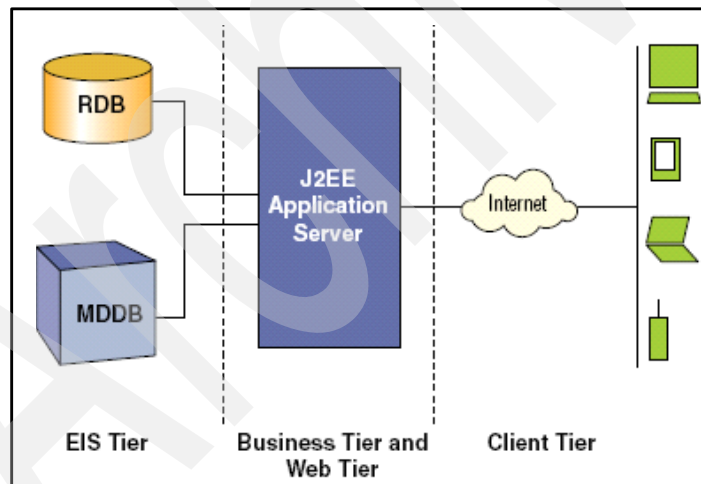


Figure 8-9 Three-tier configuration

Within the J2EE framework, DB2 Alphablox runs as an application within the application server, as shown in Figure 8-10, leveraging existing server resources such as process management, application management, and request management. Analytic applications enabled with DB2 Alphablox run as standard application server applications and the Web pages are served directly through the application server.

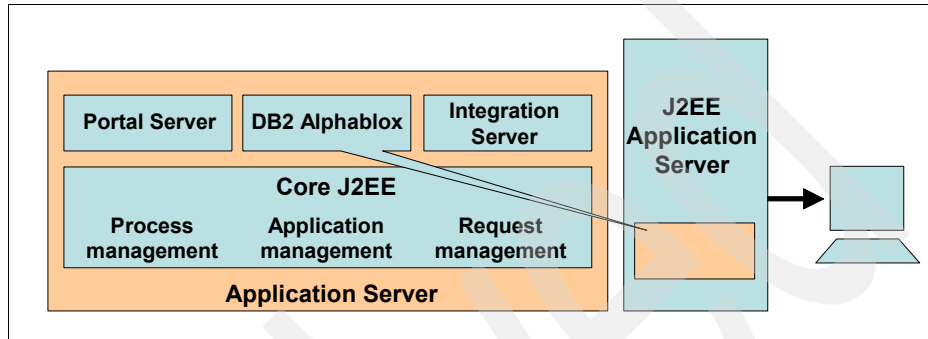


Figure 8-10 DB2 Alphablox running on an application server

Components of a DB2 Alphablox-enabled application

Once installed, DB2 Alphablox provides a comprehensive set of components and application templates for developing analytic solutions. The modular *building Blox* approach enables fast delivery of personalized and customized applications.

Analytic applications enabled with DB2 Alphablox appear as a collection of Web pages that serve as containers for the following application components:

- ▶ Standard HTML tags and page elements (logos, text, or images) to enhance the user interface
- ▶ Blox necessary to deliver the required application functionality
- ▶ JavaScript for extended application and user interface (UI) logic (optional)
- ▶ Java servlets for customized business logic (optional)

Application building Blox

To promote the creation of custom analytic solutions, DB2 Alphablox includes a set of generic application building Blox, as depicted in Figure 8-11. Application building Blox are prebuilt, high-level JavaBean components that provide the functionality required by analytic applications. Blox allow developers to perform data manipulation and presentation tasks and build dynamic, and personalized analytic applications. Because Blox are modular and reusable in design, they are easily built into a variety of analytic solutions.

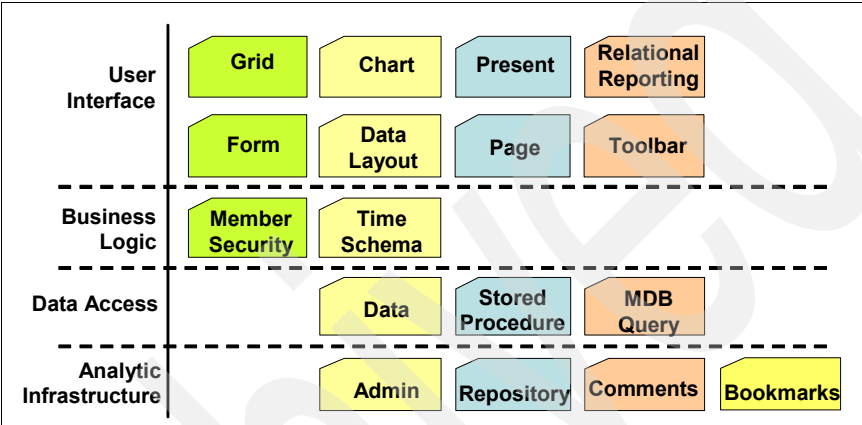


Figure 8-11 DB2 Alphablox application building Blox

Each Blox provides broad functionality through its properties and associated methods, which allow the Blox appearance and behavior to be specified and controlled. Event filters and event listeners are available for performing pre- event and post-event processing for user events such as drilling up or drilling down, pivoting, changing the page filter, loading a bookmark, or changing the data value in a grid cell.

8.1.3 Deploying DB2 Alphablox

Detailed descriptions of the aspects for deploying analytic applications enabled with DB2 Alphablox are described in the following sections.

Administration

DB2 Alphablox is designed for integration with an existing application server environment to help leverage a robust architecture to deliver analytic solutions. To facilitate installation, DB2 Alphablox provides a cross-platform, graphical interface to install the DB2 Alphablox server and administration pages. Once installed, you have centralized administration of analytic solutions through a set of administration pages. These pages enable application developers to manage

DB2 Alphablox services that use the same resources and are also complementary to the administration provided through the application server.

For example, application developers use the DB2 Alphablox administration pages as a convenient way to register and set up new applications. When creating an application from the DB2 Alphablox home page, DB2 Alphablox creates the application definition in the DB2 Alphablox repository. In addition, the J2EE context and directory structure are created.

When running analytic applications enabled with DB2 Alphablox, the application server passes the user name and associated roles to DB2 Alphablox. To allow personalization, the user profile can be configured to allow the application developer to define custom properties for the user.

DB2 Alphablox administration pages can also be used to configure DB2 Alphablox specific settings such as data sources, relational cubes, groups, and DB2 Alphablox server settings. These administration pages, packaged as a J2EE-compliant application, can be managed through the same mechanisms as any other Web application in the environment.

DB2 Alphablox can be administered either through Web pages under the DB2 Alphablox home page or through a standard command console accessible through any Telnet terminal software. Using either method, administrators can create users, data sources, and other DB2 Alphablox objects. This design enables remote server administration. For example, an application developer can define a new application from a workstation using the Web pages, or an administrator can perform routine monitoring tasks on a remote computer.

Setting up the application

Once the DB2 Alphablox-enabled application is completed, it is a self-contained J2EE Web application that authorized users can access as they would any other Web page. The application developer defines the DB2 Alphablox-enabled analytic application through the appropriate DB2 Alphablox administration page. The application developer specifies information such as application context, display name, home URL, and default saved state. Based on this information, DB2 Alphablox creates the application definition in the DB2 Alphablox repository and the application is made available to users through the application server.

The application context is a J2EE term referring to the descriptor that uniquely identifies the Web application or module. The application context serves as a container for J2EE applications that run within a J2EE server. Because they are standard J2EE applications, it is easy to package them as Web archive (WAR) or enterprise archive (EAR) files so they can be deployed to various application servers.

As specified in the J2EE standard, each DB2 Alphablox-enabled analytic application has a WEB-INF directory that houses the configuration information and the supporting resources necessary to deploy the application. These resources include components like Java classes (Java archive files) and JSP tag libraries.

The WEB-INF directory also includes the Web application descriptor file web.xml. The web.xml file, a standard file in all J2EE applications, is an XML file that contains markups that define the application behavior internally and as it relates to the application server. Included in the web.xml file are application-specific properties and their values, servlet mappings, and security constraint information. This file enables the deployment into application servers, because it includes everything the application server needs to know. The application server reads the web.xml file during initialization of the application.

Managing metadata in the DB2 Alphablox repository

The Metadata Repository Manager controls the contents of the DB2 Alphablox repository. The repository is a database that holds application-specific metadata for applications and users. It also includes information about data sources, relational cubes, user groups, roles, applications, and application states. When a user saves an application or Blox state, it is stored in the repository. The repository also stores bookmarked Blox properties that enable collaboration between users and groups, as well as XML representations of saved spreadsheet Blox.

DB2 Alphablox at runtime

To support a widely dispersed user community, DB2 Alphablox provides a high degree of flexibility by allowing the developer or end user to choose the delivery format of analytic applications enabled with DB2 Alphablox at runtime. The same application can be deployed in different modes at different times to meet different requirements throughout the enterprise. This arrangement enables all users to leverage analytic solutions, regardless of any network bandwidth or client-side software limitations. It also allows applications to be optimized according to the application function and analytic capability required by the end user.

DB2 Alphablox application deployment options

Consider the following deployment scenarios, which are depicted in Figure 8-12 on page 331:

- ▶ **Static HTML:** The application is delivered over an extranet or a narrowband network, providing users with simple data views. No significant client processing is required, and the information is presented in static HTML.
- ▶ **Dynamic HTML:** This mode utilizes JavaScript and cascading style sheet (CSS) to support the full range of data analysis functionality with a highly

usable and customizable graphical user interface. It does not require any plug-ins or downloads of Java class files.

- ▶ **XML rendering:** The application data needs to be integrated with transactional application servers or delivered to clients such as cell phones or pagers. The application is rendered in XML for delivery to XML-enabled applications and clients.
- ▶ **Ready for print:** Users can request that pages be rendered for printing. The application presents the information, formatted for printing.
- ▶ **Ready for PDF:** The application user requires a report with greater control over page layouts, storage, and printing. The application view is converted to PDF.
- ▶ **Export to Excel® or spreadsheet Blox:** The application provides data to be analyzed in Excel or spreadsheet Blox, and exports the data to the chosen application.

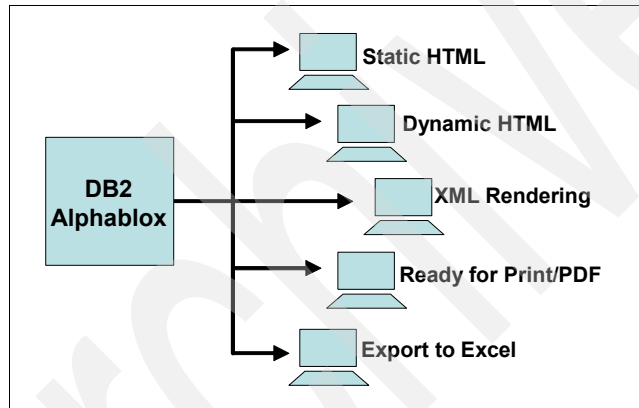


Figure 8-12 DB2 Alphablox scenarios

8.1.4 Services

In addition to application building Blox, the DB2 Alphablox platform consists of several services that help manage the applications, as seen in Figure 8-13. Each DB2 Alphablox service is responsible for a particular aspect of the application operating environment. In this section we take a closer look at each of these services.

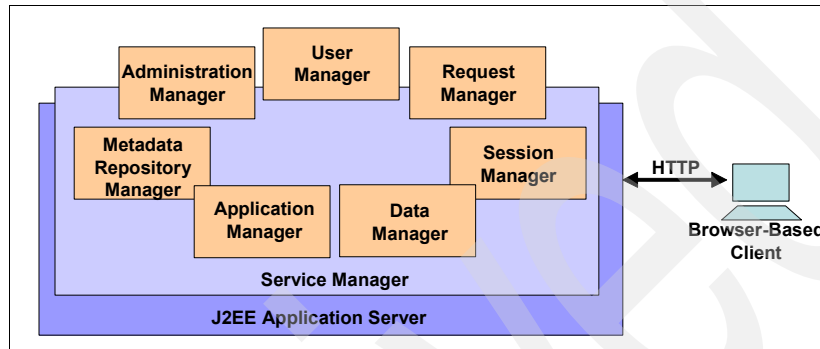


Figure 8-13 DB2 Alphablox services

Service Manager

As the focal point for server administration and monitoring, the Service Manager starts, stops, and provides access to the other managers, passes service requests to the correct manager, and monitors DB2 Alphablox resources.

Request Manager

The application server processes the original HTTP request. If there is DB2 Alphablox content, it is passed to the Request Manager for processing. If the request is from a user for which there is no active session, the Request Manager passes the request to the Session Manager. The Request Manager processes the application and Blox names, and then passes this information to the Application Manager for further processing.

As the application runs, the Request Manager coordinates communications between Blox on the application page and their server-side peers. The Request Manager also creates, monitors, and manages threads for each request.

Session Manager

The Session Manager establishes a session for each new DB2 Alphablox browser instance. If an individual user has multiple DB2 Alphablox browsers open, the user would have multiple concurrent sessions. The Session Manager creates and manages session objects and tracks which applications a user visits.

It also maintains a mapping between the DB2 Alphablox session and the session being maintained by the application server. The Session Manager also terminates dormant sessions after first saving the current state of each application and releases session resources.

User Manager

The application servers pass the user name to the User Manager, which gets the user information from the request object and then interacts with the application server through standard APIs to ensure that the user is authenticated. It controls all users of DB2 Alphablox services and creates and manages user instances. The User Manager also monitors resources allocated to each user and maintains user information, such as which applications are accessed, by which users, and for how long.

The DB2 Alphablox User Manager manages user authentication and authorization as well as provides personalization capabilities for customizing application content. By default, DB2 Alphablox uses its repository and the J2EE Security API to manage user and group information.

DB2 Alphablox also provides an out-of-the-box Lightweight Directory Access Protocol (LDAP) integration solution. This solution allows DB2 Alphablox to authenticate and authorize the users by using an LDAP directory server to recognize DB2 Alphablox users, groups, and custom properties. The DB2 Alphablox User Manager is built on top of the personalization engine called the Extensible User Manager. For environments in which custom security is desired, the Extensible User Manager personalization engine provides interfaces that allow the extension of either the out-of-the-box security solutions DB2 Alphablox repository or LDAP. It is also possible to plug in another external user manager such as NTLM or some existing Enterprise JavaBeans™ (EJBs).

Application Manager

The Application Manager is responsible for creating or modifying the DB2 Alphablox application definition from the DB2 Alphablox administration applications page. The Application Manager verifies user privileges for application access, starts application instances, manages each application instance, and supervises page processing before a page is presented to the user. The application design determines the exact page processing that occurs.

Application instance

The application instance controls the running state of each application. There is an application instance for each DB2 Alphablox browser instance in which the application is running. It is important to understand the difference between an application and an application instance, as depicted in Figure 8-14.

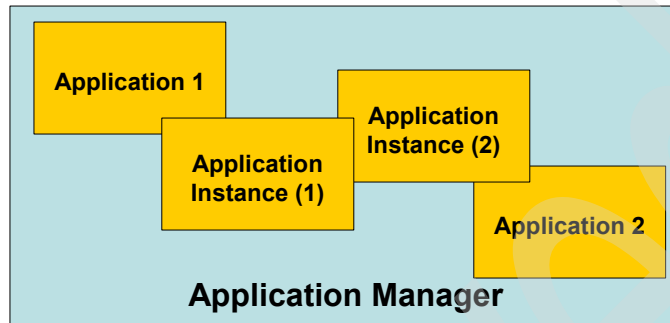


Figure 8-14 Application services

An application is the entity JSP, HTML pages, images, servlets, and so on that the application developer creates and stores on the DB2 Alphablox server. An application instance is the running state of that application, appearing in a browser window and interacting with a single user. The instance remains active until the client or administrator stops the application or the session times out.

The application instance maintains information about the state of each Blox in the application as well as information about the application as a whole. A user can choose to save the state of the entire application or simply that of an individual Blox. This feature can enhance collaborative analysis by enabling users to return to the saved state of the application and to share their results with others.

Data Manager

The Data Manager controls application requests for data sources and is responsible for accessing, browsing, querying, and retrieving data from data sources, as depicted in Figure 8-15. It is uniquely designed for connectivity to a variety of data sources. Through a single API for both multidimensional and relational sources, the Data Manager translates the data into dimensions, rows, columns, and pages the components typically used in multidimensional analysis. The Data Manager then presents this data for processing and manipulation by various Blox. Regardless of the data source, users perform analysis with the same analytic application front end.

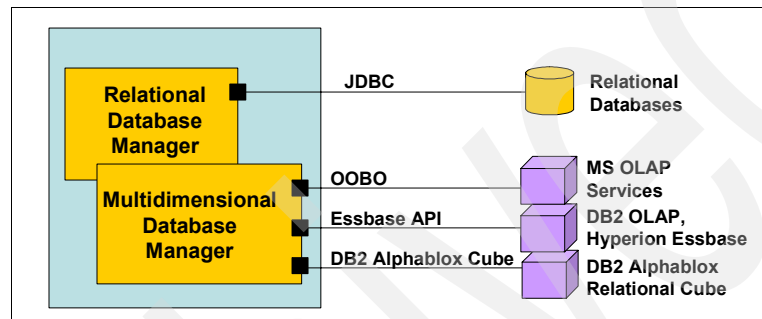


Figure 8-15 DB2 Alphablox data sources

The Data Manager architecture enables other databases to be connected through plug-in adapters. Each adapter encapsulates database-specific information for connection and processing. Data source definitions that identify the adapter are stored and administered centrally on DB2 Alphablox. If information for a data source changes, the application administrator changes the information in a single, central location.

The Data Manager and its associated data adapters provide support for:

- ▶ Browsing a collection of predefined data source connections, such as DB2 Alphablox named data sources
- ▶ Exposing the available databases within each data source
- ▶ Managing the database connections for user sessions
- ▶ Translating query objects into the underlying native query language
- ▶ Executing queries against the databases
- ▶ Modifying result set displays through user-invoked pivoting and drilling
- ▶ Write-back to the underlying data

In addition, the Data Manager allows for traversal of a result set and metadata. Users can retrieve data from the data source, traverse it using the DB2 Alphablox server-side result set and metadata APIs, and then take appropriate action. For instance, the application can be built to use the server-side APIs to traverse through the data looking for a certain condition based on an alert (for example, if actual inventory drops below plan). If the data meets the condition, a workflow process can be established that notifies the affected user (in this case the buyer for the product). The user can then write-back to the data source (order more product), closing the loop on the specific business process.

Content Manager

The Content Manager handles the setup of applications and examples that exist in the DB2 Alphablox Application Studio library of templates and tools. It has the ability to set up data sources and register applications.

8.1.5 Blox server/client structure

Each Blox has a server-side peer that contains the majority of the Blox functionality. Blox have the ability to render information to the client in a variety of formats. Server-side peers connect to a data source, obtain a result set, and deliver it to the client in the requested run-time format. Client-side components and their server-side peers work together to provide data access, presentation, and manipulation through the built-in user interface of the grid Blox and other presentation Blox.

Using server-side peers (as depicted in Figure 8-16) and client-side components optimizes the performance of analytic applications enabled with DB2 Alphablox. DB2 Alphablox manages the application logic, separating it from the user interface presentation, thus reducing the burden on the client machine.

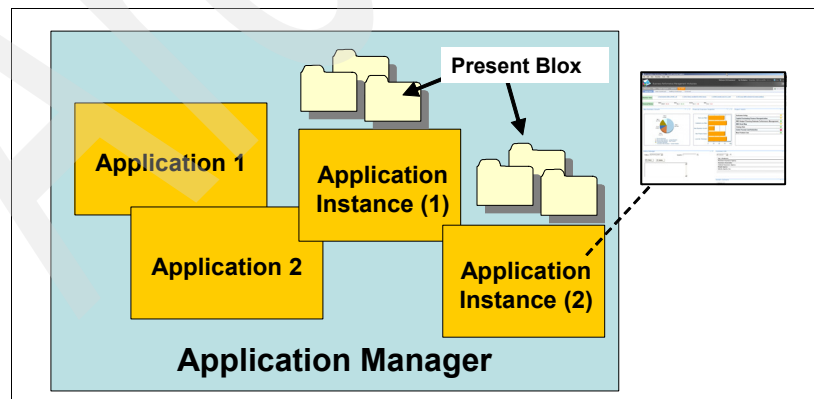


Figure 8-16 DB2 Alphablox server-side peers

Application delivery session flow

Numerous tasks are accomplished between the tiers as an application is accessed, and dynamically assembled and delivered to the client's Web browser. The page processes vary based on the page type and content.

The application server is responsible for the following tasks:

- ▶ Network management
- ▶ Management of connections
- ▶ User authentication and security
- ▶ Processing and serving HTML files
- ▶ Processing and compiling JSP files using its servlet/JSP engine
- ▶ Serving the entire processed page back to the Web browser

DB2 Alphablox is responsible for the following tasks:

- ▶ Data access and manipulation
- ▶ Dynamically building and deploying the user interface that provides interactive analytic applications
- ▶ Managing the data session
- ▶ Personalizing the data view

Security

DB2 Alphablox leverages robust security models through the use of J2EE standard APIs (JAAS, JNDI). For example, if the application server uses LDAP, NTLM, Kerberos, or another security mechanism, DB2 Alphablox will leverage that mechanism, as seen in Figure 8-17. In addition, DB2 Alphablox leverages the roles that are given to application server users.

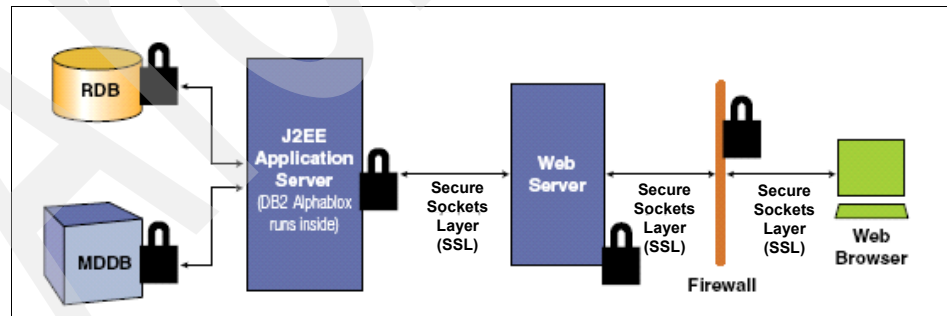


Figure 8-17 DB2 Alphablox security model

DB2 Alphablox users are managed via the application server administration, enabling them to be managed in the same way as users of other applications. This capability allows a developer to construct an application that uses the

personalization capability from the application server, combined with information from DB2 Alphablox to dictate the content seen by a user.

By leveraging the application server security mechanisms, DB2 Alphablox developers can implement the security model that works best with their application server. In addition, DB2 Alphablox does not impose additional security constraints. For example, a DB2 Alphablox user ID can be generated by the username passed from the application server.

The only information passed to DB2 Alphablox from the application server is the user name. Typically, user names and passwords are also kept in the DB2 Alphablox repository to help enable personalization and to provide for database access. DB2 Alphablox supports and honors database security. Application users must provide valid database authorization to access application data. When the DB2 Alphablox password is synchronized with the database password, the user can also access the database without a separate sign-on to the database. DB2 Alphablox also works with Secure Sockets Layer (SSL) capabilities, if they exist on any of the servers. SSL encrypts and authenticates all messages sent between the browser, and the Web and application server pipe.

8.2 Integration with DWE OLAP

Querying relational databases can be made easier for users if a dimensional model is used, because dimensional models make it easier to pose business questions related to a particular business process or business area. The structure of data, when organized in dimensional hierarchies, is more intuitive to users and their understanding of relationships among data categories. DWE OLAP, described in detail in Chapter 5, “Enabling OLAP in DB2 UDB” on page 79, is used to define the dimensional model. DWE OLAP and DB2 Alphablox work in synergy to provide end-to-end multidimensional analysis in DWE without the need for a separate multidimensional online analytic processing (MOLAP) or hybrid online analytic processing (HOLAP) server to persist physical cubes. DWE OLAP is tightly integrated with the DB2 Alphablox Relational Cube Server, making it easy to rapidly build customized analytic applications.

8.2.1 Metadata bridge

The metadata of the multidimensional cube that is modeled during design time using DWE OLAP can be conveniently imported into the Alphablox Relational Cube Server (described in 8.2.2, “DB2 Alphablox Relational Cubing Engine” on page 340) through the metadata bridge, as shown in Figure 8-18.

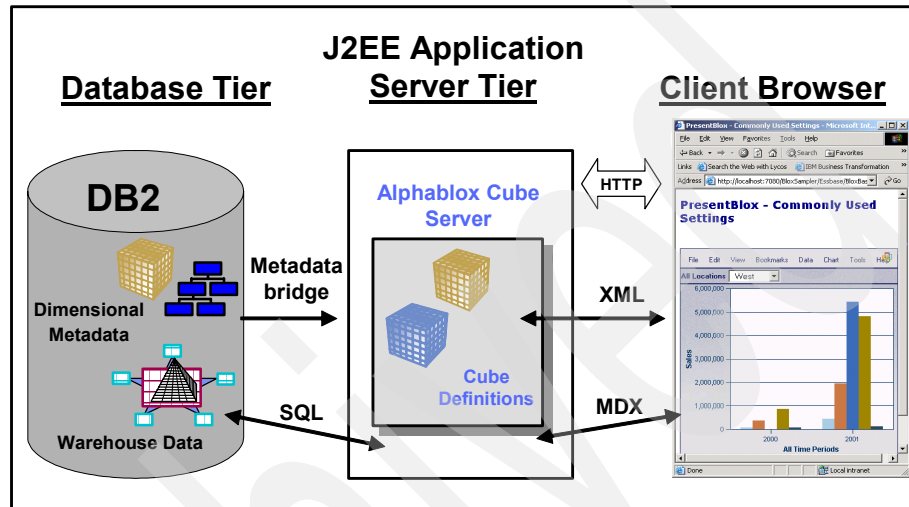


Figure 8-18 DWE Analytics architecture

The dimensional metadata model shown on the left-hand side of Figure 8-18, within DB2, maps to the underlying DB2 UDB tables. The OLAP Optimization Advisor, described in Chapter 5, “Enabling OLAP in DB2 UDB” on page 79, designs an optimal MQT based on the model. When the administrator executes the metadata import function from the Alphablox Administration Console, the Alphablox Cube Server invokes a stored procedure in DB2 to import the same cube model as metadata in XML format. This is represented by the upper set of lines in Figure 8-18. Now DB2 Alphablox and DB2 are synchronized on the same cube model.

The Alphablox Cube Server can then manage and tune the imported cubes. Adding and modifying measures, dimensions, and calculated members can be done within the Alphablox Cube Server.

At runtime, when the user invokes a query by performing an operation such as drill-down on the Alphablox UI, the Alphablox client generates a Multi Dimension Expressions (MDX). MDX is a highly functional expression syntax for querying multidimensional data. It provides the full OLAP navigational richness such as pivot, slice-and-dice, drill-down, and drill-across. The MDX is to multi-dimensional data sources what SQL is to relational data sources. These

MDX queries are compiled by the Alphablox Cube Server and then translated to the appropriate SQL queries to retrieve the fact data from base tables in the database tier. This is represented by the lower set of arrows in Figure 8-18 on page 339. On the database tier, the DB2 Optimizer evaluates the queries for eligible MQTs and finds the DWE OLAP designed MQTs. Since these MQTs were based on the same cube model that Alphablox is using, a high rate of MQT exploitation, known as *coverage*, is assured. The performance is very much on par with any other commercial cubing engine. Alphablox can subsequently cache the query results as a virtual cube on the application server, delivering even better performance during future analysis against the same cube model.

8.2.2 DB2 Alphablox Relational Cubing Engine

Depending upon the size and complexity of the dimensional model and the business requirements, users may need the power of a dedicated OLAP server. In these cases, data is extracted from relational databases and built into dedicated high-speed cubes that provide advanced analytic functions. In the cases where the full power of a dedicated OLAP server is not necessary, but you want to provide your users with the power of OLAP analysis functionality, you can use the relational cubing engine capability of the DB2 Alphablox Relational Cubing Engine.

The DB2 Alphablox Relational Cubing Engine, also referred to as the DB2 Alphablox Cube Server, is a high performance, scalable cubing engine, designed to support many users querying many different OLAP cubes. It is designed to enable quick multidimensional access to relational data stored in a data warehouse or a data mart. It is useful for creating multidimensional data sources for which you do not have the time and resources to develop full featured OLAP databases, and it is very good at presenting relatively small cubes, even if they are built from large databases. The DB2 Alphablox Cube Server is designed to provide a multidimensional view of data sources in a relational database.

When the DB2 Alphablox Cube is started (as depicted in Figure 8-19), the cube metadata is loaded from the OLAP tables in DB2. Also, the dimensional data is loaded from the DB2 tables to memory or optionally persisted on disk in the application server.

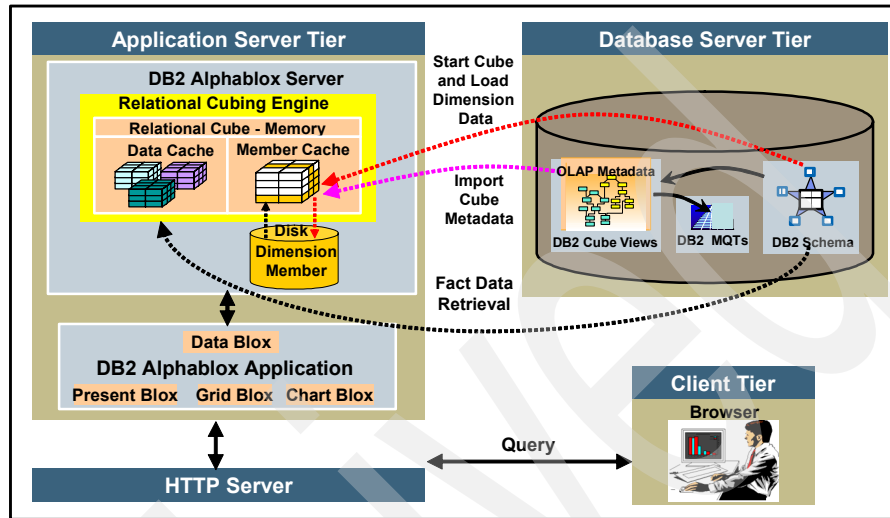


Figure 8-19 Data movement between database and DB2 Alphablox Cube Server

During runtime, DB2 Alphablox Cube Server stores precomputed results in memory in the form of cubelets, not on disk. Any results not stored in memory remain in the underlying database. The cube server retrieves results on an as-needed basis by sending SQL queries to the database. The results from the query are then stored in memory and are instantly accessible to the DB2 Alphablox applications.

While the Alphablox Cube Server is not intended as a replacement to a full-featured OLAP Server, it provides an intelligent subset of functionality. We fully expect this subset of functionality to grow over time. Recent enhancements to the Alphablox Cube Server (Version 8.4) include support for ragged and unbalanced hierarchies, persistent and non-persistent calculated members, default dimension members, Named Sets, Attributes and Client Member Order Specification, and support for non-empty keyword, which suppresses missing rows and columns. Expanded support for MDX functions is now available to partners and clients, including Order, Topcount, Toppercent, Bottomcount, Bottompercent, Item, Hierarchize, Topsum, and Bottomsum.

8.2.3 Application performance

While the query performance within the application is dependent on the complexity of the query that needs to be processed, the configuration of the Alphablox Relational Cubing Engine also plays a significant role. Hence, tuning the Alphablox Relational Cubing Engine is a very important function in the setup of the analytic application enabled with DB2 Alphablox. There are several parameters that can be altered to improve the query performance, and these can be administered from the Administration User Interface and the Cube Manager (shown in Figure 8-20).

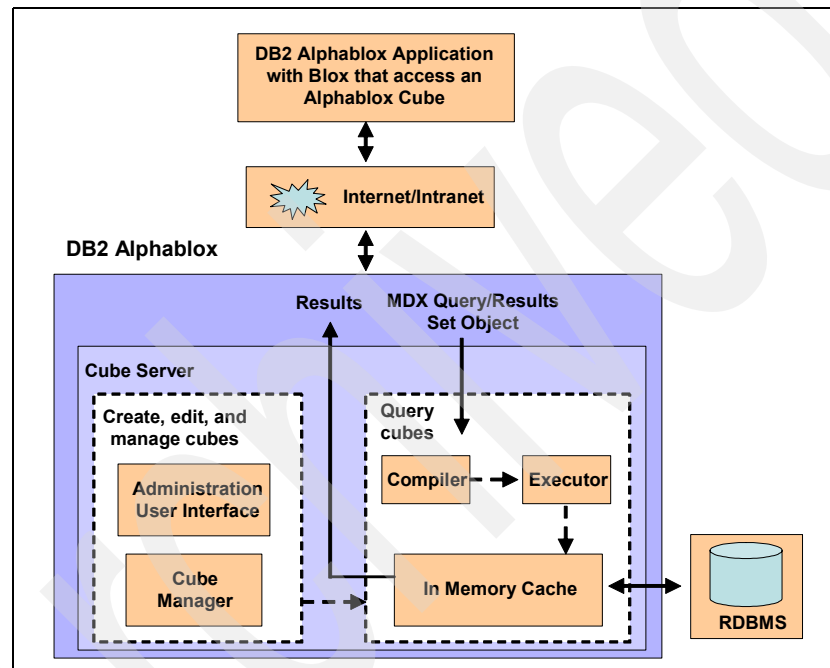


Figure 8-20 DB2 Alphablox Cube Server architecture

The data cache contains the cubelets of the result set and can be set to *unlimited*, which means that its limit is the total amount of memory available on the application server. Or the data cache can be completely *disabled*, which means that nothing is stored in memory and every query needs to be directed to the DB2 tables.

The dimension metadata cache can be set to complete or partial caching during cube start-up. This cache can be set to dynamic, which means that the dimension metadata is stored on disk in the application server as compressed files as it is dynamically cached in memory as needed. Or the cache can be set

to static, which means that all the dimension metadata is cached in memory cube during start-up.

There is an in-memory cache for the intermediate work area that stores the intermediate result set of the query. This cache, as shown in figure Figure 8-20 on page 342, is populated with queries made to the underlying relational database. If a query against an DB2 Alphablox Cube requests data that is not already stored in the cache, then the data is retrieved from the underlying database, and, if necessary, old data is aged out of the cache. The system performs all of these caching functions automatically. The performance of complex queries with large amounts of intermediate results are dependent on the amount of memory available on the application server.

The query performance can also be impacted by the following:

- ▶ Complexity of SQL queries generated at runtime to fetch cube cell data
- ▶ Number of SQL statements generated from the MQX query
- ▶ Presence of MQTs in the database
- ▶ The queries effectively utilizing (routing to) the MQTs available

The design of cube model impacts performance in the following ways:

- ▶ The presence of a large number of dimensions slows down the performance.
- ▶ A large fan-out of dimension members means broad dimensions with only a few levels, which slows down the performance.
- ▶ Having calculated members expressed in cube definitions rather than having them in the MDX query improves the query performance.

To learn more about DB2 Alphablox Cube Server configuration, features, and functions, refer to the DB2 Alphablox Information Center at:

<http://publib.boulder.ibm.com/infocenter/ablxhelp/v8r4m0/index.jsp?topic=/com.ibm.db2.abx.cub.doc/abx-c-cube-cubingconceptsoverview.html>

8.3 Developing an application

DB2 Alphablox is a premier application development platform rapidly building and broadly deploying custom analytic solutions across the enterprise. DB2 Alphablox provides a set of analytic components and supporting services to make it easy to rapidly assemble analytic applications using JSP tags. DB2 Alphablox developers use adapters to connect to all types of data sources, and then use data Blox to retrieve data from those data sources, and visual Blox to create highly interactive graphs, charts, and reports tailored to suit user needs.

8.3.1 Creating a data source

In this section we show how to create and manage the DB2 Alphablox application through the DB2 Alphablox Administration Console. Launch the DB2 Alphablox Administration Console in the Windows client machine from **Start** → **All Programs** → **IBM DB2 Alphablox 8.4** → **AlphabloxAnalytics** → **DB2 Alphablox Administrative Pages**. You can also launch the page from the `http://hostname:9080/AlphabloxAdmin`. Figure 8-21 shows the DB2 Alphablox Administrative Console home page.

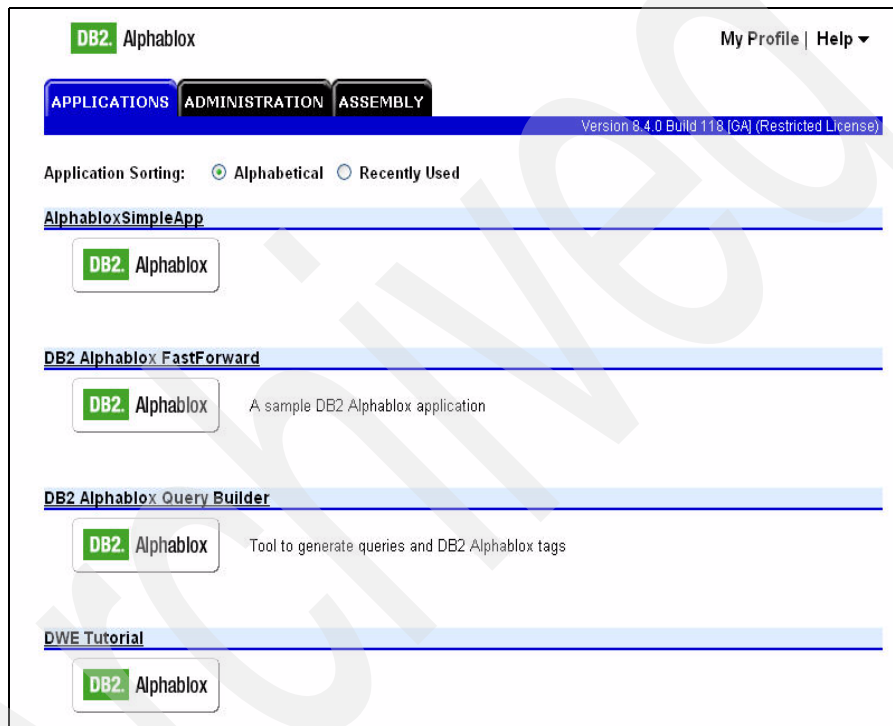


Figure 8-21 DB2 Alphablox administrative page

Click the **Administration** tab. The submenu that you see under the Administration tab is General, Groups, Users, Applications, Data Sources, Cubes. Click **Data Sources** to configure the underlying data source, as shown in Figure 8-22.

The screenshot shows the DB2 Alphablox Administration interface. At the top, there's a header with 'DB2 Alphablox' and a 'Help' dropdown. Below this is a navigation bar with tabs: 'APPLICATIONS', 'ADMINISTRATION', and 'ASSEMBLY'. Under 'ADMINISTRATION', there are sub-tabs: 'General', 'Groups', 'Users', 'Applications', 'Data Sources' (which is highlighted), and 'Cubes'. The 'Data Sources' section is titled 'Data Sources' and contains a 'filter:' dropdown set to 'Canned'. To the right of the filter is a list of data sources. Below the list are buttons: 'create', 'edit', and 'delete'. The main area is titled 'Create Data Source' and contains a form with the following fields: 'Data Source Name' (DATA_WH), 'Description' (Data Warehouse Database), 'Adapter' (IBM DB2 JDBC Type 4 Driver), 'Server Name' (localhost), 'Port Number' (50000), 'Database Name' (DATAWH), 'Default Username' (db2inst1), 'Default Password' (masked with dots), 'Use DB2 Alphablox Username and Password' (No), 'Maximum Rows' (10000), 'Maximum Columns' (1000), and 'JDBC Tracing Enabled' (No). At the bottom of the form are 'SAVE' and 'CANCEL' buttons.

Figure 8-22 Configuring a data source in the Alphablox Administration page

For details on configuration, refer to the white paper “Integrate DB2 Alphablox and DB2 Cube Views to build multidimensional OLAP Web Apps” located at:

<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0602wen/index.html>

It is a good practice to ensure that the data source was created without any errors by selecting the **Test Selected Data Source**. It will return a screen indicating whether the test was a success.

An additional data source needs to be defined with *Alphablox Cube Server Adapter* as the adapter in order to access the OLAP cube. The configuration details to set up this data source are also in the white paper.

8.3.2 DB2 Alphablox Cube

Although it is possible to define an OLAP cube in the DB2 Alphablox Administration page (Cube tab), we recommend that you define the cube using the DWE Design Studio OLAP feature. The Design Studio can do the performance optimization on the cube metadata to recommend MQTs and MDCs as well as build the analytic application. This will be very useful when dealing with large databases and a large number of concurrent users.

Importing the OLAP cube

Given that the OLAP cube has been defined using the DWE Design Studio OLAP feature, the next step is to feed the OLAP metadata to the DB2 Alphablox application. On the DB2 Alphablox Administration Console, you can do this by selecting the **Administration** → **Cubes** tabs. On the DB2 Alphablox Cube page, enter a cube name. Select the relational data source that you created. Check **Enabled**. Leaving **Enabled** unchecked will not allow you to start the cube successfully. Checking **Enable DB2 Cube Views Settings** will make all the cube models and cubes designed in DWE Design Studio available in the drop-down lists. Select the appropriate cube model and cube names from the drop-down lists. In order to make the OLAP application show meaningful names of dimensions and measures, select the **Use Business Names** radio button. Then select **Import Cube Definitions** to import the cube metadata including measures and dimensions. You should see the metadata objects on the left pane. Select **Show Import Log** to see the result of the metadata import step. Finally, check **Import cube definitions on start, rebuild, and edit** to automatically import the changes made to the cube metadata in the DWE Design Studio every time the DB2 Alphablox Cube is restarted or rebuilt. See Figure 8-23 on page 347 for details.

Starting the cube

The next step is to start the cube that was created. For this, select the **Administration** tab in the DB2 Alphablox Administration Console and select the **General** tab. Select **DB2 Alphablox Cubes** under the Runtime Management section. A list of cubes will appear on the right pane with the options start, stop, and details. Select the cube you wish to use and select **Details**. It will tell you the status of the cube and whether it is currently running or stopped. If the cube is stopped, select the **Start** button. The result would be as depicted in Figure 8-24. Now the DB2 Alphablox cube is available for OLAP navigation such as drill-down, drill-through, and slice and dice operations.

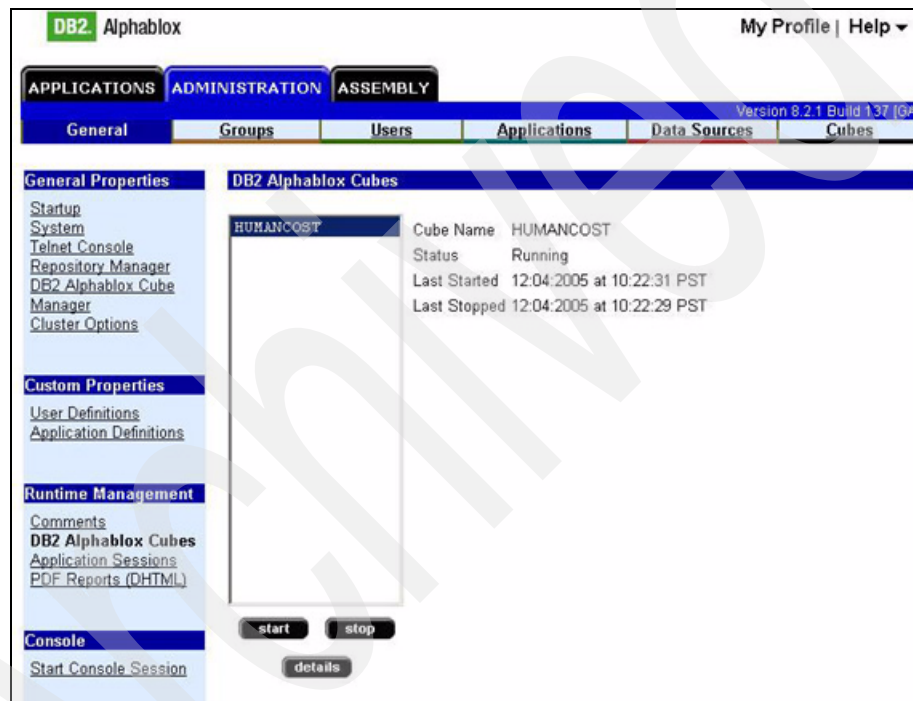


Figure 8-24 Starting an DB2 Alphablox Cube

8.3.3 Developing JSP code for the DB2 Alphablox application

After the data sources have been created in DB2 Alphablox, we will access the OLAP cubes through a J2EE Web application. Developing an DB2 Alphablox Web application is no different from developing any Java Server Pages (JSP) application. DB2 Alphablox components are added to the JSP page by embedding JSP tags into the page. Each tag has a number of properties that can be specified to customize the look, feel, and behavior of an DB2 Alphablox component.

DB2 Alphablox does not dictate the use any particular development tool. Any tool that supports the development of J2EE applications can be used from a simple editor such as Notepad to a sophisticated Integrated Development Environment (IDE) such as the Rational Application Developer. IDEs typically provide many advanced development, test, and deployment functions. In this book we look at the Rational Application Developer and the DWE Design Studio Java editor.

A JSP page is essentially a Web page containing HTML code (see Figure 8-25) with .jsp as the suffix to the file name.

```
<HTML>
<HEAD>
<Title> My Company </Title>
</HEAD>

<BODY>
<!--Top Banner-->

<!-- Page Body -->
< TABLE width="100%" height="80%">
  <TBODY>
    <TR>

    </TR>
  </TBODY>
</TABLE>
</BODY>
</HTML>
```

Figure 8-25 Web page containing HTML code

Within the JSP page you can embed one or more sections of code that is DB2 Alphablox specific (see Figure 8-26).

```
<!-- DB2 Alphablox TreeFormBlox -->
<bloxform:tree id="PoTMenu" rootVisible="false" textWrapped="true">
<bloxform:folder>
  <bloxform:item label="Home" href="index.jsp" />
  <bloxform:folder label="Metrics" expanded="true">
    <bloxform:item label="Profit Analysis" href="profit.jsp" />
    <bloxform:item label="Sales Analysis" href="sales.jsp" />
    <bloxform:item label="Expense Analysis" href="expense.jsp" />
    <bloxform:item label="Channel Report" href="channelrpt.jsp" />
  </bloxform:folder>
</bloxform:folder>
</bloxform:tree>
```

Figure 8-26 Sample DB2 Alphablox JSP tag

The first step is to generate the JSP code for the DB2 Alphablox components, also referred to as the Blox tag. There are two common ways to generate the Blox tag. One way is to use the DB2 Alphablox Query Builder to design a report and generate the corresponding Blox tag. This Blox tag can be inserted in the code of a JSP page. The second way is to write the code for the Blox blog code using an IDE such as the DWE Design Studio or Rational Application Developer.

DB2 Alphablox Query Builder

The DB2 Alphablox Administration Console has a utility to rapidly build a report against the data sources defined within it. This utility, known as the DB2 Alphablox Query Builder, can be accessed from the Applications tab of the Administration Console. Within the DB2 Alphablox Query Builder, you connect to a data source, a cube source if available, providing the appropriate login information. During login, there is an option to Execute Default Query, or you can write your own query in the Query window (see Figure 8-27).

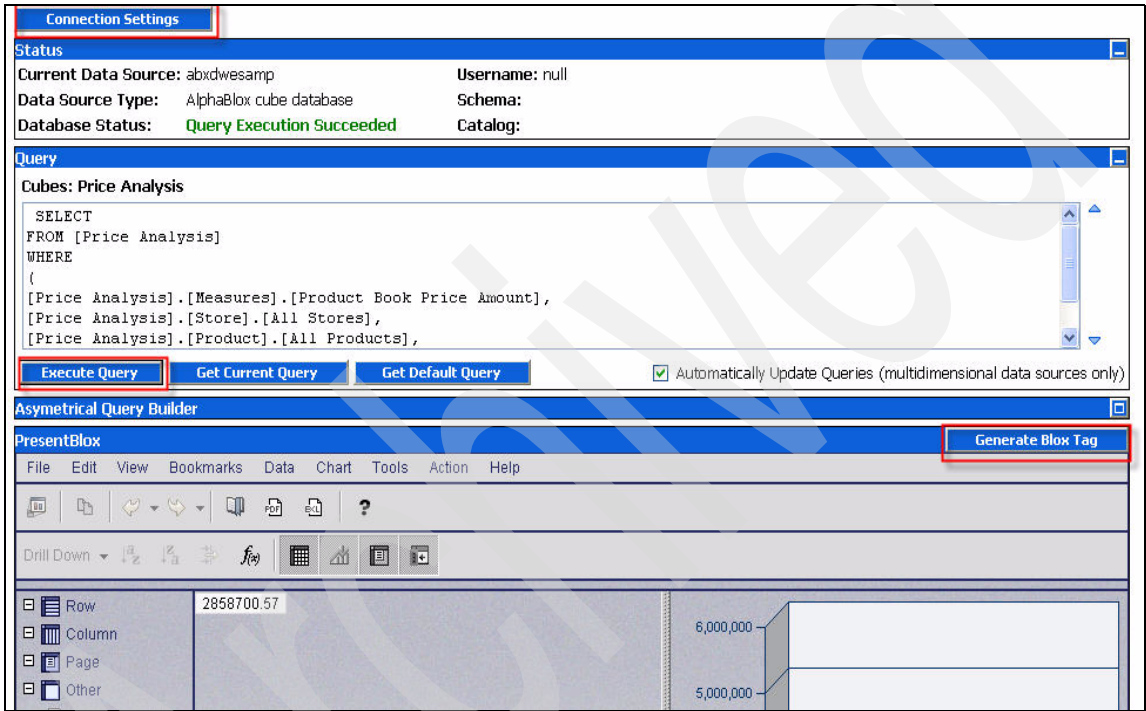


Figure 8-27 DB2 Alphablox Query Builder

This Query Builder provides a fully interactive grid within which you can drag and drop measures and dimensions on to the row, column, and page axis; and do drill-downs, drill-ups, and pivots. After you have created a report that is satisfactory, you can generate the Blox tag for that report by selecting **Generate Blox Tag**. This function brings up a dialog box and provides the Blox tag equivalent of the report you just created (see Figure 8-28).

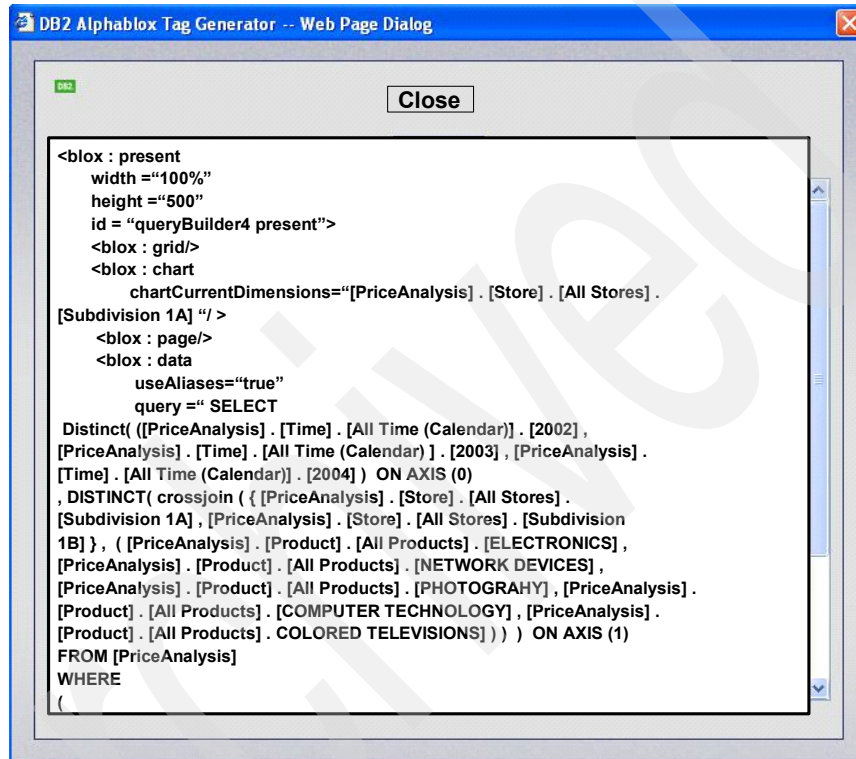


Figure 8-28 JSP tag produced by the Generate Blox Tag function

Copy the Blox tag from the dialog box and paste it into the JSP page where you wish to embed the interactive DB2 Alphablox report. The DWE Design Studio can be used to edit the JSP page of the Web application with which you are working.

Rational Application Developer

At one time there was no prescribed editor for DB2 Alphablox application development, so developers would use their favorite Java editor. Now the Rational Application Developer (RAD) has plug-ins for DB2 Alphablox that make development very convenient. The easy-to-install DB2 Alphablox plug-ins provide

significant enhancements for developers, especially to those who are familiar with the Rational development environment.

Note: RAD is not a part of the DWE V9.1 package, but is sold as a separate product.

RAD enabled for DB2 Alphablox provides the developers with rapid development features that include:

- ▶ Automatic inclusion of DB2 Alphablox tab libraries
- ▶ Automatic tag completion
- ▶ Property sheets for configuration management
- ▶ Portal development
- ▶ Debugging in the test environment

RAD Integrated Test Environment includes the Web development component, the application development component, and the test server (see Figure 8-29). DB2 Alphablox can be installed in the local application server, either WebSphere Application Server (WAS) 5.1 or 6.0, which is supplied with RAD. This makes debugging the DB2 Alphablox code very simple, as the developer does not have to export the code to a remote server for testing the code.

The version of RAD that is compatible with the DB2 Alphablox plug-ins is V6.0.0.1 or later.

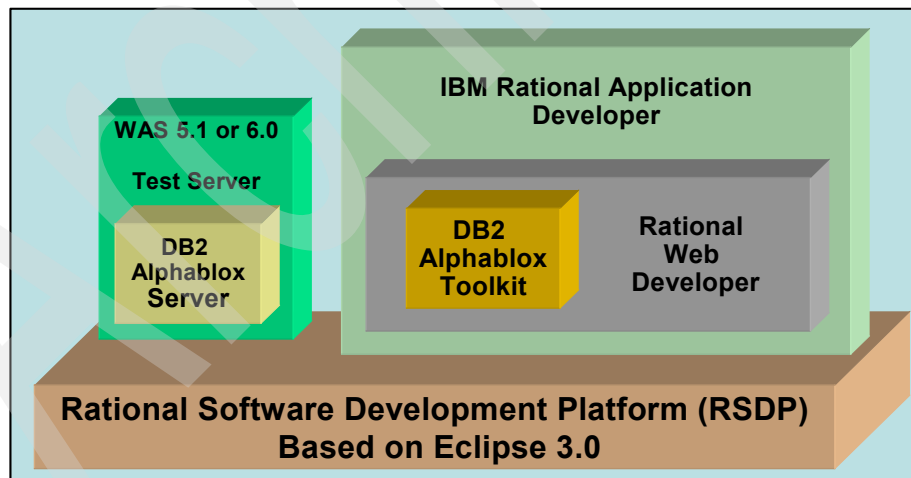


Figure 8-29 DB2 Alphablox and RAD Integrated Test Environment architecture

The installation and configuration of the RAD environment for DB2 Alphablox development involves the following steps:

1. Install Rational Developer. The ideal scenario involves installing RAD. But there are other alternatives to install the basic Web development environment, Rational Web Developer (RWD), or the more sophisticated Rational Software Architect (RSA).
2. Install the WAS Test Server, and WAS V6.0 is installed by default. If you require the WAS V5.1, it must be installed.
3. Perform updates on RAD. The Rational Developer needs to be V6.0.0.1 or later.
4. Install the DB2 Alphablox Server, and DB2 Alphablox is installed in the WAS Test Environment.
5. Install the DB2 Alphablox Toolkit for RAD. After the development environment is prepared with the necessary software, install the DB2 Alphablox plug-ins for RAD.

Using the DB2 Alphablox toolkit in RAD requires you to:

1. Configure RAD and the WAS Test Server.
2. Create a new dynamic web project.
3. Create a new JSP file.
4. Add DB2 Alphablox tag libraries.
5. Add `<blox:header>` and other Blox tags to the JSP file as needed. When RAD sees the tag `<blox:header>` it knows that it is a DB2 Alphablox tag and the DB2 Alphablox Code Assist feature becomes available (see Figure 8-30 on page 355). This helps to simplify code development by providing automatic tag completion.
6. Execute and debug within the RAD integrated test environment.

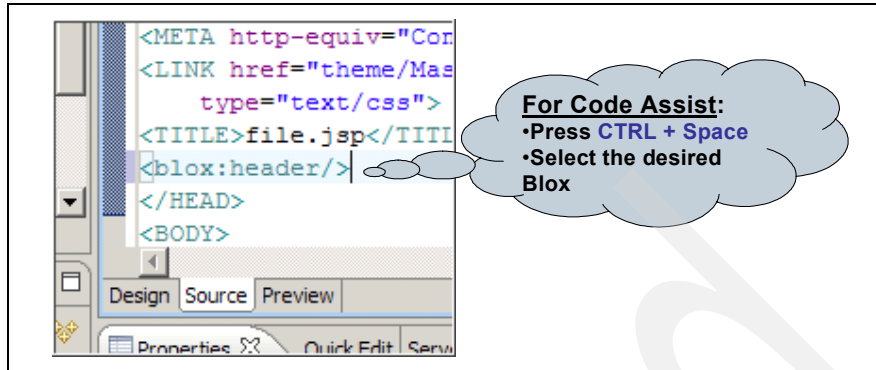


Figure 8-30 DB2 Alphablox plug-in for RAD provides code assist

For more information about building DB2 Alphablox applications using RAD, refer to DB2 Alphablox infocenter at:

<http://publib.boulder.ibm.com/infocenter/ablxhelp/v8r4m0/index.jsp?topic=/com.ibm.db2.abx.gst.doc/abx-c-start-20.html>

A detailed example of developing a DB2 Alphablox application can be found in the technical article located at:

<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0501bendel/>

The article demonstrates developing a DB2 Alphablox application using WebSphere Studio Application Developer, which is a predecessor of RAD.

DWE Design Studio Java Editor

There are several approaches to developing an analytic application enabled with DB2 Alphablox and deploying it on an application server such as WebSphere Application Server (WAS). The common denominator in all of these approaches is that the analytic application enabled with DB2 Alphablox will exist as an Enterprise Archive (EAR) file in the file system after it has been deployed. An EAR file is a compressed file, similar to a zip file, which contains all the files needed to deploy the application in a J2EE environment such as WAS.

In an implementation, you may use an existing DB2 Alphablox application as a starting point for a new application. In this section we show you how to use the DWE Design Studio Java editor (see Figure 8-31) to modify the JSP pages of the DB2 Alphablox application to include the desired DB2 Alphablox tags. We continue the discussion assuming that you are using a Windows development environment with DWE Design Studio installed and that the Windows machine has access to the file system of the application server.

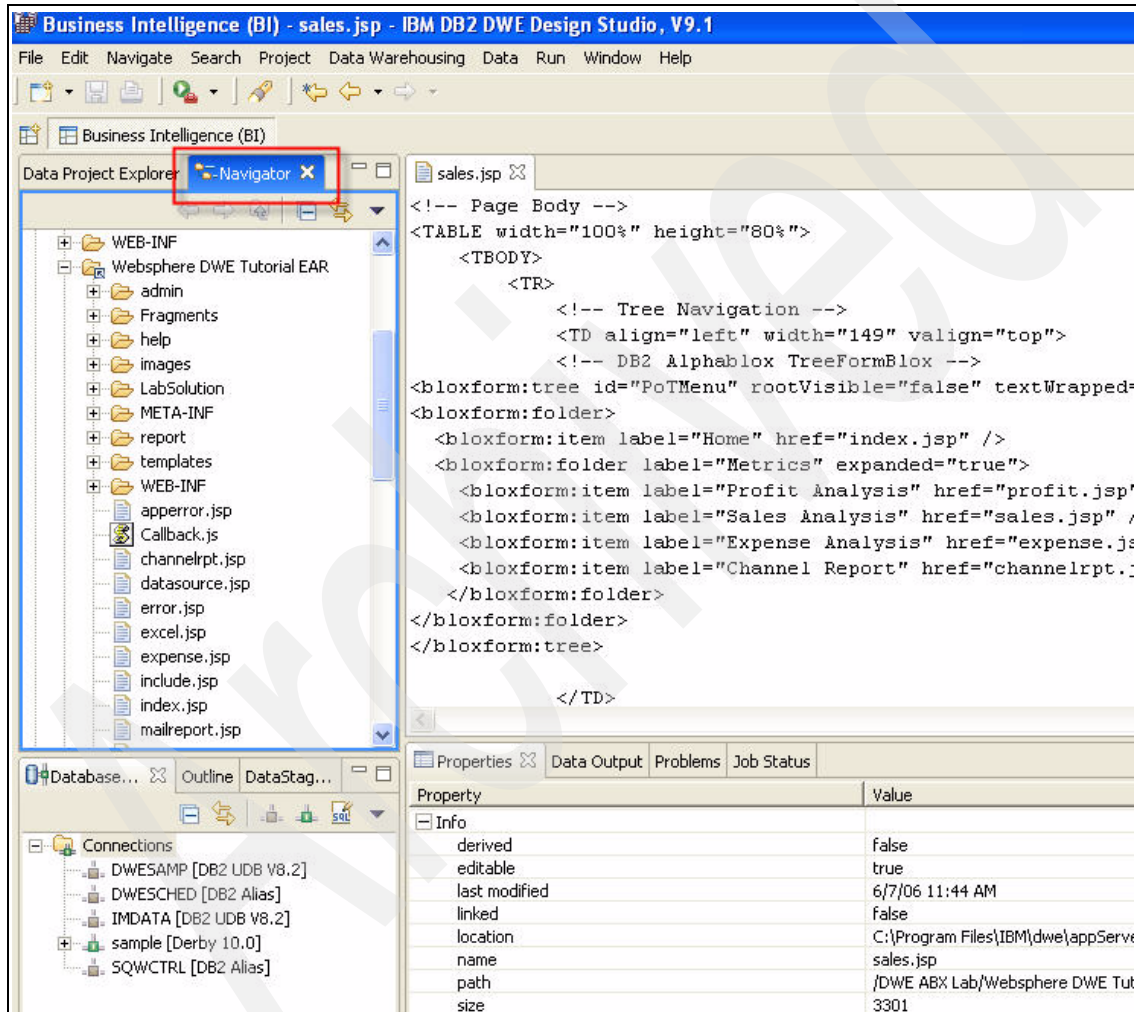


Figure 8-31 DWE Design Studio Java Editor

In the DWE Design Studio, the first task is to enable the use of linked resources in your workspace. For this, select **Windows** → **Preferences**, expand **Workbench** → **Linked Resources**, and check **Enable Linked Resources**.

The next task is to create a new Java project. In your workspace go to **File** → **New** → **Project** → **Java** → **Java Project**, give it a name, select **Finish**, select **No** to “Do you want to switch to this perspective?” You will see the new Java project in your Navigator tab.

The next task is to access the WAR file within the EAR of the DB2 Alphablox application from the new Java project that you just created. Select the **Navigator** tab, right-click the new java project, select **Properties** → **Java Build Path** → **Source** → **Add Folder**, give it a name (similar to your DB2 Alphablox application), select **Advanced**, select **Link to folder in the file system** → **Browse** to the DB2 Alphablox application **.ear** file in the file system and select the **.war** folder. Then select **OK**, answer **No** to “Do you want to remove the project as a source,” select **OK** for Source Folder Added, and click **OK** to close the Properties window. See the highlighted areas in Figure 8-32.

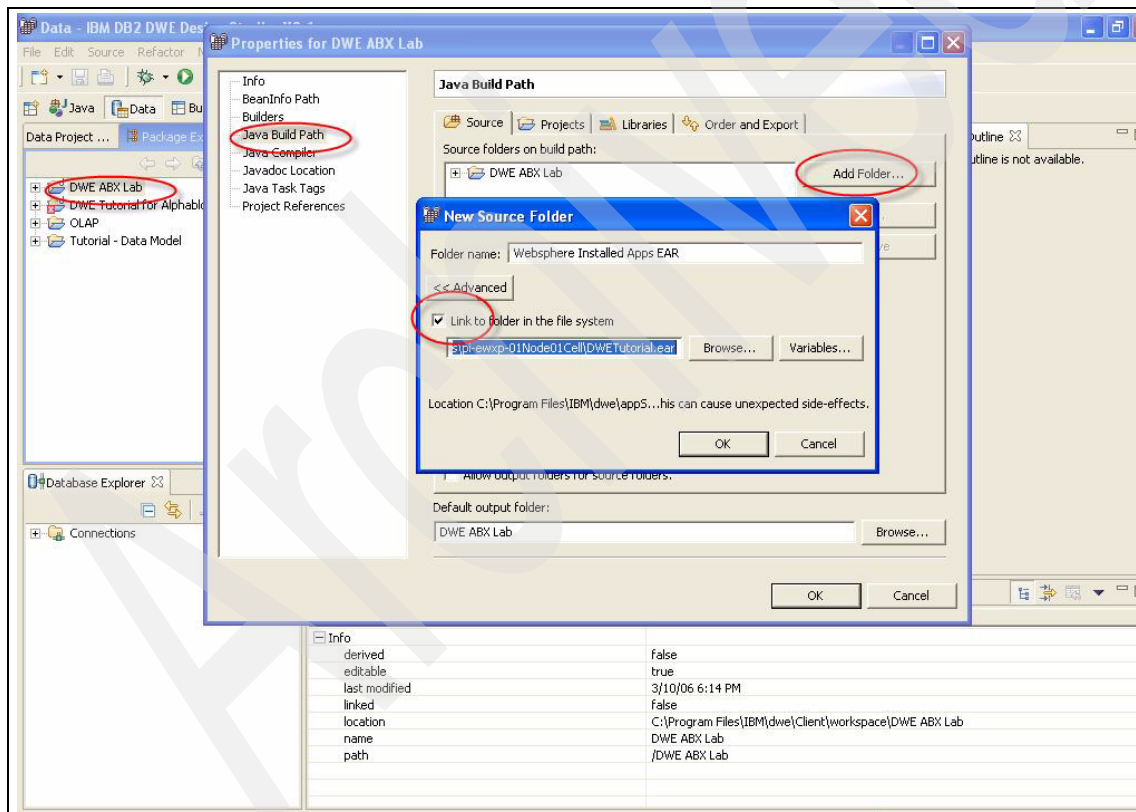


Figure 8-32 Configuring the DWE Design Studio to work with the DB2 Alphablox Application EAR file

Now you should be able to expand the new Java project and see the JSP files within the DB2 Alphablox application ear file. Double-click the file that you wish to

modify and it will be launched in the canvas on the right-hand side. Modify the file as necessary and when you are done select **File** → **Save**. This will directly modify the file in the application ear file. You can see the changes immediately by launching the Alphablox application on a Web browser or refreshing the page if it is already open.

8.4 Integration with DWE Mining

Embedded data mining, as was described in detail in Chapter 7, “DWE and data mining” on page 237, is a mechanism for delivering data mining to non-conventional platforms such as portals and customized reports for use by business analysts, management, and the line of business. Generating Web-based reports with data mining information allows you to share the discovery information that was previously hidden in your data. When data mining components are implemented as DB2 extenders, they can be accessed via SQL. Any reporting or query tool that can export SQL can become a data mining platform, including DB2 Alphablox, Business Objects, Cognos, Microstrategy, and Excel spreadsheets.

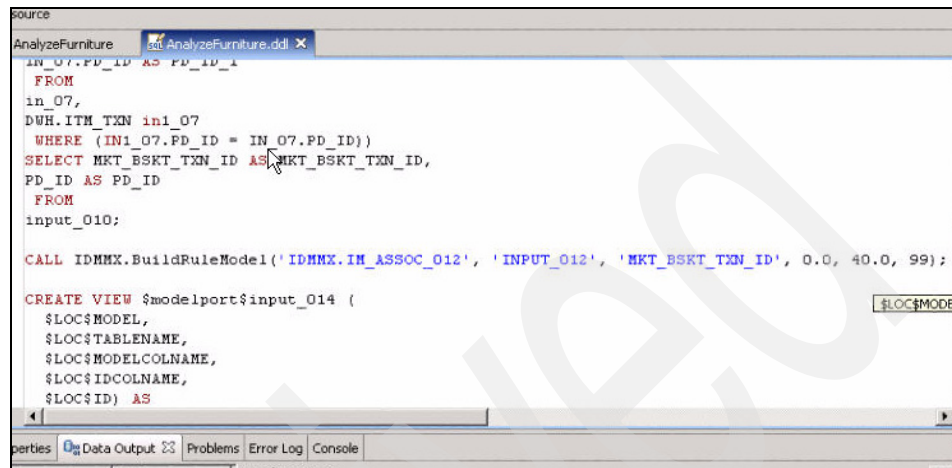
With the strategic approach of embedded data mining in mind, data mining has now been incorporated into the DWE V9.1 suite. DWE provides a graphical user interface for the DWE Data Mining capabilities. This is a significant improvement in ease of use. Now the data mining components can be integrated with other BI tools such as DWE OLAP, while still maintaining accessibility for integration with DB2 Alphablox customized applications and with IBM Business Partner tools and applications, such as SPSS Clementine and SAS.

DWE V9.1 has defined some simple mining techniques, deployed using icons, to make sure that the business analysts understand the quality of the models they create. These mining techniques are discussed in detail in Chapter 7, “DWE and data mining” on page 237.

A significant landmark in embedded data mining is the addition of Mining Blox and custom tag libraries for mining. They simplify Web development of the mining visualization applets with the rich graphical mining results such as the associations visualizer. They let you leverage the DB2 Alphablox UI chart components for simpler charts and build mining Blox by re-using the DB2 Alphablox infrastructure (DHTML rendering), including bookmarks and personalization of views. It also enables you to leverage DB2 Alphablox grid and report components for tabular mining results for association and sequence rules, deviations, clusters, and numeric predictions.

You can also build a mining model using the drag-and-drop interface of DWE Design Studio. Once the model is tuned and tested, you can generate the code

corresponding to the mining model using the **Mining Flow** → **Generate SQL Code** menu option. An example of the code generated is shown in Figure 8-33. The generated SQL script contains the preprocessing steps and the invocation of the DWE Mining stored procedures.



```

Source
AnalyzeFurniture AnalyzeFurniture.dtl x
IN_07.PD_ID AS PD_ID_1
FROM
in_07,
DUH.ITH_TXN in1_07
WHERE (IN1_07.PD_ID = IN_07.PD_ID)
SELECT MKT_BSKT_TXN_ID AS MKT_BSKT_TXN_ID,
PD_ID AS PD_ID
FROM
input_010;

CALL IDMHX.BuildRuleModel('IDMHX.IN_ASSOC_012', 'INPUT_012', 'MKT_BSKT_TXN_ID', 0.0, 40.0, 99);

CREATE VIEW $modelport$input_014 (
$LOC$MODEL,
$LOC$TABLENAME,
$LOC$MODELCOLNAME,
$LOC$IDCOLNAME,
$LOC$ID) AS

```

Figure 8-33 Mining model SQL code generated

You can also use the SQL code in the DB2 Alphablox application. The result is a mining application such as the one shown in Figure 8-34. The business user can access the mining model from a Web browser and make simple selections, which are the input parameters to the model, and then run the report.

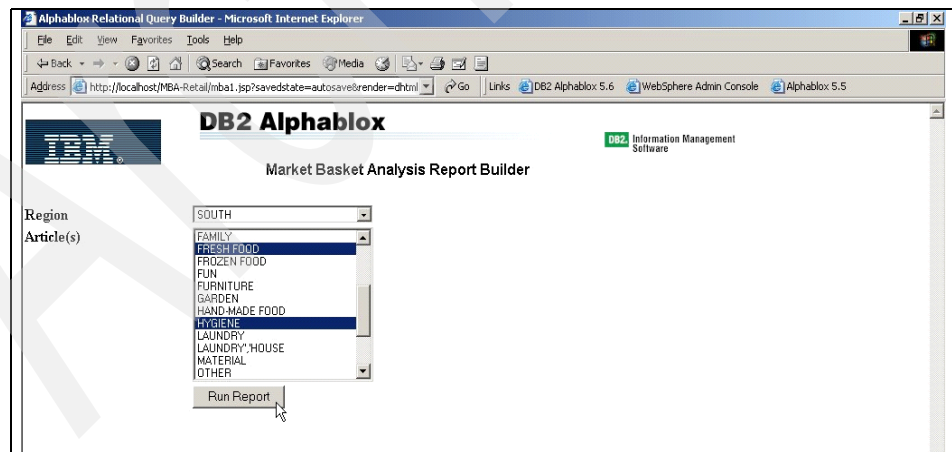


Figure 8-34 Market basket analysis mining application embedded in DB2 Alphablox

A sample report that is generated by the DB2 Alphablox mining application is depicted in Figure 8-35.

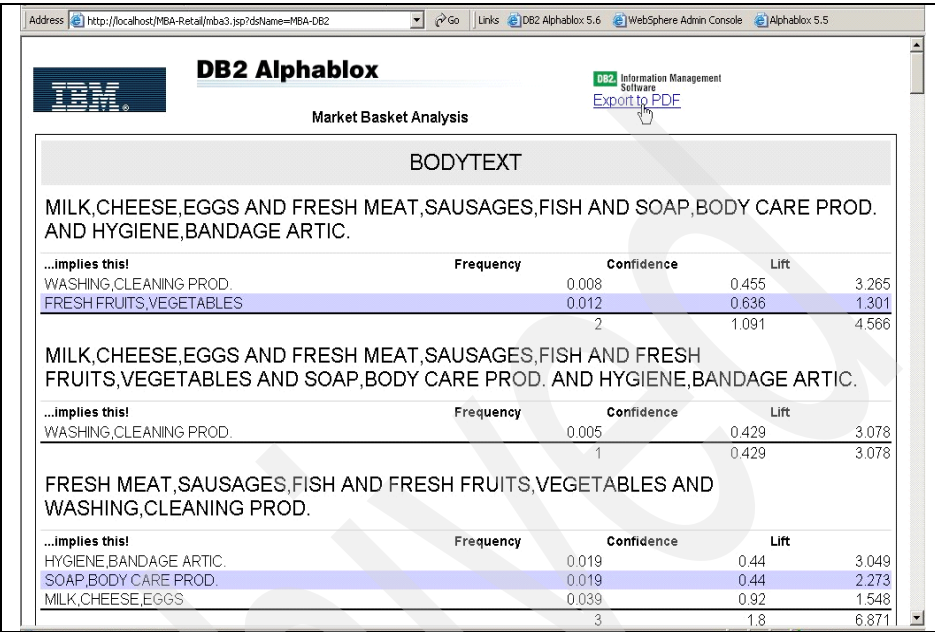


Figure 8-35 Results from the market basket analysis in DB2 Alphablox

Now the mining results are available through an easy-to-use Web interface to access the powerful mining capabilities. The Web interface leverages the power of the mining flow to deliver repeatable analysis for multiple areas without the need for additional IT investment.

8.5 Integration with portal applications

Portals make it possible to integrate information from a variety of applications, content, processes, and people into a single point of interaction. With the ability to manage themes, personalization, profiles, and security, portal applications have become a popular choice for building interactive Web sites, workflows, executive dashboards (as shown in Figure 8-36), among other services.

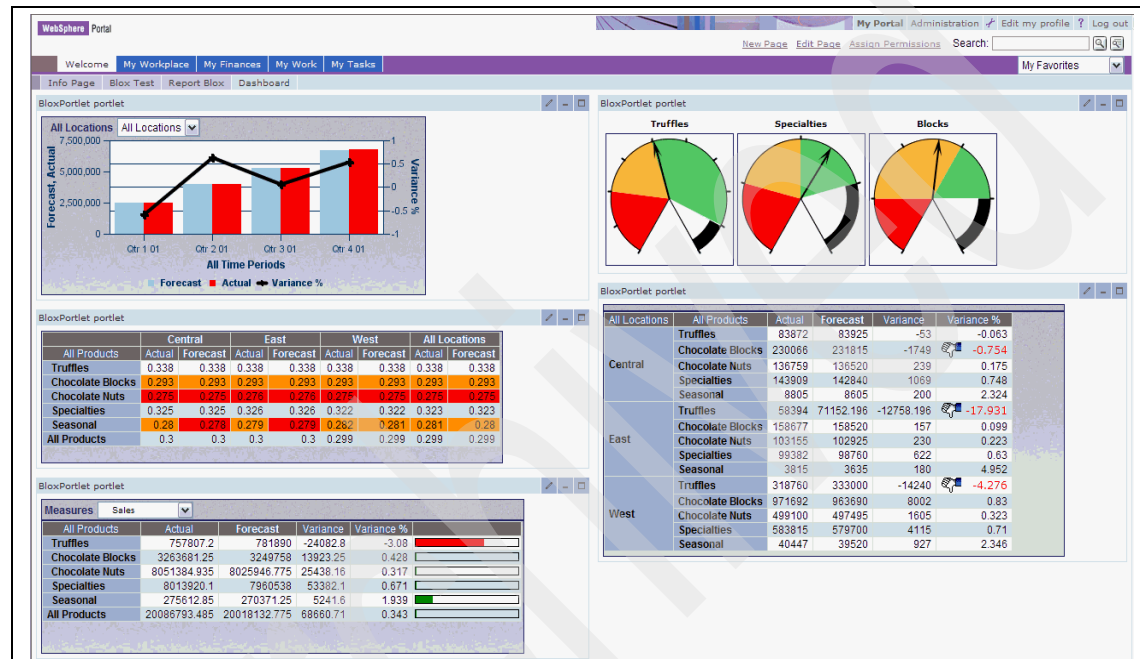


Figure 8-36 Dashboard built using DB2 Alphablox and WebSphere Portal Server

Traditionally, DB2 Alphablox content could be displayed in portals as iFrames. This approach had some drawbacks such as scroll bars to view the portlet content, multiple logins, and complex development. Now DB2 Alphablox is integrated with WebSphere Portal Server (WPS). Integration with WPS provides you with a wide variety of functionality while taking advantage of advanced capabilities of WPS. This functionality includes:

- ▶ Support for portlet containers
- ▶ Out-of-the-box DB2 Alphablox portlet
- ▶ Developing DB2 Alphablox portlets in JSP
- ▶ Using portletAPI tag library for portlet-specific functions
- ▶ Existing tag libraries for putting Blox in portlets
- ▶ DB2 Alphablox tag library for Blox to portlet actions
- ▶ Blox-to-Blox communication

- ▶ Click2Action portlets
- ▶ Support for both IBM and JSP 168 portlet APIs
- ▶ Multiple logins not needed

Note: WebSphere Portal Server is not included in DWE Version 9.1 and must be purchased separately. The currently supported version of WebSphere Portal Server is V5.1.

Users can interact with data using the Alphablox UI and create bookmarks for different data views. Public bookmarks created within one portlet are available to other instances of the same portlet on the portal page. This feature allows portal users to compare different views of the data without having to leave the page. A video that effectively demonstrates some of this functionality in DB2 Alphablox can be accessed online at:

http://demos.dfw.ibm.com/on_demand/Demo/en/IBM_Demo_DB2_Alphablox_Alpha_Financial-Aug05.html

In DB2 Alphablox Version 8.4, which is included with DWE Version 9.1, a new portlet is provided for immediate use. This portlet offers the options to dynamically switch data sources and turn on and off the menu bar and toolbar, as shown in Figure 8-37.

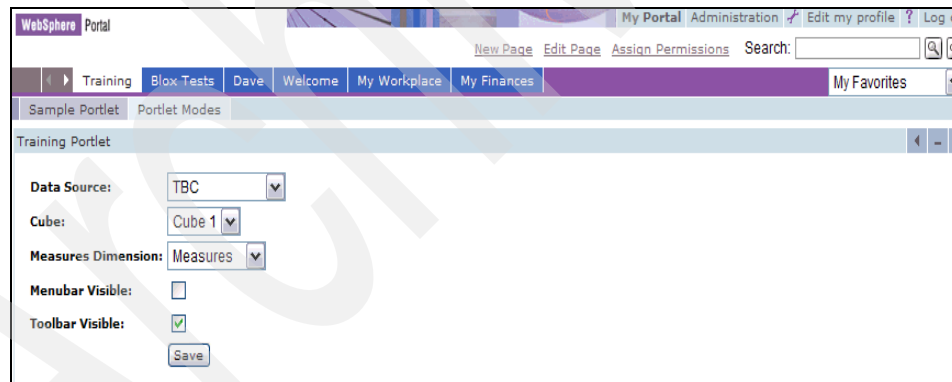


Figure 8-37 Easy-to-modify configuration options in DB2 Alphablox portlet

Portlet-to-portlet communication can occur between two portlets defined on the same page, as shown in Figure 8-38. This feature allows a portlet to send a message to another portlet by name. It also allows a portlet to broadcast a message to all portlets on the page. For example, you could select a product in portlet 1 and could broadcast the product ID to all other portlets on the page, where portlet 2 might display any open sales orders of the selected product. Both the sender and receiver portlets must understand the format and content of the message.

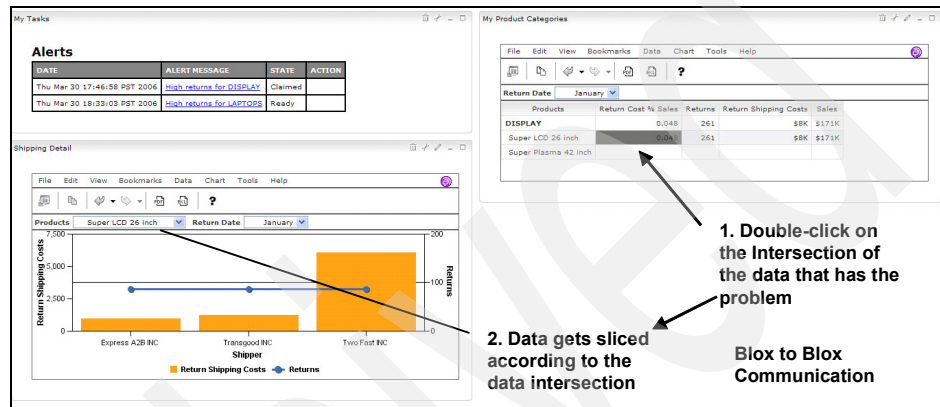


Figure 8-38 Example of DB2 Alphablox portlet-to-portlet communication

Portlets defined within the same portlet application can broadcast messages to the others. They can also share complex data types (java.lang.Object). On the other hand, portlets defined in different portlet applications, but on the same page, can send messages to the other portlets. They can only share text messages (java.lang.String).

Documentation on installing and developing the DB2 Alphablox portlets, including installing WPS, can be found in the DB2 Alphablox Information Center at:

<http://publib.boulder.ibm.com/infocenter/ablxhelp/v8r4m0/index.jsp?topic=/com.ibm.db2.abx.rln.doc/abx-c-relnotes-fftemplates.html>

Refer to the getting-started tutorial titled “Building your first Portlet with Blox components.” The *Blox Portlet Tag* reference can be found in the Developer’s Reference section.

Another good reference for implementing DB2 Alphablox portlets with code examples is the IBM Redbook titled *Improving Business Performance Insight . . . with Business Intelligence and Business Process Management*, SG24-7210.

Section 8.6.2 in the book contains the detailed portlet setup, and Appendix A shows portlet implementation code examples.

DB2 Alphablox can also be deployed on portal applications other than WebSphere Portal Server, using iFrames. But you must make sure that the version of Application Server on which that portal server is running is supported by DB2 Alphablox. For a list of supported application servers, refer to the Systems requirements, Server Configuration section in the DB2 Alphablox Information Center. An example of an implementation on DB2 Alphablox as iFrames can be found on IBM developerWorks® site at:

<http://www-128.ibm.com/developerworks/library/i-bpm4/>

Deploying and managing DWE solutions

To provide the data required for analysis a DB2 Data Warehouse Edition (DWE) solution typically contains several components. These components have been discussed in the previous chapters of this book. The starting point in a DWE implementation is the development of a physical data model to hold the required data. Once a data model is in place the OLAP models can be added to provide optimized access to the data and mining models can be created to enable data-mining analysis of the data. DWE further provides the ability to access the data through InLine analytics, which allow the OLAP and mining information to be seamlessly embedded into Web pages. Underpinning all of this is the necessity to have SQL Warehousing (SQW) data flows and control flows in place to ensure that the data being used for analysis is current and being accessed in the most efficient way.

Once a solution has been developed DWE provides the ability to deploy and manage it in a production environment. The range of tasks required for each component of the solution varies from ensuring that the production databases are enabled for OLAP and mining through to the actual running of the routines to maintain the data. The component within DWE that is provided to deploy and manage these tasks is the DWE Administration Console (Admin Console).

The Admin Console component is installed within WebSphere Application Server, which is the infrastructure used for the runtime environment. The SQW

data flows and control flow to maintain the production data have to be deployed and managed through the Admin Console. For other tasks, such as the deploying of the physical model, OLAP models, and mining models, we recommend using the Admin Console. However, other methods, such as directly deploying SQL scripts, are available.

In this chapter we discuss and describe deploying and managing solutions using the DWE Administration Console. Topics included are:

- ▶ DWE Administration Console
- ▶ Deployment of development code into a test or production environment
- ▶ Managing the solution with the DWE Administration Console
 - SQL Warehousing
 - OLAP functionality
 - Data mining
 - Alphablox
- ▶ Location and use of diagnostic information

9.1 The DWE Administration Console

The DWE Administration Console (Admin Console) is the centralized Web-based administration component of DWE and enables the management and monitoring of production activities through a single client interface. The Admin Console provides the capability to deploy and manage SQL Warehousing (SQW), OLAP models, and mining models. There is a dedicated SQW runtime component that supports the administration of the SQW component. Furthermore, there are DWE runtime components that support general DWE administration tasks, OLAP administration tasks, and mining administration tasks. As shown in 9.6 “Managing DB2 Alphablox within DWE” on page 455, the administering of an Alphablox application is through a separate console that can be accessed from the Admin Console.

Figure 9-1 depicts the introductory screen for the Admin Console, which shows the components of a DWE solution that can be accessed and managed.

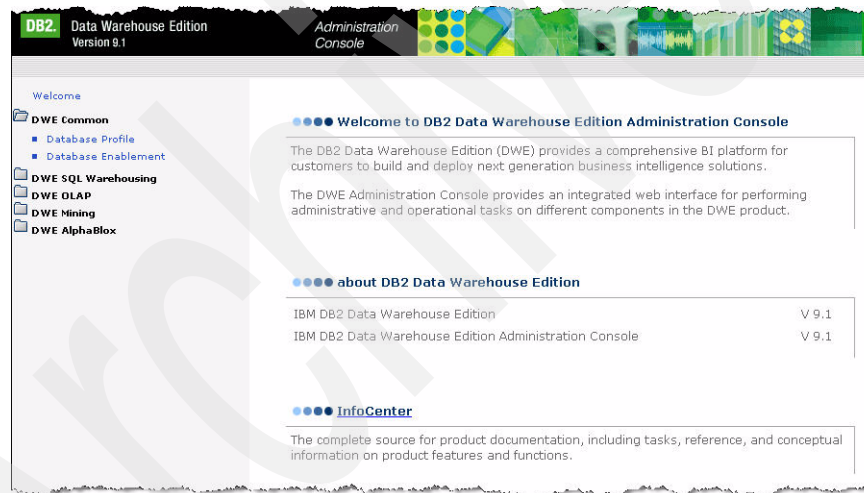


Figure 9-1 DWE Administration Console

In this section we discuss and describe the following:

- ▶ Functionality provided by the Admin Console
- ▶ Architecture of the Admin Console
- ▶ Deployment of the Admin Console
- ▶ Security considerations
- ▶ General administration tasks
- ▶ Locating and using diagnostics

9.1.1 Functionality provided

The functionality offered by the Admin Console is divided into several sections, as illustrated in Figure 9-1 on page 367. Each section is specific to one of the components of DWE that has been used to develop a data warehousing solution, with the exception of the DWE Common section, which is used to define and set up any required databases. The following list introduces the sections of the Admin Console described in detail in this chapter:

- ▶ The DWE Common section provides for the defining of any required data sources and the enabling of these data sources for OLAP and mining. Before a database can be used for OLAP, mining, or SQW activity, it has to be defined within the Admin Console. The underlying connection to a database can either be through a Type 2 JDBC driver, which requires a local DB2 client and the remote database to be cataloged locally, or a Type 4 JDBC driver.
- ▶ The DWE SQL Warehousing section controls the deployment, running, scheduling and monitoring of data warehouse applications that were created in the DWE Design Studio. The Admin Console provides for the viewing of statistics and logs associated with processes and for the troubleshooting of any runtime issues.
- ▶ The DWE OLAP section provides the functionality to import and export OLAP models, and invokes the DB2 DWE OLAP Optimization Advisor. The Advisor identifies materialized query tables (MQTs) and generates the DDL. The OLAP section executes that DDL to create the MQTs. It also allows for displaying the cube model metadata, such as tables, joins, measures, and attributes.
- ▶ The DWE Mining section provides the functionality for the viewing, exporting, updating, and deleting of models in the mining database. Mining models can be imported into the database and models loaded into the cache. The Admin Console further provides a mining visualization tool that provides graphic representations of the results of the mining model.
- ▶ The DWE Alphablox section starts the Alphablox Administration tool, which is used for the deployment and management of Web portal-based analytics, such as scorecards and dashboards.

9.1.2 Architecture

The Admin Console is a platform for running production SQW routines and managing the other DWE components in a runtime environment. To support this functionality the architecture of the Admin Console interfaces with the individual DWE components, as well as the WebSphere Application Server (WAS) environment.

Admin Console interfaces with DWE components

The DWE Admin Console is a J2EE application that is installed locally to a WebSphere Application Server as part of the DWE installation process. The console provides a Common Web interface that is based on Java Server Pages (JSPs)/Java Server Faces (JSF) technology. This common Web interface is combined with a common administration infrastructure that provides services between the Web client and the underlying administration interfaces for the SQL Warehousing (SQW), OLAP, mining, and Alphablox. These interfaces are illustrated in Figure 9-2.

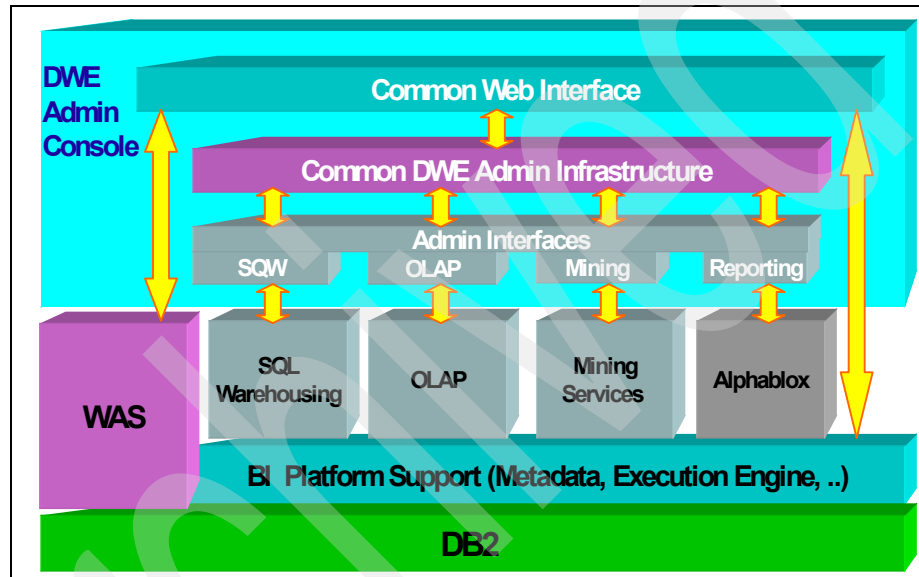


Figure 9-2 Administration Console components

Use of the WAS environment

Within the WAS environment the Admin Console is packaged as an Enterprise Archive file (EAR) entitled *DWEAdminConsole.ear*. This archive contains the Admin Console interface as well as the individual components for SQL Warehousing, OLAP, mining, and an Enterprise Java Bean (EJB) for scheduling. The topology of the Administration Console can be viewed from the WebSphere Administration Console by selecting **Enterprise Applications** → **DWEAdminConsole**.

Being an enterprise application, the Admin Console takes advantage of WebSphere technologies such as:

- ▶ **Scheduling:** The WAS scheduler works in conjunction with the scheduling defined as part of DWE and is used to schedule the SQL Warehousing processes.
- ▶ **JDBC Providers:** WebSphere Application Server provides configurable data source properties including connection pooling and J2C authentication.
- ▶ **Mail providers:** WebSphere Application Server provides a default SMTP mail provider for DWE mail services.

9.1.3 Deploying in a runtime environment

When the Admin Console is deployed as part of a DWE solution, then consideration is required as to where the DWE components and required databases should be created. The topology of the runtime environment is important to ensure the optimum performance of the runtime components, in particular the SQL Warehousing.

It is essential that the WebSphere server containing the Admin Console has access to the databases being used or that the SQW component has access to the appropriate databases. It is also important to have the runtime environment configured in such a way that performance is maximized. The maximizing of performance of a data warehousing solution can often be achieved by simply avoiding unnecessary movements of large amounts of data across networks. However, the collocation of data sources is not always possible, and doing it simply to avoid resource contention may not always be desirable. A commonly used example is that having WAS and DB2 installed on the same server may result in resource contention.

Ensuring access to the necessary databases

As the means of deploying and managing a DWE solution, the Admin Console must have access to the following logical databases, which are illustrated in Figure 9-3 on page 371:

- ▶ The *warehouse source database* contains the source data for the data warehouse application. This can be DB2 database or any other supported database.
- ▶ The *warehouse target database* contains the target data for the data warehouse application.
- ▶ The *SQL* execution database is declared as a property within a data flow in the DWE Design Studio (Design Studio) and is a critical property in the run profile for the flow. This is because the code generated by the Design Studio data flow will be executed within that execution database. In a data

warehouse application that contains multiple data flows, different SQL execution databases might be used to run each data flow. This would be beneficial, for example, in an environment where there are multiple SQL Warehousing processes moving data between several sources and targets. The SQL execution database must be a DB2 database.

- ▶ The *scheduling* database is a DB2 database that is used to schedule the various warehouse applications through WAS. Although used by the Admin Console, the scheduling database is primarily a WAS resource that may contain scheduling information for other enterprise applications.
- ▶ The *DWE Control database* is a DB2 database used to store the warehousing metadata about the warehouse source and target, OLAP, and mining databases. This metadata also includes the process that are currently running and historical data relating to average runtimes of processes.
- ▶ The *OLAP database* is a DB2 database that contains the OLAP information. This database will have the star schema and associated DWE OLAP metadata that has been defined in the Design Studio. This is described further in Chapter 5, “Enabling OLAP in DB2 UDB” on page 79.
- ▶ The *mining database* is a DB2 database that is used to store the mining models. These mining models are created in the Design Studio and used within SQL Warehousing process and Alphablox applications.

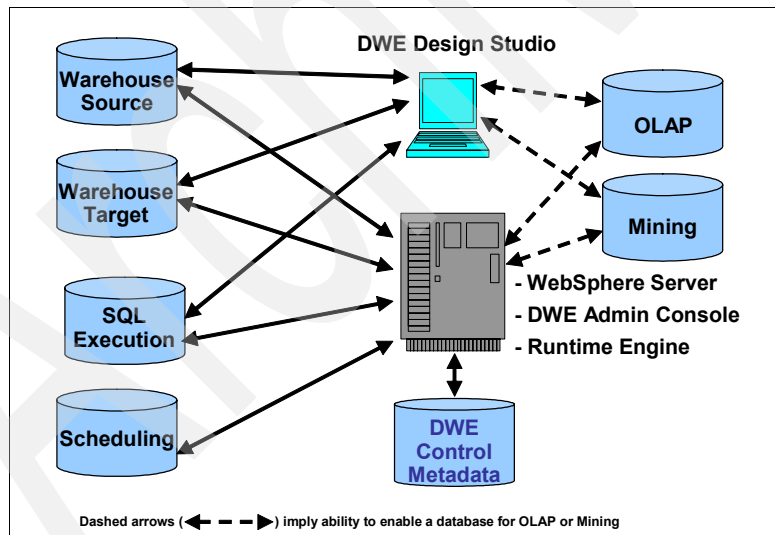


Figure 9-3 Logical databases in a typical runtime environment

The runtime engine accesses the DWE Control database and uses the information stored there to trigger an activity in the warehouse source or target databases. The DWE Admin Console triggers the scheduling database and

results in the creation of schedules within WebSphere, which are then automatically executed by the WebSphere Scheduler. Databases can be enabled for OLAP or mining using either the DWE Admin Console or the DWE Design Studio.

Depending on the characteristics of the environment, many of these logical databases can be combined into one physical database. For instance, the data warehouse target database may also be the OLAP database and include the DWE Mining models.

Note: These databases fulfill a role necessary for a DWE solution and do not replace any existing DB2 databases. One example is the DB2 Tools database, which is primarily used to schedule DB2 maintenance activities.

DWE components in a runtime environment

In a runtime environment the DWE components can be grouped together as either being part of the data warehouse server, the application server, or the client. Figure 9-4 illustrates these three components distributed across three separate systems. The Admin Console and the associated runtime environment for the SQW processes are installed as part of the application server, but are used for the movement of data to the data warehouse server. The actual optimal configuration of a specific DWE runtime environment will largely depend on the type of SQL Warehousing activities that will be executed. The considerations for this are discussed in 9.3.2 “Runtime architecture of DWE SQL Warehousing” on page 391.

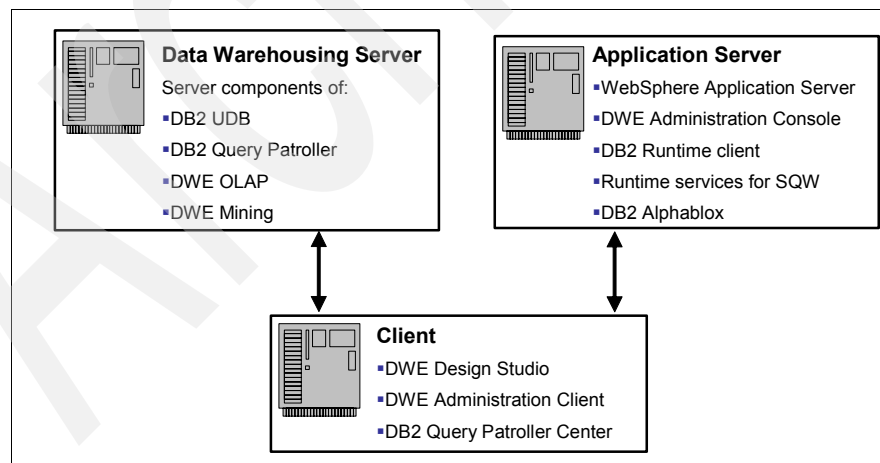


Figure 9-4 Logical groups of DWE components

9.1.4 Security considerations

The DWE Administration Console is an installed WebSphere application, with user and group access managed by the WebSphere Administration Console. As part of the DWE post-installation configuration process, WebSphere Global Security is enabled by default, which ensures that all users log on to the Administration Console. This post-installation process also maps local OS groups to roles within the DWE Administration Console. Roles within the Administration Console are used to categorize the type of operations that users can perform. The requirement for management of user access to the Administration Console and access to data sources can also be managed by WebSphere.

In this section we discuss:

- ▶ Role-based security for the DWE Administration Console
- ▶ WebSphere-managed access to data sources
- ▶ Security housekeeping

Role-based security for the DWE Administration Console

The DWE Administration Console utilizes role-based security, which allows users to be given administrator, manager, and operator roles. The allocation of these roles is performed through the WebSphere Administration Console and allows the performance of a variety of administrative tasks from a single, unified set of console pages. These three roles perform the following list of tasks:

- ▶ Administrator
 - Creates data sources and system resources for SQW applications
 - Installs and uninstalls DWE SQL Warehouse applications
 - Maintains user IDs and passwords for data sources and system resources
- ▶ Manager
 - Manages and maintains installed applications and processes
 - Schedules processes for production runs
 - Monitors results of process executions
- ▶ Operator
 - Monitors process execution status
 - Reports abnormal process execution
 - Restarts processes

The tasks within these roles can be customized by the permissions.xml file, which is located at: <WAS_ROOT>/profiles/default/installedApps/cell-name/DWEAdminConsole.ear/ETLAdmin.war/WEB-INF/classes/secconfig/permissions.xml, and depicted in Figure 9-5.

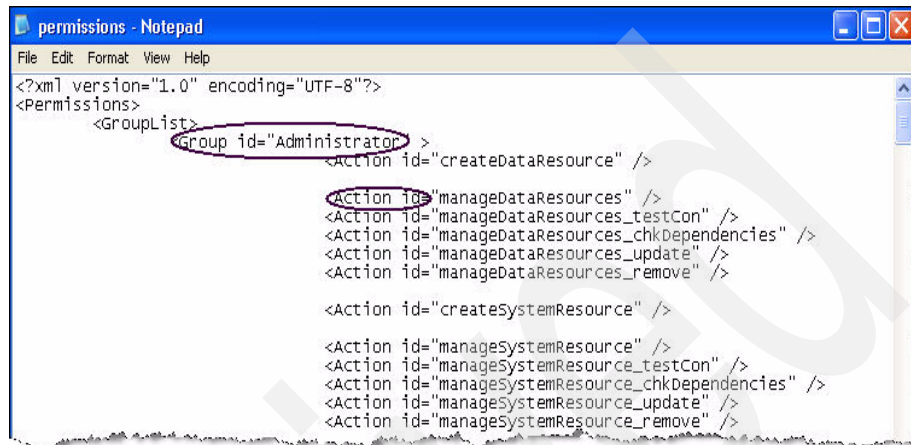


Figure 9-5 Permissions.xml file used to customize Administration Console roles

Within this file the *group ID* refers to the application role, for example administrator, and the *action ID* refers to the SQL Warehouse privilege. If the file is changed, then after saving the file the DWEAdminConsole application needs to be restarted in the WebSphere Administration Console.

Tip: It is a good practice to make a copy of the permissions.xml file before it is altered.

Access to data sources

Databases used or managed by the DWE Administration Console can have this access managed through WebSphere. A Java 2 Connector (J2C) authentication alias can be defined that stores the alias for the data source, the user ID, the password, and a description of the data source. The database can then be accessed using this J2C alias. An alternative to J2C authentication is to use a standard-database log on where access to the database is based on the operating system provided user ID and password.

Note: If data sources are managed by the SQW Administration then the user ID and password will be stored with the SQW metadata and the system administrator will need to keep the password current.

Further information about J2C authentication and detailed instructions on how to set it up can be found in the WebSphere Application Server manuals, located online at:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/csec_j2csecurity.html

Security housekeeping

There are two main types of housekeeping that are required for the DWE Admin Console. If the operating system (OS) password of a user changes, then the password stored within WebSphere will also have to be changed. There may also be a need to update the access to the DWE Administration Console if there is a new user, or if the security requirement for an existing user has changed.

In this section we discuss:

- ▶ Required activities when an operating system password changes
- ▶ Changing user access to the DWE Admin Console

Required activities when an operating system password changes

When the operating system password is changed, the following will also need to be changed if they are being used. If J2C authentication is not being used, then that value would not need to be updated.

- ▶ If WebSphere Global Security is enabled (this is the default):
 - The global server user password needs to be updated in the WebSphere Administration Console. This password is changed by selecting **Security → Global Security → User Registries → Local OS → Server user password**. Apply this change and then select **Save Changes** to the master configuration.
 - The properties file used by SOAP applications should be updated. This file is located at `WAS_ROOT/profiles/default/properties/soap.client.props`.
 - i. Replace the current encrypted password with the new password for the `com.ibm.SOAP.loginPassword` parameter
 - ii. Run the command to encrypt the new password:

```
<WAS_ROOT>/profiles/default/bin/PropFilePasswordEncoder.bat  
<WAS_ROOT>/profiles/default/properties/soap.client.props  
com.ibm.SOAP.loginPassword
```

Where `<WAS_ROOT>` is the installation directory of WAS.
 - The alternative to updating the WebSphere Administration Console, or the configuration file, is to invoke the DWE Configurator to re-enable global security. The DWE configurator can be found in the config directory where the installed image for DWE was copied.

On Windows you would run <installation image directory>\config\config.exe, while on AIX and Linux you would run <installation image directory>/config/config.

- ▶ If a J2C Authentication Alias is used for data sources then the WebSphere Administration Console is used to change the password for the user ID:
 - a. Update the password for the user ID by navigating to **Global Security** → **Authentication** → **JAAS Configuration** → **J2C Authentication**.
 - b. Choose the user ID whose password is being updated.
 - c. Change the password.
 - d. Apply this change and then select **Save Changes** to the master configuration.
- ▶ If a DB2 server is running as a Windows service, the password for the user running the DB2 services will need to be updated to include this new password in **Control Panel** → **Services**.

Note: If passwords have been changed, then WAS should be restarted.

Changing user access to the DWE Administration Console

Changing access to the DWE Administration Console will be necessary if WebSphere administration decides that they want to add a new operating system user or group to access the DWE Admin Console. After the initial install of DWE, three WebSphere roles will be mapped to one or more operating system groups. It may be that during the post-installation configuration the same operating system group is used for all three roles or there may be a separate group for each role.

In any event, these assignments can be changed through the WebSphere Administration Console, assuming that WebSphere global security has been enabled:

- ▶ If a new operating system user/group is required to access the DWE Administration Console, they need to be added to WebSphere:
 - a. They are added in the WebSphere Administration Console by navigating to **System Administration** → **Console Settings** → **Console Users or Console Groups**.
 - b. Choose to **Apply** the new user/group.
 - c. Save the change to the master configuration.

Note: The OS user/group must have already been created, otherwise an error is returned.

- ▶ Once the user/group has been defined to WAS, they need to be mapped to the DWE Admin roles.
 - a. Navigate to **Enterprise Applications** → **DWEAdminConsole** → **Map security roles to users/groups**.
 - b. Select a role.
 - c. Choose to either **Look up users** or **Look up groups**.
 - d. Search for the users or groups to add.
 - e. Select the user or group to add.
 - f. Choose **OK** and save to master configuration.
- ▶ If a user/group needs to be prevented from accessing the DWE Admin Console:
 - a. Navigate to **Enterprise Applications** → **DWEAdminConsole** → **Map security roles to users/groups**.
 - b. Select a role.
 - c. Choose to either **Look up users** or **Look up groups**.
 - d. In the “Selected list of users/groups” select the user/group you want to remove access from, and move them back to the *Available list of users/groups*.
 - e. Choose **OK** and save to master configuration.

Figure 9-6 shows an example of the WebSphere Administration Console where one user is mapped to the Administrator role and the same group is mapped to the Administrator, Manager, and Operator roles.

Enterprise Applications

Messages

Changes have been made to your local configuration. Click [Save](#) to apply changes to the master configuration.

The server may need to be restarted for these changes to take effect.

Enterprise Applications > DWEAdminConsole > Map security roles to users/groups

Map security roles to users/groups

Each role that is defined in the application or module must map to a user or group from the domain user registry.

Look up users Look up groups

| Select | Role | Everyone? | All authenticated? | Mapped users | Mapped groups |
|-------------------------------------|---------------|--------------------------|--------------------------|------------------|----------------|
| <input checked="" type="checkbox"/> | Administrator | <input type="checkbox"/> | <input type="checkbox"/> | PI-EWXP-01\angus | Administrators |
| <input type="checkbox"/> | Manager | <input type="checkbox"/> | <input type="checkbox"/> | | Administrators |
| <input type="checkbox"/> | Operator | <input type="checkbox"/> | <input type="checkbox"/> | | Administrators |
| <input type="checkbox"/> | Everyone | <input type="checkbox"/> | <input type="checkbox"/> | | |

OK Cancel

Figure 9-6 User/group assignment to WAS roles

Note: If changes are made to the WebSphere master configuration then the change should be confirmed by clicking the **Save** option, as indicated in Figure 9-6.

9.1.5 General administration tasks

The DWE Administration Console primarily works with databases. One of the key initial tasks is to define the required databases and enable these databases for OLAP activities, mining activities, or both. The definition of a database does not create the database. Rather, it links to an existing local or remote database. Databases required for use by data warehousing processes need to be defined by the Create Data Source section within DWE SQL Warehousing. However, a warehousing data source can use a WebSphere data source that has been defined within the DWE Common section of the DWE Administration Console.

The DWE Common section of the DWE Administration Console carries out two database-related tasks:

- ▶ Defining databases in the DWE Administration Console
- ▶ Enabling defined databases for OLAP and mining

Defining databases in the DWE Administration Console

Defining a database within the DWE Administration Console is done through the DWE Common → Database Profile section. This section contains database connection information, which can be shared and referenced by the OLAP and data-mining functions. The defined data sources can already have been defined to WebSphere or can be defined within WebSphere using this section of the DWE Administration Console. The DWE Administration Console provides functionality to create data sources and update existing data sources. It also allows us to remove selected data sources from the DWE Administration Console or to test the connectivity of a data source.

Figure 9-7 shows how a new database profile is created. As depicted, the names given to the database profile, data source, and the JNDI name can be different, but the database name must match the actual database name.

The screenshot displays the 'Create Database Profile' dialog in the DWE Administration Console. The 'Database Profile' tab is selected, showing the following fields and options:

- Database Profile Name:** WarehouseDB
- Profile Create Option:** Two radio buttons are present: 'Create Profile using an Existing Data Source' (unselected) and 'Create Profile with a New Data Source' (selected).
- JDBC Provider:** DB2 Universal JDBC Driver (XA) (selected from a dropdown menu)
- Data Source Name:** WHSDb
- JNDI Name:** jdbc/dwe/whsdb
- Database Name:** PRODDb

A callout box with an arrow pointing to the 'Database Name' field contains the text: 'Database name must match the actual database name'.

Figure 9-7 Creating a database profile

Tip: When creating a new data source we recommend that you prefix the JNDI name with `jdbc/dwe/`, as this will allow easy identification of the data source.

A new data source can be defined as using either the DB2 Universal JDBC driver (Type 4) or the CLI-based JDBC driver (Type 2). The main difference is that the Type 4 driver directly accesses the DB2 server, while the Type 2 driver uses a locally cataloged database.

There is also the option to select the XA version of the driver for both the Universal JDBC driver and the CLI-based driver. This driver allows for distributed transactions and should be used if two-phase commit operations are likely.

Enabling defined databases for OLAP and mining

Once the database access has been configured, the database can be enabled for mining, OLAP, or both. Until a database is enabled for OLAP and mining, it lacks the necessary objects, such as stored procedures and tables, to support those activities. Enabling a database within the DWE Administration Console is performed through the **DWE Common** → **Database Enablement** section. The Administration console can first connect to a database profile and check whether OLAP/mining has been enabled. If the database has not been enabled, then the enabling process can be invoked to create the required database objects.

- ▶ Enabling a database for OLAP: When enabling a database for OLAP the only prompts are for the database name to be enabled. After a short period of time, during which the objects are being created, the database is enabled.
- ▶ Enabling a database for mining: When a database is enabled for mining there are some options before the enabling starts. One option is whether the mining routines should be deployed as fenced or unfenced. The difference is that fenced routines are invoked directly by DB2 while unfenced routine are invoked externally. If you plan to use caching for mining models, which is a performance feature discussed in 9.5.3 “Caching the mining model for performance” on page 454, then the database must be enabled in fenced mode. This is because the unfenced mode for routines does not support model caching.

Another decision prior to enabling is whether any database configuration parameters should be changed from the defaults to values better suited for mining. Assuming the database has a default configuration, then choosing to update the parameters will result in the database configuration parameters being changed, as illustrated in Example 9-1.

Example 9-1 DB CFG parameters updated by enabling a database for mining

```
APPL_CTL_HEAP_SZ from 128 to 10000
APPLHEAPSZ from 256 to 1000
LOGSECOND from 3 to 7
```

9.1.6 Locating and using diagnostics

The DWE Administration Console is an application within WebSphere Application Server (WAS), which means that the WAS tools can be used to provide diagnostic information. The primary source for WAS diagnostic information is the WebSphere Administration Console, which in a default installation can be found at the secure site:

<https://localhost:9043/ibm/console>

Or at the non-secure site:

<http://localhost:9060/ibm/console>

This console provides logging and tracing for WAS, runtime messages, and the ability to trace data sources that have been defined within WAS.

- ▶ Runtime messages

The runtime messages are accessed from the WebSphere Administration Console by navigating to **Troubleshooting** → **Runtime Messages**. These messages are grouped as either errors, warnings, or messages.

- ▶ Logging and tracing

For each instance of the WebSphere Application Server, the logging and tracing can be viewed by initially navigating to **Troubleshooting** → **Logs and Tracing** and then selecting the server for which diagnostics are to be viewed.

There are five options for logging and tracing:

- Diagnostic trace

This option allows the viewing and modifying of the properties for the diagnostic trace service. By default, the diagnostic tracing is enabled and has a maximum file size of 20 MB.

- JVM™ logs

This allows viewing and modifying of the settings for the Java Virtual Machine (JVM) System.out and System.err logs for the server. The JVM logs are created by redirecting the System.out and System.err streams of the JVM to independent log files.

The System.out log is used to monitor the health of the running application server. The System.err log contains exception stack trace information that is useful when performing problem analysis. There is one set of JVM logs for each application server and all of its applications.

- Process logs

The process logs are created by redirecting the standard out and standard error streams of a process to independent log files. These logs can

contain information relating to problems in native code or diagnostic information written by the JVM. By default, the logs are named `native_stdout.log` and `native_stderr.log` and can be viewed from within the console.

- IBM service logs

The IBM service log contains both the WebSphere Application Server messages that are written to the `System.out` stream and special messages that contain extended service information that can be important when analyzing problems. There is one service log for all WebSphere Application Server Java virtual machines (JVMs) on a node, including all application servers, and their node agent (if present). A separate activity log is created for a deployment manager in its own logs directory. The IBM Service log is maintained in a binary format and is primarily used by support personnel.

- Change log detail levels

This section is used to configure and manage log level settings. Log levels allow you to control which events are processed by Java logging. The default setting is `*=info`, which means that all traceable code running in the application server, including WebSphere Application Server system code and client code, is processed and that all informational messages are captured.

- Database diagnostics

Database-specific diagnostics are available through log files or through tracing of specific database.

- Database diagnostic log files:

There are two primary database diagnostic files, the notification log file and the DB2 Instance diagnostic log (`db2diag.log`) file. These two files are created for each instance of DB2 and are located on the database servers.

- DB2 notification log

This log file contains information about the DB2 instance that will be useful to DBAs. On a Windows system this information is written to the Application Event log, which can be found by selecting **Start** → **Settings** → **Control Panel** → **Administrative Tools** → **Event Viewer** and choosing the **Application** option.

If the log is sorted by source, the DB2 messages can be easily examined.

On a Unix/Linux system the notification log can be found in the location specified by the `DIAGPATH` database manager configuration parameter. The file name is `<instance_name>.nfy`, and a typical example would be `/home/db2inst1/sqllib/db2dump/db2inst1.nfy`.

- DB2 Instance diagnostic log (db2diag.log)

This log file contains detailed information about the operation of the DB2 instance. This log file is intended for use by IBM support personnel in the event that an issue requires being reported.

On Windows, this file is located in

<install_dir>\SQLLIB\<instance_name>\db2diag.log, with a typical example being C:\Program Files\IBM\SQLLIB\DB2\db2diag.log.

On UNIX/Linux this file is located in

<instance_home>/sqlib/db2dump/db2diag.log, with a typical example being /home/db2inst1/sqlib/db2dump/db2diag.log.

– Database tracing

If required, tracing can be enabled for the databases using either WebSphere-managed JDBC tracing or by using DB2 CLI traces.

- WebSphere JDBC tracing

If the data source has been defined as using the Type 4 JDBC driver, then the trace can be set up by navigating to **Resources** → **JDBC Providers** → *provider_name* → **Data Sources** → *database_name* → **Custom properties**.

The DB2 JDBC driver properties include traceFile, traceFileAppend, and traceLevel. These properties and others are discussed in the DB2 Information Center, and can be found by selecting **Developing** → **Database Applications** → **Programming Applications** → **Java** → **DB2 Universal JDBC driver** → **Properties**.

- DB2 CLI/ODBC/JDBC tracing

If the DB2 Type 2 JDBC driver is being used and the database connection is not established through WAS, then the tracing facilities in the DB2 CLI driver can be used. Details on how to enable this tracing can be found in the DB2 Information Center by referring selecting **Developing** → **Database Applications** → **Programming Applications** → **CLI** → **Diagnostics and error handling** → **CLI/ODBC/JDBC trace facility**.

The DB2 Information Center can be accessed online at:

<http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp>

9.2 Deploying the physical data model

When moving from a development DWE environment to a production environment, the first part of a deployment is the database structure that has

been created from the physical model. There are a number of available methods to deploy the database structure, whether they are part of the DWE Administration Console or through some other method.

In this section we discuss deployment using:

- ▶ The DWE Design Studio
- ▶ The Administration Console
- ▶ Native DB2 functionality

9.2.1 Deployment using the DWE Design Studio

The DWE Design studio has the capability to deploy the physical data model directly to a target database or to generate the model to a file. The physical data model can be generated by right-clicking a database in the DWE Design Studio after navigating to **Projects** → **Databases** → **Model Name** → **Database**.

If the DWE Design Studio has a connection to the production database and the users have the correct authority, then the physical model can be directly deployed to the production database. The option to generate a file should, however, be considered for a variety of reasons:

- ▶ Does the data model require being entered into a source control system?
- ▶ If deployment is to be done as part of deploying the warehousing applications then it needs to be included as a DDL file.
- ▶ Will the production tablespaces have the same naming and paths as development?
- ▶ Can the performance of the physical data model be enhanced by modifying the DDL to add multi-dimensional clustering (MDC) or additional indexing?

The generation or deployment of a data model is shown in Figure 9-8 on page 385. In this example the table CVSAMPLE.LOCATION has been defined without any indexes and the tablespace is the default of USERSPACE1. The option to *Open the DDL file for editing* has been chosen so the values can be changed in the generated model.

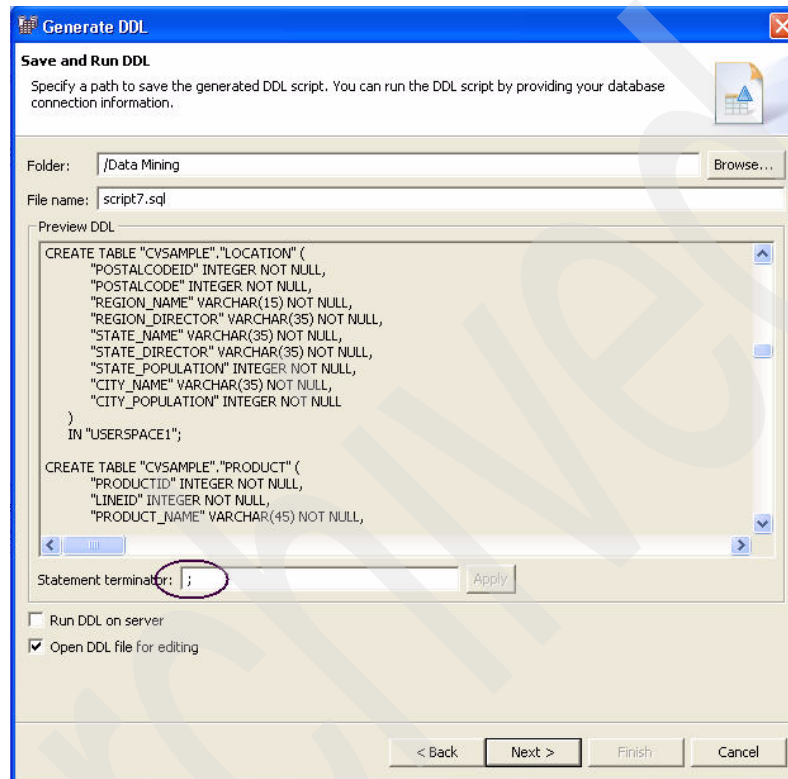


Figure 9-8 Generating DDL for deployment

9.2.2 Deployment using the DWE Administration Console

The DWE Administration Console provides two methods for deploying a data model. These two methods both require that the data model be provided in a file, whether this file is generated by the DWE Design Studio or by some other method, such as *db2look*. The file containing the model can be deployed as part of an SQL Warehousing deployment or run directly as an SQL file.

In this section we discuss the following two deployment options:

- ▶ Deploying the data model within a warehousing deployment
- ▶ Deploying the data model as an SQL file

Deploying the data model within a warehousing deployment

It is possible to package the data model together with the SQL Warehousing applications that will use the deployed tables. If one or more DDL files are included with the deployment package in the Design Studio then the files will be executed by the Administration Console as part of the application deployment.

The process for this is as follows:

1. Within the project in the DWE Design Studio the DDL can be generated from the development database into a file or a file can be imported containing the DDL into the project.
2. When a new *data warehouse application* is created, the packaging of the deployment code allows the specification of one or more DDL files that will be included with the application. When the deployment .zip file is created, the DDL files that have been specified are contained within the /etl/misc/ folder.
3. When a data warehouse application is deployed within the DWE Administration Console there is an option in step 1 (General) called *Execute Deployment Code Units*. This option executes any files in the /etl/misc/ folder and is set to yes by default.

Tip: The statement terminator for the DDL file should be defined as a semi-colon (;).

The deployment of a data warehousing application is discussed in more detail in 9.3.4 “Deploying and managing warehouse applications” on page 409.

Deploying the data model as an SQL file

The deployment of the data model as part of a warehousing deployment is the recommended means of initially creating the tables on a production system. If there is a requirement for creating tables after a deployment or performing additional SQL operations then the DWE Administration Console provides this functionality. The ability to run SQL statements is provided as part of the DWE OLAP functionality and is designed to run statements containing optimizations for any OLAP models.

However, any SQL statements can be run, including DDL, from **DWE OLAP → OLAP Optimization → Run SQL Script**. The only restrictions are that the target database has been enabled for OLAP and that if the database is remote then it must be cataloged locally to the DB2 client.

Tip: The statement terminator for the DDL file should be defined as a semi-colon (;).

9.2.3 Deployment using native DB2 functionality

As the generated DDL file is a standard DB2 file, it can be deployed using:

- ▶ DB2 Command Center: The DDL file can be opened in the DB2 Command Center and executed. The statement terminator specified in the Command Center needs to match the terminator in the DDL file.
- ▶ DB2 CLP: The DDL file can be executed from a DB2 command line. If the default statement terminator, as highlighted in Figure 9-8 on page 385, is used, then the command would be:

```
db2 -tf ddlscript.sql
```

If a different terminator is specified, then the command would be:

```
db2 -td<terminator> -f ddlscript.sql
```

9.3 DWE SQL Warehousing

Now that the database objects have been created within a data warehouse, there is a need to design, develop, and deploy a means of maintaining the data within these objects. If a table is created and initially populated with data, then the contents of this table have to be maintained to ensure that we are using the correct data. For example, if in one of the dimensional tables there is a product that moves within its hierarchy then this change needs to be reflected in the dimensional table. If this change is not reflected, then incorrect business decisions may be made based on an old version of the data.

There are a number of ways to maintain the contents of database tables, such as operating system scripts, SQL scripts, or an ETL tool such as Ascential DataStage. The DWE Design Studio provides the functionality to develop, validate, and test warehousing applications that primarily work with data once it is within a database. The DWE Administration Console then takes these applications that have been created and deploys them into a production environment. Once the application is deployed, the Administration Console can be used to run and monitor data warehouse applications that contain specific executable processes. The Administration Console can also be used to set up scheduling of these processes, view execution statistics, and analyze log files.

Before continuing, we want to define what we mean by an application and a process:

- ▶ An *application* represents one or more processes that the Design Studio users have assembled into a package for deployment. These processes might consist of a set of control flows that build or modify a data warehouse according to a fixed or on-demand schedule. Alternatively, an application

might contain a single data flow inside a single control flow that updates one dimension table.

- ▶ An individual *process* in the runtime environment is equivalent to a control flow in the design-time environment. When a schedule or process is started, all of the activities within a particular control flow, including its data flows, are executed.

In this section we concentrate on the SQL Warehousing functionality in the DWE Administration Console, the runtime deployment of warehousing applications, and the following:

- ▶ An overview of the SQL Warehousing components
- ▶ Runtime architecture considerations for SQL Warehousing
- ▶ How to manage database and system resources
- ▶ Details on how to deploy and manage the warehouse applications
- ▶ Details on how to manage the warehouse processes
- ▶ How to find the warehousing diagnostics
- ▶ How to use the warehousing diagnostics

Full details on the development of an SQL Warehousing application can be found in Chapter 6, “Data movement and transformation” on page 137.

9.3.1 An overview of the SQL Warehousing components

The creation of an SQL Warehousing application and the subsequent deployment of this application are part of the same overall process, which has the aim of populating and maintaining the production databases. However, the different requirements of developing and deploying a warehousing application requires different components and processes. Figure 9-9 illustrates the components involved in the building of an application in the DWE Design studio and the deployment in a runtime environment.

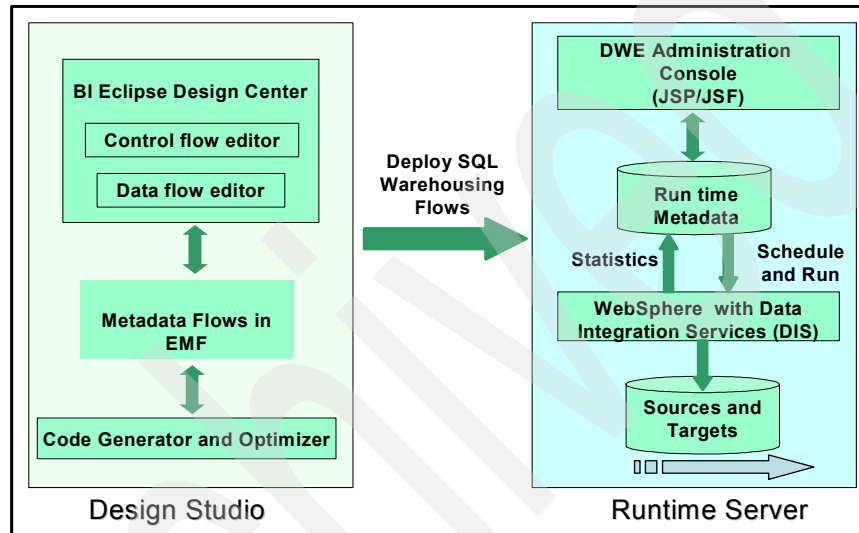


Figure 9-9 SQL Warehousing components in development and production

In this section we introduce:

- ▶ SQL Warehousing in the DWE Design Studio
- ▶ SQL Warehousing components in a runtime environment

SQL Warehousing (SQW) in the DWE Design Studio

The DWE Design Studio uses SQL Warehouse operators to model data flows and control flows. A data flow transforms operational data into warehousing data. An example of this is to take the contents of a source table and copy the current day data into a target table. A control flow contains logically related data flows and non-data flow activities, such as e-mail and FTP activities, to describe the work flow for building a warehousing application. Within a control flow the data flows can be conditional based on the results of the previous data flow. For example, if a data flow is successful the next data flow can be executed. Or if the data flow fails, an e-mail can be sent to the support team. The control flow and data flow metadata are displayed in the DWE Design Studio as graphical process

flows and are stored in an XML format for future retrieval and updates. When you deploy the data flow or control to the runtime environment, it is this source XML file that is deployed.

After the control flows have been validated and verified with the corresponding activities/data flows, they can be packaged into SQL Warehouse applications. The packaging of the application generates the runtime code for the control flows and data flows, which is stored as an *execution plan graph* (EPG). An execution plan graph describes the execution of a data flow, control flow, and mining flow as a sequence of nodes. Different types of nodes in an EPG depict the various execution points in a flow, for example, the start of execution, end of execution, and start of database transaction. Once packaged, the application can be deployed by the DWE Administration Console into the runtime environment.

More details on the building of an SQL Warehousing application can be found in Chapter 6, “Data movement and transformation” on page 137.

SQL Warehousing components in a runtime environment

The starting point for the runtime deployment of an application is the packaged SQL Warehousing application that has been created within DWE Design Studio. The runtime environment that the application is deployed to consists of the individual components in Figure 9-9 on page 389 with an overall topology as shown in Figure 9-10 on page 391.

- ▶ The DWE Administration Console is used to deploy and manage the SQL Warehousing applications.
- ▶ The *runtime metadata database* is a DB2 database that is used to store the metadata about the running of processes and the storing of runtime statistics.
- ▶ All interaction with the SQL Warehouse metadata, source and target tables is handled by WebSphere or DWE Data Integration Service (DIS), which is a part of the Administration Console and used as the runtime environment for SQL Warehousing. DIS can access the data sources via a WebSphere application interface using JDBC drivers or a DB2 application interface (DB2CMD).

- The *sources* and *targets* are databases that are referenced by data flow activities and can be local or remote to the DWE Admin Console. The source and target databases for SQL script activities can also be local or remote to the DWE Admin Console, and the connections to these target databases are managed by DIS. In Figure 9-10 the source, target, and runtime databases are all remote to the WAS processor.

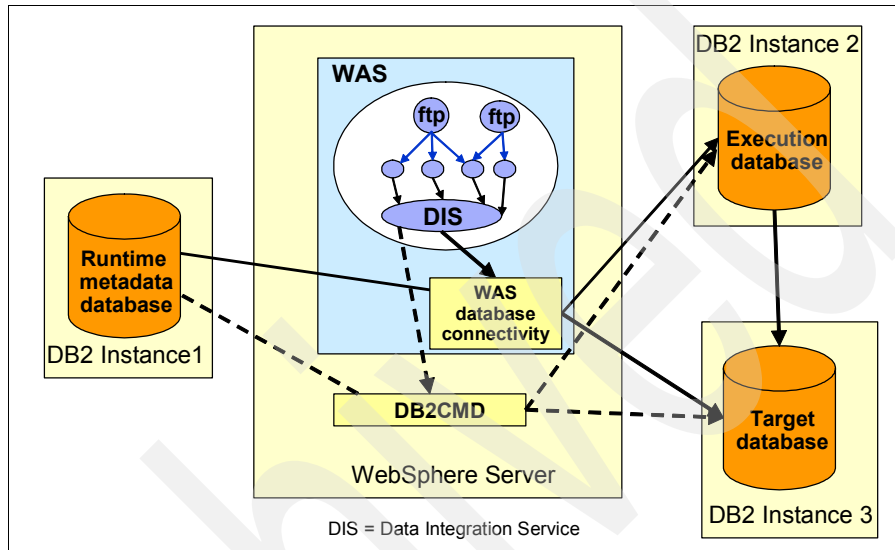


Figure 9-10 A deployment topology for the SQL Warehousing components

9.3.2 Runtime architecture of DWE SQL Warehousing

Before the deployment of an SQL Warehousing application, the architecture of the runtime environment should be carefully considered. In this instance architecture relates to where the different DWE components are installed in relation to each other. The DWE components are flexible enough to be deployed on separate processors or all to be installed on the same processor. In Figure 9-2 on page 369 we showed a typical layout of the main DWE components with the data warehouse server and WebSphere server being on different physical processors. This is a typical layout. However, the goal is to introduce SQL Warehousing scenarios that may benefit from a different layout.

When we create a runtime environment there are a number of factors that can influence performance. Factors such as network traffic between the servers, read and write speed for the data on disk, and the specification of the hardware that the software has been installed on can all impact performance. Although important, these factors are not specific to a DWE environment and are outside the scope of this discussion. There are, however, components within a DWE

environment whose location can have a significant impact on the runtime performance of a SQL Warehousing application.

These components are the SQL Warehousing runtime environment (DIS) and the SQL Warehousing execution database. In this section we explain the roles of these components and make deployment recommendations. These deployment recommendations also include suggestions for the configuration of databases used for DWE SQL Warehousing.

In this section we discuss:

- ▶ The location of the SQL Warehousing runtime environment
- ▶ The location of the SQL Warehouse execution database
- ▶ Deployment considerations

Note: The execution database is specified while developing an application in the Design Studio and can be mapped to a runtime database when the SQW application is deployed.

Location of the SQL Warehousing runtime environment

When the DWE Administration application is installed in the WebSphere Application Server, one of the components provides the runtime environment for the SQL Warehousing activities. This component is known as the SQL Warehousing runtime environment, or the Data Integration Service (DIS). This is the component that runs the Design studio operators such as:

- ▶ Execute a select statement against a source database.
- ▶ Insert data into a target database.
- ▶ Execute an SQL or OS script.
- ▶ Perform the FTP operation from and to a remote system.
- ▶ Wait for a file before commencing the next process.

The location of the SQL Warehousing runtime environment in relation to the location of the commands is important. This is because the SQW runtime environment will carry out activities from the system where the SQL Warehousing runtime environment is installed. In the following section we describe two examples of how a command is run based on how the runtime environment is configured.

- ▶ Deploying an FTP operator
- ▶ Deploying an SQL script

Deploying an FTP command

An FTP operator is commonly used to move data from one system to another, at which point the flat file can be used as an input for a LOAD command. For more details on defining an FTP operator, refer to 9.3.3 “Managing warehouse

resources” on page 405. When using the SQL Warehousing FTP operator in a runtime environment, the execution process depends on where WAS/DIS has been installed. Figure 9-11 illustrates two scenarios that show how the process of an FTP command depends on where the WebSphere server is installed in relation to the FTP files.

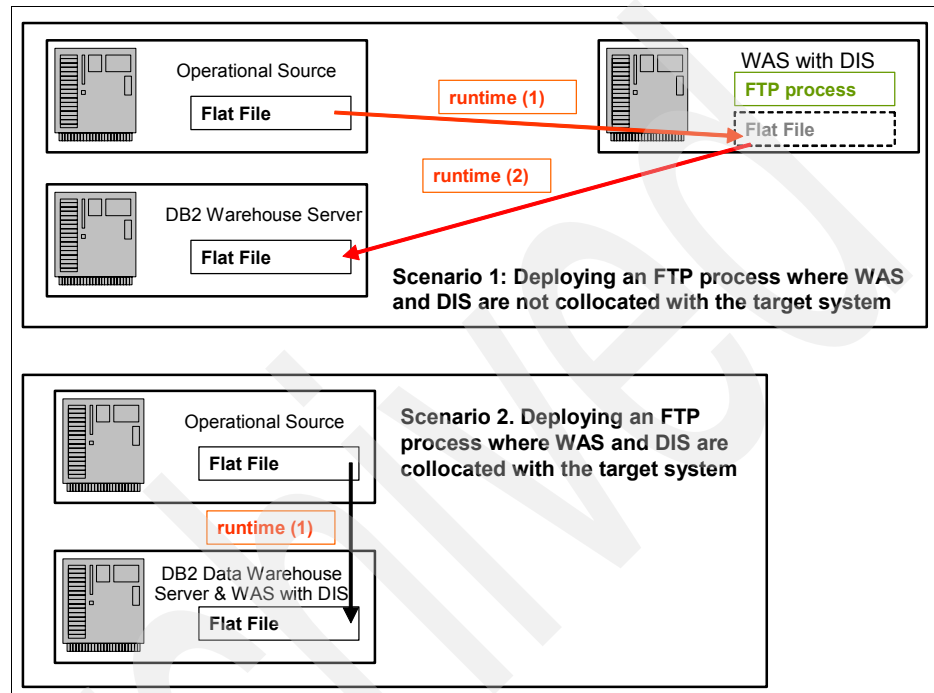


Figure 9-11 Using an FTP operator in two different runtime environments

In scenario 1 (Figure 9-11) the file will be fetched from the Operational Source and written to a local file residing on the WebSphere Application Server system. An FTP operation then transmits the local file on the WAS system to the data warehouse server. In scenario 2 (Figure 9-11) the file is fetched from the operational source and then written directly to the WebSphere Application Server, which is the same server as the data warehouse. By collocating WAS with the data warehouse server, the flat file was only moved once.

Deploying an SQL script

An SQL script can be used to execute any SQL statements against a DB2 database. If an SQL script-based system was being migrated to a DWE environment, then the control flows will initially be comprised of SQL scripts. Within the DWE Design Studio the command operator can be a DB2 SQL script, a DB2 shell command, or custom executable code. When the SQL Warehousing

operator is used in a runtime environment the execution will depend on where the WAS/DIS has been installed.

Figure 9-12 shows how the execution of an SQL operator depends on where the WebSphere server is installed in relation to the warehouse database.

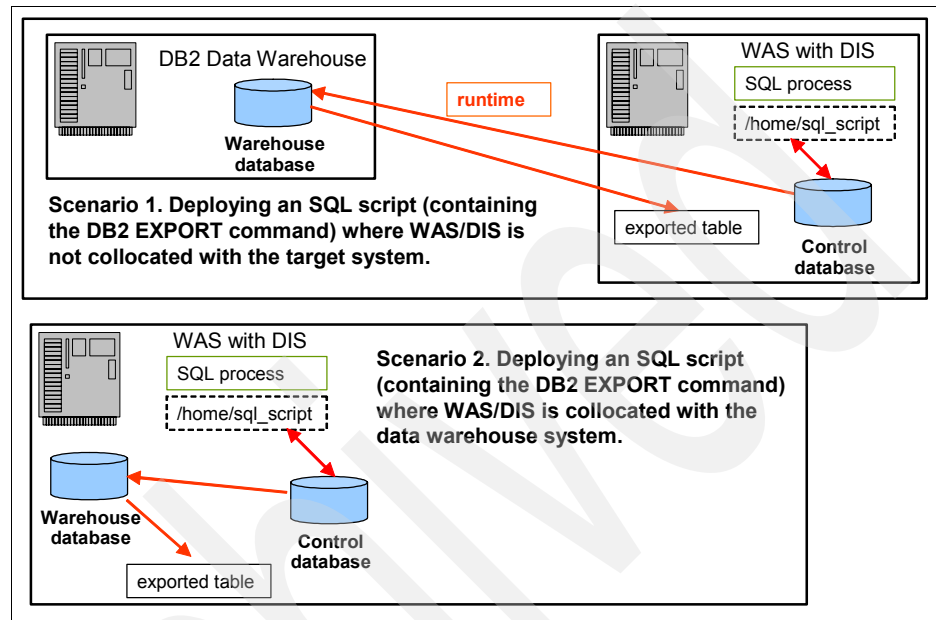


Figure 9-12 Deploying an SQL script

When the SQL Warehousing application is deployed, the necessary runtime code units are written to the user-specified directory on the application server system and the references to these code units (EPGs) are stored in the SQL Warehousing runtime metadata in the DB2 control database. In scenario 1, the SQL command is run locally at the server where the DWE Administration Console is installed. This means that the data is exported from the data warehouse to the WAS processor. In scenario 2 the movement of data is again simplified by collocating the data warehouse and the WAS processor.

The location of the SQL Warehousing execution database

The execution database has been introduced in 9.1.3 “Deploying in a runtime environment” on page 370, and is defined during the development of a data flow in the DWE Design Studio. When the application is deployed into production the development databases can be mapped to the production equivalents, which is discussed in 9.3.4 “Deploying and managing warehouse applications” on page 409.

The location of the execution database is important, as all of the work within the data flows (processes) is run as SQL within the execution database. For example, if an application is created with a source and target on separate DB2 instances and we specified an execution database on another DB2 instance, which is collocated with WebSphere. The flow of data in this example will be:

1. From the source database into the execution database
2. From the execution database into the target database

Therefore the location of the execution database is important to ensure that data is transferred as efficiently as possible.

To illustrate the different scenarios that may be encountered in defining the execution database, a fairly typical warehousing example can be used. In this example, there is a source fact table for which certain elements of data need to be copied into the target fact table. The source and target tables are identical and the filter SQL from the source table contains a simple `WHERE` clause. As the volumes are low, an insert will be performed into the target table based on the `SELECT` from the source. The only differences between each scenario are the location and type of the source and target tables.

The possible database configurations are as follows:

- ▶ Source and target are both DB2 and in the same database
- ▶ Source and target are both DB2 and in different databases
- ▶ Source is a non-DB2 database and the target is a DB2 database

Note: The configurations examine the behavior of `INSERT` statements between tables. The operation of other statements, such as `UPDATE` and `DELETE`, within configurations is not directly considered. To fully understand the flow of individual processes the EPG code for all data flows should be generated and viewed in the DWE Design Studio.

Source and target are both DB2 and in the same database

This scenario may be the result of a previous external extract, transform, and load (ETL) processing that has taken disparate sources and populated a transactional layer of a data warehouse. Once the data has been loaded into database tables it can then be very efficient to leverage the use of SQL. In this scenario, illustrated in Figure 9-13, the execution database should be the same database as the one containing the source and target tables.

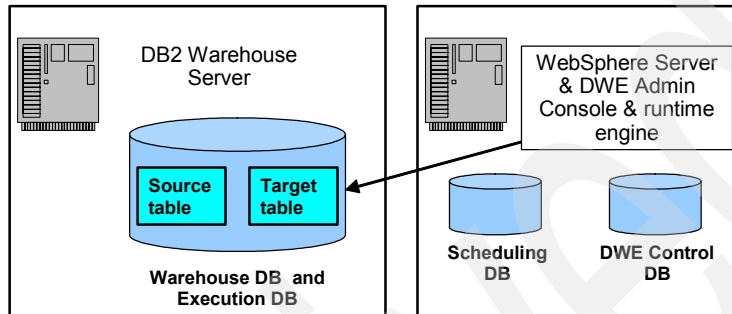


Figure 9-13 Execution database is the same as the warehouse table

This configuration ensures that the data transfer between the source and target tables is carried out within the warehouse database. This can be checked during development in the DWE Design Studio by opening your data flow and selecting **Data Flow** → **Generate Code**, which shows the deployment code.

An example of the code can be seen in Example 9-2, where we are directly selecting from the source table `CVSAMPLE.SALESFACT` and inserting into the target table `CVSAMPLE.SALESFACT2` in one SQL statement. Although the SQL statement will be issued by the runtime engine within WAS, which in this example is on a separate server, the SQL is being run on the warehouse server.

Example 9-2 Generated code from an intra-database insert from select

```
CODE_UNIT:JDBC
( )
{
    INSERT INTO CVSAMPLE.SALESFACT2
    TIMEID, STOREID, PRODUCTID, SALES, COGS, ADVERTISING)
    SELECT
    TIMEID AS TIMEID,
    STOREID AS STOREID,
    PRODUCTID AS PRODUCTID,
    SALES AS SALES,
    COGS AS COGS,
```

```

ADVERTISING AS ADVERTISING
FROM
CVSAMPLE.SALESFACT INPUT_03
WHERE (STOREID = 30)

}

```

Source and target are both DB2 and in different databases

This scenario may be where the transaction and data warehouse systems are both DB2 but are in different databases. This could be a different database in a different instance or in the same instance, as illustrated in Figure 9-14. In this situation we recommend that you have the execution database collocated with the source database.

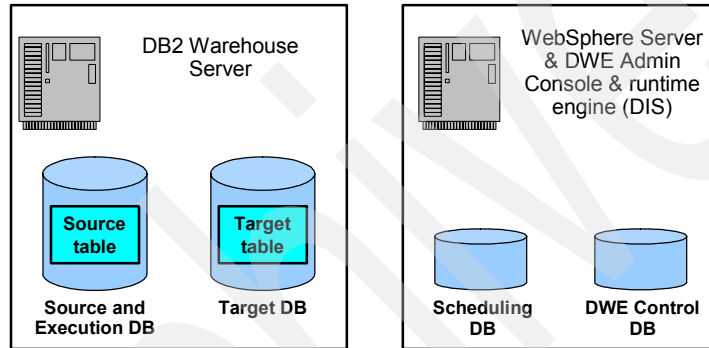


Figure 9-14 The execution database is the same as the source database

The reason for collocating the source database and the execution database is that DIS can issue the predicated select against the source table and then insert the data into the target table. The generated SQL in Example 9-3 illustrates how this configuration would work.

The EPG fragment in Example 9-3 is executed by DIS, which connects to both the source and target databases. The SQL select is issued against the source database with the data then being inserted into the remote database using a JDBCInsert class from the runtime environment. The use of the Java runtime unit class shows that the selected data is flowing to the WebSphere server where the data is then being inserted into the target database.

Example 9-3 Generated code for a DB2 insert from select across two databases

```

EPGTXN ( ) : type TXN : node /graph15
  (db connection = [SOURCE, TARGET])
  {

```

```

( ):CODE_UNIT, node /graph15/node9;
CODE_UNIT:JAVA
(Source Database Name = SOURCE

Source SQL Query = SELECT
TIMEID AS TIMEID,
STOREID AS STOREID,
PRODUCTID AS PRODUCTID,
SALES AS SALES,
COGS AS COGS,
ADVERTISING AS ADVERTISING
FROM
CVSAMPLE.SALESFACT INPUT_03
WHERE (STOREID = 30)

Target Database Name = TARGET

Target Insert Statement = INSERT INTO
CVSAMPLE.SALESFACT2(TIMEID, STOREID, PRODUCTID, SALES, COGS,
ADVERTISING) VALUES (?, ?, ?, ?, ?, ?)

NumTargetColumns = 6

)
{
Java Runtime Unit Class:
com.ibm.datatools.etl.dataflow.baselib.runtimeunits.JDBCInsert
}
}

```

Note: The JDBCInsert class inserts data 1,000 rows at a time, but the data is committed at the end of the transaction.

Source is a non-DB2 database and the target is a DB2 database

This scenario may be where there is data in a non-DB2 warehouse table that is required for a DB2 data mart. This configuration is made possible by the ability of the DWE Design Studio and the Administration Console to connect to JDBC data sources. In this scenario the execution database should be the same database as the DB2 target database. This is because the execution database has to be a DB2 database. This configuration is illustrated in Figure 9-15.

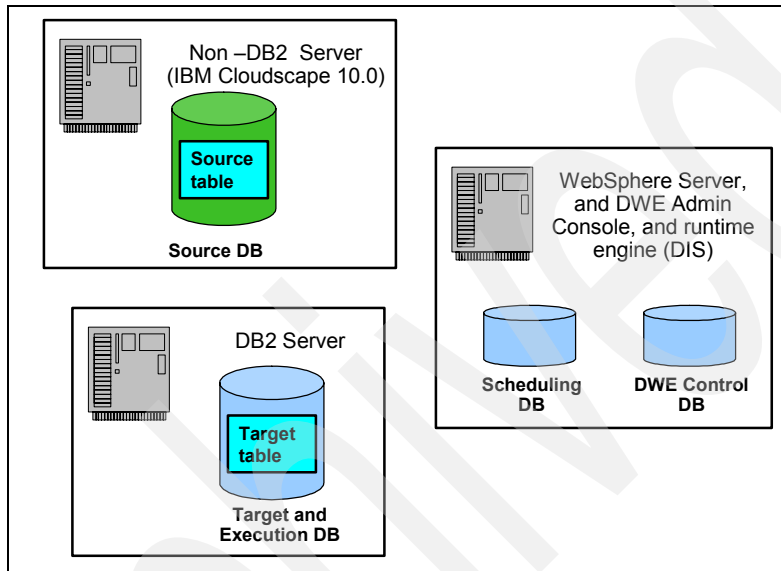


Figure 9-15 The execution database is the same as the DB2 target database

To test the scenario, a Cloudscape™ V10 data source is used. The code that will be generated in such a scenario is illustrated in Example 9-4. The target database is TARGET and initially a DB2 Declare Global Temporary Table (DGTT) will be created in this database to store the data read from the source database, derbydb. The data is then selected from the derbydb database and inserted into the DGTT on the TARGET database by DIS. The contents of the temporary table are then selected with the WHERE clause added and inserted into the TARGET database using a JDBCInsert class from the runtime environment.

Example 9-4 Generated code selecting from Cloudscape and inserting into DB2

```
EPGTXN ( ) : type TXN : node /graph15
  (db connection = [TARGET])
  {
    ( ):CODE_UNIT, node /graph15/node9;
    CODE_UNIT:JDBC
    ( )
```

```

    {
        DECLARE GLOBAL TEMPORARY TABLE SESSION.IBM_ETL_TEMP_3(
            TIMEID INTEGER,
            STOREID INTEGER,
            PRODUCTID INTEGER,
            SALES DECIMAL(7,2),
            COGS DECIMAL(7,2),
            ADVERTISING DECIMAL(7,2))
        NOT LOGGED ON COMMIT PRESERVE ROWS

    }
}
....
EPGTxn ( ) : type TXN : node /graph18
    (db connection = [derbydb, TARGET])
    {
        ( ):CODE_UNIT, node /graph18/node9;
        CODE_UNIT:JAVA
        (Source Database Name = derbydb

        Source SQL Query = SELECT TIMEID AS TIMEID,
            STOREID AS STOREID,
            PRODUCTID AS PRODUCTID,
            SALES AS SALES,
            COGS AS COGS,
            ADVERTISING AS ADVERTISING
        FROM CVSAMPLE.SALESFACT2 SALESFACT2

        Target Database Name = TARGET

        Target Insert Statement = INSERT INTO
        SESSION.IBM_ETL_TEMP_3(TIMEID, STOREID, PRODUCTID, SALES, COGS,
        ADVERTISING) VALUES (?, ?, ?, ?, ?, ?)

        NumTargetColumns = 6

    )
    {
        Java Runtime Unit Class:
        com.ibm.datatools.etl.dataflow.baselib.runtimeunits.JDBCInsert
    }
}

```

```

.....
EPGTXN ( ) : type TXN : node /graph21
  (db connection = [TARGET])
  {
    ( ):CODE_UNIT, node /graph21/node9;
    CODE_UNIT:JAVA
    (Source Database Name = TARGET

    Source SQL Query = SELECT
    TIMEID AS TIMEID,
    STOREID AS STOREID,
    PRODUCTID AS PRODUCTID,
    SALES AS SALES,
    COGS AS COGS,
    ADVERTISING AS ADVERTISING
    FROM
    SESSION.IBM_ETL_TEMP_3 INPUT_03
    WHERE (STOREID = 30)

    Target Database Name = TARGET

    Target Insert Statement = INSERT INTO
    CVSAMPLE.SALESFACT2(TIMEID, STOREID, PRODUCTID, SALES, COGS,
    ADVERTISING) VALUES (?, ?, ?, ?, ?, ?)

    NumTargetColumns = 6

  )
  {
    Java Runtime Unit Class:
    com.ibm.datatools.etl.dataflow.baselib.runtimeunits.JDBCInsert
  }
}
....

```

Deployment considerations

Now that scenarios have been shown in which the location of the runtime environment and the execution database affect how an application will run, the considerations for the runtime architecture can be discussed.

In this section we discuss:

- ▶ General recommendations
- ▶ Database configurations
- ▶ Deployment considerations.

General recommendations

We generally recommend that the server where WebSphere is installed also has a full installation of DB2 UDB. Even if the runtime architecture is to have a separate warehouse server and WebSphere server, this is recommended. In a full production environment, using only one DB2 installation for both the data warehouse server and application server might significantly impact performance. This is because the runtime environment for SQL Warehousing needs to access a database for the process metadata and scheduling. If these runtime databases are on a remote server, each time some runtime metadata is being updated then it will need to be done remotely. Another consideration is that the DB2 Warehouse server may have different availability and backup criteria than a write-intensive online database such as the control database for SQL Warehousing.

Database configuration

When a warehousing application is deployed to a runtime environment, then the configuration of the databases being used needs to be considered. As with all aspects of the deployment, the EPG code in the Design Studio should be examined to understand how each database is being used. For instance, are declared global temporary tables (DGTTs) being used? Or are large volumes of data being inserted?

Specific considerations include:

- ▶ Use of declared global temporary tables (DGTTs)

If the EPG code is using a DGTT to stage any data, then the database hosting the DGTT has to have a user temporary tablespace. A tablespace of this type does not exist by default and therefore must be created before the process can be run. The runtime tablespace used for the DGTT needs to be sized appropriately so that in Example 9-4 on page 399, `SESSION.IBM_ETL_TEMP_3` can hold the contents of the `CVSAMPLE.SALESFACT2` table from the `derbydb` database. The DGTT could be created independently in the runtime environment or the code or its creation could be included with the application deployment package.

The performance of DGTTs can also be improved by adding indexing and performing runstats. With the example in Example 9-4 on page 399, the selecting of the data would be improved by an index on the `STOREID` column since this is the column that the `WHERE` clause is using to restrict the data. If

an index is created, then a *runstats* statement would have to then be executed so that the DB2 Optimizer can use the index.

► Database logging

The size of the DB2 logs can be crucial when the warehousing application is performing a large insert, update, or delete. The LOGFILSIZ, LOGPRIMARY, and LOGSECOND database configuration parameters should be configured to allow the largest SQL operation to successfully complete. The amount of log space required can be worked out, as illustrated in Table 9-1, where the number of rows being worked with is 500 K and the size of each row is 254 bytes. If the log space is insufficient to complete the operation then the transaction will roll back and this error will be reported in the process logs within the Administration Console.

Table 9-1 Working out the log space required for insert/update or delete operations

| Description | Values |
|------------------------------|--|
| Number of rows | 500000 |
| Row length (bytes) | 254 |
| Log overhead (bytes) | 18 |
| Include the undo log | $((254 * 2) + 18) = 526$ |
| Total log size required (MB) | $((526 * 500,000) / 1024) / 1024 = 242 \text{ MB}$ |

Note: In a partitioned (DPF) environment, the log space on each partition needs to be considered. In Table 9-1 if there were four partitions, each with 125 K rows, then there would need to be $(242/4)$ 60.5 MB of log space in each data partition.

Deployment considerations

If a runtime environment has applications that fit certain profiles then it can be worthwhile considering the collocation of the WebSphere server and the DB2 Data Warehouse server. In this section we discuss two common application profiles, based on the scenarios discussed in “Location of the SQL Warehousing runtime environment” on page 392 and “The location of the SQL Warehousing execution database” on page 394.

► Applications with multiple command operators

If the deployed warehouse applications execute a large volume of commands such as FTP, SQL scripts, or OS scripts then it is worth considering collocating the WAS and DB2 Data Warehouse server. In Figure 9-11 on page 393 and Figure 9-12 on page 394 we illustrated how collocating the two

servers can ensure that data is moved as efficiently as possible. The key consideration is that the commands are executed by the runtime component of DWE, which is on the WebSphere server. This is only a consideration rather than a recommendation, because good practice generally is that a runtime WAS environment and a runtime DB2 Data Warehouse should be on separate servers to mitigate any resource contention. If, however, the quantity of the commands being deployed is large, or if large volumes of data are being moved, then it may be worth collocating the two servers.

► Applications with multiple data flows

If the deployed warehouse applications primarily contain SQL data flows then the runtime performance can be impacted by the location of the execution database and the location of the WAS server.

- If the source and target database are the same then as long as the execution database is also in that database it is not necessary to have WAS collocated. As was shown in Example 9-2 on page 396, the runtime environment will execute the SQL statements on the data warehouse server. Only the commands will be flowing between WAS and DB2, and data will not be moving between the two environments.
- If the source and target database are in different databases, then the WAS server will be used to move the data between the two databases. In Example 9-3 on page 397 and Example 9-4 on page 399, the classes within DIS were used to move data between the two databases. If the source and target tables are in the same database then the execution database should be defined as being the same database. If the source and target database are separate and the process is doing a predicated select, such as a *SELECT from table where value = x*, then the execution database should be collocated with the source database. This collocation allows the pushing down of the predicate, as shown in Example 9-3 on page 397, rather than using a temporary table, as shown by Example 9-4 on page 399.

The collocation of WAS with the DB2 data warehouse server could also be considered if there are large volumes of data flowing through DIS. If the source and target databases are on physically separate servers then the performance would be optimized by having WAS on the target machine to perform any inserts locally.

This is only a consideration rather than a recommendation because good practice generally is that a runtime WAS environment and a runtime DB2 data warehouse should be on separate servers to mitigate any resource contention. If the quantity of data being moved is sufficiently large then it may be worth collocating the two servers.

The best way to decide what runtime topology may best suit the data flows is to view the *generated code* for a data flow within the DWE Design Studio. If the EPG code is only making one database connection, as in Example 9-2 on page 396, then the collocating of the DB2 and WebSphere is probably not required. If the generated EPG code contains multiple database connections and use of *Java runtime unit classes* such as `com.ibm.datatools.etl.dataflow.baselib.runtimeunits.JDBCInsert`, then data will be flowing through the WebSphere server.

While collocation may not always be desirable or possible it is important to understand that the connections to the multiple databases are made by the SQL Warehousing runtime environment. It follows that as the connections are made by the runtime environment that the insert classes are being executed on the WebSphere server.

9.3.3 Managing warehouse resources

In this section we discuss the creation and management of data warehouse resources for use by the runtime warehousing applications. These resources can be defined within this SQL Warehousing section or can be defined in the DWE Common section, as described in 9.1.5 “General administration tasks” on page 378. These resources can either be databases or system resources such as an FTP server or a DataStage server that is being used to run DataStage parallel jobs. Before a data warehouse application can be deployed, the appropriate resources required must either be created or defined for use by the application.

Although the data flows and control flows contain initial references to resources, the actual resources to be used during execution might not be known at design time. During the deployment an administrator must map these initial references to the actual references. This resource management task is not part of deployment so you must define the resources first, then select them during the deployment process.

Warehouse resources only require being created if your applications contain control flows that access these resources. If the control flows do not have FTP commands or access DataStage jobs then there will not be a need to define system resources. If SQL scripts are to be deployed to perform tasks such as LOAD and EXPORT, the database has to be managed by SQW and the user ID and password (stored in a DWE-specific encrypted format) provided in the data source definition. If the database is local and no user ID and password is specified then implicit connect will be used.

Create data sources

The creation of a data source for SQL Warehousing is done within the DWE Administration Console at **DWE SQL Warehousing → Resources → Create Data Resources**. The data sources that are required by the warehouse applications can be defined using an existing WAS data source. WAS Data sources for SQL Warehousing need to have been defined either in the WebSphere Administration Console or the *Database Profile* section of the DWE Administration Console. If a WAS data source is not being used then the required information for the standalone connection must be provided.

Figure 9-16 shows a new warehouse data source being created that is not an existing WebSphere data source. Once the data source is specified as not being a WAS data source, the further connection information is required to be specified. In this example an existing WAS data source is not being used, so the user ID and password must be provided.

The figure displays two screenshots of the DWE Administration Console interface. The left screenshot shows the 'Attach to WAS Data Source' dropdown set to 'Yes', with fields for 'JNDI Name' (jdbc/dwe/sample), 'User ID' (db2admin), and 'Password' (masked). The right screenshot shows the dropdown set to 'No', with the same 'JNDI Name' and 'User ID', but the 'Password' field is also masked. Below the 'Attach to WAS Data Source' dropdown, there are fields for 'Display Name' (SAMPLE), 'Description' (sample database), 'Database Type' (DB2), and 'Database Name' (Generic). A 'Test Connection' button is visible at the bottom of the right panel.

Figure 9-16 Creating a new data source for warehousing applications

The database type can either be DB2 or generic. The DB2 database type needs to have its library paths in the system path so that it can be accessed by WebSphere. The generic database type can be any data source type supported by and managed by WebSphere. The library paths for the generic database type must also be defined in the system path. When specifying the database name in Figure 9-16, it has to match the name of the data source.

If an existing WAS data source is selected, the default settings, including J2C authentication, will be used to connect to the data source through WAS. We recommended that any remote DB2 databases be cataloged in the local DB2 server or administration client rather than using the pure Java type 4 JDBC connectivity. This is to ensure that the Administration Console can run processes

that require DB2 command-line access to those source and target databases. The DB2 command line is used frequently to run SQL scripts that contain LOAD and EXPORT statements.

Note: The development and runtime JNDI names can be different since the database mapping is specified as part of the application deployment.

Manage data sources

The DWE Administration console provides four options for data sources that have been defined in the create data source section. These options can be accessed by selecting **DWE SQL Warehousing** → **Resources** → **Manage Data Resources**.

- ▶ The test connection option returns a message that confirms whether the application server can make a live network connection to the specified database.
- ▶ The check dependencies option returns a list of applications and processes that use the specified data source.
- ▶ The update option displays a page where you can edit data source properties and apply changes. You can also test a connection to the updated data source.
- ▶ The remove option allows you to remove a previously created system resource.

Create system resources

This section of the DWE Administration Console is accessed by selecting **DWE SQL Warehousing** → **Resources** → **Create System Resources** and defines system resources that the warehousing application requires. A system resource can define a computer that is used in an FTP operation or a DataStage server that runs parallel jobs. Figure 9-17 depicts an example of setting up a system resource and also illustrates the two types of resources that can be defined.

The screenshot shows a 'Create System Resource' dialog box with the following fields and values:

- Display Name**: * FileSource
- JNDI Name**: * FileSource
- Resource Type**: * System (dropdown menu is open, showing 'System' and 'DataStage server')
- OS Type**: * Windows (dropdown menu is open, showing 'Windows')
- Host Name**: * pi-ewxp-01.ibm-pi-e.com
- User ID**: * db2admin
- Password**: *
- Description**: Location of source flat files

Buttons at the bottom: Test Connection, Finish, Cancel.

Figure 9-17 Creating a system resource

The *resource type* in Figure 9-17 is a *system* resource, which translates to being an FTP server.

Manage system resources

The DWE Administration console provides four options for system resources that have been defined in the create system resource section. These selections are:

- ▶ *Test connection* returns a message that confirms whether the application server can make a live network connection to the FTP server or DataStage server. This test is performed by a Telnet connection being attempted to the specified system resource.
- ▶ *Check dependencies* returns a list of applications and processes that use the specified resource.
- ▶ *Update* displays a page where you can edit system resource properties and apply changes. A connection test can also be made to the updated system resource.

- *Remove* allows a previously created system source to be removed.

9.3.4 Deploying and managing warehouse applications

Now that the required data and system resources have been created the warehouse applications developed in the DWE Design Studio can be deployed to the runtime environment. The preparation of the warehouse application takes place in the DWE Design Studio, but the actual deployment takes place in the Administration Console. The deployment of the new application makes the application processes available for scheduling, execution, and administration. The resources that are referenced in the control flows and data flows must be ready to accommodate the new application. The process for creating these resources has been discussed in 9.3.3 “Managing warehouse resources” on page 405. Each application to be deployed will contain one or more control flows, which will in turn contain data flows or other activities such as SQL scripts, executables, FTP commands, or OS scripts. When the application is deployed each control flow becomes a runtime process.

In this section we discuss the:

- Contents of the warehouse deployment file
- Deployment process
- Structures created by the deployment
- Management of the applications after deployment

Contents of the warehouse deployment file

Before the deployment of the warehouse application is discussed, it is worthwhile to understand the contents of the deployment package. The deployment package is a zip file that is generated by the DWE Design Studio for deployment. This zip file contains data flows and control flows that were created during the design phase. These data flows are compiled and validated based on an application profile, and the appropriate deployment package containing the code is generated as a result. An application profile is a combination of control flows, database definitions, variables, and script files.

For example, you can package as many control flows into a data warehouse application as desired. Multiple applications can contain the same flows. Flows contained within the same application share the same application home/log/work directory after the data warehouse archive has been deployed in DWE Admin. They can also be enabled and disabled together, but are otherwise independent of each other.

Figure 9-18 illustrates the contents of a deployment package and its directory structure.

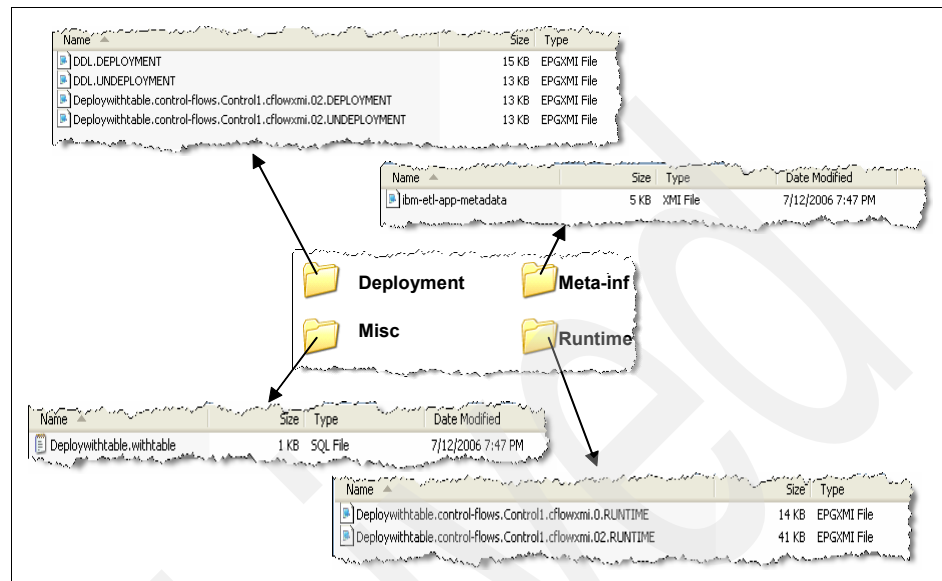


Figure 9-18 Contents of a deployment file

The deployment package file, illustrated in Figure 9-18, contains:

- ▶ **Deployment files:** The deployment folder contains two control flow EPG (.epgxmi) files for each control flow — one for the deployment of the application and one for the undeployment of the application. The deployment EPGs are run once (and only once) when the data warehouse application is deployed to the SQL Warehousing application-server-based runtime. The undeployment EPGs contain code that is run when the data warehouse application is uninstalled from the SQL Warehousing runtime. It is the opposite of the deployment EPG and is again only running once.
Two DDL(epgxmi) files are also created for each application, which contains any DDL files that are to be deployed as part of the application.
- ▶ **Application metadata (meta-inf):** The meta-inf folder contains an ibm-etl-app-metadata.xml file and is generated for each application that contains information, such as the location of log files, what variables are being used, and which databases are being accessed.
- ▶ **Applicable miscellaneous files (Misc):** The misc folder contains deployment files that are database setup files that are bundled with the data warehouse application. They are to prepare the database for use before the control flows are executed. Commands in these files only need to be run once rather than

each time a process is run. Examples of the types of commands in these files would be DDL statements to create tables or a database configuration update.

- ▶ The runtime execution plan graphs (EPGs): The runtime folder contains one runtime EPG (.epgxml) file for each operator in the control flow, whether they be a data flow, e-mail, or a DB2 command. This file is the XML representation of the graphical element within the DWE Design studio and is run each time the process is invoked.

Deployment process

The deployment process extracts code units from the zip file generated by the DWE Design Studio and creates a deployment file structure. The process also populates control tables in the SQL Data Warehousing control database.

The four steps to deploying the application are:

1. General: The first task is to locate and select the deployment .zip file. This file can either be local to the client system or local to the application server. Once selected the .zip file is read and elements of the application metadata are presented and can be changed if required. These elements include the application name, the application home directory, and the application log and temp directories. Care should be taken when deciding where to place the application files, since they are used at runtime to execute the operators within them.
2. Data sources: In this step you can map the data sources to be used for the target system where the installed application is going to run. This step requires that the data resources have already been defined. The resource names do not have to be the same, as long as the mapping is done to the correct JNDI name on the application server. If the database name was TEST on development, this process allows you to map this database to the appropriate JNDI name in the runtime environment.
3. System resources: In this step you can map the system resources used in the development environment to the system resources where the installed application is going to run. If an FTP server has been defined in the DWE Design Studio then this would be mapped to a production FTP server. The system resource being mapped to must have already been defined, and the system resource information will only be displayed if the control flow contains DataStage elements or any commands with a type of FTP.
4. Variables: A variable is a user-defined name that represents data that can be changed in a data or control flow. The flexibility of variables allows the definition of certain properties to be deferred until a later time. Variables can be defined within the DWE Design Studio to require a value at varying phases of the deployment cycle.

These phases are:

- Design time: The variable is populated during the design phase and does not require input during the production deployment.
- Deployment prep: The variable is populated during the preparation of the deployment package and does not require input during the production deployment.
- Deployment: The variable needs to be populated during the deployment of the application. An example of this would be when a data flow is deployed to a test system and later to a production system where the production system is very similar to the test system except for the table schema names. A designer could then choose to use deployment phase variables for the table schema names so that the tested SQW application can be deployed safely into the production system.
- Runtime: The variable can be populated during deployment, but this is not required. Instead the variable can be populated by updating the process after the application is deployed. A runtime level variable could be used where there are multiple control flows that all use the same variable, but setting this at a general application level may not be desired.
- Execution instance: The variable can be populated during deployment, but this is not required. Instead, the variables must be populated when the individual process instance is run. The setting of a variable at the execution instance means that a new value can be, but does not necessarily have to be, specified each time the process runs. The setting of an execution variable requires a process profile to be defined, which is described in “Manage process profiles” on page 421. An example of the use of an execution instance is if a user needed to have the same data flow run against different databases. The designer can use an execution phase variable for the database so that each process instance can choose a database. This removes the need to design or deploy identical data flows with different database names.

Deployment file structure

Once the application is installed, the relevant application, process, activity, and variable parameters are written to the SQL warehouse control tables. Also, the deployment and execution code units, which will be used at process execution time, are written to the application home directory.

Figure 9-19 illustrates what the application deployment directory will look like. The deployment file mirrors the folder structures within the packaged application.

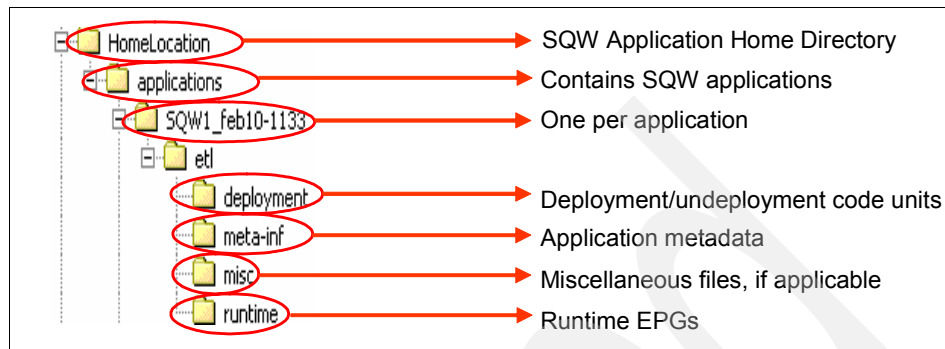


Figure 9-19 Deployment file structure for a warehousing application

During application deployment the log directory and temp directory are also created. The log directory is the default location for the log files generated when a runtime process is executed. The temp directory stores files such as a record of the deployment. The SQW application deployment also writes log and error messages in a file with *application name*.html into the log directory specified at application deployment time.

Managing the applications after deployment

The DWE Administration Console can be used to manage the data warehouse applications that have been deployed. During the life cycle of an application it may be necessary to update application properties and eventually uninstall the application. When navigating to **DWE SQL Warehousing → Data Warehouse Applications → Manage Data Warehouse Applications** there are four options for the selected application:

- ▶ **Update**: This option opens a properties page for the application, where attributes can be changed, such as the log directory for the application or the current value of a variable.
- ▶ **Enable**: This option enables an application that is currently disabled. An application has to be enabled as a prerequisite for running the processes within the application. When an application is deployed the default is that they are enabled, but you can choose that they not be enabled.
- ▶ **Disable**: This option disables an application, making its processes unavailable for execution.
- ▶ **Uninstall**: This option performs the opposite set of tasks of the deployment task. After an application has been uninstalled, you can no longer use it from the console. The deployment history for an application contains a reference to

the fact that the application was undeployed, but all of the metadata about the application is physically removed from the WAS environment.

An application can also be directly selected by clicking it, which allows the properties of the application to be viewed and updated, as illustrated in Figure 9-20.

Application General Information

Processes packaged in the application*

Data sources referenced*

System resources referenced*

Variables referenced

General Processes Data Sources System Resources Variables

Application Name DWEADMIN_lab_DWEAdmLab02 Home Directory C:/HomeLocation

Log Directory C:/Logs Working Directory C:/temp

Mail Provider mail/dwe Retain Instance Statistics No

Updated Tue, Feb 28 2006 23:43:53 PST Last Updated By

Description Simple control and data flow to demo DWA administration Comment Deployed for DWE 9.1 T3

Apply Cancel

* Read-only

Figure 9-20 Properties of a deployed application

The following are brief descriptions of the tabs in Figure 9-20:

- **General:** allows the log directory and working directory to be updated. You can also configure the mail provider for e-mail activity and choose whether to retain the statistics for each run of the process. The properties and comments can also be updated and the tab also displays information about when and by whom the application was last updated.
- **Processes:** shows whether the application is enabled and when it is next scheduled to be run. You can also select the individual process and bring up the properties for the process. This process dialog information is discussed in 9.3.5 “Warehouse processes” on page 415.
- **Data Sources:** allows you to view the data sources used by the application. This information is read-only, as any updates to the data sources need to be done via the **DWE SQL Warehousing → Resources → Manage Data Sources** section of the Administration Console.
- **System Resources:** allows viewing of the system resources used by the application. This information is read-only, as updating the system resources is

done via the **DWE SQL Warehousing** → **Resources** → **Manage System Resources** section of the Administration Console.

- ▶ Variables: allows the viewing and inserting of new values for the variables that are used by the application. Each variable can be selected, which then brings up a page with more information about the variable value and definition.

Note: This is the only location where a user can modify variables in the runtime and execution phases.

9.3.5 Warehouse processes

When an application is deployed to a runtime environment it may contain one or more processes. A process is a control flow from the DWE Design studio that performs the warehousing activities such as populating tables, invoking DataStage jobs, or sending out confirmation e-mails. The management of these tasks includes updating process properties such as variable values, and verifying that data source connectivity exists for specific processes. A process can also be scheduled within the Administration Console, which in turn uses the scheduling capabilities of WebSphere.

The DWE SQL Warehousing → Processes section of the Administration Console provides the functionality to:

- ▶ Manage processes.
- ▶ Manage process profiles.
- ▶ Run processes.
- ▶ Manage process schedules.
- ▶ Monitor process instances.

In this section we discuss these parts of the Administration Console by introducing their functionality and in what circumstances they will be used.

Manage processes

When an application has been deployed the processes are displayed within the Administration Console. If there are multiple processes in an application then each process will be displayed with the corresponding application. The processes can be viewed, and can potentially be changed by navigating to **DWE SQL Warehousing** → **Data Warehouse Applications** → **Processes** → **Manage Processes**.

This is depicted in Figure 9-21 on page 416, which shows where an administrator would enable or disable a process. A process must be enabled to allow it to run or be scheduled. When an application is deployed the default is that the underlying processes are enabled, although they can be disabled by an

administrator if required (for example, if there is on-going maintenance for one of the data sources used by the process). The validate option checks that any data sources and system resources within the process can be connected successfully. If any of the resources fail the test connection, an SQW02204I error is returned, indicating that at least one or more resources has failed the test. Be aware that the process is not automatically disabled if the test fails.

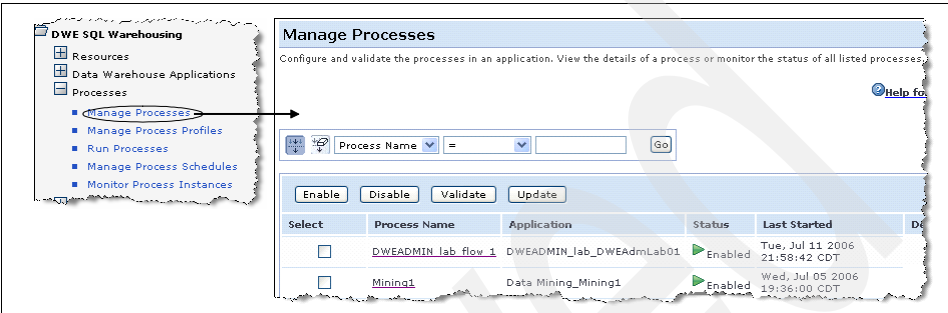


Figure 9-21 Managing a warehouse process

If a process is selected or chosen to be updated, then the Properties window for the process depicted in Figure 9-22 is given.

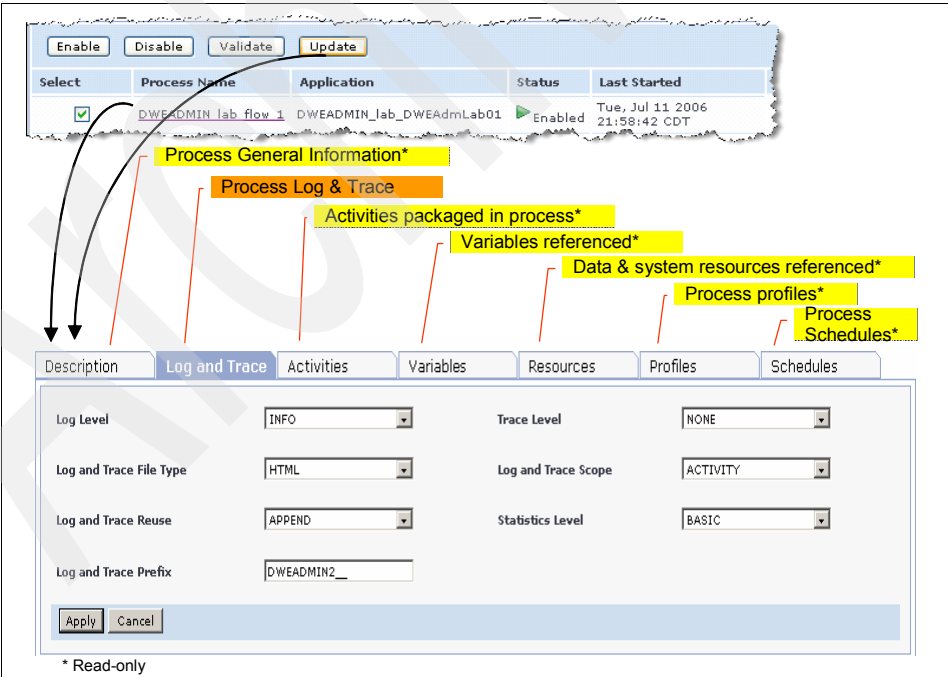


Figure 9-22 Properties of a process

The properties of a process are divided into multiple tabs, as follows:

- **Description:** This contains read-only information about the process that was selected. The information includes when the process was last run and when it was created. From this tab the statistics for the last runtime instance of the process can be retrieved, which includes information such as whether the process instance was successful and how long it took to run. Instance statistics are discussed in more detail in 9.3.6 “Warehousing diagnostics” on page 431.
- **Log and Trace:** This is where an administrator can update the diagnostic settings for a process. These settings would typically be changed if additional information was required for a process due to runtime performance issues or if the process has been newly deployed into the runtime environment. Table 9-2 contains the list of variables and their various options.

Table 9-2 Options for logging and tracing processes

| Variable | Values | Comments |
|-------------------------|--|---|
| Log Level | INFO WARNING ERROR | The default is INFO, which includes all INFO, WARNING, and ERROR messages. The WARNING level contains all WARNING and ERROR messages. The ERROR level contains ERROR messages only. |
| Log and Trace File Type | NONE HTML XML | The default type is HTML. Be aware that selecting NONE could also suppress error messages. |
| Log & Trace Reuse | OVERWRITE APPEND | The default is APPEND and is recommended to avoid the possibility of concurrency issues. |
| Log and Trace Prefix | Use a maximum of 20 alphanumeric characters. | The log location is set at an application level so if two applications use the same log directory and they have a process with the same name then any log output will be combined. To avoid this situation the log file name should be made unique by providing a prefix. |
| Trace Level | NONE METHOD CONTENT BOTH | The default is NONE. Tracing the METHOD means that only the code unit calls will be traced for each activity in the process. Tracing the CONTENT means that only the SQL statements will be traced including the calls to metadata procedures and tables. BOTH meant that both METHOD and CONTENT will be traced. |

| Variable | Values | Comments |
|---------------------|---------------------|---|
| Log and Trace Scope | ACTIVITY PROCESS | An activity is contained within a processes. A process is the runtime equivalent of a control flow, while an activity is either the runtime equivalent of a data flow or some other unit of work that is defined in a control flow and runs independently, such as a shell script. The default trace scope is set to ACTIVITY. |
| Statistics Level | NONE BASIC | BASIC collects the row counts for SQL statements that are executed via JDBC. The default is BASIC. |

In the scenario where an application has been recently deployed, or is performing poorly, the most detailed level of logging would be required, which would be to set:

- Log level = INFO
- Trace level = BOTH
- Log and trace scope = PROCESS
- Statistics level = BASIC

The rest of the settings can be kept at the default level.

Tip: If this process is to be run several times in a system test or pre-production environment then we recommend that you change the log file prefix to easily identify each run of the process.

- **Activities:** This shows the individual activities that comprise the process. An individual activity could be a data flow, a DB2 SQL script, or an e-mail. Each separate activity can be selected, and a properties dialog is brought up for the activity, as illustrated in Figure 9-23 on page 419.

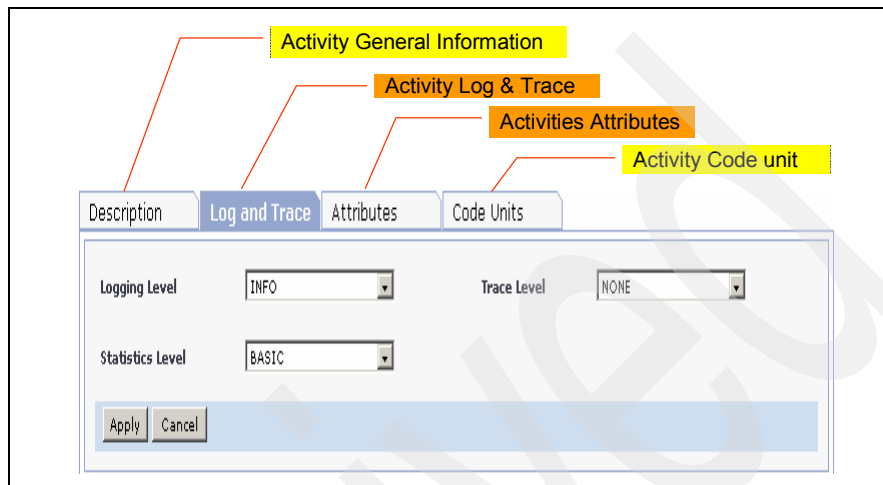


Figure 9-23 Attributes of an activity within a process

The information available on each activity is presented in the following tabs:

- *Description* contains read-only information about the activity that was selected. The information includes the activity name and the type of activity. If the activity is a data flow then this dialog includes the name and location of the underlying XML file.
- *Logs and Trace* contains the diagnostic operations for the individual activity. This is a subset of the options available for the full process and is comprised of the logging, trace, and statistics levels.
- *Attributes* contains the variables and constants that are used for the activity. Any attribute that has a type of runtime or execution_instance can be changed here, but any deployment level attributes can only be viewed.
- *Code Units* correlates the code unit in the EPG to which this activity relates. This mapping is useful for monitoring and debugging runtime instances.

- *Variables* allows the variables for a process to be viewed. If the change phase for the variable is defined as *runtime* then the contents of the variable can be altered at this time. If the variable is selected then a window is launched that contains detailed information about each variable. Figure 9-24 on page 420 shows the process of viewing the current contents of a variable, and since the *change phase* is runtime the value of CVSAMPLE can be updated.

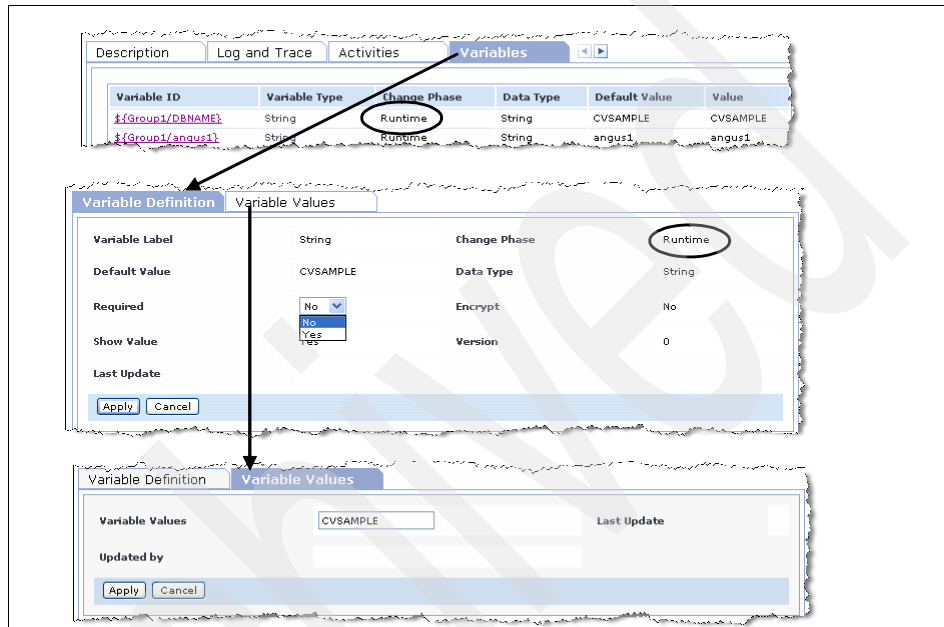


Figure 9-24 Viewing the contents of a process variable

This variable window shown in Figure 9-24 has the following tabs:

- *Variable Definition* contains details of the variable, including the data type and when it was last updated.
- *Variable Values* allows the current value of the variable to be updated after deployment if the change phase has been defined as *runtime* or *execution instance*.
- *Resources* allows the resources that are used in the process to be viewed. These resources may be data sources or system resources that have been defined for the runtime environment. A JNDI name can be selected, the properties can be viewed, and the connection to each resource can be tested.
- *Profiles* contains the profiles defined for the process. A process profile is a set of variable values for one or more process instances to use at runtime. By selecting each profile, a Properties window is displayed. This is discussed in

“Manage process profiles” on page 421. If no process profile has been created then this tab will contain no information.

- *Schedules* contains the schedules that have been defined for the process. By selecting each schedule, a properties window is displayed. This is further discussed in “Manage process schedules” on page 424. If no schedule has been created, this tab will contain no information.

Manage process profiles

When a process is deployed into a runtime environment there may be a requirement for a process profile. A process profile allows a set of variables to be defined for one or more process instances to use at runtime. In this way instances of the same process can be started with different profiles or multiple instances can be started with the same profile depending on the runtime requirements. The main purpose for defining process profiles is to facilitate scheduling. If the same process needs to be run twice under slightly different conditions, then two instances can be scheduled to use two different profiles with different values.

If the deployed application does not use instance variables then it is not necessary to define a process profile. However, if the process does contain instance variables then a process profile must exist to allow the process to be scheduled.

The section within the Administration Console for the management of a process profile is shown in Figure 9-25 and is located by selecting **DWE SQL Warehousing** → **Data Warehouse Applications** → **Processes** → **Manage Process Profiles**.

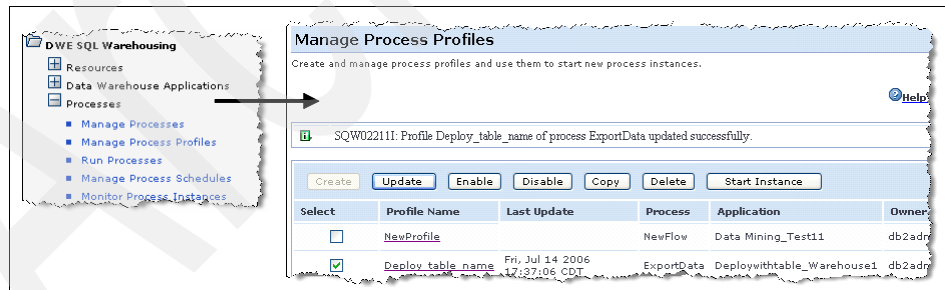


Figure 9-25 Managing a process profile

The administrator can manage the use of profiles with the following options:

- *Create* is used to create a new process profile. A new process profile would be need to be created after an application has been deployed and before it is scheduled if it contains instance variables. One process profile is associated

with one warehousing process. When the profile is created new values can be specified for the instance variables, or the defaults from deployment can be kept.

- ▶ *Update* is used to update an existing profile. This would typically be used when a variable value in the profile will change over time. The profile can also be switched between *active* or *inactive*, which is the same as *enabling* or *disabling* a process. A profile has to be active to be used when scheduling the execution of a process.
- ▶ *Enable* sets a profile to be enabled, which means that it is available for use when scheduling a process.
- ▶ *Disable* sets a profile to be disabled, which means that it is no longer available for use with a process.
- ▶ *Copy* duplicates the contents of an existing profile into a new profile. This option could be used to create multiple similar profiles for use by a process, for example, on separate days. Or the schedule that runs during the week may need to have different variable values on the weekend.
- ▶ *Delete* removes a profile so new schedules cannot use the profile.
- ▶ *Start Instance* launches an instance of the process with the associated profile. This option is typically used to test a profile before committing it to a schedule.

Tip: When process profiles are used with schedules, the profile values that are used by the scheduler are the values that were provided when the process schedule was created. If a profile is updated, the changes only take effect for new schedules that are created after the profile is updated. If there is a requirement to make new profile information available to an existing schedule, then the schedule itself must be updated.

Run processes

When an application is deployed it consists of processes that can be scheduled or started from the Administration Console, as shown in Figure 9-26 on page 423. A process is the runtime equivalent of a control flow and contains activities that are the runtime equivalents of data flows or operators such as FTP or SQL scripts. The manual execution of a process in a runtime environment can be useful in a pre-production system or when running *ad hoc* schedules during the day. When a process is run a unique process instance is created that can then be monitored with the Administration Console.

The section within the Administration Console to run processes is depicted in Figure 9-26 on page 423 and is located by selecting **DWE SQL Warehousing** → **Data Warehouse Applications** → **Processes** → **Run processes**.

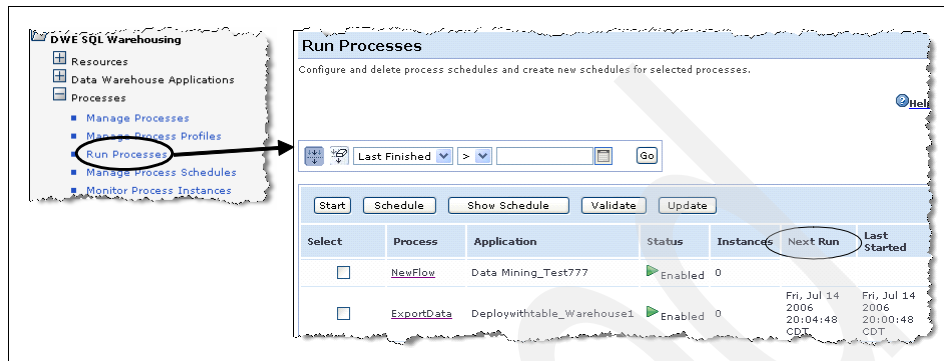


Figure 9-26 Running a process

This section enables the administrator to work with the runtime processes, with the following capabilities:

- ▶ *Start* is used to start an instance of the process. The name of the process instance can be specified, but the system default may also work as it concatenates the process name and an automated numerical value. The process can be started with directly specified variables or by using either a pre-existing or new process profile. Once the instance has been started the Administration Console displays the Monitor Process Instance screen, which is discussed further in 9.3.6 “Warehousing diagnostics” on page 431.
- ▶ *Schedule* is used to create a schedule for a particular process. Scheduling a process involves specifying when the process will run, how often, and what variable values should be used at runtime. In “Manage process schedules” on page 424, we discuss the details of how to create a schedule.
- ▶ *Show Schedule* shows a list of schedules for the selected process. Any schedules associated with the process are displayed along with the number of runs remaining.

Note: Processes with schedules can be easily identified, as they have entries in the Next Run column

- ▶ *Validate* allows an administrator to check that all the resources (systems and data) can be connected to successfully.
- ▶ *Update* is used to update parameters for the selected process. This functionality has already been discussed in “Manage processes” on page 415.

Manage process schedules

Within the DWE Administration Console schedules for the running of processes can be created or altered. A process can be scheduled to run once or multiple times. Each scheduled process has no dependency on the successful completion of another schedule. This is important when designing a process, as any conditional processing should be within a process.

For example, an application could have two processes — one process to populate a table and the other process to export the contents of the table to a flat file to be used in a third-party tool. If the two processes were actually two activities within a single process then the conditional processing that can be created in DWE Design Studio can be used to ensure that the exporting only happens after the table is populated successfully. If, however, the two processes were used, then the export could be scheduled to take place after the table was populating, but this would not be conditional on the insert being successful. In most cases this would be satisfactory, but if for any reason the insert process failed then the export process would still be executed.

Although the DWE SQL Warehousing schedules are accessed through the DWE Administration Console, the actual scheduling of the process takes advantage of WAS scheduling. The scheduler database and schema used by WAS to store scheduling information is created by running the config tool after DWE has been installed. Once a schedule has been created through the Admin Console the details are entered into the scheduling database and the schedule is managed by WAS.

The section in the Administration Console used to manage schedules is shown by Figure 9-27, and is located at **DWE SQL Warehousing → Data Warehouse Applications → Processes → Manage Process Schedules**.

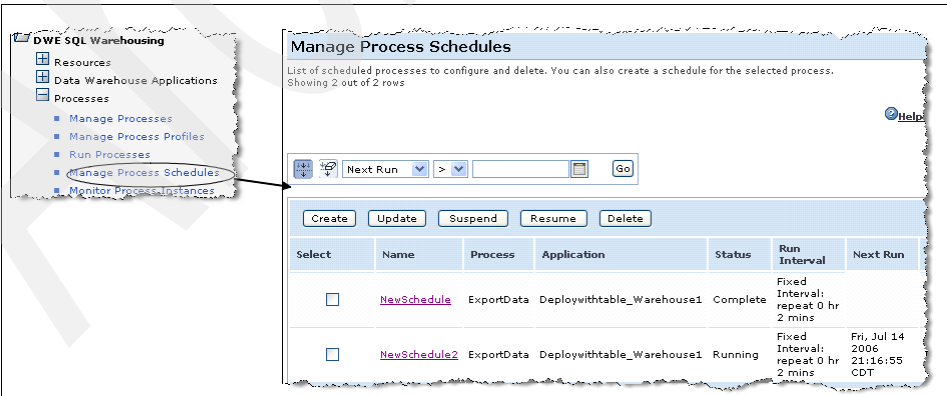


Figure 9-27 Scheduling a process

This section of the Administration Console allows an administrator to create and manage schedules using the following options:

- *Schedule* is used to create a new schedule for a process. When a scheduled process is run, the instance ID is the process name concatenated with the runtime. For example, a schedule called NewSchedule would have an instance ID of NewSchedule2-0607151309 if it was run at 13:09 on the 15th of July 2006.

A schedule can be run repeatedly or only once. If the schedule is to be run more than once then this can be configured in three ways:

- **Daily:** A daily schedule will execute every day at the same time that was specified for the first run of the schedule. If the original run was scheduled on 2006/07/15 at 14:16:00 then the subsequent run will be scheduled to start at 2006/07/16 at 14:16:00.

The schedule can be created to run indefinitely or for a specific number of days.

- **Weekly:** A weekly schedule will execute on the days specified at the same time that was specified for the first run of the schedule. If the first run of a schedule is on a Monday then the schedule will be set to next execute on the following Monday. The schedule can be set to run on more than one day, so if the original run was on a Monday then the schedule can be set to run on Monday through Friday. The use of a weekly schedule is useful when processes must execute during the week but not on the weekends.

The schedule can also be created to run indefinitely or for a certain number of weeks.

- **Fixed interval:** A fixed interval schedule is run initially at a specified date and time and thereafter at a specified repeat interval for the schedule (that is, after how many hours and minutes should the schedule be run again). The repeat interval is from when the job started, so if the process started at 07:00:00 and finished at 07:10:00 and the repeat interval was two hours, then the process would run at about 09:00:00. The process may not run exactly at 09:00:00, because by default the WAS scheduler has a poll interval of 30 seconds to check for scheduled jobs to run.

The schedule can be created to run indefinitely or for a certain number of intervals.

Note: The repeat interval can be 23 hours 59 minutes or less.

- *Update* is used to update an existing schedule. For example, the schedule for a process can be changed from a daily schedule to a weekly schedule. A schedule will also have to be updated when the process profile is changed.

So if the value for a variable needs to change or a different profile needs to be used, then this change is performed here.

Changing a schedule is much like creating one in that a new date and time can be specified for the process to initially run and then the new type of schedule will commence. For example, if an existing daily schedule was set to next execute at 09:00:00 on the 16th of July and the schedule was changed to a fixed interval, then by default the first run of the new schedule would be at 09:00:00 on the 16th of July. The starting time and date of the updated schedule can be changed but they need to be greater than or equal to the current date and time.

- ▶ *Suspend* is used to stop a current schedule, which prevents any scheduled processes that have not yet started.
- ▶ *Resume* is used to start a schedule that has been previously suspended.
- ▶ *Delete* is used to remove a schedule. If there are a large number of schedules then the deleting of completed schedules is a useful means of housekeeping.

Monitor process instances

Each instance of a runtime process can be monitored through the DWE Administration Console whether the instance was invoked through a schedule or an *ad hoc* run. The Monitor Process Instance screen, shown in Figure 9-28 on page 427, contains information about the execution of the process and the ability to manage failed instances. Each instance that is run adds one row to this screen so the included filter allows the amount of data shown on-screen to be restricted. The example in Figure 9-28 on page 427 has a filter to only show processes that were started after a certain date, which resulted in 10 out of 109 rows being displayed (not 148). For an administrator or support personnel, the State column is useful, as it quickly shows the current state of the instance. Examples include *started*, *finished*, *finished with warning*, *terminated*, and *failed*.

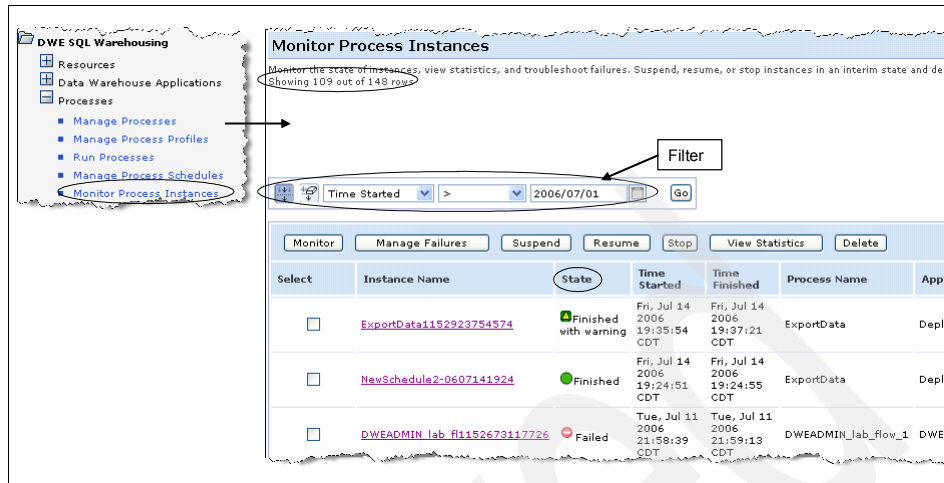


Figure 9-28 Managing a process instance

The section used to monitor the progress of a process instance is illustrated by Figure 9-28 and is located within the DWE Administration Console at **DWE SQL Warehousing → Processes → Monitor Process Instances**.

The section of the Administration Console depicted in Figure 9-28 allows the monitoring and management of the instances that are running or have run:

- *Monitor* shows the progress of each activity in the process. There are two runtime instances and if [NewSchedule2-0607142116](#) is selected to be monitored then the state of each activity is shown. For this instance the [Data_Flow_04](#) activity has failed but the subsequent [Command_02](#) operator has successfully completed. If the process was in a *running* state then the activity information would show which operators have finished, which one was running and what was still to run. Further details on the activity can be retrieved by clicking the **Activity Name**.

The details of an activity are depicted by Figure 9-23 on page 419.

- *Manage Failures* is used to work with any instances that have a state of *failed*, as depicted in Figure 9-29 on page 428. If the instance had *finished with warning*, it would not be included because the instance has to have failed. Viewing the details of a failed instance is discussed in 9.3.6 “Warehousing diagnostics” on page 431.

| Select | Instance Name | State | Time Started | Time Finished | Process Name | Application |
|-------------------------------------|-------------------------|-----------------------|-------------------------------|-------------------------------|--------------|----------------------------|
| <input type="checkbox"/> | ExportData1153005959562 | Started | Sat, Jul 15 2006 18:25:59 CDT | | ExportData | Deploywithtable_Warehouse1 |
| <input checked="" type="checkbox"/> | NewSchedule2-0307142116 | Finished with warning | Fri, Jul 14 2006 21:16:56 CDT | Fri, Jul 14 2006 21:17:06 CDT | ExportData | Deploywithtable_Warehouse1 |

| Activity Name | State | Type | Start Date | End Date |
|---------------|----------|------------------|-------------------------------|-------------------------------|
| Data_Flow_04 | Failed | DataFlowActivity | Fri, Jul 14 2006 21:16:58 CDT | Fri, Jul 14 2006 21:16:58 CDT |
| Command_02 | Finished | DB2SQLScript | Fri, Jul 14 2006 21:16:59 CDT | Fri, Jul 14 2006 21:17:06 CDT |

Figure 9-29 Monitoring a process instance

- *Suspend* suspends an instance that is running or in the process of being stopped. The instance goes into a suspended state after the activity that was running when the Suspend button was pressed has finished. Suspending an instance is different from stopping it, as suspending allows the instance to be restarted.

To illustrate how the suspending of an instance works an example process can be used that has two activities, such as in Figure 9-29, where one activity is a data flow followed by a command operator.

Example 9-5 gives these operators more detail to help illustrate the behavior of the *suspend* option.

Example 9-5 Pseudo code to insert data then export to a flat file

```
//Source table contains 500K rows
Data Flow:
insert into target.table select * from source.table

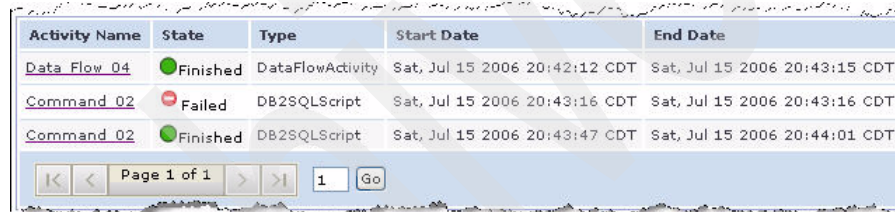
Command operator:
export to file.del of del select * from target.table
```

If an operator or administrator *suspended* the instance while the data flow in Example 9-5 was executing, the data flow would finish as normal and populate the 500-K rows into the target table. The instance would then be placed into a *suspended* state with the command operator not having been

run. If the instance was in a *stopping* phase then resumed during the data flow then the data flow would continue and populate the 500-K rows and then execute the command operator.

Once *suspended*, an instance can be *resumed*, allowing subsequent activities to be executed or *stopped*, which means that subsequent activities cannot be run.

- *Resume* puts a failed, stopping, suspending, or suspended instance back in the queue to be executed again. A suspended instance will be resumed at the start of the next activity while a failed instance can be restarted with the failed activity or with the next one. The exception to this is a failed activity in an iteration that always restarts with the start iterator of the outermost iteration. In Example 9-5 on page 428 if the data flow was successful but the command operator had failed then the resumption of the instance would result in the command operator being executed (but the data flow would not be executed again, as it had already run successfully). The results of the successful resumption of a failed instance are shown in Figure 9-30.



| Activity Name | State | Type | Start Date | End Date |
|---------------|----------|------------------|-------------------------------|-------------------------------|
| Data Flow_04 | Finished | DataFlowActivity | Sat, Jul 15 2006 20:42:12 CDT | Sat, Jul 15 2006 20:43:15 CDT |
| Command_02 | Failed | DB2SQLScript | Sat, Jul 15 2006 20:43:16 CDT | Sat, Jul 15 2006 20:43:16 CDT |
| Command_02 | Finished | DB2SQLScript | Sat, Jul 15 2006 20:43:47 CDT | Sat, Jul 15 2006 20:44:01 CDT |

Page 1 of 1 1 Go

Figure 9-30 Resumption of a failed instance

If the instance was stopping either the data flow or command flow, then the activity can be resumed and can then complete successfully. This is because resuming on a stopping instance resets the state, and until this request is completed the state can be switched back again. If an instance has stopped then it cannot be resumed.

- *Stop* either stops or terminates an instance. When a failed instance is stopped the result is a *terminated* instance. If the instance was running or was suspended then the result is a *stopped* instance. The instance will stop after the activity that was running at the time has concluded. Until that activity has concluded, the instance will be in a *stopping* state. With Example 9-5 on page 428, if the stop button was pressed during the data flow then this would complete (the 500-K rows would be loaded) but the command flow would not be executed. As a stopped instance cannot be resumed, it should only be used if there is no requirement to resume the process.
- *View Statistics* displays statistics at the process, activity, and underlying code unit levels for the selected instance. Viewing the statistics for an instance is discussed within 9.3.6 “Warehousing diagnostics” on page 431.

- *Delete* deletes the process instance from the list of monitored instances. This could be considered for housekeeping purposes where instances that are a certain vintage may be removed. An alternative might be that instances that have been successful may not need to have their details kept in this section. Even though an instance has been removed, the runtime statistics are retained for diagnostic and reference purposes. These runtime statistics can be deleted using the **History and Statistics** → **View Statistics** section of the Admin Console, which is discussed in 9.3.6 “Warehousing diagnostics” on page 431.

Note: If the instance is deleted from the monitoring and statistics sections and the application has *retain instance statistics* set to no, then the instance is permanently deleted from the control database. If *retain instance statistics* is set to yes, the instance is only removed from the monitoring and statistics pages in the Administration Console. Instance statistics are also removed upon removal of the SQW application.

9.3.6 Warehousing diagnostics

Knowing what diagnostic information is available and where to find it is important for the support of a production DWE solution. In this chapter we have discussed the defining of resources required by the deployed applications, the deploying of the applications, and the configuring and running of the processes within the applications. In looking at the processes, the monitoring of process instances was introduced. In this part of the chapter we continue looking at the diagnostics that are available in the three remaining sections of the SQL Warehousing area of the Administration Console:

- **Troubleshooting:** Under the Troubleshooting heading the Manage Failed Instances section can be accessed by selecting **DWE SQL Warehousing** → **Troubleshooting** → **Manage Failed Instances**. This section lists the process instances that have failed and supplies further information so that any issues can be diagnosed. Figure 9-31 on page 431 illustrates the steps involved in viewing the instances that have failed.

The screenshot shows the 'Manage Failed Instances' page in the SQL Warehousing Administration Console. The page is divided into several sections:

- Top Section:** Contains buttons for 'View Details', 'Delete', and 'Restart'. Below these is a table listing failed instances. The first instance is 'NewFlow1151025738534' with application 'Data Mining_Test10', activity type 'DFLOW', and a failure time of 'Thu, Jun 22 2006 20:22:59 CDT'. The error message is 'DB2 SQL error: SQLCODE: -286, SQLSTATE: 42727, SQLERRMC: 4096;DB2ADMIN'.
- General Section:** A tabbed interface with 'General', 'Code Unit Details', and 'Variable Information'. The 'General' tab is active, showing details for the selected instance:
 - Application Name: Data Mining_Test10
 - Process Name: NewFlow
 - Process Instance Name: NewFlow1151025738534
 - Activity Name: Data_Flow_02
 - Owner: db2admin
 - Execution Plan Graph: [C:/HomeLocation/applications/Data Mining_Test10/et/runtime/Data.Mining.control-Flows/NewFlow.sflowxml.03.RUNTIME.epg.xml](#)
 - Process Log: [C:/Logs/Data Minin_NewFlow.html](#)
- EPG Contents Section:** A pop-up window showing the contents of the execution plan graph (EPG) file for the failed instance. It contains a 'Back' button and a list of commands:
 - Command_09 () : type GRAPH : node
 - PREP () : type BLOCK : node /graph9
 - CODE_UNIT, node (node12)
- Log Section:** A table showing the execution log for the process. It includes timestamps and messages:
 - Jun 27, 2006 3:29:21 PM: NewFlow
 - Jun 27, 2006 3:29:21 PM: SQW03415I: Flow execution started for process
 - Jun 27, 2006 3:29:22 PM: NewFlow:Data_Flow_02
 - Jun 27, 2006 3:29:22 PM: SQW03509I: Statement 0 in unit /graph12/n
 - Jun 27, 2006 3:29:22 PM: SQW03503E: Failure to execute statement fo
 - Jun 27, 2006 3:29:22 PM: A default table space could not be found with "DB2ADMIN" is authorized to use.
 - Jun 27, 2006 3:29:22 PM: SQLState: 42727, SQLCode: -286

Figure 9-31 Managing a failed instance

- Initially, the list of failed instances is presented with options to *view details*, *delete*, or *restart*. Deleting the instance removes it from the page and

changes its status so that it cannot be viewed on this page. Restarting a process allows either the failed activity to be restarted or restarting the activity that follows the failed activity. Viewing the details opens the Failed Instance Details window.

- The Failed Instance Details window has three tabs, General, Code Unit Details, and Variable Information. Variable information contains the variable values at the time of the instance failure. Code Unit Details contains the information about the code unit that failed, such as the code unit name and the code unit type. The General tab provides links to the Execution Plan Graph (EPG) and the process log.
 - The explain plan graph file has the text version of the EPG.
 - The process log file either contains multiple instances or a log for each instance, depending on whether the log has been selected to be removed before a new instance is started. The log contains details of the failing operation, such as additional information about the SQL error code that was shown on the initial screen.
- *History and Statistics*: This section of the Admin Console contains deployment information about applications and instance level and summary statistics about processes that have been run.
- *View Deployment History* displays information about all of the deployments that have occurred since the Administration Console was installed. This information could include when the application was deployed, when it was attempted to be deployed, and when it was uninstalled. There can be multiple entries per application, so if an application has been deployed and subsequently uninstalled then there will be two deployment entries. Selecting the application name provides details of the deployment activity including the status (success or failure) and the location of the zip file used for deployment.
 - *View Statistics for Process Instances* shows the elapsed time and other statistical attributes for a specified process. There is a set of statistics associated with every process that is run, and with many processes running a process filter should be used. This filter contains a *time started* option, which will only show statistics before or after a certain date. The performance of the individual activities within the process can be viewed as well as the corresponding performance of the code unit elements.

Figure 9-32 on page 433 illustrates the statistics that are available for the process instance. In particular it shows:

- The initial statistics screen that contains all of the processes that have been run on the Administration Console. The filter allows the number of processes to be limited by a variety of methods. The example limits the displayed processes to only those instances that started after the

specified date. The information provided by this screen is whether the instance ran successfully and when it started and finished. By selecting a particular instance more detailed information can be accessed.

- The *Process Instance Statistics* tab has details on the overall runtime of the process.
- The *Activity Instance Statistics* tab has details for each activity in a process. The process contains a DB2 command script that deletes the contents of a table and a data flow that populates the table with new data. This tab provides the elapsed times for each of these activities.
- The *Code Unit Instance Statistics* tab has details for each element of the EPG that ran as part of the process.

Process Instance Statistics

| Instance | Process Name | Application | State | Time Started | Time Finished |
|---------------------------|--------------|---------------------|----------|-------------------------------|-------------------------------|
| ControlFlow11151518238823 | ControlFlow1 | RedbookDW_MoveData1 | Finished | Wed, Jun 28 2006 13:10:38 CDT | Wed, Jun 28 2006 13:10:55 CDT |
| ControlFlow11151518238824 | ControlFlow1 | RedbookDW_MoveData1 | Finished | Wed, Jun 28 2006 13:10:47 CDT | Wed, Jun 28 2006 13:10:55 CDT |

Process Instance Statistics

| | | | |
|------------------|------------------------------------|---------------|-------------------------------|
| Process Instance | ControlFlow11151518238823 | Process Name | ControlFlow1 |
| Application | RedbookDW_MoveData1 | Status | Finished |
| Time Started | Wed, Jun 28 2006 13:10:38 CDT | Time Finished | Wed, Jun 28 2006 13:10:55 CDT |
| Elapsed Time | 0 hour(s) 0 minute(s) 16 second(s) | | |

Activity Instance Statistics

| Activity Name | Activity ID | Type | Status | Time Started | Time Finished | Elapsed Time |
|---------------|-------------|------------------|----------|-------------------------------|-------------------------------|--|
| Command_02 | 0 | DB2SQLScript | Finished | Wed, Jun 28 2006 13:10:43 CDT | Wed, Jun 28 2006 13:10:51 CDT | 0 hour(s) 0 minute(s) 7 second(s) 844 millisecond(s) |
| Data_Flow_03 | 0 | DataFlowActivity | Finished | Wed, Jun 28 2006 13:10:51 CDT | Wed, Jun 28 2006 13:10:55 CDT | 0 hour(s) 0 minute(s) 4 second(s) 500 millisecond(s) |

Code Unit Instance Statistics

| Unit Name | Unit Type | Status | Time Started | Time Finished | Elapsed Time | Rows Processed |
|----------------|-----------|----------|-------------------------------|-------------------------------|--|----------------|
| /node12 | SQLSCRIPT | Finished | Wed, Jun 28 2006 13:10:43 CDT | Wed, Jun 28 2006 13:10:43 CDT | 0 hour(s) 0 minute(s) 0 second(s) 47 millisecond(s) | N/A |
| /graph12/node9 | JDBC | Finished | Wed, Jun 28 2006 13:10:51 CDT | Wed, Jun 28 2006 13:10:51 CDT | 0 hour(s) 0 minute(s) 0 second(s) 47 millisecond(s) | 0 |
| /graph15/node9 | JDBC | Finished | Wed, Jun 28 2006 13:10:51 CDT | Wed, Jun 28 2006 13:10:55 CDT | 0 hour(s) 0 minute(s) 3 second(s) 797 millisecond(s) | 103231 |

Figure 9-32 Viewing statistics for an instance of a process

- *View Summary Statistics for Process Instances* shows the summary statistics for all of the instances of a specified process. These summary statistics can be useful in identifying bottlenecks in a batch window or to do additional performance analysis and measurements on process instances. Figure 9-33 on page 435 illustrates the summary statistics for a process. In particular they show:
 - The initial summary statistics screen containing all of the processes that have been run on the Administration Console. In the example, the list of processes has been filtered to only show one process. By selecting a particular process more detailed information can be accessed.
 - The Process Statistics tab has details on the average runtime of a process, the minimum and maximum time, and which instances were the fastest and slowest.
 - The Activity Statistics tab has details for each activity within a process. This tab provides the average elapsed time for each of the completed runs of these activities.
 - The Code Unit Statistics tab has summary statistics for each element of the EPG that ran as part of the process.

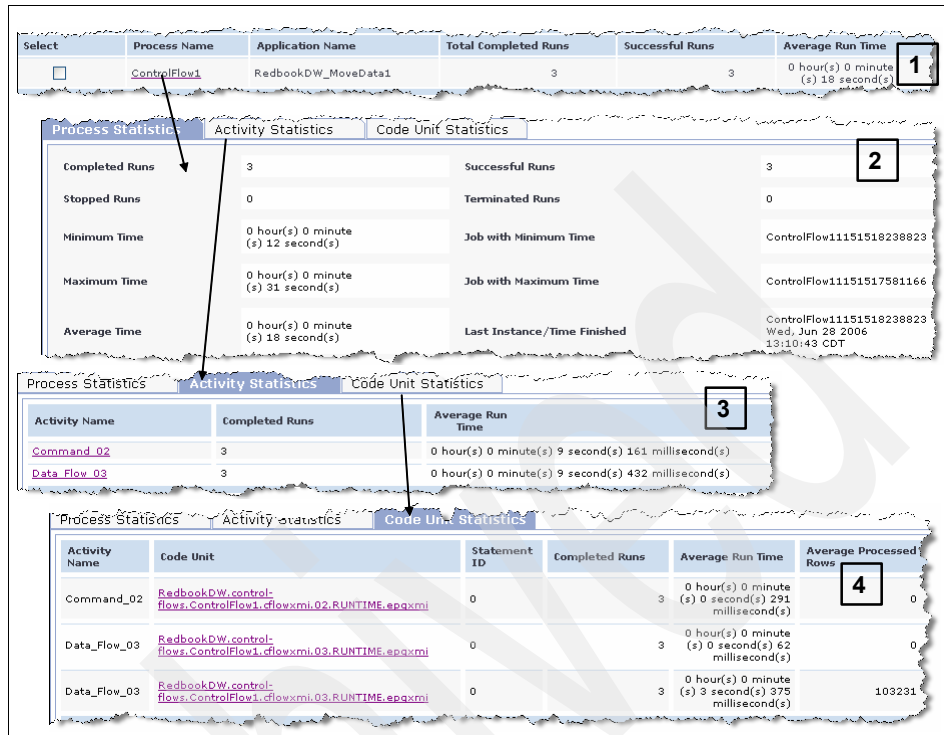


Figure 9-33 Viewing summary statistics for an instance

- **Logs and Traces:** This section of the Administration Console provides direct access to the logs for each process, the logs for the DWE Administration Console, and the WebSphere system and error logs. The process logs would typically be the first set of logs to be examined when diagnosing issues since most common problems such as database connectivity or availability of a system resource would be reported here. The Admin Console log would typically be looked at next to view the log and trace information that is specific to the Admin Console. And, finally, the WebSphere JVM log files would be looked at since they contain information for the entire WebSphere server in addition to the DWE components.

Selecting **View process logs** enables the user to access the logs for all of the processes on the server. The log file path column indicates where the log is located and what it is called. The log location is updatable and managed at the application level, while the log name is also updatable but managed at the process level. The contents of the process log depend on what level of logging and tracing has been defined for the process. In the example shown

in Figure 9-34, the log level was defined as INFO and the trace level as NONE. This means that informational messages would be captured but the detail of what the process is doing was not captured.

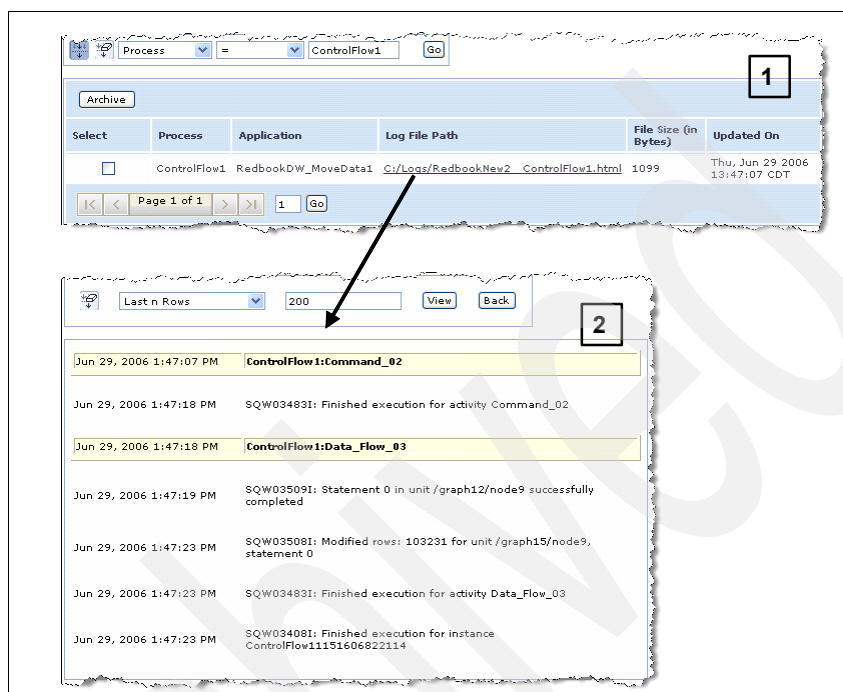


Figure 9-34 Viewing the logs for a process

The example shows the following:

- The View Process Logs screen (1) contains the logs for all of the processes that have been run on the Administration Console. If the log name has been changed for a process then only the current log is displayed. The log can also be archived from this screen, which copies the log file to a location on the Admin Console while retaining the contents of the log.
- The Log file tab views the process log (2), which contains the details on the execution of the process. With a larger file we recommend initially using the *Last n Rows* option to only view the last 200 rows. This value can then be incremented to view earlier entries in the log.

Selecting **View Server logs** allows users to view the logs for the server rather than just the logs for a particular process. These server logs are segmented in the log specific to the Admin Console and also the WebSphere JVM logs. As with the process logs, all of these logs can be viewed by a certain number

of rows at a time. The Admin Console log can also be archived, which copies the current contents to a defined file and restarts the log.

Figure 9-35 shows the server logs for the Admin Console, and WebSphere, reporting the same database connection issue.

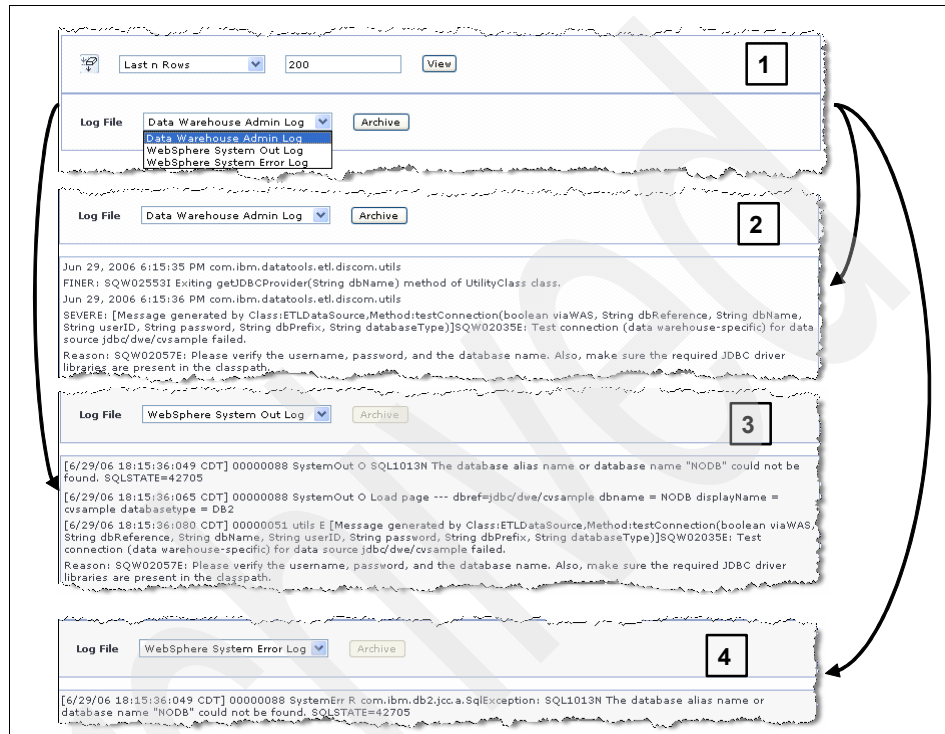


Figure 9-35 Viewing the server logs

In particular, the example shows the following;

- The View Server Logs screen enables the specific server log to be chosen as well as to specify how many rows of the log should be displayed. The data warehouse administration log is archived from this screen.

Note: The WebSphere JVM logs are not archived from the Admin Console since they are managed in the WebSphere Administrative Console.

- The *data warehouse admin log* is specific to the data warehousing components of the DWE Admin Console. The location of the log file is contained in the WAS config.properties file, which is located in <WAS_HOME>\lib\ext. A typical example on Windows would be:

C:\Program Files\IBM\dwe\appServer\lib\ext\config.properties

This file has a variable called ETL_ADMIN_LOG, which contains the location of the data warehouse Admin Log. The file also contains a variable, LOGLEVEL, which by default is set to ALL. This is the most detailed level of application logging. The others, in order, are FINEST, FINER, FINE, INFO, WARNING, and ERROR.

The example shows the data warehousing application calling the getJDBCProvider class that has failed for the JNDI data source jdbc/dwe/cvsample.

- The WebSphere system out log is used to monitor the health of the running application server. Its location is also specified in the config.properties file and in the WebSphere Administration Console under **Logging and Tracing** → *server_name* → **JVM Logs**.

In the example, the physical database NODB has been mapped to the jdbc/dwe/cvsample name but it cannot be found, and so cannot be connected.

- The WebSphere system error log has exception stack trace information that is useful when performing problem analysis. Its location is also specified in the config.properties file and in the WebSphere Administration Console under **Logging and Tracing** → *server_name* → **JVM Logs**.

In the example, the log reiterates what the system out log found, which is that the application cannot connect to the NODB database.

9.4 Managing the OLAP functionality

The OLAP section of the DWE Administration Console enables the deployment and management of the DWE OLAP solution. Specifically, a user can import and export an OLAP model, use the Optimization Advisor to recommend MQTs, check whether queries are using the defined MQTs, and explore existing cubes and cube models. The Admin Console shares much of the OLAP functionality with the DWE Design Studio. The main difference is that the OLAP models and cubes are created in the Design Studio with the Admin Console managing them once they have been created.

An example is that within the Admin Console you cannot specify a slice and query type (for example, drill-down) that is likely to occur when a user looks at

data in the model, while in the Design Studio these advanced options are available.

9.4.1 Creating the OLAP environment

The OLAP Management section in the Admin Console is where a production database can import OLAP metadata from an XML file. Typically, this XML file will be from a database containing OLAP models whose metadata has been exported from the Design Studio. However, the metadata can also have originated from a partner product that supports the XML format used by DWE OLAP.

This section of the Admin Console also allows for OLAP metadata to be exported from a database, which could then be required by another vendor application or another DB2 database. Importing and exporting of OLAP metadata can also be performed in a development environment using the Design Studio.

Note: Databases used for DWE OLAP in the Admin Console need a profile defined and be enabled for OLAP

9.4.2 Optimizing the OLAP environment

The OLAP Optimization section, shown in Figure 9-36, in the Admin Console performs the creation of aggregations to optimize user queries against the OLAP model. The section also includes a means of checking whether the aggregations are being used by the user queries. There is also functionality for running aggregation scripts that have been recommended.

OLAP Optimization

Select a database profile to connect to the database.
Database Profile: **Test**
Database Name: SAMPLE

Select a cube model.

| Name | Schema | Description |
|--------------------------------------|----------|--|
| <input type="checkbox"/> Sales Model | CVSAMPLE | Sales Model cube model is a representation of a CVSAMPLE schema that groups Market, Product and Time dimension objects around a central Sales Facts object |

1 Page 1 of 1

Figure 9-36 OLAP optimization options

Advising on DB2 summary tables

The *advise* option is where an OLAP model, imported from XML metadata, can be optimized by the use of DB2 summary tables, enabling improved performance

of aggregate queries. DB2 summary tables can improve query performance because they contain precomputed and pre aggregated results from one or more tables that can be used in a query. A summary table is a special type of a materialized query table (MQT) that specifically includes summary data.

The summary tables recommended by the advisor are based on an OLAP model. The advisor allows the user to specify in which DB2 tablespace the recommend summary tables should be placed and whether the summary tables are to be refresh deferred or immediate. Immediate summary tables are continually synchronized with the base tables. The changes to the base tables are tracked by DB2 so that the summary tables can be incrementally updated by changing the portion of the summary tables that corresponds to the changed portion of the base tables. Deferred summary tables are not automatically updated, rather they are *refreshed* at an interval defined by the user. Refreshing a summary table completely rebuilds the summary by either using the DB2 REFRESH command or another population method such as the DB2 LOAD statement.

There are considerations when specifying the update strategy for the summary tables. If the base tables are updated on a regular basis then the refresh immediate option should be considered, as the synchronization of base and summary tables will be managed by DB2. Deferred refreshes are preferable when the base tables contain large amounts of data or where a batch process is the desired mechanism for controlling when the summary tables are updated.

Note: Even though the immediate option is chosen the advisor may not recommend refreshing immediate summary tables due to the greater functionality offered by refresh deferred summary tables.

If refresh deferred summary tables are to be used then the DB2 database configuration parameter DFT_REFRESH_AGE must be set to a value other than the default of 0. The value for this parameter contains a time duration, which specifies how long since the summary table was refreshed, that can be used in the optimization of a query. The highest setting is ANY, which means that all summary tables can be considered by the DB2 Optimizer regardless of when they were recreated.

The Optimization Advisor also needs input as to how much of the system resources should be devoted to recommending and deploying the summary tables. By default the advisor will take as long as is required to recommend suitable summary tables and potentially recommend summary tables that take up considerable disk space. These default settings can be changed so that a time limit can be placed on how long can be spent considering the optimizations and how large the recommended summary tables can be.

After the Optimization Advisor is complete, the recommendations for the creation of the summary tables, any associated indexes, and the maintenance scripts can be downloaded and executed using the *Run SQL Scripts* option.

Tip: The recommended summary tables will have default names such as DB2INFO.MQT0000000001T01, which can be altered.

The benefit of recommending summary tables in the runtime environment rather than using a script generated from the development environment is that the Optimization Advisor has based the recommendations on runtime data volumes and any additional indexes that may exist. The benefit of using a script from the development environment is that the maintenance of the summary tables, assuming they are not refresh immediate, can be incorporated into a control flow. This control flow can ensure that the summary tables are only refreshed when the preceding data flow has successfully completed. If the summary tables are based on the production Optimization Advisor then it is advisable to include this script in a development data flow and redeploy the warehousing application. This process would ensure that the recommended summary tables are as optimal as possible while ensuring that they are maintained as part of the larger warehouse environment.

Checking whether DB2 summary tables are being used

The *summary table usage* option allows the testing of how much of a SQL workload is being routed to a summary table on the system. This feature makes it possible to quickly evaluate whether a deployed MQT is being used by user queries. The evaluation works with each OLAP model that is present in the OLAP-enabled database. As has been discussed in Chapter 5, “Enabling OLAP in DB2 UDB” on page 79, there may be multiple OLAP cubes that are based on one underlying model and the optimization evaluation works with the model. As summary tables can be created without the Design Studio and the Admin Console, these other summary tables will also be considered when the usage is analyzed.

Tip: If summary tables are created independently of the Optimization Advisor they could have a different naming convention so as to be easily distinguishable when running this analysis.

Testing the summary table utilization usage is based on the queries that have been, or will be, run against the database. Existing queries can be used as an input by specifying the time period for which you want to examine queries. Typically, a user might want to look at how the summary tables are being used during the day rather than overnight when a batch may be running. The input queries can also be specified in a file containing SQL statements.

Tip: This input file should separate the SQL statements using a semi-colon (;)

Whether the queries have already been executed against the database or they are being provided as part of an input file, they both require that the database explain tables have been created. The evaluation process can only consider the previously executed queries that have had their access plans explained while the queries contained in the input file are executed against DB2 in explain mode only. Explain mode is a means of running SQL against DB2, which does not execute the query but does generate the query execution plan, which is inserted into the DB2 explain tables. These explain tables need to be created in every database that will require the evaluation of the usage of summary tables. If the explain tables do not exist on the database where the cube mode has been deployed then the Admin Console returns a DWE4418E error.

The explain tables can be created as detailed in Example 9-6.

Example 9-6 Creating EXPLAIN tables

Windows:

```
C:\Program Files\IBM\SQLLIB\MISC>db2 -tf EXPLAIN.DDL
```

Linux/Unix:

```
/home/db2inst1/sqllib/misc: db2 -tf EXPLAIN.DDL
```

To create these tables on a remote database from the DWE Admin Console, the database needs to be cataloged locally to enable the use of the DB2 CLP. Further details on the explain tables and their usage can be found in the DB2 manuals, which are available online at:

<http://publib.boulder.ibm.com/infocenter/db2luw/v8//index.jsp>

Once the selected SQL statements have been read from the explain tables the Admin Console displays a Summary Table Usage screen, as depicted in Figure 9-37.

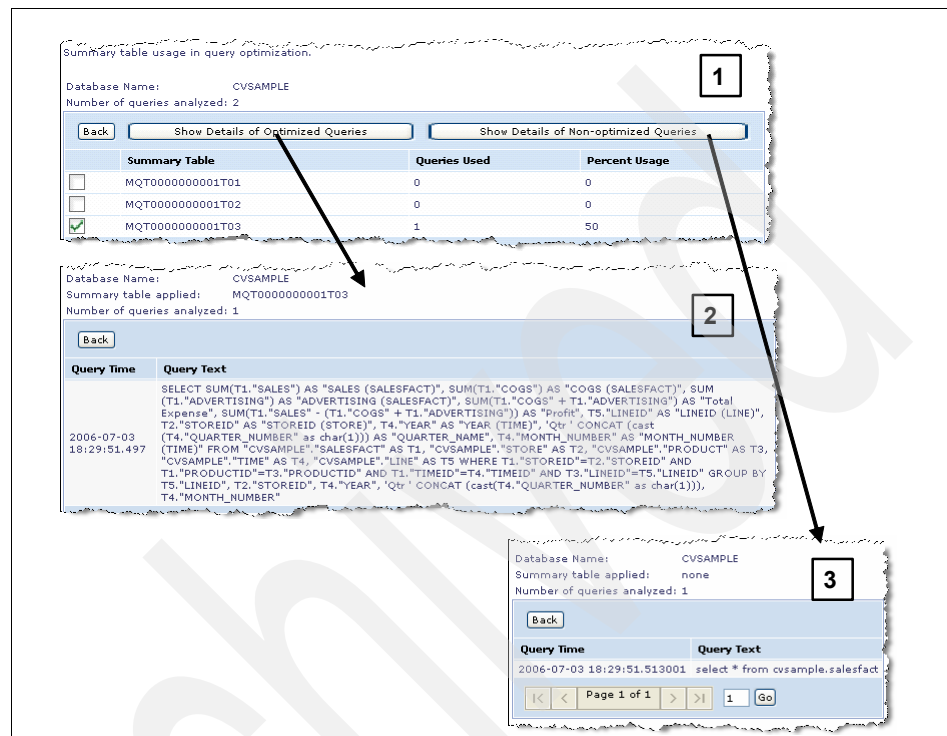


Figure 9-37 Usage of summary tables

This illustrates a scenario where the OLAP database has three summary tables (MQTs) and the input file contained two queries:

- ▶ One summary table should be selected before choosing to view queries that were or were not optimized by the summary table. The *queries used* and *percent usage* values are based on the current workloads only. With two input queries and one summary table being used, the percent usage is set to 50%.
- ▶ The time of the query and the original SQL for the optimized query is shown.
- ▶ The time of the query and the original SQL for the query that was not optimized by the summary table is shown.

Running of SQL scripts to create DB2 summary tables

The *run SQL script* option allows the running of DB2 SQL scripts to create the summary tables or to run an SQL script to refresh the summary tables. The script will be executed on the database specified in the initial OLAP Optimization

screen. If this is a remote database then it has to be cataloged locally so that a DB2 CLP script can be executed against it.

The SQL script does not need a CONNECT statement or a CONNECT RESET statement since these will be added at runtime. Each SQL statement in the script should be separated by the semi-colon character (;), assuming the script is run by the Admin Console. Each executed script generates a log that can be viewed or saved.

9.4.3 Viewing the OLAP environment

The View OLAP Contents section in the Admin Console allows the viewing of the OLAP metadata from a Web browser. This capability can be used as an alternative to viewing the metadata through the DWE Design Studio or directly in the tables as they exist in the DB2 database. In Chapter 5, “Enabling OLAP in DB2 UDB” on page 79, we discuss the various OLAP objects in some detail so here we only provide a summary and highlight what information is available in the Admin Console.

As with the preceding sections on the OLAP models, the metadata is viewed one model at a time. To aid the user when navigating, the Admin Console contains a location map, which is enabled by default. The location map is shown in Figure 9-38, and indicates the level of the OLAP hierarchy for the current information being displayed.

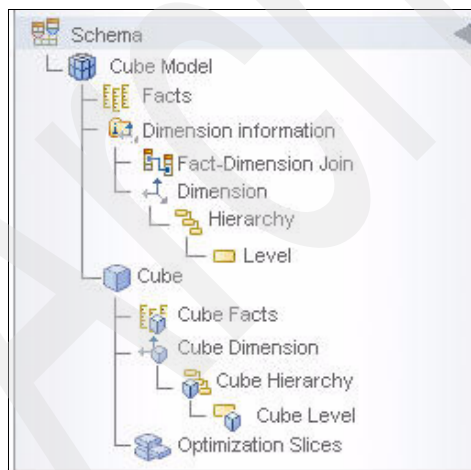


Figure 9-38 Viewing the current level of the OLAP metadata

The viewing of the OLAP metadata starts at the schema level. The schema is the grouping of database objects under which the OLAP objects have been created.

Each schema can potentially contain multiple cube models and the Admin Console allows the relevant model to be selected for further analysis.

The following are the remaining objects in the schema:

- ▶ *Cube model*: an OLAP model that has been created in the Design Studio that groups several dimension tables around a central fact table. One or more OLAP cubes can be created from a single cube model.
- ▶ *Facts* contain the fact information for a cube model and the fact table is comprised of several objects. The tables that make up the fact table are presented, as well as any joins that exist between the fact tables (should there be more than one fact table). The details of the fact attributes are displayed, which are generally the lookups to the dimension tables. Finally there is a listing of the measures that are part of the OLAP model, which includes the data type of the column, the business name, and how the column is aggregated. The measures information includes the source of the column and whether any expressions have been used in its creation.
- ▶ *Dimension Information* has summary level details for each dimension, including the name of the dimension, what comments have been left by the designer, and the type of the dimension. A dimension can either be a *regular* or a *time* dimension. The latter has special characteristics that allow applications to intelligently and appropriately perform time-related functions.
- ▶ *Fact-Dimension* join contains the detail of the join between the fact and dimension tables. This information includes both table names, cardinality of the join, and whether it was inner or outer join. Additional join details can be examined that show the operator and columns that comprise the join.
- ▶ *Dimensions* contain the detailed information about the facts that are present in a cube model and is comprised of several objects. The tables that comprise the dimensions are presented as well as any joins that exist between the tables.
- ▶ *Hierarchy* defines relationships among a set of attributes that are grouped by levels in the dimension of a cube model. These relationships provide a navigational and computational means of traversing dimensions, and there may be multiple hierarchies defined for each dimension. Each hierarchy also has a type and a deployment that describe the relationships among the levels within the hierarchy.
- ▶ *Level* consists of a set of attributes that work together as one logical step in a hierarchy's ordering. A level contains one or more attributes that are related and can function in one or more roles in the level. The relationship between the attributes in a level is usually defined with a functional dependency. An example is where a store key is the unique identifier of a row in a dimensional table, but there are other attributes, such as store size and store address, which may be used in queries.

- ▶ *Cube* contains a subset of metadata objects that are based on the metadata objects in the cube model. The cube facts object and list of cube dimensions are subsets of those in the referenced cube model. Furthermore, cubes are appropriate for tools and applications that do not use multiple hierarchies because cube dimensions allow only one cube hierarchy per cube dimension. The Admin Console provides a summary of the cubes in the database and a route to additional details about the cube.
- ▶ *Cube Facts* contains a listing of the measures that are part of the cube, which includes the data type of the column, the business name, and how the column is aggregated. The measures information includes the source of the column and whether any expressions have been used in its creation.
- ▶ *Cube Dimensions* reference the cube model dimension from which it is derived and the relevant cube hierarchy for the given cube. A cube dimension can have only one cube hierarchy.
- ▶ *Cube Hierarchy* can have a set of cube levels that are a subset of the parent levels from the cube model. In general, a cube hierarchy has the same hierarchy type and deployment mechanism as the cube model from which it is derived.
- ▶ *Cube Level* is a subset of a level in a cube mode that is used in a cube. A cube level references the level from which it is derived and inherits the level key attributes and default attribute that are defined for the cube mode level. The cube level can also have a set of attributes that are a subset of the related attributes from the cube model.
- ▶ *Optimization Slice* is defined and used by the Optimization Advisor in the Design Studio to provide summary tables that are focused on the most important regions of the cube model. An optimization slice is defined by a set of one or more levels, and the type of query expected at the slice, such as drill-down, report, MOLAP extract, hybrid extract, or drill through. The Admin Console depicts the optimization slices that have been defined for the cube. This is displayed in Figure 9-39. It shows that the optimization slice has been designed primarily for reporting and at a monthly level of the time hierarchy.

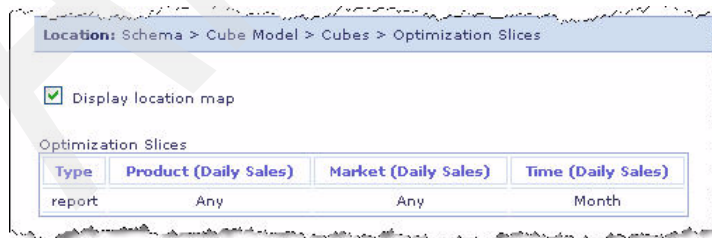


Figure 9-39 Optimization slices in an OLAP cube

When optimization levels are defined in the Design Studio they assist in the creation of improved summary tables due to additional information being provided to the OLAP optimization process.

9.5 Data mining

The rationale behind data mining and its implementation in DB2 Data Warehouse Edition (DWE) is discussed extensively in Chapter 7, “DWE and data mining” on page 237. The purpose of this section is to introduce the data mining features present in the DWE Administration Console. The mining model management section in the Admin Console allows the user to view, export, update, and delete models in the mining-enabled database. Mining models can also be imported into the mining database and loaded into a cache for quicker retrieval in the future. The mining visualization tool in the Admin Console further provides a graphic representation of the results of the mining model.

Before examining the details of the mining functionality in this section it is important to reiterate what constitutes a mining model:

- ▶ How mining is implemented in DWE, and the various mining models
- ▶ Deploying and managing the mining model
- ▶ Caching the mining model for performance
- ▶ Mining diagnostics in the Admin Console

9.5.1 Data mining modules

A mining model is one of three data mining modules included in DWE, the other two being scoring and visualization. The typical flow of how these features work together in a data mining process is as follows:

1. A mining task is defined using the modeling functionality.
2. An initial mining run is done to build a model using the modeling functionality.
3. This model is visualized using graphical functionality provided by the visualization functionality.
4. New data is scored against the existing model, which scores this new data using the scoring functionality.

Details of the data mining modules

Mining models created by the Design Studio are Predictive Model Markup Language (PMML) models that are stored in the mining tables created when a database is enabled for mining. PMML is an industry standard and the visualization and scoring modules within DWE work with models of this type.

There is also an industry standard SQL interface, SQL/MM, to the mining models that simplifies access by applications such as the scoring and visualization modules. When the mining models are stored in DB2 tables, each model is stored in a separate table.

The visualization module is a Java application or applet that can visualize all mining models created by DWE. As the visualization module reads a PMML model it can potentially read models from other providers who also support this standard. The module can read files that have been created in PMML format and also models that are stored in DB2 tables. Visualization of the mining models is available as part of the Design Studio, and as a Java applet on a Web server.

The scoring module can use two provided APIs. There is an *easy mining interface* that uses Java UDFs. It has a simplified interface to do scoring with a single call to an SQL stored procedure. Then the result of the scoring can be stored as a database view. The more detailed interface uses the SQL/MM API, which uses DB2 User Defined Types (UDTs) for the models, result specs, and scoring results and DB2 User Defined Functions (UDF) to import models into the database, score data against models, and analyze the scoring result. The results of the scoring are stored in DB2 tables that are created when the database is enabled for mining.

In Example 9-7 the easy mining interface is being used to score against a data model, where:

- ▶ IDMMX.ApplyClusModel is the stored procedure name.
- ▶ importedClusModel is the name of the model.
- ▶ BANKING_SCORING is the data source.
- ▶ BANKING_RESULT is the name of the view to be used as the output.
- ▶ CLUSTERID, QUALITY, and CONFIDENCE are the columns in the view.

Example 9-7 Simple scoring

```
CALL IDMMX.ApplyClusModel( 'importedClusModel', 'BANKING_SCORING',  
                           'BANKING_RESULT', 'CLUSTERID', 'QUALITY',  
                           'CONFIDENCE');
```

Types of mining models

The mining model is created in the DWE Design Studio using supplied mining functions to gain insight about the data in the data warehouse. The classification of mining functions is the same as the models that are created by them, being:

- ▶ Association
- ▶ Sequence rules
- ▶ Clustering

- ▶ Regression
- ▶ Classification

The goal of an association mining function is to find items that are consistently associated with each other in a meaningful way. For example, purchased transactions can be analyzed to discover combinations of goods that are often purchased together. The associations mining function answers the question: If certain items are present in a transaction, what other items are likely to be present in the same transaction?

A *sequence rules* mining function can be used in various business areas. An example in the manufacturing industry is where combinations can be found of components, materials, production machines, and external conditions that might lead to increased error rates. This information can then be used to proactively prevent subsequent errors.

A *clustering* mining function searches the input data for characteristics that frequently occur in common. The input data is then grouped into clusters with each member of the cluster having similar properties. There are no preconceived notions of what patterns exist within the data, since clustering is purely a discovery process.

A *regression* mining function predicts the value of a numerical data field, the target field, in a given data record from the known values of other data fields of the same record. The known values of other data fields are called input data fields or explanatory data fields and can be numerical or categorical. The predicted value might not be identical to any value contained in the data used to build the model. A regression model is created and trained based on known data sets of data records whose target field values are known.

This trained model can be applied to known or to unknown data. In unknown data, the values of the input fields are known. However, the value of the target field is not known. The application of the model to known data is called test mode. The main purpose of the test mode is to verify the quality and the predictive power of the model by comparing its predictions to the known actual target field values of the test data. If you are working with a mining model for a long time, then there should be regular test runs to verify that the model is still useful.

Classification is the process of automatically creating a model of classes from a set of records that contain class labels. The classification technique analyzes records that are already known to belong to a certain class, and creates a profile for a member of that class from the common characteristics of the records. A scoring tool enables the user to predict whether new records belong to that particular class. The classification mining function is widely used in client relationship marketing (CRM). Commercial applications of this mining function

include credit card scoring, ranking of clients for directed mailing, the prediction of credit risk in new clients, and attrition prediction.

9.5.2 Deploying and managing the data mining models

The ability to import and manage the mining models within the Admin Console is important, as they can be used in a variety of ways. Mining models can be embedded into a mining flow, which can then be a part of a deployed data warehouse application. An example of this would be where new data is processed through the database tables, and this data is then scored against an existing mining model. For this application to run in a production environment the mining model needs to have been deployed to the appropriate database. Mining models can also be used within a Java class as part of a Web-based application. For this application to be able to score or visualize the data, the mining model again needs to be deployed into the correct production database.

In this section we discuss the deployment and management of the mining models, which correspond to the DWE Mining → Model Management section of the Admin Console, as shown in Figure 9-40.

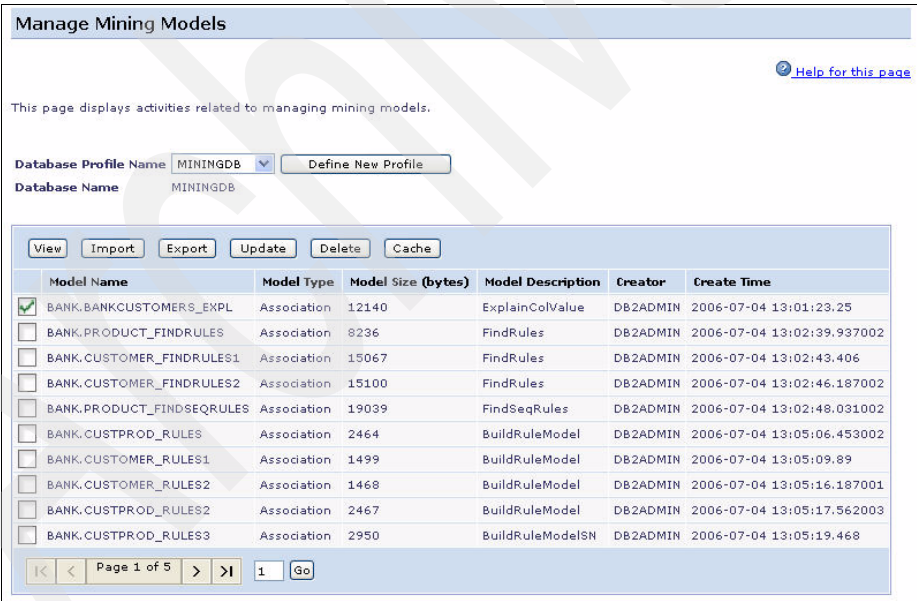


Figure 9-40 Managing mining models

If mining models are already in the database profile that has been selected, then they will be presented in this initial screen, as depicted in Figure 9-40. The model type is shown that corresponds to the different types of mining models introduced

earlier in this chapter. From this initial screen the database that is being used to store the mining models can be managed by selecting the following options:

1. View
2. Import
3. Export
4. Update
5. Delete
6. Cache

Viewing a mining model invokes a DWE-supplied Java applet to visualize the model, which is then displayed in the Web browser.

Note: If the browser does not have the required Java plug-in, a link is provided to download it from the Sun™ Microsystems™ Web site.

This Java applet incorporates the visualization functionality for the mining models and allows the details of the model to be displayed as well as the graphical representations of the data. Common to all the types of mining models is the ability to print the details and to save the model as a visualizer (.vis) file. Another common feature is that a help function is provided for each type of visualization, which can assist with the explanation of mining terms and how to interpret the numerical results. The tabs in the applet and the information contained in the tabs are specific to each type of model, with Figure 9-41 being an example of viewing the contents of a clustering model.

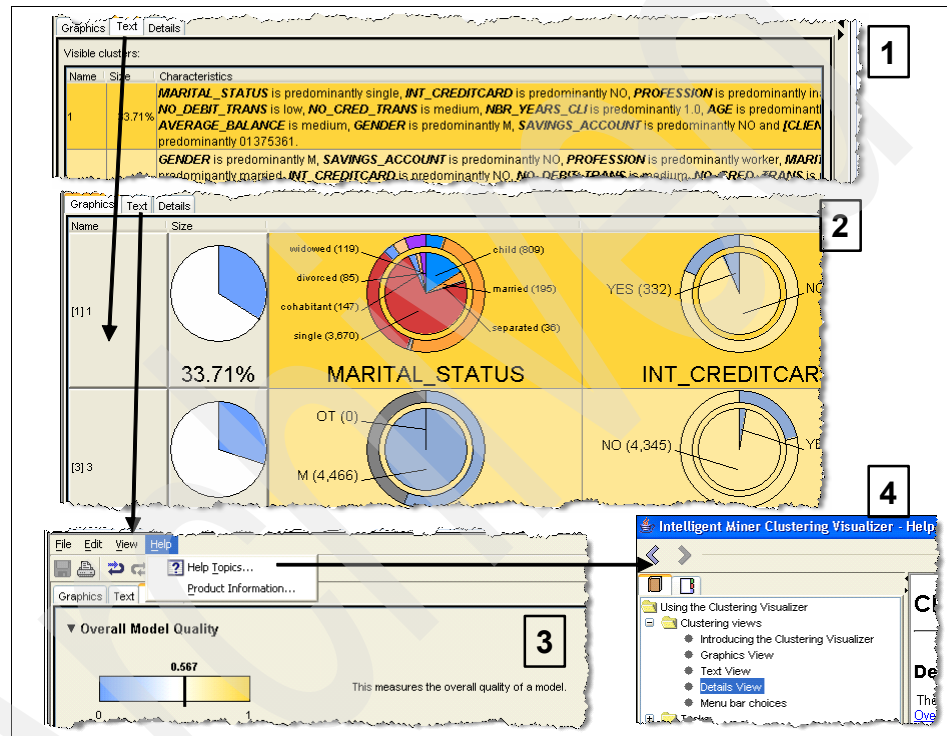


Figure 9-41 Viewing the contents of a clustering model

In Figure 9-41 a clustering model is depicted within the Admin Console. We discuss the details of a model in the list below:

1. The *Text* view provides details of how the many clusters of data have been created (there are four in this example) and what percentage of the total data each cluster contains. Each individual cluster has its own set of characteristics that provide textual interpretations of how the data is distributed within the cluster.

2. The *Graphics* view shows multiple graphs for each cluster, with each graph showing how the data is distributed within that characteristic. For the `MARITAL_STATUS` characteristic you can see how the underlying data supports the *predominantly single* interpretation in the text view.
3. The *Details* view shows further information about the data in the model, such as what the mean, min, and max is of fields such as age and number of years as a client (`NBR_YEARS_CLI`). The Details view also provides a numerical value as to the overall quality of the model, which is explained in the help section.
4. There is a help section specific to each of the visualizer modules, which in this case is the cluster visualizer. Looking up the meaning of the *overall model quality*, the help explains the significance of the numerical value, which is that a value of nearly 0.6 means that a pair of records in the same cluster have the same values in 60% of the active fields.

Importing a model allows a PMML mining model to be imported from the client to the mining tables on the database server. As part of the process of importing this XML file, the model name, model type, and business name can be specified. If the mining model is being used in a warehousing application then the name given to the imported model must be the same name as the model used in the development mining flow.

Exporting a model takes a model that is stored in the mining tables in DB2 and exports it to a PMML format flat file. This exported file can be used with appropriate partner products or used to transfer a mining model to another DB2 database.

The updating of a model provides the ability to specify a new PMML file to be used for the model and to change the model description and business name. Updating a model may be required where you have a production mining flow running that has been changed on the client database and needs to be propagated to the server database. The type of model has to be the same for both the original and the updated mining model. If an attempt is made to update an association model with a clustering model, the Admin Console will raise a DWE3229E error.

Deleting a model removes the selected mining model from the mining tables in the database and prevents its further use by warehousing applications and embedded Java objects. The deletion of a model is permanent. So, unless the original imported file is readily available it is a good practice to export a copy of the model before deletion.

9.5.3 Caching the mining model for performance

DWE provides a memory cache for mining models that is designed to improve the performance of scoring runs and allow real-time scoring. The use of memory caching allows the mining models to be kept in memory for subsequent scoring runs. When the model is permanently stored in memory, a scoring run is faster because the initial loading, decompressing, and parsing of the compressed XML model from the mining tables can be skipped. When a model is not stored in the cache, the performance of real-time scoring can be affected as the mining model is deleted from memory after the scoring run is finished. The mining model cache is for the entire database instance, so if there are multiple mining-enabled databases then all of the models can be placed in the same cache.

A mining model can either be explicitly cached or cached through use in a prior scoring run. An explicitly cached model is known as a *pinned* model, while one that has been implicitly cached is known as an *unpinned* model. Unpinned models are restricted by a *cache size* parameter, which sets how many models can be in the mining cache. The number of pinned models in the cache can exceed this maximum but only after replacing any existing unpinned models in the cache. Pinned models can also appear more than once in the cache, which can improve performance in an environment with many concurrent scoring users and multiple processors.

Figure 9-42 depicts the process of inspecting a model cache (1), adding new models to the cache (2, 3), and the resizing the cache (4). Increasing the cache size will have an impact on the memory footprint used for the mining database. The default cache size of 1 should only be increased to the number of models that are actively used in your query workload, if there are many single-row scoring statements.

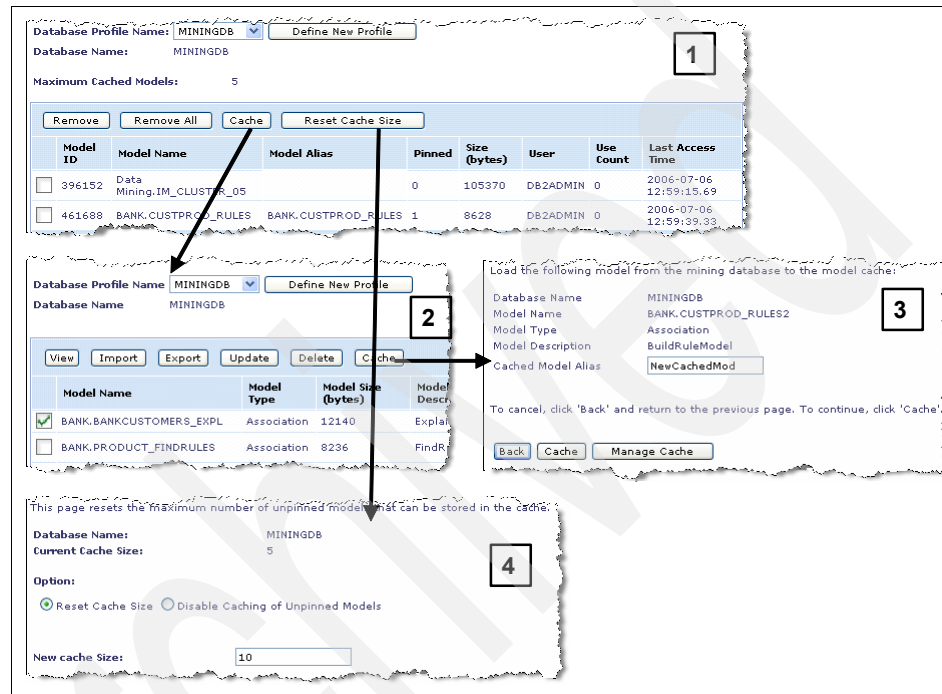


Figure 9-42 Caching a mining model and managing the mining cache

The cache functions in Figure 9-42 are contained in the DWE Mining → Model Cache Management section of the Admin Console, which links to DWE Mining → Model Management to add models to the cache.

9.6 Managing DB2 Alphablox within DWE

DB2 Alphablox is the analytical component of a DWE data warehouse application and can be accessed and managed from the Admin Console. Alphablox builds on the other components of DWE to provide the interface through which a business accesses its enterprise data. Alphablox uses the DWE OLAP component of DWE for multidimensional analysis since a database needs to be enabled for OLAP and populated with OLAP metadata before it can be used to

create an Alphablox cube. The DWE OLAP component can in turn rely on the DWE SQL Warehousing component to maintain the tables used by the OLAP metadata. If there is any embedded mining within an Alphablox application then this will be deployed and managed through the DWE Administration Console. Once the underlying infrastructure has been created the access to the data through Alphablox allows the development and subsequent support of information dashboards and interactive OLAP data analysis.

The building of such analytical applications and full coverage of DB2 Alphablox is discussed in Chapter 8, “InLine Analytic Applications” on page 315. The aim of this section is to present a summary of the key aspects of Alphablox so that a system administrator should have enough knowledge of the functionality to understand how to administer an Alphablox environment.

9.6.1 Defining DB2 Alphablox

DB2 Alphablox provides the ability to rapidly create customized analytic components that are embedded into existing business processes and Web applications. Applications that you build with DB2 Alphablox run in standard Web browsers from where users can perform real-time, highly customizable multidimensional analysis. Alphablox differs from traditional query and reporting tools, as the customized analytical applications that are created are tailored to the unique needs of business users, which can make them more intuitive.

DB2 Alphablox can access and interact directly with DB2 and can also connect to the OLAP metadata that has been defined in the DWE OLAP environment. From these relational sources Alphablox can create structured reports and provide a variety of charts to graphically display the data. These structured reports are flexible enough to allow users to interactively display the exact view of their relational data that they want by using functions such as filters and drill-down. All of this functionality can be packaged in a single application, which can access multiple data sources while providing an intuitive user interface to make this analysis easier.

Alphablox architecture

The DB2 Alphablox server and any Alphablox applications run as J2EE-compliant applications within WebSphere Application Server. This allows both administration and reporting functionality to be via a Web browser. Figure 9-43 shows a typical Alphablox deployment architecture with the separate presentation, data sources, and application services layers.

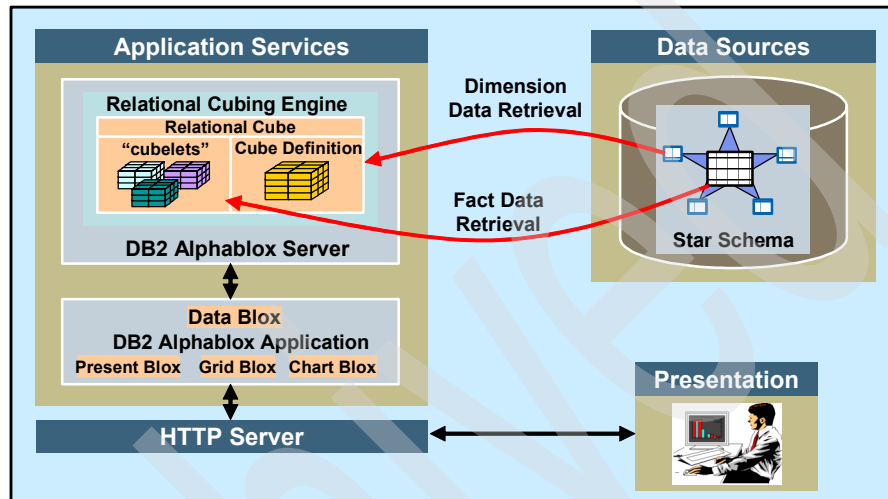


Figure 9-43 Alphablox runtime architecture including the relational cubing engine

The application services layer is the *DB2 Alphablox Server*, which is comprised of a number of components, including The *relational cubing engine*.

DB2 Alphablox Server

The DB2 Alphablox Server is installed in WebSphere as an enterprise application (EAR) that contains a server module and an admin module. The server is comprised of several discrete services that run independently and are managed by the Alphablox Administration Console. One of the key services is the *relational cubing engine*. Others include the server manager, which controls the requests for application pages, and the session manager, which controls and monitors users' interactions with DB2 Alphablox applications. These and other components are discussed in more detail in 8.1.4 "Services" on page 332.

Relational cubing engine

The Alphablox Relational Cubing Engine, also referred to as the Alphablox Cube Server, is a high-performance, scalable cubing engine, designed to support many users querying many different OLAP cubes. The cubing engine is designed to provide a multi-dimensional view of data sources in a relational database. The cubing engine uses the DWE OLAP component as the source for its

multidimensional metadata rather than having to create it from base relational tables.

Another key feature of the relational cubing engine is the ability to cache elements of the cube. When an Alphablox Cube is started, as shown in Figure 9-43 on page 457, the cube metadata is loaded from the OLAP tables in DB2, and the dimensional data is loaded from the DB2 tables to memory. Furthermore, during runtime the cubing engine stores results in memory in the form of *cubelets*. These cubelets can be used to provide instant results to users requesting previously viewed data, with any new data being seamlessly retrieved from the database.

DB2 Alphablox application

A DB2 Alphablox application is J2EE applications, which means that it can contain any components that are available to J2EE applications. There are, however, generally two common components in an Alphablox application, JavaServer™ Pages™ (JSPs) and Blox components.

- ▶ **JavaServer Pages (JSPs)**

JSP technology is the mechanism by which an application developer typically provides DB2 Alphablox functionality to an application. Interactive Web pages are developed using JSP, Blox custom tag libraries, HTML, and any Java or JavaScript code as needed for any particular applications. JSP pages are compiled at runtime on the application server. DB2 Alphablox processes any DB2 Alphablox requests and then the Web server returns dynamic content to the user.

- ▶ **Blox components**

Blox components are DB2 Alphablox components that provide functionality such as accessing data sources and displaying data in grids and charts. You add Blox components to a JSP page using the Blox custom tag libraries. The Blox tag libraries allow you to control many aspects of the Blox. For example, you can change an attribute of ChartBlox to specify the type of chart displayed. Blox components can also be manipulated programmatically, using either Java or JavaScript.

Notwithstanding how the Alphablox application is created, any users with a Web browser can access the application without the need for any plug-ins. This is due to Alphablox using standard Dynamic HTML technology, which utilizes JavaScript and Cascading Style Sheet (CSS). This supports a full range of data analysis functionality with a highly usable and customizable graphical user interface.

9.6.2 DB2 Alphablox Administration Console

Now that Alphablox has been introduced, and its key features highlighted, the use of the Alphablox Administration Console to manage the Alphablox environment can be shown. The Alphablox Administration Console is a J2EE application that can be used to develop, deploy, and manage applications. The Admin Console can be used to access the Alphablox Console, depicted in Figure 9-44, by selecting **DWE Alphablox** → **Alphablox Administration**.

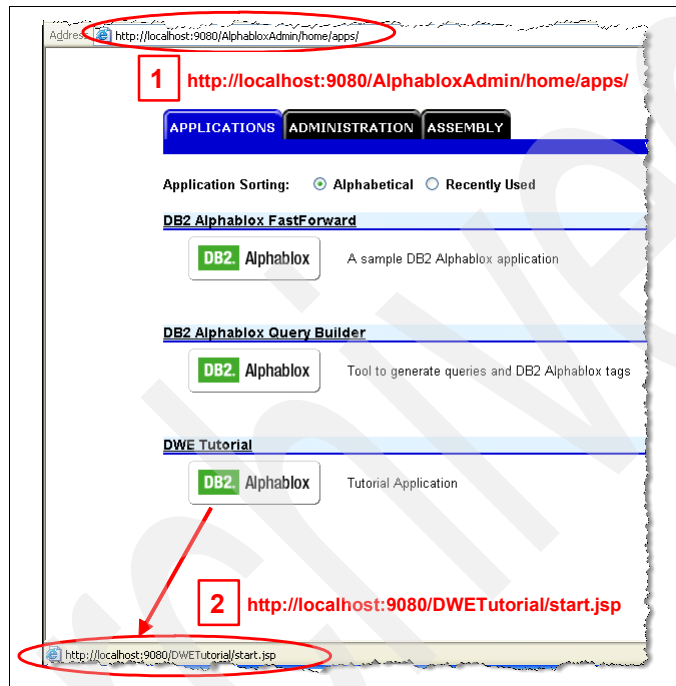


Figure 9-44 Alphablox Administration Console

The Alphablox Administration Console has three main sections: applications, administration, and assembly, and are summarized below. Details can be accessed by the context sensitive help, such as that depicted in Figure 9-45.

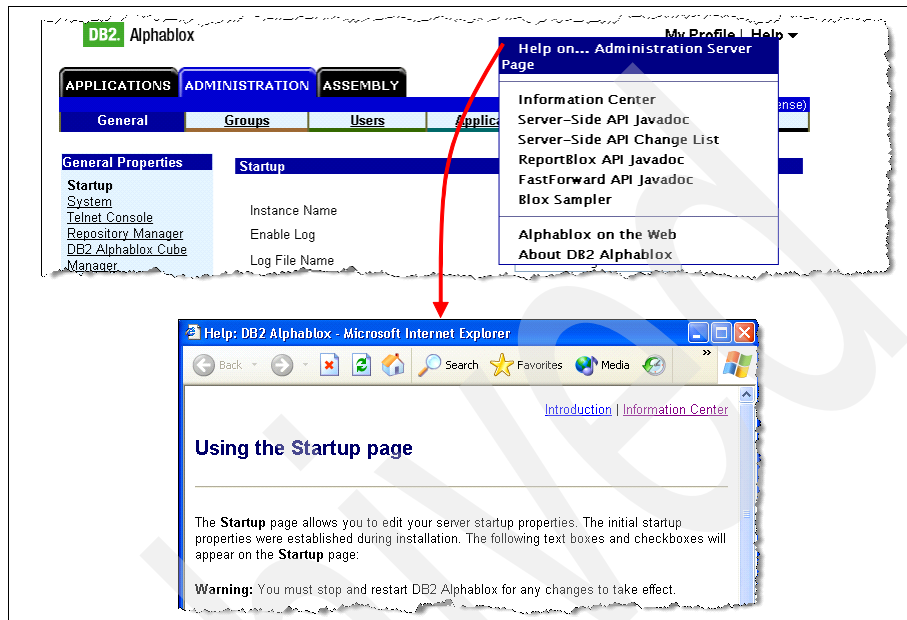


Figure 9-45 Alphablox help

The Applications section, as shown in Figure 9-44 on page 459, is the default section of the console and provides a link to each of the Alphablox applications. The location of the console within WAS is indicated by (1) while the location of each deployed Alphablox application is shown by (2) when the application is highlighted. An application can be launched by selecting it and, depending on your security settings, can then be altered or viewed.

The Administration section, pictured in Figure 9-46, groups together various components that administer different facets of an Alphablox server.

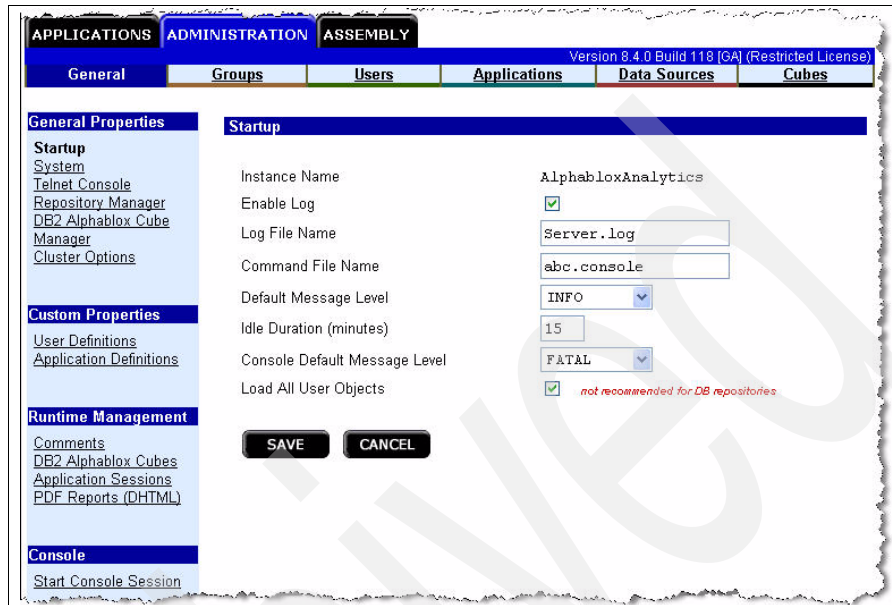


Figure 9-46 Administration of an Alphablox environment

These components of the environment are:

- ▶ **General:** This page contains the following sections:
 - **General Properties:** In this section are server tasks, and the Alphablox repository that is used to keep track of applications, users, groups, bookmarks, and other information. This repository can reside either in the operating system file system or in a relational database. An administrator can also specify the maximum number of Alphablox cubes and where on disk the dimension data for the Alphablox cube is cached.
 - **Custom Properties:** Can be defined for both applications and users.
 - **Runtime Management:** Allows the user to start and stop Alphablox data cubes and also view which Alphablox applications are currently being used.
 - **Console:** DB2 Alphablox can be administered through its Console or from Web pages under the Administration tab of the DB2 Alphablox home page. Most administrative activities (for example, creating and editing users and groups) that are configured through the DB2 Alphablox home page user interface can also be configured through console commands.

- ▶ **Groups:** The Groups page manages the allocation of application permissions to groups of users and the membership of these groups. An Alphablox group can be comprised of users or other groups.
- ▶ **User:** The user page allows an administrator to create a user profile for each user that will log into the Alphablox application. The user profile defines the user name, password, what Alphablox groups the user belongs to and whether there are any user properties specific to an Alphablox application.
- ▶ **Applications:** The Application page is where an administrator manages the DB2 Alphablox applications. Before a user can access an application from the Applications tab, an administrator must define the application using this section. Defining an application does not create the application, it creates a reference to an existing application and defines its properties. An administrator can create application definitions, edit existing definitions, and delete definitions.

A common way of creating an application definition is to import an existing J2EE application and define it as an DB2 Alphablox application. After importing a J2EE application to DB2 Alphablox, an application has access to all of the DB2 Alphablox application services such as Blox rendering.

- ▶ **Data Sources:** The Data Sources page is where an administrator manages each data source definition used by an application. An administrator can create data source definitions, edit existing definitions, and delete definitions. Alphablox can define two types of data sources, multidimensional and relational. DB2 Alphablox uses data adapters to connect to multidimensional data sources and drivers to connect to relational data sources.
- ▶ **Cubes:** The Cube page allows administrators to create a multidimensional representation of data that resides in a relational database. A DB2 Alphablox cube is a data model often used in online analytical processing (OLAP) to represent business data that is typically analyzed over multiple dimensions. Within the Cubes page you can create cube definitions, edit existing definitions, and delete definitions.

The Assembly section, pictured in Figure 9-47, provides access to the Alphablox Application Studio, which has a collection of examples and query-building utilities that are designed to accelerate your application development.

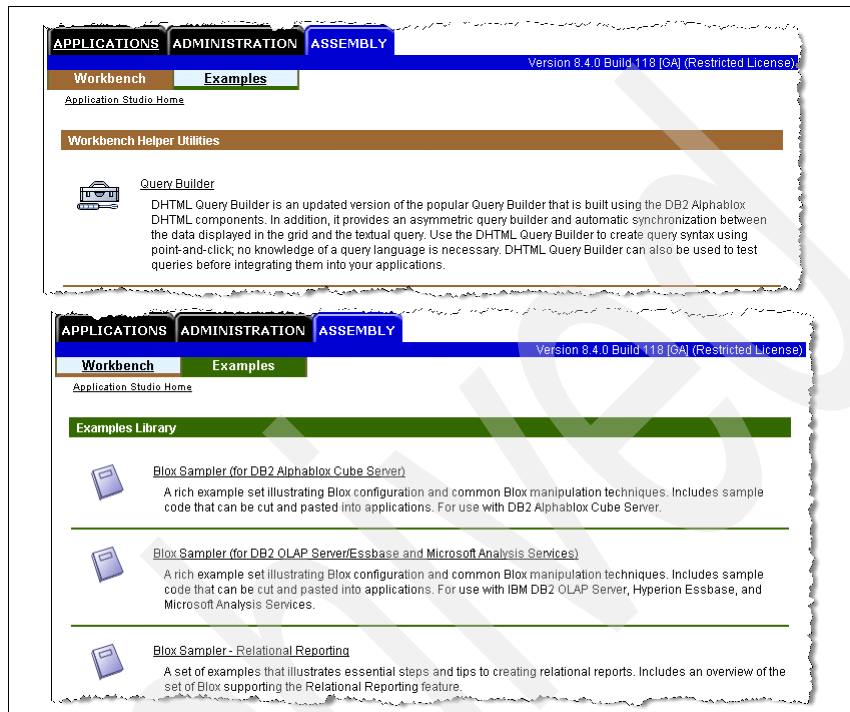


Figure 9-47 Creating applications from the Alphablox console

The Workbench, or Query Builder, uses the Alphablox DHTML client technology to generate queries, and it allows these queries to be created using point-and-click with no knowledge of a query language being necessary. This DHTML Query Builder can also be used to test queries before integrating them into your applications. The examples are an assortment of Blox designed to report against Alphablox Cube Servers such as DWE OLAP Metadata, OLAP cubing engines such as Hyperion Essbase, and reporting against relational databases.



DB2 UDB Data Warehouse features

DB2 Data Warehouse Edition is built on DB2 UDB Enterprise Server Edition (ESE). In this chapter we describe some of the key data warehouse features within DB2 UDB that should be taken into consideration as part of any large-scale data warehouse implementation. New capabilities for data warehousing introduced in DB2 V9 are also included.

10.1 DB2 UDB ESE

DB2 Enterprise Server Edition (DB2 ESE) is a full-function premier Web-enabled client/server RDBMS. It is available on all supported UNIX operating environments, including AIX, Solaris™, and HP-UX, as well as Linux and Microsoft® Windows.

DB2 ESE is positioned for large and mid-sized departmental servers. It has the ability to partition data within a single server or across multiple servers with the Database Partitioning Feature (DPF). This capability must be properly licensed by purchasing an optional Database Partitioning Feature license or be licensed with the DB2 Data Warehouse Enterprise Edition.

10.2 Architecture

DB2 has a shared-nothing architecture, as depicted in Figure 10-1 on page 467, as opposed to a shared-disk or shared-memory architecture, which is used by some other commercial database products. The advantage of the shared-nothing architecture is the ability to spread all the data across multiple servers in a near linear fashion. This architecture has been proven as the best way to spread a query workload evenly over all available system hardware (disks, processors, switches, and memory).

It supports environments in the following fashion:

- ▶ Uniprocessor: A single processor accessing individual memory and disk storage
- ▶ Symmetric Multiprocessor (SMP): Multiple processors accessing the same memory and all the available disk storage
- ▶ Cluster: Distributed memory processors capable of accessing all the available disk storage
- ▶ Massively Parallel Processor (MPP): Multiple loosely coupled processors linked by a high-speed interconnect

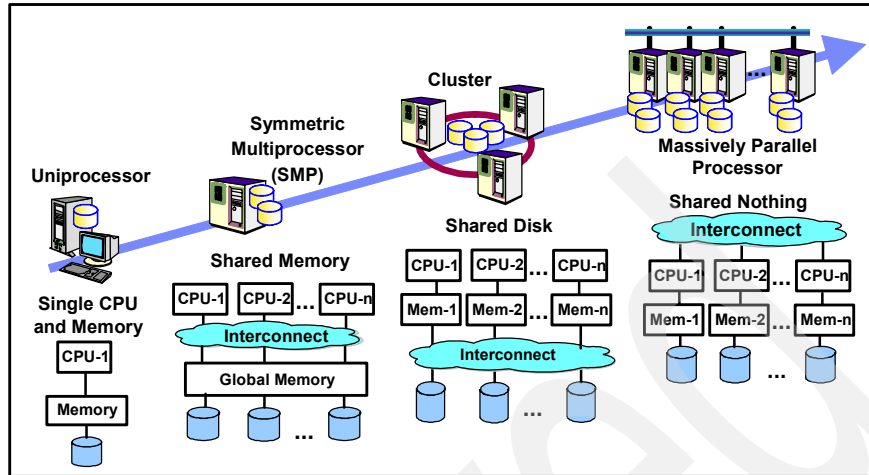


Figure 10-1 Software and hardware architectures

DB2 runs on a variety of hardware topologies including a large number of clustered small or large SMP servers. For example, clustered SMPs could be comprised of dozens of 4-CPU systems, 200 2-CPU boxes, or four 24-CPU SMPs. In order for DB2 to run in a shared-nothing architecture, a high-speed interconnect between servers is required. The interconnect connects the various servers in the cluster such that a coordinated query can be effectively executed over a shared-nothing architecture. Switches connecting the cluster are also diverse, and DB2 has supported the Intel® Virtual Interface Architecture and the Sun Cluster Interface since 1998. NUMA-Q® Fabric support was added in 1999, and Logic Infiniband in 2001. The DB2 shared-nothing implementation on UNIX and Windows has been proven to scale to well over 50 servers on audited TPC-H, TPC-C, and TPC-D benchmarks, and over two hundred servers in client environments.

Clustering for high availability is a task that can be done by the operating system for any data service, such as a database. On the other end of the spectrum, clustering for scalability requires intelligence in the database engine. Choosing shared-nothing for scaling on UNIX systems, or Intel SMPs clustered with Windows or Linux, offers two advantages:

- ▶ **Linear scaling:** Shared-nothing is modular and adaptive. As the system grows, it is easy to add more hardware to the cluster. Data redistribution can be done incrementally if it has to fit within any maintenance windows, or as a background task.
- ▶ **High availability (HA):** HA is provided with redundant machines, even commodity hardware, for very high availability. Workload ownership

(including an IP address) and data ownership can move quickly from one machine to another or from one machine to many to ensure workload balancing during a hardware failure. Ownership of the data does not have to be arbitrated by a separate software layer, because ownership is associated with the IP address of the node.

10.3 Balanced Configuration Unit

IBM has years of experience building large data warehousing infrastructures. The Balanced Configuration Unit (BCU) is a proven approach for building successful data warehouse infrastructures and reducing the time required to get them installed, configured, and running. The BCU uses Industry-standard components for ease of installation and implementation, balanced performance for delivering scalability, and fault tolerance for high availability. The end result is a solution that provides faster time to market, lower total cost of ownership, and high reliability. The BCU can be thought of as an appliance-like (complete and packaged) solution, which uses IBM commodity components and is delivered as a database-ready solution.

The BCU is a complete solution for building a data warehousing infrastructure using a modular, building-block design with a focus on balanced performance throughout the hardware and software configuration. The BCU is a combination of processors, memory, I/O, storage, DB2 database partitions, and DB2 configuration parameters under a single operating system representing a scalable ratio of disk I/O to memory to CPU to network, as shown in Figure 10-2.

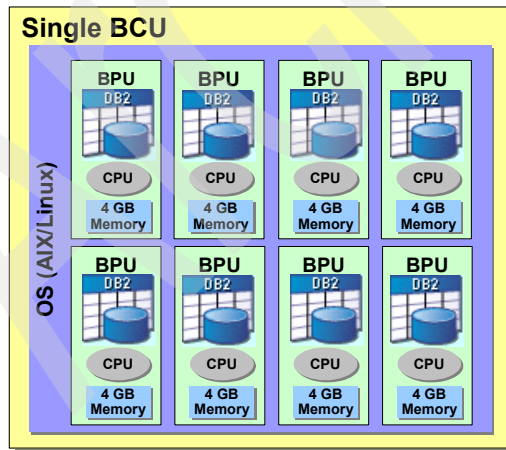


Figure 10-2 Single Balanced Configuration Unit

The building blocks start with a virtual entity, the Balanced Partition Unit (BPU). The BPU is one DB2 partition and its related components. Components refer to the processes and hardware resources that service each individual partition (BPU) and consist of a DB2 partition, CPU, memory, and disk.

Balanced collaboration is the key to success, whereby all of the components are chosen to fit with each other for a mixture of compatibility, performance, and reliability reasons. The idea of the BCU is to choose a combination of processors, memory, I/O storage, DB2 partitions, and a single operating system, which make up a single building block that is scalable. Larger systems are then built by combining numerous BCU building blocks, as shown in Figure 10-3.

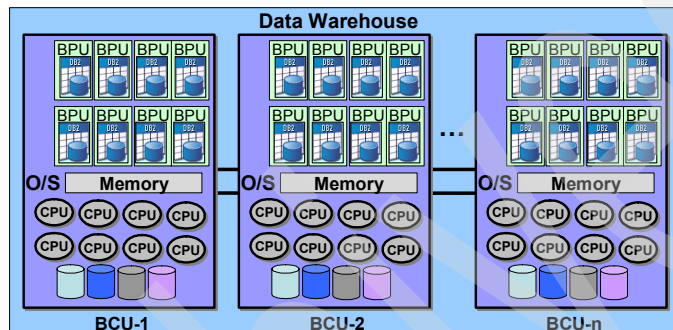


Figure 10-3 BCU building blocks

A large data warehouse utilizes the shared-nothing DB2 architecture and consists of a number of virtually identical scalable BCU building blocks. These blocks are combined into a single DB2 system image configuration to support the required data and query volumes.

The BCU is composed of specific software levels and hardware that IBM has integrated and tested as a pre-configured building block for data warehousing systems. The BCU reduces the complexity, cost, and risk of designing, implementing, growing, and maintaining a data warehouse and BI infrastructure.

Attention: More information about the BCU solution and methodology can be obtained by engaging the IBM BI Best Practices team. Contact the team via your local IBM representative for current BCU recommendations.

10.4 Partitioning

Partitioning a database is often done to provide increased scalability, improve query performance, or ease database maintenance. Partitioning typically

involves logically or physically dividing up a table into smaller units to increase database, CPU, and I/O parallelism. DB2 offers several methods to partition a database. One or all methods can be used for data warehousing. In all cases, the partitioning scheme is handled through the data definition language (DDL).

10.4.1 Hash partitioning

DB2 UDB with DPF uses hash partitioning to efficiently support very large databases and exploit large hardware environments. Database partitioning refers to splitting a database into separate physical partitions, which can be distributed across or within SMP and clustered servers, while still appearing to the end-user and database administrator as a single-image database.

DB2 UDB with DPF divides the database into partitions. By searching these database partitions in parallel, elapsed times for queries can be dramatically reduced. Data is distributed across multiple database partitions, thereby enabling database scaling into the terabytes. A crucial factor in attaining the performance advantages of a parallel environment is being able to intelligently distribute the data across the various database partitions. DB2 UDB with DPF supports intelligent database partitioning in that it evenly distributes and remembers the location of each data row.

SQL operations are performed in parallel across all partitions and therefore operate much faster. DB2 UDB with DPF provides *parallel everything* such that all transactions (selects, updates, deletes, inserts), as well as utility operations, all operate in parallel across all partitions.

The architecture design divides and executes the workload to enhance scalability through enabling partitioned workloads to have their own private data and computing resources. In this way, near-linear scalability is achieved as resources across data partitions are not shared, and there is no central locking of resources.

A database partition is a part of a database that consists of its own data, indexes, configuration files, and transaction logs. A database partition is sometimes called a node or a database node. A single-partition database is a database having only one database partition. All data in the database is stored in that partition. In this case, database partition groups, while present, provide no additional capability.

A partitioned database is a database with two or more database partitions. Tables can be located in one or more database partitions. When a table is in a database partition group consisting of multiple partitions, some of its rows are stored in one partition, and other rows are stored in other partitions. Usually, a single database partition exists on each physical node, and the processors on

each system are used by the database manager at each database partition to manage its part of the total data in the database.

Because data is divided across database partitions, you can use the power of multiple processors on multiple physical nodes to satisfy requests for information. Data retrieval and update requests are decomposed automatically into sub-requests, and executed in parallel among the applicable database partitions. The fact that databases are split across database partitions is transparent to users issuing SQL statements.

User interaction occurs through one database partition, known as the coordinator node for that user. The coordinator runs on the same database partition as the application, or in the case of a remote application the database partition to which that application is connected. Any database partition can be used as a coordinator node.

DB2 supports a partitioned storage model that allows you to store data across several database partitions in the database. This means that the data is physically stored across more than one database partition, and yet can be accessed as though it were located in the same place. Applications and users accessing data in a partitioned database do not need to be aware of the physical location of the data.

Partition groups

A partition group is a logical layer that allows the grouping of one or more partitions to perform operations on all the partitions in the group. A database partition can belong to more than one partition group. When a database is created, DB2 will create three default partition groups that cannot be dropped:

- ▶ **IBMDEFAULTGROUP:** This is the default partition group for any table you create. It consists of all database partitions as defined in the `db2nodes.cfg` file. This partition group cannot be modified. Tablespace `USERSPACE1` is created in this partition group.
- ▶ **IBMTEMPGROUP:** This partition group is used by all system temporary tables. It also consists of all database partitions as defined in the `db2nodes.cfg` file. Tablespace `TEMPSPACE1` is created in this partition.
- ▶ **IBMCATGROUP:** This partition group contains the catalog tables (tablespace `SYSCATSPACE`), and thus it only includes the database catalog partition. This partition group cannot be modified.

To create new database partition groups you can use the `create database partition group` statement. This statement will create the database partition group within the database, assign database partitions that you specified to the partition group, and then record the partition group definition in the database system catalog tables.

Example 10-1 shows an example of creating a partition group pgrpall on all partitions specified in the db2nodes.cfg file.

Example 10-1 Partition group pgrpall

```
create database partition group pgrpall on all dbpartitionnums
```

To create a database partition group pg0123 consisting of partitions 0,1, 2, and 3, issue the command in Example 10-2.

Example 10-2 Partition group pg0123

```
create database partition group pg0123 on dbpartitionnums (0,1,2,3)
```

Tablespaces in a DPF environment

A tablespace can be created in specific partitions by associating it to a given partition group. The CREATE TABLESPACE statement with the IN DATABASE PARTITION GROUP clause can be used for this purpose. This allows users to have flexibility as to which partitions will actually be storing their tables. Example 10-3 shows a tablespace mytbls, which spans partitions 0, 1, 2, and 3.

Example 10-3 Tablespaces

```
create regular tablespace mytbls in database partition group pg0123  
managed by automatic storage
```

Partitioning keys

A partitioning key is a column (or group of columns) that is used to determine the partition in which a particular row of data is stored. A partitioning key is defined on a table using the CREATE TABLE statement, as depicted in Example 10-4.

Example 10-4 Partitioning key

```
create table mytable (col1 int, col2 int, col3 int, col4 char(10))  
in mytbls1  
partitioning key (col1, col2, col3)
```

Partition maps

When a database partition group is created or modified, a partitioning map is automatically created by DB2. A partitioning map, in conjunction with a partitioning key and a hashing algorithm, is used by DB2 to determine which database partition in the database partition group will store a given row of data.

Where a new row is inserted is determined by hashing the row's partitioning key values to an entry in the partitioning map, which contains the partition number to use, as shown in Figure 10-4.

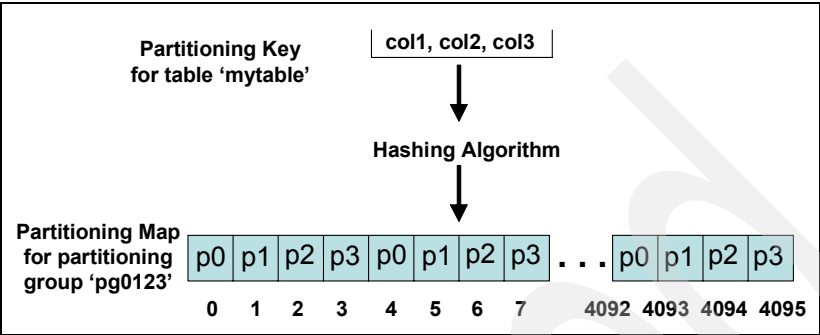


Figure 10-4 Hash partitioning map

Partition group pg0123 has been defined on partitions 0, 1, 2, and 3. An associated partitioning map is automatically created, that is, an array with 4096 entries containing the values 0, 1, 2, 3, 0, 1, 2, 3.... (Partition numbers are stored in round-robin fashion by default, although this can be changed.) Table mytable has been created with a partitioning key consisting of columns col1, col2, and col3. For each row, the partitioning key column values are passed to the hashing algorithm, which will return an output number from 0 to 4095. This number corresponds to one of the entries in the partitioning map array that contains the value of the partition number where the row is to be stored. If the hashing algorithm had returned an output value of 7, then the row would have been stored in partition p3.

The partitioning map is an internally generated array containing either 4,096 entries for multiple-partition database partition groups, or a single entry for single-partition database partition groups. The partition numbers of the database partition group are specified in a round-robin fashion.

This hash partitioning methodology can be viewed also as a load-balancing tool. By changing the partition key and partition map, the number of rows in any database partition can be adjusted. In the event that there is data skew after the initial distribution, DB2 UDB utilities can automatically analyze and correct the skew. DB2 UDB with DPF can automatically create a new partition map to evenly distribute the currently skewed data by changing the partition assignments. Affected data rows are automatically moved to their newly assigned database partition while the system is online. More data can be placed in partitions that have faster processing ability, thereby enabling load balancing on nodes of different processing power in a non-uniform cluster configuration. The DB2

partition map is used to assign proportionally more data to those partitions with more processing power.

Incremental database growth is supported by adding additional partitions to the database (along with more processing power), changing the partition map to include these new database partitions, and automatically redistributing the data. The DB2 UDB with DPF utilities provide this capability, allowing for a well-defined growth path.

DB2 UDB is aware of the partitioning scheme in the DDL, data manipulation SQL, and at runtime. The DB2 UDB Optimizer takes advantage of partitioning knowledge to evaluate the costs of different operations and thus choose the optimal execution strategy for a given SQL statement.

Choosing partitioning keys

Given a particular DPF workload and configuration, the biggest single performance factor is the choice of partitioning keys for the tables. However, there may not be one absolute and obvious best set of them, as one set may optimize some queries and another set others, so you must look at the overall workload and know what is most important to optimize. You may also have to try various alternatives before getting the best results.

Here are some quick tips for partitioning key selection. For more assistance refer to the IBM Redbook *DB2 UDB ESE: Partitioning for Performance in an e-Business Intelligence World*, SG24-6917.

Do:

- ▶ Include frequently used join columns.
- ▶ Spread data evenly across partitions.
- ▶ Have a broad range domain.
- ▶ Use integer, which is more efficient than character, which is more efficient than decimal.
- ▶ Use the DB2 Design Advisor.

Do not:

- ▶ Use LOB or LONG fields.
- ▶ Have a unique index or primary key unless it is a super-set of the PK.
- ▶ Allow ALTERations of the PK.
- ▶ Update columns.

Important: It is very important to understand that DPF partitioning is based on hashing, with the intent to spread rows evenly over partitions and the available hardware resources to maximize performance. It is not wise to choose partitioning keys in an attempt to implement range partitioning.

Here are some other considerations involved in choosing partitioning keys:

- ▶ Once a table is created as partitioned, you cannot directly change its partitioning key, but you could use one of these approaches:
 - Unload the table to a file, drop and re-create the table with the new partitioning key, and reload the table.
 - Create a new version of the table with a temporary name and the new partitioning key, load the new table from the old one (the fastest way is: "declare mycursor for select * from t1", followed by "load from mycursor of cursor replace into t1_new"), drop the old table, rename the new table to the real name, and recreate indexes and re-do other steps as when the table was originally created.
- ▶ With ALTER TABLE you can add or drop a partitioning key, but only for a non-partitioned table.
- ▶ The columns in any unique or primary key constraint defined on the table must be a super-set of the partitioning key.
- ▶ Avoid unbalanced distribution of rows across partitions by choosing partitioning key columns with high cardinality. If this cannot be achieved, try to use columns with uniformly distributed values. After a table is populated you can check how many of its rows are in each partition, and how many of its rows map to each entry in the partitioning map (for possible redistribution purposes) by running queries like those in the DBPARTITIONNUM example and the HASHEDVALUE example. Unless you have extremely large tables, differences of a few percent in cardinalities per partition can be ignored.
- ▶ Partitioning keys should not include columns that are updated frequently. Whenever a partitioning key value is updated, DB2 may need to drop the row and re-insert it into a different partition, as determined by hashing the new partitioning key value.
- ▶ Unless a table is not critical or you have no idea what a good partitioning key choice would be, you should not let the partitioning key be chosen by default. For the record, the default partitioning key is the first column of the primary key, and if there is none, the first column that has an eligible data type.
- ▶ Make sure that you understand collocation and the different join types. For information, see the *DB2 Administration Guide: Performance*, SC10-4222, topics "Join strategies in partitioned databases" and "Join methods in partitioned databases."

- ▶ Here are the requirements for collocation. Collocated tables must:
 - Be in the same database partition group, one that is not being redistributed. (During redistribution, tables in the database partition group may be using different partitioning maps. They are not collocated.)
 - Have partitioning keys with the same number of columns.
 - Have the corresponding columns of the partitioning key be partition compatible.
 - Be in a single partition database partition group, if not in the same database partition group, defined on the same partition.
- ▶ Partition compatibility is defined between the base data types of corresponding columns of partitioning keys. Partition-compatible data types have the property that two variables, one of each type, with the same value, are mapped to the same partitioning map index by the same partitioning function. Partition compatibility has the following characteristics:
 - Internal formats are used for DATE, TIME, and TIMESTAMP. They are not compatible with each other, and none is compatible with CHAR or VARCHAR.
 - Partition compatibility is not affected by columns with NOT NULL or FOR BIT DATA definitions.
 - NULL values of compatible data types are treated identically. Different results might be produced for NULL values of non-compatible data types.
 - The base data type of a UDT is used to analyze partition compatibility.
 - Decimals of the same value in the partitioning key are treated identically, even if their scale and precision differ.
 - Trailing blanks in character strings (CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC) are ignored by the system-provided hashing function.
 - CHAR or VARCHAR of different lengths are compatible data types.
 - REAL or DOUBLE values that are equal are treated identically even though their precision differs.

10.4.2 Multidimensional data clustering (MDC)

Multidimensional data clustering is an innovative method of organizing and storing data along multiple dimensions. Organizing data in this manner can greatly improve certain query types and reduce reorg and index maintenance.

The MDC feature, introduced with DB2 UDB Version 8, provides a method for automatic clustering of data along multiple dimensions. Prior to Version 8, DB2 UDB only supported single-dimensional clustering of data, through clustering

indexes. Using a clustering index, DB2 maintains the physical order of data on pages in the key order of the index, as records are inserted and updated in the table. Clustering indexes greatly improves the performance of range queries that have predicates containing one or more keys of the clustering index. With good clustering, only a portion of the table needs to be accessed. In addition, when the pages are sequential, more efficient prefetching can be performed. Secondary indexes can be created to access the tables when the primary key index is not applicable. Unfortunately, secondary indexes perform many random I/O accesses against the table for a simple operation, such as a range query.

The MDC feature addresses this important deficiency in database systems. MDC enables a table to be physically clustered on more than one key (or dimension) simultaneously. MDC tables are organized physically by associating records with similar values for the dimension attributes in an extent (or block). DB2 maintains this physical layout efficiently and provides methods of processing database operations that provide significant performance improvements.

Figure 10-5 presents a conceptual diagram of multi-dimensional clustering along three dimensions: region, year, and color.

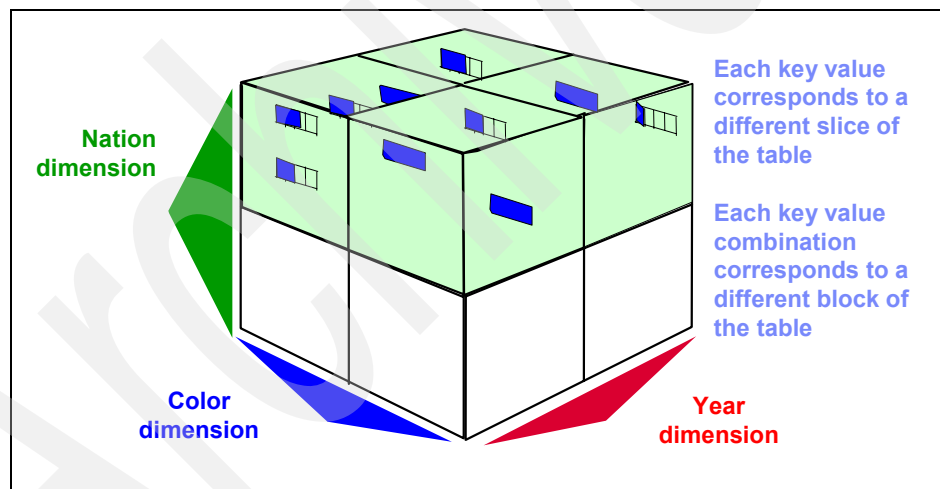


Figure 10-5 MDC conceptual diagram

Each block contains only rows that have the same unique combination of dimension values. The set of blocks that have the same unique combination of dimension values is called a cell. A cell may consist of one or more blocks in the MDC table. As depicted in Figure 10-5, a cell can be described as a combination of unique values of year, nation, and color, such as 1997, Canada, and Blue. All records that have 1997 as year, Canada as nation, and blue as color will be stored in the same extents of that table.

With MDC tables, clustering is guaranteed. If an existing block satisfies the unique combination of dimension values, the row is inserted into that block, assuming there is sufficient space. If there is insufficient space in the existing blocks, or if no block exists with the unique combination of dimension values, a new block is created.

Example 10-5 shows the DDL for creating an MDC table. The ORGANIZE keyword is used to define an MDC table.

Example 10-5 MDC example

```
CREATE TABLE sales (  
    cust_id INTEGER,  
    year CHAR(4),  
    nation CHAR(15),  
    color CHAR(12),  
    amount INTEGER)  
PARTITIONING KEY(cust_id)  
ORGANIZE BY (nation,color,year)
```

MDC introduced a new type of index, called a block index. When you create an MDC table, the following two kinds of block indexes are created automatically:

- ▶ A dimension block index per dimension, which contains pointers to each occupied block for that dimension.
- ▶ A composite block index, which contains all columns involved in all dimensions specified for the table, as depicted in Figure 10-6. The composite block index is used to maintain clustering during insert and update activities. It can also be used for query processing.

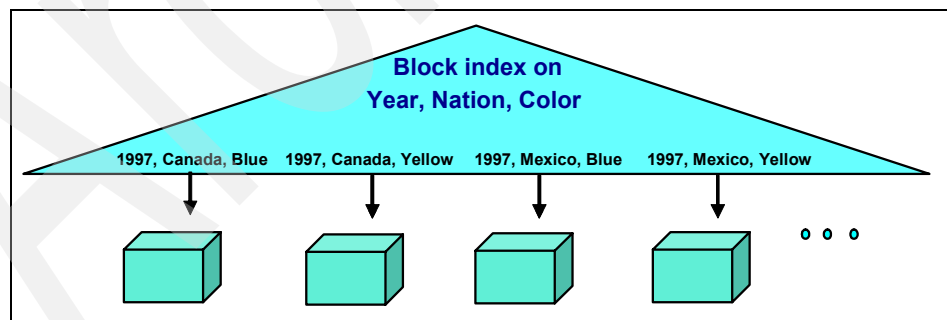


Figure 10-6 Block index

With MDC, data is organized on disk based on dimensions. Queries can then skip parts of the tablespace that the optimizer has determined do not apply. When data is inserted, it is automatically put in the proper place so you no longer

need to reorganize the data. In addition, because one index entry represents the entire data page (versus having one index entry per row with traditional indexes), MDCs reduce the overall size of the space required for the indexes. This in turn reduces disk requirements and produces faster queries because of the reduced amount of I/O needed for a query. MDCs also improve delete performance because DB2 now only has to drop a few data pages. Inserts are also faster because DB2 rarely has to update an index page, only the data page.

DB2 manages MDC tables by blocks according to dimensions instead of row IDs (RIDs) as implemented in clustering indexes. This is depicted in Figure 10-7.

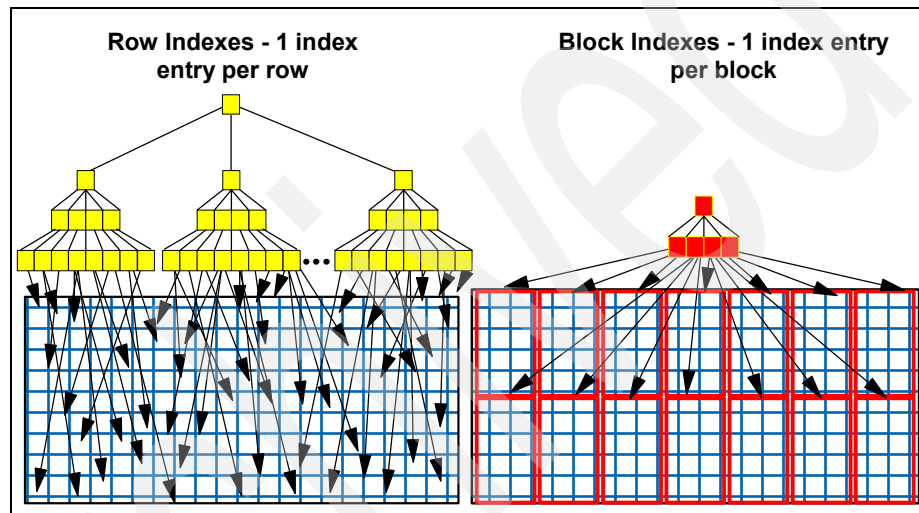


Figure 10-7 MDC page and block retrieval comparison

RID indexes require one RID per data record. Block indexes (BIDs) only have one index entry pointing to the block. Consequently, the size of the BID index is dramatically smaller than the RID index. This means the probability that a BID index page needed is in memory cache when needed is much higher than with RID indexes. You can have multiple block indexes on the same fact table since these indexes are small, so there is very little disk space required to have many indexes.

DB2 has the ability to perform fast rollouts of data from within MDCs. For a rollout deletion, the deleted records are not logged. Instead, the pages that contain the records are made to look empty by reformatting parts of the pages. The changes to the reformatted parts are logged, but the records themselves are not logged. There is no change in the logging of index updates, so the performance improvement depends on how many RID indexes there are. The fewer RID

indexes, the better the improvement is as a percentage of the total time and log space.

Multidimensional clustering is a unique capability that is targeted for large database environments, providing an elegant method for flexible, continuous, and automatic clustering of data along multiple dimensions. The result is significant improvement in the performance of queries, as well as a significant reduction in the overhead of data maintenance operations (such as reorganization) and index maintenance operations during insert, update, and delete operations.

Performance considerations

The performance of an MDC table is greatly dependent upon the proper choice of dimensions and the block (extent) size of the tablespace for the given data and application workload.

A poor choice of dimensions and extent size can, as would be expected, result in low disk storage utilization and poor query access and load utility performance.

Choosing dimensions

The first step is to identify the queries in the existing or planned workloads that can benefit from multidimensional clustering. For existing applications, the workload may be captured from DB2 Query Patroller or the dynamic SQL snapshot and the SQL statement Event Monitor. This workload can be used as input into the DB2 Design Advisor to analyze the workload and recommend dimensions for the MDC tables.

For a new table or database, you will need a good understanding of the expected workload. The following are typical dimension candidates:

- ▶ Columns in range or equality or IN-list predicates
- ▶ Columns with coarse granularity
- ▶ Roll-in and roll-out of data
- ▶ Columns referenced in GROUP BY and ORDER BY clauses
- ▶ Foreign key columns or join clauses in fact table of a star schema database
- ▶ Combinations of the above

Usually there will be several candidates. You should rank them and evaluate the impacts on the workload and cell density. When there are no good low or medium cardinality dimensions, or the chosen dimension has high cardinality, consider using generated columns. The generated expressions used should be monotonic (increasing/decreasing) for those columns used in the range queries.

Choosing the extent size

Extent size is related to the concept of cell density, which is the percentage of space occupied by rows in a cell. Since an extent only contains rows with the same unique combination of dimension values, significant disk space could be wasted if dimension cardinalities are very high, for example, when there is a dimension with unique values that would result in an extent per row.

The ideal MDC table is the one where every cell has just enough rows to exactly fill one extent, but this can be difficult to achieve. The objective of this section is to outline a set of steps to get as close to the ideal MDC table as possible.

Note: The extent size is associated with a tablespace, and therefore applies to all of the dimension block indexes as well as the composite block index. This makes the goal of high cell density for every dimension block index and the composite block index very difficult to achieve.

Defining small extent sizes can increase cell density, but increasing the number of extents per cell can result in more I/O operations, and potentially poorer performance when retrieving rows from this cell. However, unless the number of extents per cell is excessive, performance should be acceptable. If every cell occupies more than one extent, then it can be considered excessive.

At times, due to data skew, some cells will occupy a large number of extents while others will occupy a very small percentage of the extent. In such cases it would signal a need for a better choice of dimension keys. Currently, the only way to determine the number of extents per cell requires the DBA to issue appropriate SQL queries or use db2dart.

Performance might be improved if the number of blocks can be reduced by consolidation. Unless the number of extents per cell is excessive, this situation is not considered an issue.

Note: The challenge is to find the right balance between sparse blocks/extents and minimizing the average number of extents per cell as the table grows to meet future requirements.

10.4.3 Range partitioning

Table partitioning (TP), more often known as range partitioning, was introduced in DB2 UDB V9. TP is a data organization scheme in which table data is divided across multiple storage objects, called data partitions (not to be confused with database partitions (DPF), according to values in one or more table columns.

These storage objects can be in different tablespaces, in the same tablespace, or in a combination of both.

DB2 UDB supports data partitions or data ranges based on a variety of attributes. Commonly used partitioning schemes are to cluster together data partitions by date, such as by year or month, or have numeric attributes for partitioning. For instance, records with IDs from one to one million are stored in one data partition, IDs from one million to two million in another data partition, and so on. Or, for example, records for clients with names starting with A–C could be in one data partition, D–M in the second data partition, N–Q in a third data partition, and R–Z in the last data partition.

Although you have the option of referring to data partitions by names or numbers (useful for data partition operations), they can be completely transparent to applications. That is, applications can continue to access data by specifying column and table names, and do not need to worry about where the data partitions reside.

DB2 provides flexibility for creating partitioned tables. For example, if there is one year of data, it can be partitioned by date so that each quarter resides in a separate data partition. The create table syntax in Example 10-6 illustrates how this can easily be done. The graphical DB2 Control Center can also be used for creating and managing data partitions.

Example 10-6 Create table syntax for partitioning

```
CREATE TABLE orders(id INT, shipdate DATE, ...)
PARTITION BY RANGE(shipdate)
(
  STARTING '1/1/2006' ENDING '12/31/2006'
  EVERY 3 MONTHS)
```

This results in a table being created with four data partitions, each with three months of data, as shown in Figure 10-8.

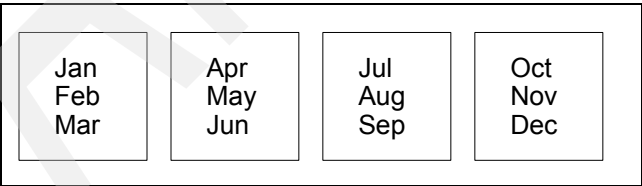


Figure 10-8 Table partitions

The benefits of table partitioning include:

- ▶ *Improved manageability*: DB2 UDB allows the various data partitions to be administered independently. For example, you can choose to back up and restore individual data partitions instead of entire tables. This enables time-consuming maintenance operations to be segmented into a series of smaller operations.
- ▶ *Increased query performance*: The DB2 Optimizer is data partition aware. Therefore, during query execution, only the relevant data partitions are scanned. This eliminates the need to scan data partitions that are not impacted by the query, and can result in improved performance. This is illustrated in Figure 10-9.

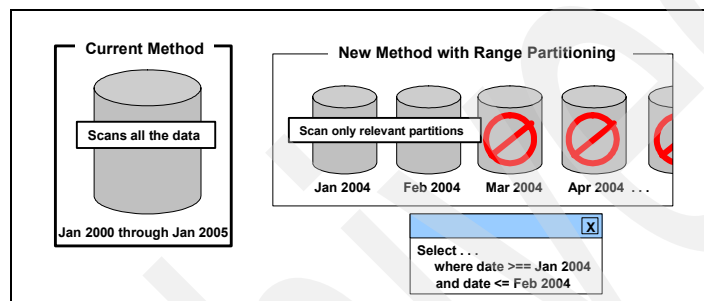


Figure 10-9 Scanning only relevant partitions

- ▶ *Fast roll-in/roll-out*: DB2 9 allows data partitions to be easily added or removed from the table without having to take the database offline. This ability can be particularly useful in a data warehousing environment where there is often the need to load or delete data to run decision-support queries. For example, a typical insurance data warehouse may have three years of claims history. As each month is loaded and rolled in to the data warehouse, the oldest month can be archived and removed (rolled out) from the active table. This method of rolling out data partitions is also more efficient, as it does not need to log delete operations, which would be the case when deleting specific data ranges.
- ▶ *Better optimization of storage costs*: Table partitioning in DB2 UDB enables better integration with hierarchical storage models. By only using the fastest and most expensive storage hardware for only the most active data partitions, DB2 UDB allows optimization of the overall storage costs and improves performance. If most queries only run against the last three months of data, there is an option to assign slower and less expensive storage hardware to the older data.
- ▶ *Larger table capacity*: Without partitioning, there are limits on the maximum amount of data a storage object, and hence a table, can hold. However, by

dividing the contents of the table into multiple storage objects or data partitions, each capable of supporting as much data as in a non-partitioned table, you can effectively create databases that are virtually unlimited in size.

- ▶ *Greater index placement flexibility:* DB2 UDB allows indexes for partitioned tables to be stored in their own storage objects (tablespaces), as opposed to being in the same storage object as the non-partitioned table. This index placement flexibility is particularly useful for performing faster index operations (such as drop index, online index create, and index reorganization), managing table growth, reducing I/O contention, and providing more efficient concurrent access to the index data for the table.

Table partitioning is similar to MDC in that it enables rows with similar values to be stored together. However, the following TP characteristics distinguish it from MDC:

- ▶ TP supports partitioning a table into data partitions along a single dimension. A common design would be to create a data partition for each month. MDC supports defining multiple dimensions.
- ▶ With TP, the user can manually define each data partition, including the range of values to include in that data partition. MDC automatically defines a cell (and creates blocks to store data for that cell) for each unique combination of MDC dimension values.
- ▶ Each TP partition is a separate database object (unlike other tables, which are a single database object). Consequently, TP supports attaching and detaching a data partition from the TP table. A detached partition becomes a regular table. Also, each data partition can be placed in its own tablespace, if desired.

Essentially, the distinct TP benefit is related to adding or removing large numbers of rows from a table. That is, roll in and roll out. For readers familiar with the use of Union All View (UAV) to partition history tables on date, TP can be considered an analogous, but superior, solution.

10.4.4 Using all partitioning schemes together

Each partitioning scheme can be used in isolation or in combination with other data organization schemes. Each clause of the CREATE TABLE statement includes an algorithm to indicate how the data should be organized. The following three clauses demonstrate the levels of data organization that can be used together in any combination:

- ▶ DISTRIBUTE BY spreads data evenly across database partitions. Use this clause to enable intraquery parallelism and distribute the workload across each database partition. This concept is known as database partitioning and is enabled using the Database Partitioning Feature (DPF) in DB2.

- ▶ **PARTITION BY** groups rows with similar values of a single dimension in the same data partition. This concept is known as table partitioning.
- ▶ **ORGANIZE BY** groups rows with similar values on multiple dimensions in the same table extent. This concept is known as multidimensional clustering (MDC).

This syntax allows for consistency between the clauses as well as allowing for future algorithms for data organization. Combining the **DISTRIBUTE BY** and **PARTITION BY** clauses of the **CREATE TABLE** statement allows data to be spread across database partitions spanning multiple tablespaces.

DB2 is the first data server to support all three methods of grouping data at the same time. This is a major innovation in improving data management and information availability. Figure 10-10 illustrates all three DB2 data organization schemes being used in conjunction with each other.

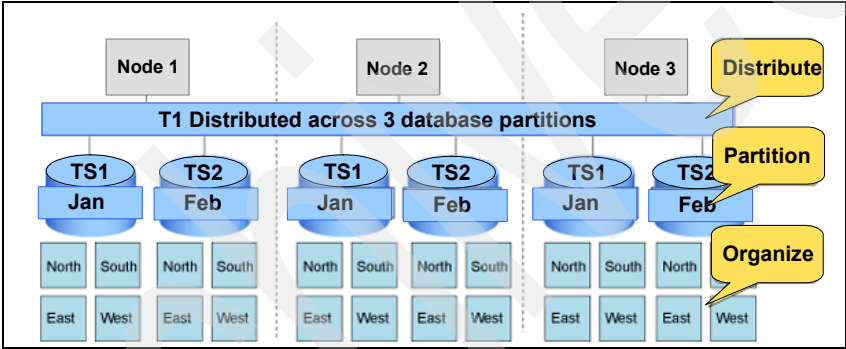


Figure 10-10 Using partitioning schemes together

10.4.5 DB2 partitioning options summary

The **CREATE table** statement now provides three complementary ways to group data in a database table, and is shown in Table 10-1.

Table 10-1 Create table options

| CREATE TABLE statement | DB2 feature |
|------------------------|-------------------------------------|
| DISTRIBUTE BY HASH | DPF - Database partitioning feature |
| ORGANIZE BY DIMENSION | MDC - Multidimensional clustering |
| PARTITION BY RANGE | TP - Table partitioning |

These clauses can be used in any combination to achieve the desired effect. Table 10-2 summarizes the feature terminology.

Table 10-2 DB2 feature terminology

| DB2 feature name | Name of part | Column used to partition the data | Other terms |
|-----------------------------------|--------------------------------------|-----------------------------------|--|
| Data partitioning feature (DPF) | Database partition | Distribution key | In prior versions, this was called the partitioning key. |
| Multidimensional clustering (MDC) | Cells, which are comprised of blocks | Dimensions | Block indexes. |
| Table partitioning (TP) | Data partition | Table partitioning key | |

Table 10-3 provides a feature comparison.

Table 10-3 DB2 feature comparison summary

| Feature | How the feature organizes data | Benefits |
|---------|---|--|
| DPF | Evenly distributes rows across database partitions | Scalability - Add computing resources (database partitions) as the database grows. |
| MDC | Groups all rows with the similar values on multiple dimensions in the same physical location in the table, called a block | Query performance - Organize data for faster retrieval, particularly for queries that involve ranges of multiple predicates. |
| TP | Groups all rows in a specified range of a single dimension in the same data partition | Data movement - Add and remove large volumes of data by adding and removing entire data partitions. |

Table design

When designing a table, the use of each feature can be considered independently. As examples:

- ▶ Determining the distribution key for DPF is not affected by whether MDC and TP are also being used.
- ▶ Whether a column should be used as table partitioning key for TP is not impacted by whether that column is also used by an MDC dimension and visa versa. Each decision can be made independently.

The way each feature works, with regard to indexing, is not changed with the introduction of new partitioning features. For example, when MDC was introduced, its index aspects did not change DPF aspects of indexes. Also, when TP was introduced, it did not change indexing related to DPF or MDC. Keeping this in mind when learning about these features can help avoid confusion.

For example, suppose that you are learning about TP and you come across the statement that *TP has global indexes*. You should not infer that there is some change in how indexes are handled in DPF. In statements such as this, the term *global* merely means that indexes are global across TP data partitions.

Fact (or history) tables in a data warehouses make excellent candidates for use with each partitioning feature, as depicted in Table 10-4.

Table 10-4 *Fact tables characteristics*

| Feature | Suitable table characteristic | Characteristic of fact tables |
|---------|---|--|
| DPF | Large tables - larger than can be handled by a single set of CPUs and I/O channels | Fact tables are the largest database tables. They often contain hundreds of millions of rows and sometimes hundreds of billions of rows. |
| MDC | Queries whose result sets return rows with similar values along multiple dimensions | Fact tables (and data warehouses in general) are designed to support these types of queries. |
| TP | Tables where large volumes of rows are added periodically then later removed after an expiry date | In fact tables, new data are often added daily. Obsolete data is usually removed monthly or quarterly. |

Rules of thumb

In this section we provide guidelines that will help in understanding the nature of design decisions.

For DPF, the top priority when selecting a distribution key is to find one that will distribute the rows evenly across the database partitions. When this situation is not achieved, the result is data skew. This means that one or some database partitions are assigned a disproportionate number of table rows, which can create a performance bottleneck. Columns with many distinct values are good candidates. Other considerations include choosing a column that maximizes the performance of joins.

Another DPF design decision concerns the number of database partitions. The number of database partitions is not primarily a table design consideration. Rather, it is an overall system design consideration based on the anticipated raw data size of the entire database and capacity of the server hardware. Many

systems need fewer than 20 database partitions. However, the largest systems may have many more. The common range can be expected to increase due to the trend toward larger data warehouses.

For MDC, a key decision is which columns will serve as MDC dimensions. The design challenge is to find the best set of dimensions and granularity to maximize grouping but minimize storage requirements. This requires knowledge of the pattern of queries that will be run. Good candidates are columns that have any or all of the following characteristics:

- ▶ Used for range, equality, or IN-list predicates
- ▶ Used to roll-in, roll-out, or other large-scale deletes of rows
- ▶ Referenced in GROUP BY or ORDER by clauses
- ▶ Foreign key columns
- ▶ Columns in the join clauses in fact table of star schema database
- ▶ Coarse granularity, that is, few distinct values

A typical design includes an MDC dimension for a column representing date plus dimensions on 0 to 3 other columns, such as region and product_type.

For TP, design decisions include selecting the column to use as the table partitioning key and number of partitions. Usually the table partitioning key will be a time-based column. Each partition will correspond to a time range of data that is rolled out at the same time. For example, a table that has monthly rollout would have a partition for each month of data. One design consideration unique to TP is the need to handle rows that fall outside the ranges of values defined in the CREATE table statement.

A typical design for a database with monthly roll-out based on sale_date would be to use sale_date as the table partitioning key and create a separate partition for each month.

Generally, one of the MDC dimensions will be a time-based column so the same column can be used for both MDC and TP. The MDC granularity can be finer-grain than the TP data partition size.

These points are summarized in Table 10-5.

Table 10-5 Design rule of thumb summary

| Partitioning feature design decision | Rule of thumb |
|---|--|
| DPF - column to use as the distribution key | The top priority is choosing a column with many distinct values. |
| MDC - columns to use as dimensions | A typical design is to choose a column that represents date plus 0 to 3 other columns such as region and product_type. |

| Partitioning feature design decision | Rule of thumb |
|---|--|
| TP - column used as the table partitioning key and number of partitions | Choose a time-based column. Define partitions that correspond to data that is rolled out at the same time. |

10.5 DB2 performance features

Many features that improve performance are built into DB2. However, there are several design features that are highly recommended for all data warehouse implementations.

10.5.1 Materialized query tables (MQT)

Sometimes a simple change in physical database structure dramatically improves query performance. In addition to indexes, DB2 UDB provides materialized query tables, which in some cases are more efficient than indexes. In prior DB2 versions, MQTs were referred to as automatic summary tables (ASTs). The terms MQT, AST, and summary tables are used interchangeably.

You can think of an MQT as a type of materialized view. Both views and MQTs are defined on the basis of a query. The query on which a view is based is run whenever the view is referenced. However, an MQT actually stores the query results as data, and you can work with the data that is in the MQT instead of the data that is in the underlying tables.

MQTs contain the results of queries, as shown in Figure 10-11. In fact, the Data Definition Language (DDL), or SQL code, used to create an MQT contains a query. MQTs are based on queries of base tables. (A base table is a regular table on which an MQT is based.) The data contained in an MQT is usually an aggregation or summary of the data in the underlying base tables.

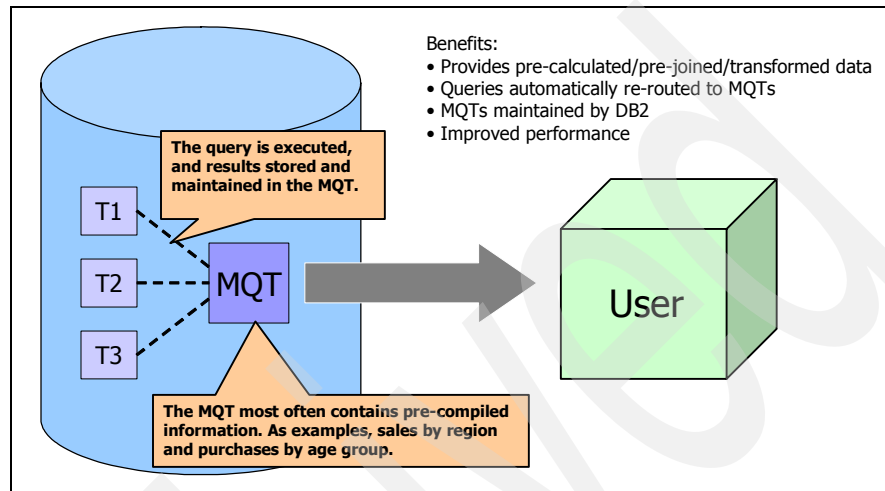


Figure 10-11 MQT

Materialized query tables can significantly improve the performance of queries, especially complex queries. If the optimizer determines that a query or part of a query could be resolved using an MQT, the query might be rewritten to take advantage of the MQT.

Consider a database where queries are run that aggregate data regularly, such as total sales over a given period for various regions. When reports are generated or queries are executed to retrieve this aggregated data, processing can take from minutes to hours each time these queries are executed. Now you can create an MQT to support these reports and queries. The processing is performed one time to aggregate the data in the MQT. All sales that were made per region per time period can be aggregated for all regions and time periods, concurrently. Now each time a query is executed to request sales data for a specific region or a report is generated for all regions, the data can be read from an MQT. When DB2 provides the data by reading from the MQT instead of recalculating aggregated data repeatedly as queries execute, response time is reduced from minutes, hours, or longer, to seconds, in many cases. The cost of (re)processing the requests is reduced to the cost of reading the aggregated data from the MQT, and overall response time is dramatically reduced.

MQTs are a powerful way to improve response time for complex queries, especially queries that might require some of the following operations:

- ▶ Aggregated data over one or more dimensions
- ▶ Join and aggregate data over a group of tables
- ▶ Data from a commonly accessed subset of data, such as a frequently accessed horizontal or vertical partition
- ▶ Repartitioned data from a table, or part of a table, in a partitioned database environment

The advantage of using DB2 MQTs is that a user does not need to know that they exist in order to use them. Knowledge of MQTs is integrated into the SQL compiler. When a dynamic SQL query is submitted, the DB2 Optimizer examines the query, the related tables, and any MQTs that are based on those tables. If the optimizer determines that an MQT can answer the query at a lower processing cost, it automatically rewrites the query to use the MQT instead of all or some of the base tables. The DB2 Optimizer changes it, transparently, without requiring special commands or changes to the application code. If an MQT is used, the EXPLAIN facility provides information about which MQT was selected.

Because the optimizer is aware of all MQTs, no changes to user queries are required after you build an MQT. Once the DBA creates the MQT, many queries may share in the benefit of the MQT and users and applications may not even know that they are exploiting this powerful feature.

An MQT is just a table and can be directly accessed by name. In this case, the DB2 Optimizer always accesses the MQT because it is explicitly requested. However, this is discouraged because, when direct dependencies are created on any MQTs by query tools, the ability of developers and system administrators to drop, alter, or otherwise refine these MQT tables is restricted.

Creating and maintaining MQTs

MQTs can be created by using the same CREATE TABLE command used for regular tables, but some specific clauses for MQTs must be included, as shown in Example 10-7. To create an MQT, a user must be granted permission to create tables. At most sites, for maintenance reasons, creating tables is restricted to developers and administrators.

Example 10-7 MQT sample

```
create table sales_summary as (select sales_person, region, sum(sales)
as total_sales
from sales group by sales_person, region)
data initially deferred refresh deferred
```

When discussing the data contained in an MQT, you must also consider data currency. That is, how current is the data? Is it in sync with the data in the underlying base tables, or have the underlying data been updated since the MQT was last populated? Does currency matter for the application? Sometimes the data needs to be kept current in as close to real-time as possible, for example, to make sure that bank balances are sufficient when a withdrawal is requested. Alternatively, data might need to be captured in a snapshot and held, for example, when generating month-end or year-end reports. While the production system might need to keep updating, the MQTs can be refreshed at the end of the time period, which gives a fixed snapshot for consistent and extensive analysis.

To support these scenarios, MQTs can be maintained automatically by DB2 or maintained manually and updated by a user or an application. Each method has advantages and disadvantages. For example, system-maintained MQTs are always current and available for use, but the data in them cannot be altered. User-maintained MQTs can be altered for what-if types of analysis or can capture a snapshot in time and be held. However, the data in the MQT might not be sufficiently current at any given time unless the user updates the data again. In the latter case, MQTs can also have staging tables where delta data can be queued until processed into the MQT by a table refresh. This can substantially accelerate the update/refresh process.

After the MQT has been created, it has to be populated and synchronized with the data in the base tables. DB2 provides two approaches to MQT maintenance:

- ▶ **Refresh immediate.** DB2 watches for changes in any of the tables that affect values maintained in an MQT. If an insert, update, or delete occurs in a source table for the MQT, DB2 includes the appropriate changes to the MQT as part of the originating transaction. In this mode, DB2 is responsible for keeping the MQT consistent.
- ▶ **Refresh deferred.** In this mode, changes to source tables do not trigger DB2 automatic MQT maintenance. This gives the database administrator (DBA) full control of when the MQT maintenance should be performed, and makes the DBA responsible for determining MQT currency. In such a case, there is latency between the contents of the MQT and the contents of the base tables. Refresh deferred MQTs can be refreshed incrementally with the use of staging tables.

A system-maintained MQT could be populated by DB2 automatically through the use of the REFRESH or SET INTEGRITY commands. A user-maintained MQT could be loaded manually by the user through the use of the LOAD command.

In addition to the features that are specific to MQTs, these tables look and function very much like standard DB2 tables. Also, MQTs can be tuned. For example, indexes can be created on MQTs and the RUNSTATS utility can be

executed on MQTs. MQTs can also be built on multidimensional clustered tables (MDCs), and they can be MDCs as well.

Note: MQTs are also discussed in 5.2.2, “Materialized Query Tables (MQTs)” on page 85.

10.5.2 Replicated tables

The key performance goal for database design with DPF (hash partitioning) is collocation. This is where the joining of tables is done within one or more individual partitions, in contrast with directed or broadcast joins, which involve rows being sent to specific or all partitions.

The partitioning key (used in the CREATE statement to define a partitioned table) is used by the system to distribute as evenly as possible the data for that table across as many database partitions as desired, so as to maximize parallelism and therefore performance. The term *collocation* is used to refer to tables that have the same partitioning key. Then, when joining these tables, the join can be done with exceptionally good performance. Therefore when designing the tables, it is advantageous to find common keys that will be good partitioning keys.

Columns are co-locatable if the following are all true:

- ▶ Are in the same partition group.
- ▶ Have the same number of partition key columns.
- ▶ Data types of partition key columns are pair-wise compatible.

If tables cannot be partitioned on the same keys as other tables and are modest in size and are typically read only, then we recommend that replicated tables be used to assist in the collocation of joins. For example, if a star schema contains a large fact table spread across twenty nodes, the joins between the fact table and the dimension tables are most efficient if these tables are collocated. If all of the tables are in the same database partition group, at most one dimension table is partitioned correctly for a collocated join. The other dimension tables cannot be used in a collocated join because the join columns on the fact table do not correspond to the partitioning key of the fact table. To achieve collocation for a particular small table, create replicated tables. This means that a full copy of the table is maintained in each partition and therefore its rows never have to be sent between partitions to do a join.

When you create a replicated materialized query table, the source table can be a single-node table or a multi-node table in a database partition group. In most cases, the replicated table is small and can be placed in a single-node database partition group, as shown in Example 10-8, where TS_SDPG is the tablespace in the single data partition group. You can limit the data to be replicated by

specifying only a subset of the columns from the table or by specifying the number of rows through the predicates used, or by using both methods.

Example 10-8 Single partition source table

```
CREATE TABLE TIME_DIM (  
    TIME_ID SMALLINT NOT NULL,  
    QUARTER CHAR(4) NOT NULL,  
    MONTH_ID SMALLINT NOT NULL,  
    MONTH_NAME CHAR(10) NOT NULL,  
    MONTH_ABBREV CHAR(3) NOT NULL,  
    SEASON CHAR(6) NOT NULL)  
    IN TS_SDPG
```

The replicated table is created using the MQT syntax with the key word **REPLICATED**, as shown in Example 10-9. This will specify that the data stored in the table is physically replicated on each database partition of the database partition group of the tablespace in which the table is defined. This means that a copy of all the data in the table exists on each of these database partitions.

Example 10-9 Multi-partitioned replicated table

```
CREATE TABLE TIME_DIM_R AS ( SELECT * FROM TIME_DIM )  
    DATA INITIALLY DEFERRED  
    REFRESH DEFERRED  
    IN TS_PDPG REPLICATED
```

A replicated MQT can also be created in a multi-node database partition group so that copies of the source table are created on all of the partitions. Joins between a large fact table and the dimension tables are more likely to occur locally in this environment than if you broadcast the source table to all partitions.

Indexes on replicated tables are not created automatically. You can create indexes that are different from those on the source table. However, to prevent constraint violations that are not present on the source tables, you cannot create unique indexes or put constraints on the replicated tables. Constraints are disallowed even if the same constraint occurs on the source table.

Replicated tables can be referenced directly in a query, but you cannot use the **NODENUMBER()** predicate with a replicated table to see the table data on a particular partition.

Use the **EXPLAIN** facility to see if a replicated materialized query table was used by the access plan for a query. Whether the access plan chosen by the optimizer uses the replicated materialized query table depends on the information that needs to be joined. The optimizer might not use the replicated MQT if the

optimizer determines that it would be cheaper to broadcast the original source table to the other partitions in the database partition group.

Replicated MQTs improve performance of frequently executed joins in a partitioned database environment by allowing the database to manage precomputed values of the table data.

You are not restricted to having all tables divided across all database partitions in the database. DB2 supports partial declustering, which means that you can divide tables and their tablespaces across a subset of database partitions in the system.

An alternative to consider when you want tables to be positioned on each database partition is to use MQTs and then replicate those tables. You can create a materialized query table containing the information that you need, and then replicate it to each node.

10.6 Compression

Disk storage systems can often be the most expensive components of a data warehouse solution. For large warehouses or databases with huge volumes of data, the cost of the storage subsystem can easily exceed the combined cost of the hardware server and the data server software. Therefore, even a small reduction in the storage subsystem can result in substantial cost savings for the entire database solution.

10.6.1 Row compression

DB2 V9 introduced technology that compresses row data to reduce storage requirements, improve I/O efficiency, and provide quicker data access from the disk. DB2 V9 uses a dictionary-based algorithm for compressing data records. That is, DB2 V9 can compress rows in database tables by scanning tables for repetitive, duplicate data and building dictionaries that assign short, numeric keys to those repetitive entries. Text data tends to compress well because of recurring strings as well as data with lots of repeating characters, or leading or trailing blanks.

DB2 examines entire rows for repeating entries or patterns, and not just particular fields or parts of rows. Take, for example, the two rows in Table 10-6. In this example, not only the repeating values (of 500) in the Dept column are compressed, but the repeating pattern (of Plano, TX, and 24355) that spans the City, State, and ZipCode columns is also compressed as a single value.

Table 10-6 Repeating patterns

| Name | Dept | Salary | City | State | ZipCode |
|------|------|--------|-------|-------|---------|
| Fred | 500 | 10000 | Plano | TX | 24355 |
| John | 500 | 20000 | Plano | TX | 24355 |

Figure 10-12 compares how DB2 would store the row normally and in compressed formats.

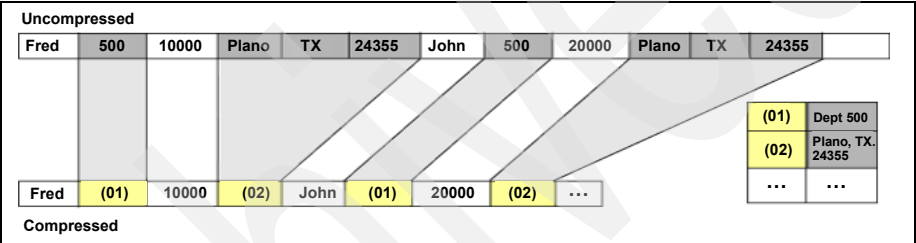


Figure 10-12 Comparison of uncompressed and compressed data storage

The dictionary for compression and decompression lookup is stored in hidden objects in the database, occupies little space, and is cached in memory for quick access. Even for extremely large tables, the compression dictionary is typically only on the order of 100 KB. However, there can be instances when certain data sets do not compress well or there are data size conditions that result in very little compression. DB2 has intelligent algorithms to determine such scenarios and does not perform compression when it does not yield any disk-space-saving benefits.

The data row compression feature in DB2 for Linux, UNIX, and Windows is similar to the compression capabilities available on DB2 for z/OS®. However, it differs from the page-level compression techniques offered by other database vendors, where a compression dictionary is built for each page in the database. By building a compression dictionary at the table rather than page level, patterns across the entire table are analyzed, generally resulting in improved disk savings with DB2.

Enabling compression

Data row compression in DB2 can be turned on when tables are created using the COMPRESS YES option. It can also be enabled for an existing table using the ALTER TABLE command, as shown in Example 10-10.

Example 10-10 Enabling compression

```
CREATE TABLE <table name> --->
      |---COMPRESS NO---|
      +-----+----->
      |---COMPRESS YES--|

ALTER TABLE <table name> --->
      +-----+-----+----->
      |--COMPRESS--+YES--+--|
      |--NO--|
```

The compression takes effect only once the table dictionary is built, which is usually during the table REORG phase, as seen in Example 10-11.

Example 10-11 Building the table dictionary during REORG

```
>--REORG--<table name>--+-----+----->
      '--INDEX--<index name>--'
      .-ALLOW READ ACCESS-.
>--+--+-----+--+-----+--+-----+--+>
      '-ALLOW NO ACCESS---' '-USE--<tbspace>-' '-INDEXSCAN-'
      .-KEEPDICIONARY---.
>--+--+-----+--+-----+--+>
      '-LONGLOBDATA-' '-RESETDICTIONARY---'
```

When compressing a large table, it may be useful to populate the table with a small set of representative or sample data first. The process of building the compression dictionary can be fairly quick using a small data set. And if the set is a good representative sample, the compression will work well even on new data that is added to the table, without DB2 having to analyze the new data. If, however, the type of data stored in the table evolves over time, the dictionary can be kept up to date using REORG.

Estimating savings

DB2 provides the INSPECT tool in order to help you determine the compression ratio estimate for a particular table or data set. It is depicted in Example 10-12. This tool collects a sample of the table data and builds a compression dictionary from it. This dictionary is then used to test compression against the records contained in the sample. From this test, compression savings are estimated.

Example 10-12 Inspect command

```
INSPECT ROWCOMPESTIMATE TABLE NAME Sales ...  
      RESULTS KEEP <filename>
```

```
db2inspf <filename> <outfile>
```

The output of the INSPECT tool needs to be formatted using a DB2 utility to see the results. The files are found in the db2dump directory. Sample output is illustrated in Example 10-13.

Example 10-13 Sample INSPECT output

```
DATABASE: TEST  
  VERSION : SQL09010  
  2005-12-01-13.12.21.232959  
  
Action: ROWCOMPESTIMATE TABLE  
Schema name: RSAHUJA  
Table name: Sales  
...  
    Percentage of bytes saved from compression: 66  
    Compression dictionary size: 2176 bytes.  
...
```

Benefits of data row compression

DB2 V9 row compression technology is capable of storage cost savings by up to 50% or more by reducing the amount of disk space (and disk sub-system peripherals) required for storing data. The size of database logs can also be reduced since DB2 compresses user data within log records.

This technology can also improve performance in some scenarios, despite compression/decompression entailing CPU overhead. Accessing data from the disk is the slowest operation in a query or transaction. By storing compressed data on disk, fewer I/O operations need to be performed on the disk to retrieve or store the same amount of data. Therefore, for disk I/O-bound workloads (for instance, when the system is waiting/idling for data to be accessed from the disk), the query processing time can be noticeably improved.

Furthermore, DB2 keeps the data compressed on both disk and memory (DB2 buffer pools), thereby reducing the amount of memory consumed, and freeing it up for other database or system operations. This can further improve database performance for queries and other operations.

Sample results

The amount of space savings using the data row compression feature in DB2 can vary depending on the data. Clients using beta versions of DB2 V9 have reported savings in excess of 50% and up to 80% for certain large database installations. For one client data set, a 179.9-GB table using 32-KB pages was reduced to only 42.5 GB, for a savings of 76.4%.

Figure 10-13 illustrates a few other examples of space-saving comparisons using the DB2 data row compression feature on different tables.

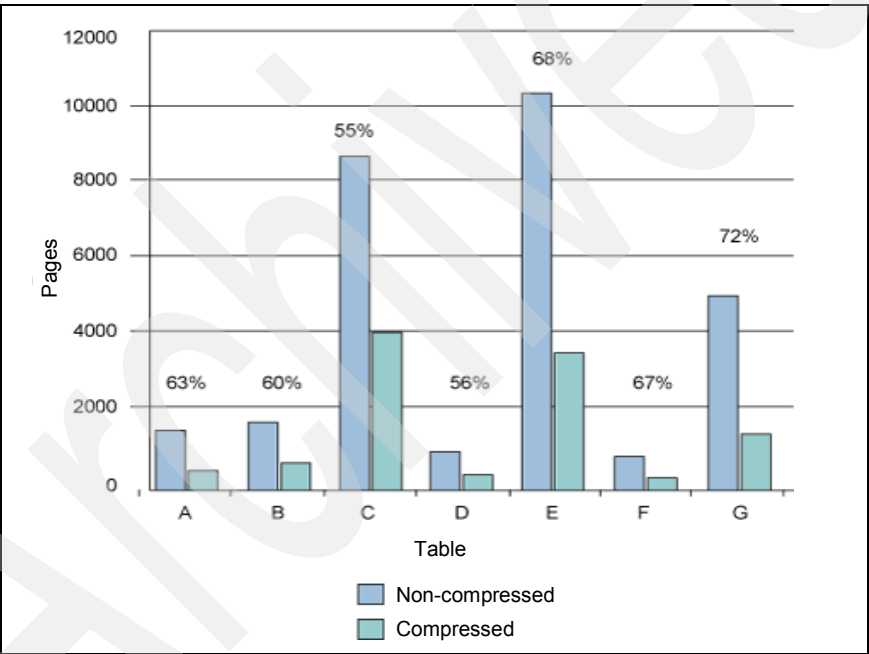


Figure 10-13 Examples of space savings with DB2 data row compression

10.6.2 Other forms of compression in DB2

Besides data row compression introduced as part of DB2 V9, DB2 already had additional mechanisms for reducing storage requirements.

NULL and default value compression

Available in DB2 for Linux, UNIX, and Windows since Version 8, this type of compression consumes no-disk storage for NULL values, zero length data in variable length columns and system default values.

Database backup compression

Available since Fixpak 4 of DB2 for Linux, UNIX, and Windows V8, this compression feature results in smaller backup images that not only reduce backup storage requirements, but also make it easier to move backups between systems.

Multidimensional clustering

A form of index compression has been available in DB2 for Linux, UNIX, and Windows since V8.1. Significant index space savings can be achieved through block indexes, where one key (or index entry) per thousands of records is used (rather than one key per record with traditional indexes).

10.7 Self-tuning memory

Tuning database memory and buffers for optimum performance is effortless with the new self-tuning memory management feature in DB2 V9. It automatically configures database memory settings and adjusts them dynamically during runtime to optimize performance and improve administrator productivity.

Database workloads seldom remain static. Workloads and the environments in which they are run can change over time due to a number of factors, including more users, change in the pattern of queries, running of maintenance tasks, changes in resources consumed by other applications, and so on. Therefore, a system tuned by even the most skilled administrator at one point in time may not be optimal at another time. And the changes may occur in seconds (rather than days or weeks), giving the administrator little time to respond. Database memory settings are especially vulnerable to such changes and can severely impact response times.

The self-tuning memory feature in DB2 V9 simplifies the task of memory configuration by automatically setting values for several memory configuration parameters at startup. The self-tuning memory manager uses intelligent control and feedback mechanisms to keep track of changes in workload characteristics, memory consumption, and demand for the various shared resources in the database and dynamically adapts their memory usage as needed. For example, if more memory is needed for sort operations and some buffer pools have excess

memory, the memory tuner frees up the excess buffer pool memory and allocates it to the sort heaps. This is depicted in Figure 10-14.

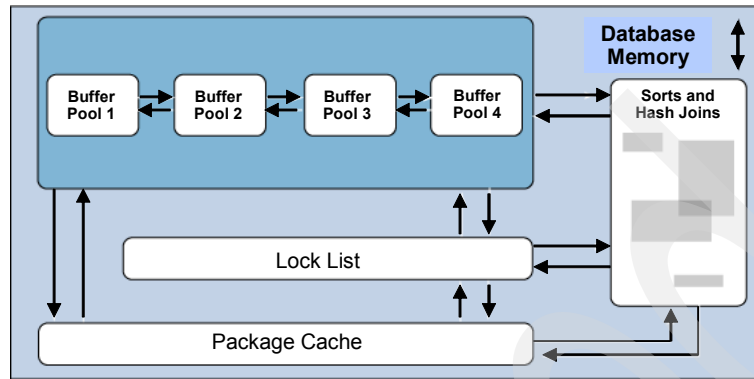


Figure 10-14 Database memory

On Windows and AIX platforms, the self-tuning memory feature can also determine the overall database memory requirements and dynamically tune the total database memory usage. That is, on these platforms, in addition to dynamically adjusting the memory utilization between the database resources, DB2 can also consume more physical memory as the workload demands it and free up that memory to the operating system for other tasks and applications when database memory requirements are low.

Enabling self-tuning memory

Self-tuning memory works on the database shared-memory resources. These resources include:

- ▶ Buffer pools (controlled by the ALTER BUFFERPOOL and CREATE BUFFERPOOL statements)
- ▶ Package cache (controlled by the pckcachesz configuration parameter)
- ▶ Locking memory (controlled by the locklist and maxlocks configuration parameters)
- ▶ Sort memory (controlled by the sheapthres_shr and the sortheap configuration parameters)
- ▶ Total database shared memory (controlled by the database_memory configuration parameter)

The self-tuning memory is enabled using the self_tuning_mem database configuration parameter. In DB2 V9, self-tuning memory is automatically enabled (for non-partitioned databases) when a new database is created. That is, the self_tuning_mem is set to ON, and the database parameters for the resources

listed above are set to AUTOMATIC. For existing databases migrated from earlier versions of DB2, self-tuning memory needs to be enabled manually. Rather than have all of the database memory resources managed automatically, you can also choose to set just the desired memory resources (parameters) to AUTOMATIC.

Traditionally, configuring database memory parameters for optimal operation can be a complex and time-consuming task. By automatically setting the memory parameters for the database, DB2 V9 simplifies the task of configuring the data server. This improves DBA productivity, freeing up the administrator to focus on other important tasks.

In addition to simplifying memory configuration, this new, adaptive self-tuning memory feature improves performance by providing a superior configuration that is dynamic and responsive to significant changes in workload.

This feature can also be very beneficial when there are several databases running on the same system, automatically allowing certain databases to consume more memory during peak times, and freeing it up for other databases and workloads when they need it more.

Self-tuning memory in action

The graphs in Figure 10-15 show DB2 self-tuning memory in action for a demanding benchmark configuration. DB2 automatically increases the database shared memory (graph on the left) to satisfy the demands of the workload, resulting in a corresponding increase in query throughput (graph on the right).

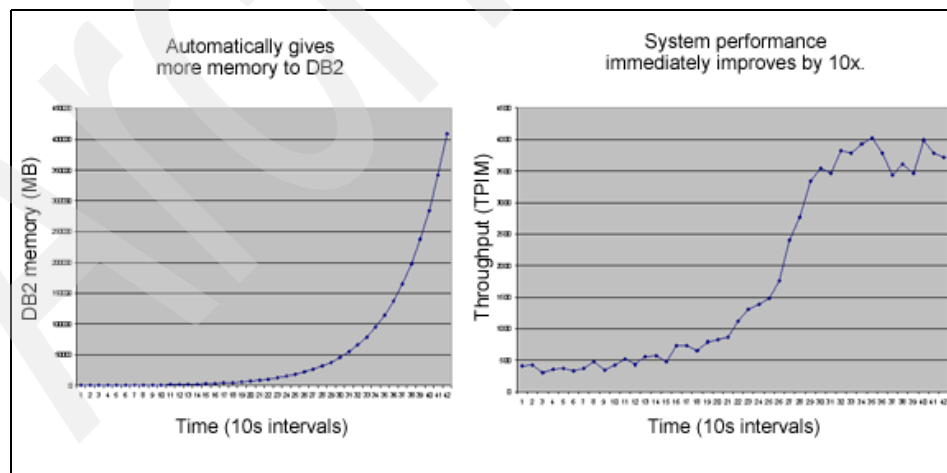


Figure 10-15 Effect of automatic tuning on system performance

Self-tuning memory in DB2 9 is a revolutionary feature that can result in significant time savings, simplify memory tuning, and automatically and dynamically optimize performance.

10.8 DB2 Design Advisor

Many DBAs would agree that the decisions made with respect to the design of a database are some of the most challenging, time consuming, and critical to make. The Design Advisor assists DBAs in making optimal and comprehensive database design decisions. This self-configuring tool greatly simplifies the design process by using workload, database, and hardware information to recommend specific performance acceleration options for routine design tasks.

Specifically, the new Design Advisor assists with the following design tasks:

- ▶ Index selection
- ▶ Materialized query tables selection
- ▶ Multidimensional clustering selection
- ▶ Partitioning selection

The DB2 Design advisor is a tool that can help you significantly improve your workload performance. The task of selecting which indexes, clustering dimensions, or partitions to create for a complex workload can be quite daunting. The Design advisor identifies virtually all of the objects needed to help improve the workload performance. Given a set of SQL statements in a workload, the Design Advisor will generate recommendations for:

- ▶ New indexes
- ▶ New materialized query tables
- ▶ Conversion to multidimensional clustering tables
- ▶ Repartitioning of tables
- ▶ Deletion of objects unused by the specified workload

You can choose to have the Design Advisor GUI tool implement some or all of these recommendations immediately or schedule them for a later time. Whether using the Design Advisor GUI or the command-line tool, the Design Advisor can help simplify tasks, for example, if you are planning for or setting up a new database or partitioning structure. Use the Design advisor to:

- ▶ Generate design alternatives in a test environment for partitioning, indexes, MQTs, and MDC tables.
- ▶ Determine initial database partitioning before loading data into a database.
- ▶ Assist in migrating from a non-partitioned DB2 database to a partitioned DB2 database.

- ▶ Assist in migrating to DB2 in a partitioned environment from another database product.
- ▶ Evaluate indexes, MQTs, or partitions that have been generated manually.

After your database is set up, you can use the Design Advisor to help meet the following types of tuning goals:

- ▶ Improve performance of a particular statement or workload.
- ▶ Improve general database performance, using the performance of a sample workload as a gauge.
- ▶ Improve performance of the most frequently executed queries, for example, as identified by the Activity Monitor.
- ▶ Determine how to optimize the performance of a new key query.
- ▶ Respond to Health Center recommendations regarding shared memory utility or sort heap problems in a sort-intensive workload.
- ▶ Find objects that are not used in a workload.

Collecting the workload from Query Patroller

Before using the Design Advisor, determine how to get a representative sample of the workload to provide as input. This can be challenging due to the ad hoc nature of data warehouse workloads. With DB2 DWE Enterprise Edition, Query Patroller (QP) can be used to select from the historical record of past queries. Select a subset that best meets your needs, such as:

- ▶ The largest 10% of queries
- ▶ Only queries from a particular application that accesses a particular set of tables (for example, a fact table and its dimension tables)
- ▶ A sample that includes queries from each end-user tool (for example, Business Objects versus SAS)
- ▶ A sample that includes queries that access each table
- ▶ A random sample

The selection of the workload should match the goals of the tuning exercise. The recommendations should be interpreted in the context of the workload provided to the Design Advisor. While you could include the entire set of queries, which could be in the 10,000 to 1 million range, this is not recommended. Usually a set of queries ranging from 10s to 100s strikes the best balance between getting good results and consuming DB2 resources.

In this case, you might use a sample of queries that run on one Monday morning when there is a lot of business objects activity, and during one night when some of the bigger SAS jobs are running. Query the Query Patroller tables to get those

queries and export the queries to a file. Use this file as input to the Design Advisor.

As an alternative, the Design Advisor has the capability to directly import the entire Query Patroller set of queries. However, this may not be the best approach if there are many queries (for example, over 10,000). One of the targeted approaches described above may be more advisable. Also, the command-line version of this tool (db2advis) has an additional option to directly obtain a workload from QP that includes queries between a start and end time stamp.

Collect and describe the workload to the Design Advisor

With the workload file in hand, launch the Design Advisor. As shown in Figure 10-16, the Design Advisor allows you to select which features should be considered, such as indexes, MQT, MDC, and partitioning. This example includes all of the options, which ensures that the Design Advisor finds a globally optimal solution by exploiting synergy among the features.

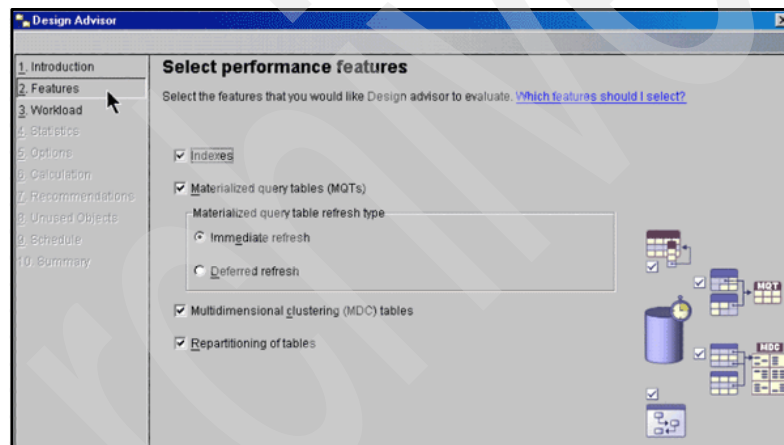


Figure 10-16 DB2 Design Advisor

Selecting all of the options can make it hard to understand why the Design Advisor recommended what it did. Some users may prefer to proceed step-by-step at the possible expense of an optimal solution. The following are examples of step-by-step approaches:

- ▶ Focus on a single application at a time — its workload and the parts of the schema that it accesses.
- ▶ Focus on optimizing access to one table at a time. That is, first identify queries that hit table X and implement only recommendations that pertain to that table. Repeat the process for subsequent tables.

- Focus on a particular feature. For example, focus only on redesigning BigFactTable using MDC and repartitioning. Leave MQTs for a future iteration, perhaps after validating the initial changes in production.

Consider MQTs with deferred refresh. While this example uses immediate refresh, deferred refresh MQTs are generally more compatible with ETL activity.

Command line (db2advvis)

The Design Advisor can also be invoked through the command line instead of the GUI, as shown in Example 10-14.

Example 10-14 db2advvis using the command line

```
db2advvis -d TPCD -i tpch_queries.in -m IMCP -k LOW -l 700 -c  
DB2ADVVIS_TBSP -f
```

Highlights include:

- -m IMCP: This specifies that the Design Advisor should consider new indexes (I), new MQTs (M), converting standard tables to MDC tables (C), and repartitioning existing tables (P). The default is indexes only.
- -k LOW: This specifies to compress the workload to a low degree. As a result, the Design Advisor will analyze a larger set of the workload provided. The default is medium.
- -l 700: This specifies that any new indexes, MQTs, and so on should consume no more than 700 MB. The default is 20% of the total database size.
- -c DB2ADVVIS_TBSP: This specifies a tablespace called DB2ADVVIS_TBSP to use as a temporary workspace for generating MQT recommendations. This option is required if you want MQT recommendations and you are running on a DPF (multiple partition) instance. Otherwise, this parameter is optional.

Another useful option (not shown) is -o output_file. This saves the script to create the recommended objects in a file.

The recommendations appear in the following order:

- Base tables that include MDC or partitioning recommendations
- MQT recommendations (first new ones, then existing ones to keep, and finally unused ones)
- New clustering indexes (if any)
- Index recommendations (new, keep, unused)

Recommendations for changing one table are shown in Example 10-15.

Example 10-15 db2adviz recommendations

```
-- CREATE TABLE "TPCD"."LINEITEM" ("L_ORDERKEY BIGINT NOT NULL,  
-- "L_PART" INTEGER NOT NULL,  
-- "L_SUPPKEY" INTEGER NOT NULL,  
-- "L_LINENUMBER" INTEGER NOT NULL,  
-- "L_SHIPINSTRUCT" CHAR(25) NOT NULL,  
--      (11 other columns omitted from this example)  
-- MDC409022109290000 GENERATED ALWAYS AS ( ((INT(L_SHIPDATE))/7) )  
-- ----PARTITIONING KEY ("L_PARTKEY") USING HASHING  
-- ----IN "TPCDLADT"  
-- ORGANIZE BY (  
-- MDC409022109290000,  
-- L_SHIPINSTRUCT )  
-- PARTITIONING KEY (L_ORDERKEY) USING HASHING  
-- IN TPCDLDAT  
--;  
-- COMMIT WORK ;
```

Note that a new partitioning key is recommended (L_ORDERKEY) to replace the current one (L_PARTKEY), which is commented out. The MDC recommendation for this table (ORGANIZE BY clause) includes two dimensions, a generated column (INT(L_SHIPDATE/7) and an existing column (L_SHIPINSTRUCT).

Next in the output are recommendations related to MQTs, as shown in Example 10-16.

Example 10-16 Recommended MQTs

```
-- LIST OF RECOMMENDED MQTs  
-- =====  
-- MQT MQT40902204140000 can be created as a refresh immediate MQT  
-- mqt[1], 0.009MB  
CREATE SUMMARY TABLE "ADVDEMO2"." MQT40902204140000"  
AS (SELECT Q6.C0 AS "C0", Q6.C1 AS "C1", ...additional details  
omitted here...)  
DATA INITIALLY DEFERRED REFRESH IMMEDIATE PARTITIONING KEY (C8)  
USING HASHING IN TPCDLDAT ;  
COMMIT WORK;  
REFRESH TABLE "ADVDEMO2"." MQT40902204140000";  
COMMIT WORK;
```

```
RUNSTATS ON TABLE "ADVDEM02"." MQT40902204140000";  
COMMIT WORK;  
-- MQT MQT409022041530000 can be created as a refresh immediate MQT  
(... DDL to create this table follows...)
```

The MQT recommendations include estimated size, tablespace to use, partitioning key (if applicable), type of refresh (immediate or deferred), and whether the table is a replica of a base table (indicated by the REPLICATE keyword), which it is not in this case.

Finally, the Design Advisor completes with the information that the DB2 Workload Performance Advisor tool is finished.

10.9 High availability

The DB2 UDB architecture offers high availability of the stored data through the support of clustered takeovers. In case of a machine failure, the activities of the failed machine can be taken over by other components of the cluster. With DB2 UDB with DPF, mutual takeover of machines in a cluster is supported. To the end user and application developer, the database still appears as a single database on a single computer.

DB2 UDB high-availability cluster support is provided for the Sun, AIX, HP-UX, Linux, and Windows platforms. High availability with clustering is provided using Microsoft Cluster Server on Windows, Veritas Cluster Server and Sun Cluster on Sun Solaris, HACMP™ on IBM AIX, HP Service Guard on HP-UX, or Tivoli® System Automation and Veritas Cluster Server on Linux. Standard high-performance mainstream products such as IBM pSeries® or Sun Enterprise™ Servers provide the infrastructure to tie together redundant hardware components with features such as data service monitoring and automatic failover. The cluster software provides transparent takeover of the disk and communications resources of the failed node. It then invokes the DB2 UDB failover support to handle the restart of the database (or the partition) from the failed node.

In a DB2 UDB with DPF environment where two system nodes are paired together, and each pair has access to shared disks (disks connected to two nodes), if one of the nodes in a pair fails, the other nodes can take over and the system continues to operate (an active-active failover scenario) and guarantees that the failover workload can be accommodated. While this method provides quick takeover of a failed node, there may be an impact on performance due to an increased load on the takeover node. An alternative to DB2 UDB with DPF mutual takeover is to have the paired node remain idle until the takeover is

required. This preserves the overall system performance (an active-passive failover scenario).

An additional feature of failover support is that only queries or connections that require access to the nodes taking part in the failover realize that a failover is in progress. Thus, other workloads are unaffected while resources are moved and recovery procedures take place.

DB2 can also exploit Windows, Linux, SUN, or AIX clustering for scalability and improved performance. To the end user and application developer, the database still appears as a single database on a single computer. This provides several benefits:

- ▶ DB2 UDB with DPF clustering enables an application to use a database that is simply too large for a single computer to handle efficiently.
- ▶ DB2 UDB with DPF partitioning avoids system limitations to exploit large SMP machines.
- ▶ DB2 UDB with DPF takes advantage of cluster, SMP, and MPP systems to provide added performance for complex queries. This is accomplished by spreading the database across multiple servers in a cluster, multiple nodes on an MPP, or multiple partitions within an SMP (or combinations of these scenarios). Individual tables can be spread across differing numbers of nodes, thus providing many tools for optimizing performance. Indexes for tables and associated logs are located with the tables so that cross-node access is minimal. The distribution of data across these nodes is transparent to the end user, who sees complete tables and databases.
- ▶ DB2 will automatically execute multiple transactions (SQL statements) in parallel by dispatching them to the multiple partitions. No keywords are needed. DB2 UDB can also automatically execute a single query (SQL statement) in parallel by breaking the query into sub-tasks and dispatching each sub-task to a different node. SELECT, INSERT, UPDATE, and DELETE statements are all executed in parallel. The key to efficient parallel processing across nodes is intelligent partitioning and parallel optimization. DB2 will automatically partition the data across the nodes such that the optimizer will know where each row is located and will be able to dispatch the processing to the node where the data is located — minimizing the movement of data between nodes. Parallelism is used during the sort phase of the SELECT statement, since sorts are local to the node and will also use parallel I/O when appropriate. Parallelism is used during the sort-merge phase, where logically possible, since the optimizer will attempt to push processing down to the individual partitions/nodes. The shared-nothing architecture is the best approach for ensuring scalability across multiple nodes, particularly when very large databases of multiple terabytes are involved.

- ▶ DB2 UDB applications can be ported to systems ranging from XP workstations, UNIX/Linux Servers, and UNIX/Linux/Windows clusters, providing for a wide range of configuration flexibility and processor scalability options. As processing requirements and databases increase, the database manager and server can grow to meet these new requirements.

Managing SQL queries with DWE

DB2 DWE includes the capability to manage the performance of queries in a data warehousing application deployment. The management of queries is important because a data warehousing system can be adversely impacted by query workloads. The balance needed allows user access to the required data with acceptable response times, while not compromising the ability of other users to have the same.

If access to the data is to be through predefined queries then performance could be managed by restricting the size of the set of users accessing that data. However, the advent of query generating tools, from numerous vendors, that are Web-enabled has meant that many users can have concurrent access to almost all of the enterprise data. Traditional database security controls access to the data, but not how it is queried. For example, a user might require access to a large fact table for structured reporting. But if the query tool offers the capability for ad hoc reporting, then the user could potentially issue a query that is very resource intensive. Allowing access to as much data as possible, while at the same time managing that access with a suitable query management tool, is becoming a necessity.

The tool that is provided with DB2 DWE for this query management is DB2 Query Patroller (Query Patroller). It can both proactively and dynamically manage user

queries. Within the DWE deployment infrastructure, Query Patroller is installed and managed through a separate process than the other DWE components.

In this section we discuss the following topics:

- ▶ What is DB2 Query Patroller
- ▶ Architecture and components of DB2 Query Patroller
- ▶ Installing and configuring DB2 Query Patroller
- ▶ Using DB2 Query Patroller in a runtime environment
- ▶ DB2 Query Patroller and the DB2 Governor

11.1 DB2 Query Patroller

DB2 Query Patroller is a query workload management component that provides server-based proactive query management by using the predictive query costing capability of the DB2 Optimizer. It does this by analyzing the predicted optimizer execution plan at submission time to determine whether, based on the predefined configuration limits, it will be allowed to execute. As Query Patroller is based on this execution plan cost of a query, it is important to keep the statistics for tables current by using the DB2 RUNSTATS statement.

Important: DB2 Query Patroller allows action to be taken before the query is run, rather than trying to minimize any impact after execution begins.

Query Patroller can be used by both administrators and query submitters:

- ▶ *Administrators* can use Query Patroller to set resource usage policies at the system level and at the user level. Once initiated, the effectiveness of these policies can be checked by monitoring the Query Patroller system to identify which queries are being intercepted. Query Patroller also allows reports to be generated that can assist in identifying trends in data warehouse usage, such as which objects are being accessed. The analysis of what is being accessed can be useful to check that, for example, MQTs that have been created are being used for the workload. The historical analysis can also reveal which individuals or groups have the longest running queries and which are the most frequent users of the data warehouse or data marts.
- ▶ *Query submitters* can use Query Patroller to monitor the queries they have submitted and have the option of storing query results for future retrieval and reuse, which effectively eliminates the need for repetitive query submission. Preferences can be set to customize the environment for query submissions by, for example, sending an e-mail notification when a query completes.

In this section we introduce the principles of Query Patroller, which are:

- ▶ Query interception and management
- ▶ Thresholds
- ▶ Using a query class
- ▶ Historical analysis

11.1.1 Query interception and management

When a query is submitted, two stages of evaluation are performed by Query Patroller to determine how to deal with the query. First Query Patroller evaluates the query to determine whether it meets the specific criteria that allows it to bypass Query Patroller. Such bypassed queries are not managed, nor are they

used for the collection of data for historical analysis. Second, once a query has been intercepted Query Patroller determines how it will be processed.

Based on the evaluation of the query, one of the following occurs:

- ▶ The query is managed and historical data is collected.
- ▶ The query is not managed, but historical data is collected.
- ▶ The query is not managed, nor is historical data collected.

The two key stages in the management of a query by Query Patroller are *query interception* and *query management*.

Query interception

Query Patroller intercepts a query if it satisfies both of the following criteria:

- ▶ The query is from an application whose queries have been specified to be intercepted. An administrator may decide not to manage certain applications. For example, the db2bp.exe application (DB2 command line) may not need to be managed, while a specific Java application may need to be managed.
- ▶ The query is from a submitter whose *submitter profile* indicates that Query Patroller should intercept the queries, but an administrator may decide that certain DB2 users should not have their queries intercepted. A common example could be the database user running an overnight batch process or any administrative users.

If Query Patroller intercepts the query, it then evaluates the query to see if the query should be managed. Query Patroller can be configured to collect data for historical analysis on intercepted queries even if they do not meet the criteria for queries that should be managed. The intercepting of queries without management is a good means for discovering useful information about the workload.

Query management

Query Patroller manages an intercepted query based on the properties that have been set for the submitter in the Query Patroller system settings. For example, if the submitter settings indicate that only queries that have a cost of 1000 or above timerons are to be managed, then any queries that are estimated by the DB2 query optimizer to be less than that value will not be managed. An example of a system threshold value is the number of queries that are allowed to be concurrently executing on the system. If a query is intercepted that will exceed the threshold, then it will be managed by Query Patroller.

The management functions that Query Patroller can perform on a query include:

- ▶ **Prioritizing a query:** Based on the profile of a submitter, the priority of a query can be set to be 0 (lowest) or 999 (highest). A higher priority query is typically required to be executed before a lower priority query.
- ▶ **Assigning a query class:** Query classes are mechanisms for grouping and running queries according to their estimated cost.
- ▶ **Running the query:** The query can either be run directly, with the application waiting for the data to be returned, or the query can run in the background and populate a results table for the user.
- ▶ **Queuing a query:** Queries may be queued when a defined maximum is reached for certain parameters. If query classes exist, then the queuing of queries is done in a query class and queries in the queue will be executed once they are eligible. An example would be where a query was queued because the submitter had already exceeded the maximum number of queries allowed. Once another query submitted by that user has completed, the previously held query can execute.
- ▶ **Holding a query:** A query will be held if, by the query being submitted, the maximum total cost allowed for that submitter profile, or the total cost allowed by the system, would be exceeded. A held query can be released through either the actions of an administrator or by a scheduled job.
- ▶ **Rejecting a query:** A query can be rejected where no suitable submitter profile has been created. An example would be where only the default *public* profile exists. If that account is suspended, then all queries submitted would be rejected by Query Patroller.

11.1.2 Query Patroller thresholds

Query Patroller relies on resource thresholds to determine the flow of the workload that can run against a database. Thresholds can be set to control the number and size of queries run by a particular submitter or group, to control the size of the overall system workload, or both. Depending on the characteristics of a particular system and workload, some or all of these thresholds can be set. The primary types of thresholds are *submitter thresholds* and *system thresholds*.

Use of submitter thresholds

Submitter thresholds are set in submitter profiles, and can be set for an individual submitter or a group of submitters. The maximum cost threshold, `MAX_COST_ALLOWED`, for a submitter determines the maximum cost of a query that a submitter can run. If a submitter tries to run a query whose estimated cost exceeds the maximum cost for the profile, then the query is held.

This parameter is typically set if there are problems with runaway queries that consume the system resources.

The maximum number of queries, `MAX_QUERIES_ALLOWED`, for a submitter determines the number of queries that can be run concurrently by a particular submitter. If a submitter tries to run another query when the maximum number of queries specified in the submitter profile are already running, the query will be queued until one of the running queries completes. This parameter can be set to allow a submitter group to submit large queries, but each individual submitter will only be allowed to submit a specific number of queries.

Use of system thresholds

The overall query workload that executes against the data warehouse can be controlled by setting system thresholds for cost using the `MAX_TOTAL_COST` parameter and number of queries using the `MAX_TOTAL_QUERIES` parameter.

`MAX_TOTAL_COST` determines the maximum size of the overall workload running against the data warehouse. The workload cost is calculated by adding the cost estimates of all the queries currently running in the system that are being managed by Query Patroller. If the execution of a new query will cause the aggregate cost of all queries currently running to exceed the maximum workload cost, the new query is placed in a queued state until the system can run the new query without exceeding the maximum workload cost. The setting of this parameter provides an upper limit on the maximum cost of the workload and can be useful in lowering the risk of overloading the data warehouse system.

`MAX_TOTAL_QUERIES` places a limit on the number of queries that can be running concurrently in the system. When this threshold is reached, additional queries are placed in a queued state where they wait until the system can run the queries without exceeding the maximum number of queries parameter that was set. This parameter is set to limit the number of concurrent queries, to avoid overloading the system.

11.1.3 Using a query class

A query class is used by Query Patroller to group together queries that have similar cost characteristics. When a query is intercepted by the system it will be allocated to a particular class. Within the query class there will be a maximum number of queries allowed to run and an upper limit for the cost in that class. For example, if there are two classes with an upper limit of 10000 timerons and 100000 timerons respectively, then an incoming query with a cost of 6000 timerons will be placed in the 10000 timerons class.

Note: The DB2 unit of measure for cost is the timeron. A timeron does not directly equate to an actual CPU execution time, but gives a relative measure of the estimated resources required to execute a particular query according to the selected access plan.

The separation of queries into different classes is typically used as a means of ensuring that smaller queries have a chance to run while also allowing larger queries to execute. This is done by separating the workload into classes and only allowing small numbers of queries to execute for the larger query classes. The number of classes, query cost in the class, and the maximum number of queries per class, depend on the individual environment and are best created after a period of time where the workload is analyzed. Using the historical analysis capability of Query Patroller allows workload-specific values to be used for the query classes, rather than a random guess. The use of historical analysis can also aid in the configuring of query classes.

11.1.4 Historical analysis

The historical analysis function provided by Query Patroller allows for the analysis of various aspects of the data warehouse use over time. This allows the gathering of information for business reports, performance tuning, and the identification of redundant database objects. The data for the historical analysis is stored in Query Patroller control tables and is typically displayed either as details or a graph in the Query Patroller Center GUI. The data includes which tables, indexes, and columns are being used and which query submitters are submitting the queries. Additional information such as the number of queries run and query execution time allows viewing resource usage by month, week, day, hour, or minute. This information is particularly useful in identifying peak query times and the tables and indexes that are frequently used.

11.2 Architecture and components

In this section we introduce the Query Patroller components and their functionality. Query Patroller works by having a server component intercept and manage queries. This server component itself is managed either with a client GUI or through the Query Patroller command-line interface.

Figure 11-1 illustrates the Query Patroller components in a typical client server environment (that is, where the DB2 data warehouse and Query Patroller are on a separate processor from the DB2 client and the Query Patroller administration GUI). The configuration shown in Figure 11-1 is for a non-partitioned environment. If the data warehouse was on a partitioned DB2 database, using DB2 with the data partitioning feature (DPF), then the Query Patroller server can be installed on all partitions or on selected partitions. For more details on the installation of Query Patroller refer to 11.3.1, “Installing the Query Patroller server” on page 524.

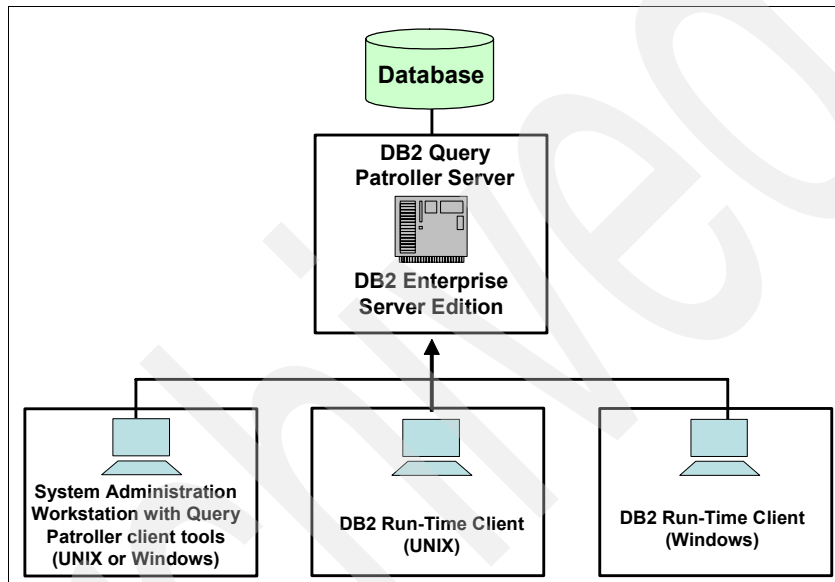


Figure 11-1 Query Patroller components

The implementation of Query Patroller in a DWE environment may differ from that of the other components, which use a Web-based administration tool. In this section we discuss how Query Patroller is implemented with DWE.

11.2.1 The server

The Query Patroller server is the mechanism by which queries are accepted, analyzed, prioritized, and scheduled. It can be installed in either a non-partitioned or a partitioned (DPF) environment.

When Query Patroller is set up to manage queries issued against a database, the DB2 Query Patroller schema, control tables, triggers, functions, and procedures are created in that database. The control tables store all of the information that Query Patroller requires to manage queries. Two tablespaces

are used to store these tables and the associated data. The *control* tablespace is used for storing the Query Patroller metadata and the *results* tablespace is used to store the tables created to contain the results of queries. The creation of these tablespaces will be discussed further in 11.3, “Installing and configuring DB2 Query Patroller” on page 524.

11.2.2 Query Patroller Center

The Query Patroller Center is a graphical client tool that provides the ability to configure and manage the Query Patroller environment. It connects to the database it is currently monitoring through a standard DB2 CLI connection. In the example depicted in Figure 11-2, the database being connected to is CVSAMPLE. The Query Patroller Center is installed separately from other DB2 client tools and is a self-contained tool that only requires that the DB2 client is present to allow connectivity.

If the Query Patroller Center is on a dedicated client machine then the connectivity to the server database can be through the runtime client. However, the Query Patroller client can also be installed on a machine with an existing DB2 Administration client. In such a scenario the Query Patroller Center can access, and be accessed from, the other DB2 tools using the Tools menu, option as shown in Figure 11-2.

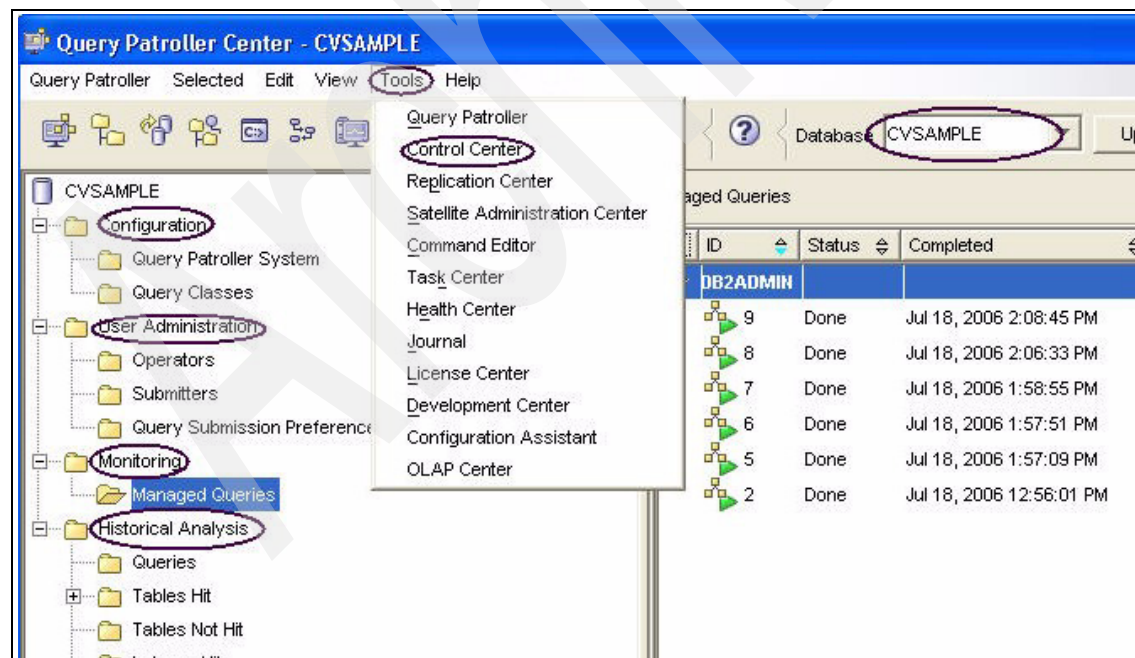


Figure 11-2 Query Patroller Center

In the Query Patroller Center there are sections for:

- ▶ Configuration: The configuration section is where *Query Patroller system* configurations are done and also where the definition and management of *query classes* is performed.
 - The Query Patroller system section is where an administrator can decide what applications are to be intercepted by the Query Patroller server. This is where the setting of the system threshold of *maximum number of queries* and *maximum workload cost* is made. An administrator can also schedule the running of any held queries, and view and change in which tablespaces the results tables are placed. Since the amount of data kept in the form of queries, historical analysis, and results tables can soon become quite large, there are options here to prune the data.

There is also an option to turn on the historical analysis for all queries, whether or not they are managed. This setting is key for an initial Query Patroller system.
 - The query classes section is where query classes can be defined or altered. We recommend that setting query classes only be done after a period of analysis on the system workload.
- ▶ User administration: The user administration section is where user privileges in the Query Patroller Center are allocated to *operators*. This section is also used to create a *submitter* and to allocate *query submission preferences* to a submitter.
 - An *operator* is created to allow DB2 users or groups different levels of access to the Query Patroller Center. An operator can either have *no access*, *view*, or *edit* access to specific administrative functions. If users are allowed to use the Query Patroller Center, then it may only be necessary to allow view access to the Historical Analysis component. If a user has DBADM on the database authority, then they automatically become a Query Patroller administrator.
 - A *submitter profile* allows an administrator to define the submitter charge-back account code, which can be useful when the departmental source of a query needs to be known. This section also allows the specification of whether queries from the submitter should be intercepted, and if they are intercepted then what resource limits should be set. The default submitter profile is PUBLIC, which includes all DB2 users.
 - *Query submission preferences* are used to specify the preferences for a specific submitter profile. If a query is managed by Query Patroller then the submitter can either be made to wait for the results to be returned or the query can operate in the background (which populates a results table). The user is then e-mailed when the results are available. The use of the results table, and whether other users can see the data in this table, is

defined here. This section also specifies whether the data in the results table should be truncated if the results set exceeds what the submitter profile has defined as the maximum number of rows in the results table.

- **Monitoring:** The monitoring section allows the viewing of queries that have been managed by Query Patroller due to their cost being equal to or below the submitters *minimum cost to manage* parameter. In Figure 11-2 on page 519 there are six queries that have been managed by Query Patroller. The details of the managed queries can be drilled down on to retrieve additional information, as depicted in Figure 11-3.

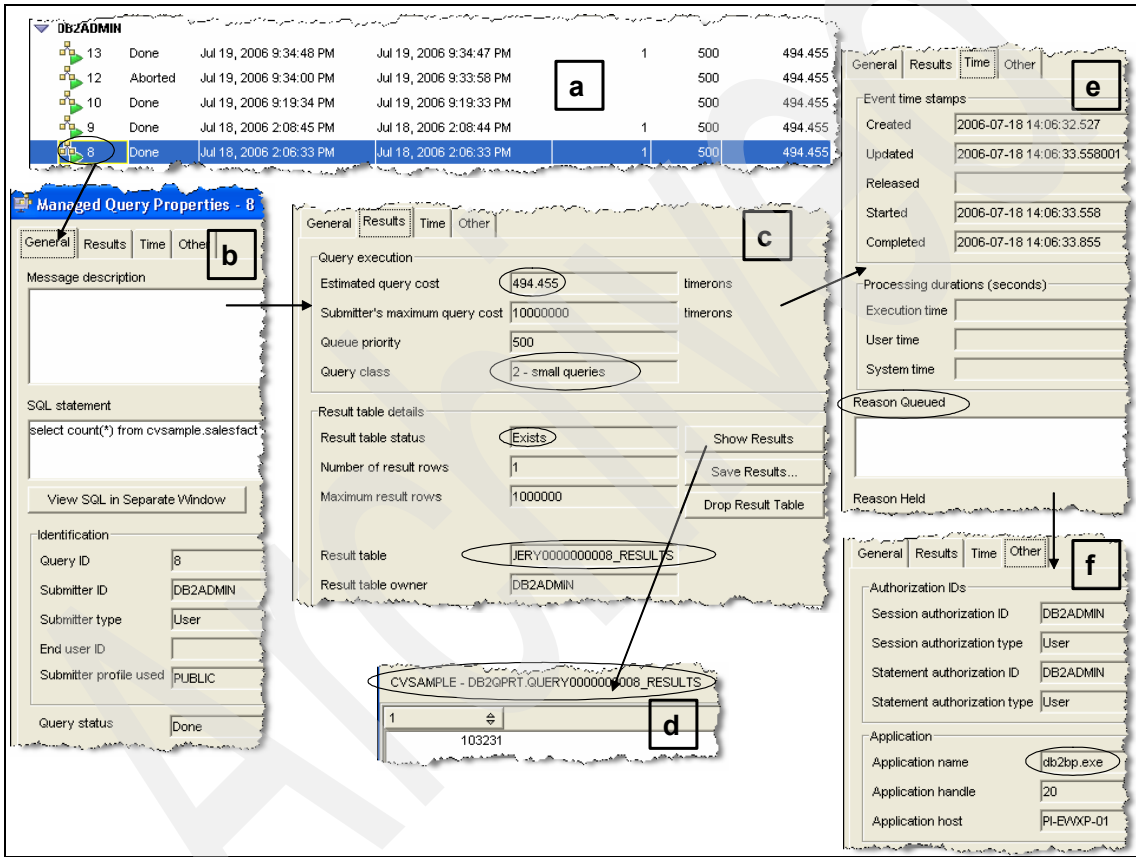


Figure 11-3 Monitoring a user query in the Query Patroller Center

The additional information about a managed query, as shown in Figure 11-3, allows an administrator to trace how the query is executing, or has executed.

- This is the overall managed queries section in the Query Patroller Center

- The General tab of the Managed Query Properties window shows the SQL that was run, who ran it, and whether it has completed.
 - The Results tab shows the cost of the query and what class it belongs to. The name of the result table is also shown, if one was used.
 - The results of the managed query are shown if a results table was used to stage them. The table that the query is run against is the results table rather than the table in the original query.
 - The Time tab shows when the query was executed and, if appropriate, the reasons for holding and queuing the query is shown.
 - The Other tab shows additional information about the query including the user and application that executed the query. This tab can be useful to see which applications are submitting the queries.
- Historical analysis: The historical analysis section (Figure 11-3 on page 521) shows the query execution information that has been retained. This data is stored in the Query Patroller system tables and the more detailed historical analysis data needs to be generated before it can be accessed. The data is generated by right-clicking **Historical Analysis** and choosing **Generate Historical Data**. The process of generating the historical data runs explain plans for the queries so that the required information about table and index usage can be obtained.

Once generated, the data for these retained queries is segmented into a number of sections:

- The Queries section provides a list of the queries that have been retained for historical analysis. The section also shows whether a query has had an explain plan ran for it and the cost in timerons. If a query is removed from this section then the statistics present in the other sections are removed also.
- The Tables Hit section shows the tables that are being used with the query workload. For each table that is used, the Query Patroller Center can show which columns and indexes are, and are not, being used.
- The Tables Not Hit section shows the tables that are not being used with the query workload.
- The Indexes Hit section shows the indexes that are being used in the query workload.
- The Indexes Not Hit section shows the indexes that are not being used in the query workload.
- The Submitters section segments the running of queries and table usage by query submitter. This provides a quick means of checking which objects are being accessed by the different users.

11.2.3 Command-line support

The Query Patroller command-line support enables Query Patroller administrators and submitters to perform most Query Patroller tasks from the DB2 CLP or from the operating system command-line prompt. Query Patroller commands can also be combined with shell scripts or languages such as Perl, awk, and REXX. The command-line interface is installed as part of the Query Patroller client and can also be executed on the server.

From the command line, as shown in Example 11-1, Query Patroller can be started and stopped, and help can be accessed for all commands or just a specific command.

Example 11-1 Typical command line options for Query Patroller

```
qpstart <dbname>
qpstop <dbname>
qp -d <dbname> help
qp -d <dbname> help list
```

11.2.4 Components in a DWE environment

In a DWE environment, the Query Patroller components are installed and managed separately. The installation of the products are discussed in 11.3, “Installing and configuring DB2 Query Patroller” on page 524.

To manage Query Patroller in a DWE environment it is important to consider where the server and client components should be installed. The Query Patroller server needs to be installed on the same system as the data warehouse database. In a topology where the DB2 Data Warehouse server and the WebSphere Server are not collocated, the Query Patroller server would only need to be installed on the data warehouse server. In the data warehouse server, Query Patroller only needs to be enabled for databases that require their queries to be monitored.

The location of the Query Patroller client depends on the functionality that is required by the administrator or user. If only the Query Patroller command line is needed then this can be executed directly on the server through a Telnet connection. For example, an administrator who only needs to start, stop, and monitor a Query Patroller-enabled database could do this through a remote connection to the server, which would not require the installation of client software. This is important, as the Web-based DWE Administration Console does not require any client software and, if the administrator machine has a network connection to the Query Patroller server, then it too can be managed remotely. If there is a need for a user or administrator to use the graphical Query

Patroller Center then each machine that requires this has to have the remote database cataloged locally through a DB2 client.

11.3 Installing and configuring DB2 Query Patroller

In this section we discuss the installation and configuration of Query Patroller for both the server and client requirements. The full details for installing the products and the post-installation configuration are discussed in the Query Patroller manual, which is part of the DB2 online documentation.

This documentation can be accessed on the Web at the following URL:

<http://publib.boulder.ibm.com/infocenter/db2luw/v8//index.jsp>

Once there, navigate to **Product Overviews** → **Database Systems** → **DB2 Query Patroller overview**.

11.3.1 Installing the Query Patroller server

The installation of the Query Patroller server components on Windows and UNIX is generally through the DB2 Setup wizard, which takes an administrator through all the steps necessary for installing Query Patroller. The Query Patroller online manuals contain the details around product versions and system requirements for the installation.

In this section we highlight a few specific features of the installation process and provide some recommendations:

- ▶ When installing Query Patroller the system must already have DB2 installed.
- ▶ The DB2 instance on the server must be stopped. This allows the Query Patroller functionality to be enabled for the DB2 instance, although this enabling feature can be left until after the installation.
- ▶ The install process allows a database to be set up to be managed by Query Patroller and to select or create the control and results tablespace. This configuration can be done after the installation process. In a multi-partitioned environment, the general recommendation is to create the *control* tablespace in a single partition tablespace, as the full benefit of using partitioning may not be realized. The *results* tablespace may benefit from being partitioned if results tables are to be heavily utilized and contain large volumes of data. The results tablespace can also be created with a different page size, and if required a different bufferpool. The page size setting can be important if the queries that will populate the results table may return rows that are too large for the default 4-K pages.

- In a partitioned DB2 environment (DPF), Query Patroller can be installed on all the machines or just on the machine where Query Patroller is to be started from and connected to by the client tools. For best performance Query Patroller should be run on the least busy machine/partition. If the coordinator partition is the least busy partition then Query Patroller can be run from there.

Note: The DB2 coordinator partition manages user connections and coordinates a query.

The installation of the Query Patroller on UNIX can also use the `db2_install` script. This method is useful if the only access to the server is through the command-line interface, such as Telnet, but does not perform any of the post-installation configuration that is done by the DB2 Setup Wizard.

11.3.2 Configuring the Query Patroller server after installation

After the server components have been installed, the Query Patroller environment can be configured to start managing incoming queries. The configuration process involves enabling the database (if required), starting Query Patroller, updating the database configuration for the managed DB2 database, checking that queries are being intercepted, and finally checking that the server can be stopped.

- Enabling the DB2 database

If the control and results tablespaces were not set up during the installation, then these need to be created and then configured by using Query Patroller. This procedure has to take place before the Query Patroller server can be started. The command and expected output is shown in Example 11-2.

Example 11-2 Setting up a database for use by Query Patroller

```

qpsetup db dev control_tablespace QPCONTROL result_tablespace
QPRERESULTS
Creating schema "DB2QP".
Successful
Creating DB2 Query Patroller control tables and the necessary control
objects.
Successful
Query Patroller system updated with the result tablespace name.
DQP2526I Package "@qpserver.lst" was bound successfully.
DQP2526I Package "clientSqlServices.bnd" was bound successfully.
DQP2523I The qpsetup command completed successfully.
```

- ▶ Starting the Query Patroller server
Query Patroller is started by issuing the **qpstart** command. For the database named dev, the command would be:
`qpstart dev`
- ▶ Updating the DB2 database configuration parameter
For a database to be managed by Query Patroller the DYN_QUERY_MGMT database configuration parameter has to be set to enable. For the database named dev, the command would be:
`db2 update db cfg for dev using DYN_QUERY_MGMT enable`
- ▶ Checking that queries are being intercepted by Query Patroller
The easiest way to check that a query is being intercepted is to set the minimum query cost to manage for the PUBLIC profile to a low level and run a simple query. If the cost of the query is below the minimum cost to manage parameter then the query can be part of the historical analysis but is not managed by Query Patroller.

This scenario is illustrated by Example 11-3, which shows a query being intercepted by Query Patroller.

Example 11-3 Checking that a query is being intercepted by Query Patroller

```
qp -d dev update submitter_profile for group 'PUBLIC' using
min_cost_to_manage 10
DQP2121I Submitter profile for group "PUBLIC" was updated
successfully.
db2 select count(*) from syscat.tables, syscat.indexes
1
-----
51480
1 record(s) selected.

qp -d dev list queries

ID          Status          Created          Completed
=====
14          Done          2006-07-21 02:19:25.842000  2006-07-21
02:19:25.904000

"1" out of "1" queries listed.
```


- Stopping the Query Patroller server

Query Patroller is stopped by issuing the **qpstop** command. For the database named dev, the command would be:

```
qpstop dev
```

11.3.3 Installing the Query Patroller client

The Query Patroller client is generally installed using the DB2 Setup wizard on UNIX and must be installed using this method on Windows. The Query Patroller online manuals contain the details around product versions and system requirements for the installation. The general recommendation is to install and configure the server components first before installing the client because the Query Patroller center can only connect to a database enabled for use by Query Patroller. Each remote machine that has the Query Patroller client software installed on it needs to have a DB2 client installed and the remote server database cataloged.

11.4 Best practices for DB2 Query Patroller

After Query Patroller has been installed and configured it is ready to be used to manage queries. The best system and user settings for Query Patroller are derived from an analysis of the query workload. This process is typically done in a development environment before an application goes into production but can also be in an existing production system that does not currently use Query Patroller. In either situation, the procedure is to initially intercept all queries that are submitted to the server. Then use this information to configure your Query Patroller environment.

In this section we discuss the capturing of query information and what to do with that captured information. We also discuss additional best practices for the use of Query Patroller, such as the pruning of the results tables.

11.4.1 Initial configuration to capture the query workload

If possible, all queries should be intercepted and historical analysis enabled for intercepted and managed queries. For the maximum effect, the thresholds should initially be set high to allow all queries to run and the information about the queries to be captured for analysis.

To facilitate this Query Patroller should be configured as follows:

- ▶ All applications should be intercepted (this is the default setting).
- ▶ Do not set any system thresholds (this is the default setting).
- ▶ Choose to save all *intercepted* queries for historical analysis (the default setting is only to save *managed* queries). This change can be made in the Query Patroller Center under the Query Patroller System menu option or through the command line using:

```
qp -d cvsamplE UPDATE QP_SYSTEM USING QUERIES_TO_SAVE 'A'  
CAPTURE_REJECTED_QUERY_INFO 'N' HISTORY_PURGE_PERIOD -1
```

- ▶ The submitter profile for the PUBLIC profile should be updated to have a minimum cost to manage of about 500000 timerons. This high setting should prevent most queries from being managed but can be increased if queries are being managed. The other defaults for the resource limits can be used and will only need to be increased if queries are being managed.

The PUBLIC profile can be updated in the Query Patroller Center under the **User Administration** → **Submitters** option. Or to update the min_cost_to_manage parameter through the command line the command would be:

```
qp -d cvsamplE update submitter_profile for g roup 'PUBLIC' using  
min_cost_to_manage 500000
```

Once the above has been completed, Query Patroller should be intercepting queries and storing them for historical analysis but not managing them. This can be checked by the following process:

1. Execute a query against the database.
2. In the Query Patroller Center check that the query does not appear in the Monitoring → Managed Queries section.
3. The query should appear in the Historical Analysis → Queries section of the Query Patroller Center.

Note: These Query Patroller settings are for initial workload analysis and are not recommended for long term use.

11.4.2 Using the historical analysis data

The length of time that the initial configuration should be used depends on the query workload, but it should be long enough to capture some periods of peak use. Once Query Patroller has sufficient information about the query workload this information can be used to make configuration changes to the Query Patroller system. Individual configurations will depend on the number and type of

queries, but in this section we provide some general principles for using the analytical information. The derived information can help make decisions about:

- The appropriate query classes

The historical analysis information shows the cost of all queries that have been executed, which can be used to specify query classes. A good starting point is to have groupings of *small*, *medium*, and *large* query classes. The maximum number of queries per class can be set, and typically more queries classified as small will be allowed to run.

As an example, if the workload indicated that most queries were 1000 timerons or less, some were 10000 timerons or less, and a few were up to 100000 timerons, then Table 11-1 illustrates a simple query class configuration.

Table 11-1 Example query class configuration

| Query class name | Maximum cost | Maximum queries |
|------------------|--------------|-----------------|
| Small | 1000 | 10 |
| Medium | 10000 | 5 |
| Large | 100000 | 2 |

- Suitable submitter profiles

The default submitter profile is PUBLIC, which is fine for an analysis, but a configured runtime environment should separate the submitter profiles into their correct operating system groups or individual users. This is an opportunity to specify that Query Patroller should not intercept queries from certain groups or users. The most common group that should not be intercepted is DB2 system and database administrators. If the authentication ID that runs the batch database population is different, then it is usually worthwhile not intercepting the queries from this user. In an example where a looping stored procedure is being used to select data, then each iteration of the loop will be caught by Query Patroller, which can make the batch job quite expensive.

- Suitable system and user thresholds

Once the historical analysis has been gathered, an administrator will have a good idea as to what thresholds might be required. If during the analysis there were times when the database slowed down, then it may be worthwhile to examine the number of queries running and the maximum cost of these queries and set these *system thresholds* accordingly.

From a *user threshold* perspective the minimum cost to manage parameter should be in line with the query class values. If the lowest query class has a

maximum value of 1000, as in Table 11-1 on page 529, then the minimum cost to manage parameter should be less than this to enable these queries to be managed. The maximum cost of a query for a submitter profile can also be configured on the basis of the historical analysis. If the system has queries that should route to MQTs, and the approximate query cost of using the MQT or not using the MQT is known, then the maximum cost setting could prevent a non-routing query from running. This particular example should be used with care, as it would prevent larger cost, but correct, queries from running.

11.4.3 Additional best practices

In conjunction with the updates to the Query Patroller configurations after using the historical analysis, there are some additional best practice recommendations that may be relevant to a particular environment.

- Results tablespace fills up

To prevent the control tablespace and results tablespace from filling up, we recommend using the *purge* job in the *Query Patroller System* section of the Query Patroller Center. The decision about how many days to keep the information depends on whether the details about the managed queries and the results data is still useful once the query has been run and the results retrieved.

Note: Results tables are easily identified as having a schema of DB2QPRT.

The historical analysis information is also stored in the control tablespace, and this should be purged after the data has been used. A good recommendation for this is once a week in a stable production environment. If there is a possibility that the data could still be useful, then it could be exported to a flat file before being purged. The purging of objects in the Query Patroller system tablespaces is important because if the underlying tablespaces fill up then Query Patroller will be unable to operate.

- Poor performance of a results table in a partitioned environment

If the preferences for a submitter are that a managed query is to return control back to an application and in the background populate a results table, then the performance of the results table should be considered.

Example 11-4 on page 531 illustrates how the use of a results table is indicated to the client. The contents of the query number can then be retrieved from the results table through the Query Patroller Center using the Managed Queries section.

```
db2 select count(*) from syscat.tables, syscat.schemata, syscat.indexes
SQL29011I Query "25" will be run in the background.
```

If the number of rows in the results tables is projected to be large (perhaps greater than 500000 rows) then in a DPF environment it is worthwhile considering that the results tablespace is a partitioned tablespace. This requirement may be mitigated by keeping the maximum number of rows that can be stored in a results table at a low number.

- MQTs not showing in the Tables Hit section of the Query Patroller Center

If the DWE environment that is managed by Query Patroller uses MQTs and these are not showing up as being used, it may be that the database statistics are not up to date. If the DB2 Optimizer is not aware of the latest statistics, updated by executing a **runstats** statement, then the explain plan may not be 100% correct. Another symptom of this could be where a new index has been added but is still being shown in the Query Patroller Center as not being used.

The version of DB2 that DWE uses (DB2 8.2.3) contains a feature called *automatic maintenance*, which allows certain database utilities, such as RUNSTATS, to run as scheduled activities outside the peak periods. This feature is particularly useful in a DWE environment where current statistics are key for the correct query paths. More details on scheduling DB2 utilities can be found on the Web at:

<http://publib.boulder.ibm.com/infocenter/db2luw/v8//index.jsp>

Navigate to **Configuring** → **Database Systems** → **Automatic maintenance of your database**.

- Too many user queries being held when using a Web-based reporting tool

Many third-party query-generating tools access the data warehouse using a 3-tier environment consisting of a Web-based user client, application server with a DB2 client accessing the DB2 data warehouse server. In such a scenario the users access the application server using their reporting user IDs, but the application server connects to DB2 using one connection ID. In an environment that is managed by Query Patroller this is worthwhile considering that the thresholds for that user will need to be set higher due to the increased usage of the single ID.

11.5 DB2 Query Patroller and the DB2 Governor

In this chapter we have introduced the components of DB2 Query Patroller and discussed its functionality in managing user queries executing against the data warehouse database. Within a DB2 environment Query Patroller is not the only

resource-based tool that can be used to manage the impacts of queries on a system. This tool is the DB2 Governor, which is provided with DB2 and enables you to better manage the use of your systems resources by placing limits on resource usage by any particular application.

The aim of this section is to:

- ▶ Introduce the DB2 Governor.
- ▶ Illustrate some differences between the Governor and Query Patroller.
- ▶ Show how the Governor and Query Patroller can work together.

11.5.1 DB2 Governor

The DB2 Governor is used to monitor the behavior of applications and users that run against the database and can change certain behavior, depending on the rules that are specified in the Governor configuration file.

A Governor instance consists of a front-end utility and one or more daemons. Each instance of the Governor that is started is specific to an instance of the database manager. By default, when the Governor is started a Governor daemon is started on each partition of a partitioned database. However, a daemon can be specified to only be started on a single partition that needs to be monitored. A common example of this is where the Governor is only started on the coordinator partition, as the majority of queries will be routed through this partition. It is usually recommended that the Governor be started on all of the partitions because if a user connects via a different partition (for example, where the Governor is not started) then it will not be governed.

Each Governor daemon collects information about the applications that run against the database. This information is checked against the rules that have been specified in the Governor configuration file for this database. The Governor then manages application transactions as specified by the rules in the configuration file. For example, applying a rule might indicate that an application is using too much of a particular resource. The rule would specify the action to take, such as to change the priority of the application or force it to disconnect from the database.

In Example 11-5 on page 533 the Governor configuration file has the standard first line that specifies the database to monitor, the interval at which to wake up, and the interval at which to write log records. The Governor processes rules in this configuration file from the top of the file to the bottom. However, if a later rule

has a setlimit clause that is more relaxed than a preceding rule, the more restrictive rule still applies. In the three lines that provide the rules in Example 11-5 on page 533:

- ▶ A default rule is set for all users, which will force off their application when these rules are breached.
- ▶ These default rules are cleared for the batch user because, in the event of multiple rules being present for a user, the most restrictive will apply.
- ▶ The rules for the batch user are defined, and are set high so as to only interfere with the batch run when necessary. The action of *priority -5* means that even when the limits are reached the user will not be forced off, only that the priority of the application will be lowered.

Example 11-5 has a default rule for all applications, so we recommend that there are rules put in place for all users/applications that should be managed differently.

Example 11-5 Governor configuration file

```
interval 1; dbname dbname; account 30;
desc "DBNAME - Standard Action for all Users with no specific rule"
setlimit rowssel 1000000 rowsread 1000000000 cpu 350 idle 3600
action force ;
desc "DBNAME - Clear all limits for scheduled maintenance activities
and batch"
authid batch
setlimit cpu -1 locks -1 rowssel -1 rowsread -1 uowtime -1 idle -1
action priority 0;
desc "DBNAME - Standard Action for scheduled maintenance activities and
batch"
authid batch
setlimit rowssel 10000000 rowsread 1000000000 uowtime 3600 idle 14400
action priority -5;
desc "more rules..."
```

The important thing to consider when using the Governor to force applications is that the Governor takes its actions after the threshold has been reached. This means that if a CPU limit has been exceeded the Governor will take some action against the application/user even if the eventual usage may have only just exceeded the threshold. If the action is to force the application off if it exceeds a limit, then all of the work is lost. And if resubmitted, the application will start the query again.

If the action associated with a rule changes the priority of the application, the Governor changes the priority of agents on the database partition where the

resource violation occurred. The changing of a priority is at the operating system level where the underlying process is affected by the changes made by the Governor. For example, a Governor action that increases the priority of a process of 20 means that on AIX the priority of the application will be increased by 20 to 80. Valid values for the priority setting are +20 to -20 (the default priority on AIX is 60).

11.5.2 Differences between the Governor and Query Patroller

Although both the Governor and Query Patroller can be used to manage user queries, there are key differences between the two:

- ▶ The Governor works by terminating a process once a predefined limit has been breached. Query Patroller can prevent the query from running at all by a check on the estimated cost of the query, prior to execution.
- ▶ Query Patroller is a purely cost-based resource management tool, while the Governor manages processes based on other factors, such as how long the unit of work has been executing and how many rows have been read by the application.
- ▶ The Governor can allow certain users and applications to only run at certain times of day or night. The closest equivalent in Query Patroller is the ability to run held queries at specific times.
- ▶ From a user and analysis perspective Query Patroller has the Query Patroller Center and the ability to store the historical data for analysis. The Governor is purely a server-side tool, which logs any action that it takes, which includes any application that it has terminated

11.5.3 Using the Governor and Query Patroller together

Query Patroller and the Governor can work together in a data warehouse environment to ensure that user queries can be categorized and managed while stopping applications that are using large amounts of resources. For example, if there is a query on the system that has low estimated costs it may not be managed by Query Patroller. In this example if the application is reading many rows of data, the Governor can stop that application from completing after the row limit has been reached.

There are, however, also scenarios where the Governor and Query Patroller can impede each other. For example, when a query is executing after being managed by Query Patroller it will still be subject to the rules defined in the DB2 Governor. There are guidelines that should be followed to keep this situation from occurring:

- ▶ Avoid a single all-encompassing general rule in the Governor configuration file.

To prevent the Governor from acting against Query Patroller, the configuration file should not just have a general rule that intercepts all users/applications by default. The preferable alternative is to explicitly include the list of user/applications to be intercepted by the DB2 Governor. In Example 11-5 on page 533 the presence of a default rule is only there to catch any users or applications that do not have a rule defined for them.

- Consider not governing the processes used by Query Patroller.

These are process such as `javaw.exe`, `java.exe`, `db2fmp.exe`, and `qp.exe` on Windows, and `Java`, `db2fmp`, and `qp` on UNIX/Linux. If these processes are included in the DB2 Governor configuration file, conflicts will result and the two performance management processes could act against each other.

For common processes, such as `Java` or `db2bp`, it may not be possible to avoid managing them altogether. In these situations, consider configuring the Governor to use the user ID rather than the application name. For an even more granular solution, a combination of user ID and application can be used to only manage the required combinations. For instance, a system administrator using the `db2bp` process might not be managed by the Governor, but a user using the `db2bp` process may be.

- Remember the dynamic nature of the Query Patroller tables.

The Query Patroller client tools use the Query Patroller control tables to retrieve information such as the historical analysis of the queries. The contents of these tables are dynamic, and there is no way to predetermine the maximum number of records in them. If this is an issue then one solution is to increase the *rowssel* and *rowsread* limits in the Governor configuration file to values that are higher than the current maximum number of records.

- Query Patroller can use dedicated processes to execute queries.

A query that is intercepted by Query Patroller can be executed either by the submitting application or by another application. These applications are called `qprunquery.exe` on Windows and `qprunquery` on UNIX/Linux. If the submission preferences specify that the submitting application should be released and the query results sent to a result table, then the query is executed by `qprunquery`. The DB2 Governor will only intercept the application if `qprunquery` is included in the DB2 Governor configuration file or if there is a default rule in place for all applications

For more help with the use of Query Patroller and the Governor, there is an ITSO technote available at:

<http://www.redbooks.ibm.com/abstracts/tips0324.html?Open>

IBM data warehousing - complementary software

The IBM DB2 Data Warehouse Edition delivers rich business intelligence functionality, and is the IBM enterprise-level offering for building demanding BI solutions. However, there still may be other infrastructure components needed. In this appendix we provide a listing of other IBM complementary data warehousing software by functionality, and the associated Web sites where you can find more information about them.

Data Modeling

In Table A-1 we list products for supporting data modeling.

Table A-1 Data modeling tools

| | |
|-------------------------------|---|
| Rational data architect | http://www.ibm.com/software/data/integration/rda/ |
| Banking data model | http://www.ibm.com/software/data/masterdata/banking/ |
| Insurance data model | http://www.ibm.com/software/data/masterdata/insurance/ |
| Retail data model | http://www.ibm.com/software/data/masterdata/retail/ |
| Telecommunications data model | http://www.ibm.com/software/data/masterdata/telecom/ |

Application development

In Table A-2 we list products for supporting application development.

Table A-2 Application development tools

| | |
|----------------------------------|---|
| Rational Application Development | http://www.ibm.com/software/awdtools/developer/application/ |
| Rational Clear Case | http://www.ibm.com/software/awdtools/clearcase/ |

Enterprise Extract Transform Load (ETL)

In Table A-3 we list products for supporting ETL.

Table A-3 Extract, transform, and load tools

| | |
|---------------------|---|
| WebSphere DataStage | http://www.ibm.com/software/data/integration/datastage/ |
|---------------------|---|

Enterprise Information Integration (EII)

In Table A-4 we list products for supporting information integration.

Table A-4 Information integration tools

| | |
|----------------------------------|---|
| WebSphere Information Integrator | http://www.ibm.com/software/data/integration/db2ii/ |
|----------------------------------|---|

Enterprise Application Integration (EAI)

In Table A-5 we list products for supporting application development.

Table A-5 Extract, transform, and load tools

| | |
|--------------|---|
| WebSphere MQ | http://www.ibm.com/software/integration/wmq/ |
|--------------|---|

Data quality

In Table A-6 we list products for supporting data quality.

Table A-6 Data quality tools

| | |
|--|---|
| WebSphere ProfileStage | http://www.ibm.com/software/data/integration/profilestage/ |
| WebSphere QualityStage | http://www.ibm.com/software/data/integration/qualitystage/ |
| IBM Anonymous Resolution | http://www.ibm.com/software/data/db2/eas/anonymous/ |
| IBM Identity Resolution | http://www.ibm.com/software/data/db2/eas/identity/ |
| IBM Relationship Resolution | http://www.ibm.com/software/data/db2/eas/relationship/ |
| IBM Global Name Analytics | http://www.ibm.com/software/data/globalname/analytics/ |
| IBM Global Name Reference Encyclopedia | http://www.ibm.com/software/data/globalname/reference/ |
| IBM Global Name Scoring | http://www.ibm.com/software/data/globalname/scoring/ |

Database Tools

In Table A-7 we list database tool products.

Table A-7 Database tools

| | |
|------------------------------|---|
| DB2 Performance Expert | http://www.ibm.com/software/data/db2imstools/db2tools/db2pe |
| DB2 Recovery Expert | http://www.ibm.com/software/data/db2imstools/db2tools/db2re/ |
| DB2 High Performance Unload | http://www.ibm.com/software/data/db2imstools/db2tools/db2hpu |
| DB2 Archive Expert | http://www.ibm.com/software/data/db2imstools/db2tools/db2archiveexpert.html |
| DB2 Test Database Generator | http://www.ibm.com/software/data/db2imstools/db2tools/db2tdbg/ |
| DB2 Change Management Expert | http://www.ibm.com/software/data/db2imstools/db2tools/db2cme/db2changemgtexpert-mp.html |
| DB2 Web Query Tool | http://www.ibm.com/software/data/db2imstools/db2tools/db2wqt/ |
| DB2 Table Editor | http://www.ibm.com/software/data/db2imstools/db2tools/db2te/ |

Storage management

In Table A-8 we list products for supporting storage management.

Table A-8 Storage management tools

| | |
|------------------------|---|
| Tivoli Storage Manager | http://www.ibm.com/software/tivoli/products/storage-mgr/ |
|------------------------|---|

Portal and application server

In Table A-9 on page 541 we list products for supporting portal and application servers.

Table A-9 Portal tools

| | |
|------------------------------|---|
| WebSphere Portal Server | http://www.ibm.com/software/genservers/portalserver/ |
| WebSphere Application Server | http://www.ibm.com/software/webservers/appserv/was/ |

Search tools

In Table A-10 we list products for supporting search tools.

Table A-10 Search tools

| | |
|--|---|
| WebSphere Information Integrator OmniFind™ | http://www.ibm.com/software/data/integration/db2ii/editions_womnifindstarter.html |
|--|---|

Master data management solutions

In Table A-11 we list products for supporting master data management solutions.

Table A-11 Master data management solutions

| | |
|---------------------------|---|
| WebSphere Product Center | http://www.ibm.com/software/integration/wpc/ |
| WebSphere Customer Center | http://www.ibm.com/software/data/masterdata/customer/ |

DWE Admin - problem determination example

In this appendix we provide an example to describe how problem determination can be performed in DWE Admin, and how to utilize the EPG information displayed and reported if a failure occurs.

There is one control flow EPG per process and one activity EPG per activity in the process. In the sample SQW application, depicted in Figure B-1, the epg with .0.RUNTIME is the control flow EPG. The second one is for activity Email_02, the third one is for Data_Flow_06, and the fourth one is for Email_08.

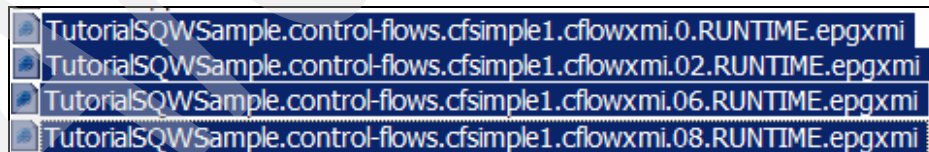


Figure B-1 Sample SQW application

When Email_02 fails, DWE Admin → Troubleshooting displays the content of the failing activity, and code unit /node12 (that is, Email_02, as described in the activity EPG XML file generated in Design Studio). You may then go to the Failed Instance → Code Unit Details and Variable Information to examine whether the proper values were used to run the instance. You can also compare the values used for the failed instance against the default values in DWE Admin

Navigator → SQW Admin → Process/Activity Properties, as depicted in Figure B-2.

```
( ):CODE_UNIT, node /node12;  
  
CODE_UNIT:EMAIL  
(OperatorTag = /flow:010/op:02  
  
OperatorLabel = Email_02
```

Figure B-2 EPG for the failed activity

The problem diagnostic information is depicted in Figure B-3.

| General | | Code Unit Details | | Variable Information | |
|--|--|-------------------|--|----------------------|--|
| Variable Name | | Variable Value | | Variable Type | |
| {mail/sender} | | xyz | | String | |
| {mail/receiver} | | xyz | | String | |
| <div><div><div><<</div><div><</div><div>Page 1 of 1</div><div>></div><div>>></div></div><div><div>1</div><div>Go</div></div></div> | | | | | |

Figure B-3 Problem information

In Figure B-4 we depict the failed process flow for the Activity EPG segment in02.RUNTIME.epgxml showing Email_02 as /node12.

```

—<nodeMap key="12">
  —<value xsi:type="com.ibm.datatools.etl.epgEPGCodeUnit" itemName="Email_02" itemTag="/node12"
    itemID="12" lastAssignedID="2" nodeType="CODE_UNIT"
      dbName="!TRANFORMDB">
        +<inputMap key="PREV_PORT"></inputMap>
        +<outputMap key="NEXT_PORT"></outputMap>
        -<codeUnit codeUnitType="EMAIL" phase="RUNTIME">
          <code/>
          +<ArgumentTable key="OperatorTag"></ArgumentTable>
          +<ArgumentTable key="OperatorLabel"></ArgumentTable>
          +<ArgumentTable key="Activity:"></ArgumentTable>
          +<ArgumentTable key="@RESOURCE"></ArgumentTable>
          +<ArgumentTable key="@SUBJECT"></ArgumentTable>
          +<ArgumentTable key="@RECIPIENT"></ArgumentTable>
          +<ArgumentTable key="@SENDER"></ArgumentTable>
          +<ArgumentTable key="@MESSAGE"></ArgumentTable>
          +<ArgumentTable key="logLevel"></ArgumentTable>
          +<ArgumentTable key="traceLevel"></ArgumentTable>
          </codeUnit>
        </value>
      </nodeMap>

```

Figure B-4 Failed process flow

To step back and look at the process flow, to see where the failed activity is within the process, you may browse DWE Admin → SQW Admin → Manage Processes → Process properties, shown in Example B-1. It shows Email_02 as node /graph12, as described in the Control Flow (process) EPG XML file. Since the process does not have a control flow image, the interpretation of the process graph in the test application will be as follows:

```

Start → Email_02
      → on Failure → Email_08
      → on success → Data_Flow_06

```

Example: B-1 Email_02

```

cfsimple1 ( ) : type GRAPH : node
{
  PREP ( ) : type BLOCK : node /graph9
  {
  }
  Email_02 ( ) : type TRY : node /graph12
  {

```

```

        ( ):EPG_REF_NODE, node /graph12/node9;
        Email_02( Email:02):EPG_REF_NODE;
    }
    CATCH_BLOCK ( ) : type BLOCK : node /graph12/graph7
    {
        ( ):EPG_REF_NODE, node /graph12/graph7/node7;
        Email_08( Email:08):EPG_REF_NODE;
    }
    FINALLY_BLOCK ( ) : type BLOCK : node /graph12/graph8
    {
    }
    ( ):EPG_REF_NODE, node /node15;
    Data_Flow_06( DataFlowActivity:06):EPG_REF_NODE;
}
CATCH_BLOCK ( ) : type BLOCK : node /graph7
{
}
FINALLY_BLOCK ( ) : type BLOCK : node /graph8
{
}

```

Glossary

Access Control List (ACL). The list of principals that have explicit permission (to publish, to subscribe to, and to request persistent delivery of a publication message) against a topic in the topic tree. The ACLs define the implementation of topic-based security.

Aggregate. Pre-calculated and pre-stored summaries, kept in the data warehouse to improve query performance

Aggregation. An attribute level transformation that reduces the level of detail of available data. For example, having a total quantity by category of items rather than the individual quantity of each item in the category.

Analytic. An application or capability that performs some analysis on a set of data.

Application Programming Interface. An interface provided by a software product that enables programs to request services.

Asynchronous Messaging. A method of communication between programs in which a program places a message on a message queue, then proceeds with its own processing without waiting for a reply to its message.

Attribute. A field in a dimension table/

BLOB. Binary Large Object, a block of bytes of data (for example, the body of a message) that has no discernible meaning, but is treated as one solid entity that cannot be interpreted.

Commit. An operation that applies all the changes made during the current unit of recovery or unit of work. After the operation is complete, a new unit of recovery or unit of work begins.

Compensation. The ability of DB2 to process SQL that is not supported by a data source on the data from that data source.

Composite Key. A key in a fact table that is the concatenation of the foreign keys in the dimension tables.

Computer. A device that accepts information (in the form of digitalized data) and manipulates it for some result based on a program or sequence of instructions on how the data is to be processed.

Configuration. The collection of brokers, their execution groups, the message flows, and sets that are assigned to them, and the topics and associated access control specifications.

Connector. See Message processing node connector.

DDL (Data Definition Language). A SQL statement that creates or modifies the structure of a table or database, for example, CREATE TABLE, DROP TABLE, ALTER TABLE, CREATE DATABASE.

DML (Data Manipulation Language). An INSERT, UPDATE, DELETE, or SELECT SQL statement.

Data Append. A data loading technique where new data is added to the database, leaving the existing data unaltered.

Data Cleansing. A process of data manipulation and transformation to eliminate variations and inconsistencies in data content. This is typically to improve the quality, consistency, and usability of the data.

Data Federation. The process of enabling data from multiple heterogeneous data sources to appear as though it is contained in a single relational database. This can also be referred to as *distributed access*.

Data mart. An implementation of a data warehouse, typically with a smaller and more tightly restricted scope, such as for a department, workgroup, or subject area. It could be independent, or derived from another data warehouse environment (dependent).

Data mart - Dependent. A data mart that is consistent with, and extracts its data from, a data warehouse.

Data mart - Independent. A data mart that is standalone, and does not conform to any other data mart or data warehouse.

Data mining. A mode of data analysis that has a focus on the discovery of new information, such as unknown facts, data relationships, or data patterns.

Data Partition. A segment of a database that can be accessed and operated on independently even though it is part of a larger data structure.

Data Refresh. A data loading technique where all the data in a database is completely replaced with a new set of data.

Data silo. A standalone set of data in a particular department or organization used for analysis, but typically not shared with other departments or organizations in the enterprise.

Data warehouse. A specialized data environment developed, structured, shared, and used specifically for decision support and informational (analytic) applications. It is subject oriented rather than application oriented, and is integrated, non-volatile, and time variant.

Database Instance. A specific independent implementation of a DBMS in a specific environment. For example, there might be an independent DB2 DBMS implementation on a Linux server in Boston supporting the Eastern offices, and another separate and independent DB2 DBMS on the same Linux server supporting the western offices. They would represent two instances of DB2.

Database Partition. Part of a database that consists of its own data, indexes, configuration files, and transaction logs.

DataBlades. These are program modules that provide extended capabilities for Informix databases, and are tightly integrated with the DBMS.

DB Connect. Enables connection to several relational database systems and the transfer of data from these database systems into the SAP® Business Information Warehouse.

Debugger. A facility on the Message Flows view in the Control Center that enables message flows to be visually debugged.

Deploy. Make operational the configuration and topology of the broker domain.

Dimension. Data that further qualifies or describes a measure, such as amounts or durations.

Distributed Application In message queuing, a set of application programs that can each be connected to a different queue manager, but that collectively constitute a single application.

Drill-down. Iterative analysis, exploring facts at more detailed levels of the dimension hierarchies.

Dynamic SQL. SQL that is interpreted during execution of the statement.

Engine. A program that performs a core or essential function for other programs. A database engine performs database functions on behalf of the database user programs.

Enrichment. The creation of derived data. An attribute level transformation performed by some type of algorithm to create one or more new (derived) attributes.

Extenders. These are program modules that provide extended capabilities for DB2, and are tightly integrated with DB2.

FACTS. A collection of measures and the information to interpret those measures in a given context.

Federated data. A set of physically separate data structures that are logically linked together by some mechanism, for analysis, but which remain physically in place.

Federated Server. Any DB2 server where the WebSphere Information Integrator is installed.

Federation. Providing a unified interface to diverse data.

Gateway. A means to access a heterogeneous data source. It can use native access or ODBC technology.

Grain. The fundamental lowest level of data represented in a dimensional fact table.

Instance. A particular realization of a computer process. Relative to database, the realization of a complete database environment.

Java Database Connectivity. An application programming interface that has the same characteristics as ODBC but is specifically designed for use by Java database applications.

Java Development Kit. Software package used to write, compile, debug, and run Java applets and applications.

Java Message Service. An application programming interface that provides Java language functions for handling messages.

Java Runtime Environment. A subset of the Java Development Kit that allows you to run Java applets and applications.

Materialized Query Table. A table where the results of a query are stored for later reuse.

Measure. A data item that measures the performance or behavior of business processes.

Message domain. The value that determines how the message is interpreted (parsed).

Message flow. A directed graph that represents the set of activities performed on a message or event as it passes through a broker. A message flow consists of a set of message processing nodes and message processing connectors.

Message parser. A program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A parser is also responsible for generating a bit stream for an outgoing message from the internal representation.

Meta Data. Typically called data (or information) about data. It describes or defines data elements.

MOLAP. Multi-dimensional OLAP. Can be called MD-OLAP. It is OLAP that uses a multi-dimensional database as the underlying data structure.

Multi-dimensional analysis. Analysis of data along several dimensions. For example, analyzing revenue by product, store, and date.

Multi-Tasking. Operating system capability that allows multiple tasks to run concurrently, taking turns using the resources of the computer.

Multi-Threading. Operating system capability that enables multiple concurrent users to use the same program. This saves the overhead of initiating the program multiple times.

Nickname. An identifier that is used to reference the object located at the data source that you want to access.

Node Group. Group of one or more database partitions.

Node. See *Message processing node* and *Plug-in node*.

ODS. (1) Operational data store: A relational table for holding clean data to load into InfoCubes, and can support some query activity. (2) Online Dynamic Server - an older name for IDS.

OLAP. OnLine Analytical Processing. Multi-dimensional data analysis performed in real time. Not dependent on underlying data schema.

Open Database Connectivity. A standard application programming interface for accessing data in both relational and non-relational database management systems. Using this API, database applications can access data stored in database management systems on a variety of computers even if each database management system uses a different data storage format and programming interface. ODBC is based on the call level interface (CLI) specification of the X/Open SQL Access Group.

Optimization. The capability to enable a process to execute and perform in such a way as to maximize performance, minimize resource utilization, and minimize the process execution response time delivered to the end user.

Partition. Part of a database that consists of its own data, indexes, configuration files, and transaction logs.

Pass-through. The act of passing the SQL for an operation directly to the data source without being changed by the federation server.

Pivoting. Analysis operation where user takes a different viewpoint of the results, for example, by changing the way the dimensions are arranged.

Primary Key. Field in a table that is uniquely different for each record in the table.

Process. An instance of a program running in a computer.

Program. A specific set of ordered operations for a computer to perform.

Pushdown. The act of optimizing a data operation by pushing the SQL down to the lowest point in the federated architecture where that operation can be executed. More simply, a pushdown operation is one that is executed at a remote server.

ROLAP. Relational OLAP. Multi-dimensional analysis using a multi-dimensional view of relational data. A relational database is used as the underlying data structure.

Roll-up. Iterative analysis, exploring facts at a higher level of summarization.

Server. A computer program that provides services to other computer programs (and their users) in the same or other computers. However, the computer that a server program runs in is also frequently referred to as a server.

Shared nothing. A data management architecture where nothing is shared between processes. Each process has its own processor, memory, and disk space.

Spreadmart. A standalone, non-conforming, non-integrated set of data, such as a spreadsheet, used for analysis by a particular person, department, or organization.

Static SQL. SQL that has been compiled prior to execution. Typically provides best performance.

Subject Area. A logical grouping of data by categories, such as clients or items.

Synchronous Messaging. A method of communication between programs in which a program places a message on a message queue and then waits for a reply before resuming its own processing.

Task. The basic unit of programming that an operating system controls. Also see *Multi-Tasking*.

Thread. The placeholder information associated with a single use of a program that can handle multiple concurrent users. Also see *Multi-Threading*.

Type Mapping. The mapping of a specific data source type to a DB2 UDB data type.

Unit of Work. A recoverable sequence of operations performed by an application between two points of consistency.

User Mapping. An association made between the federated server user ID and password and the data source (to be accessed) user ID and password.

Virtual Database. A federation of multiple heterogeneous relational databases.

Warehouse Catalog. A subsystem that stores and manages all the system metadata.

Wrapper. The means by which a data federation engine interacts with heterogeneous sources of data. Wrappers take the SQL that the federation engine uses and maps it to the API of the data source to be accessed. For example, they take DB2 SQL and transform it to the language understood by the data source to be accessed.

xtree. A query-tree tool that allows you to monitor the query plan execution of individual queries in a graphical environment.

Abbreviations and acronyms

| | | | |
|----------------|--|---------------|--|
| ACS | Access control system | DCE | Distributed Computing Environment |
| ADK | Archive Development Kit | DCM | Dynamic Coserver Management |
| AIX | Advanced Interactive eXecutive from IBM | DCOM | Distributed Component Object Model |
| API | Application Programming Interface | DDL | Data Definition Language - a SQL statement that creates or modifies the structure of a table or database. For example, CREATE TABLE, DROP TABLE. |
| AQR | Automatic query re-write | DES | Data Encryption Standard |
| AR | Access register | DIMID | Dimension Identifier |
| ARM | Automatic restart manager | DLL | Dynamically Linked Library |
| ART | Access register translation | DML | Data Manipulation Language - an INSERT, UPDATE, DELETE, or SELECT SQL statement. |
| ASCII | American Standard Code for Information Interchange | DMS | Database Managed Space |
| AST | Application Summary Table | DPF | Data Partitioning Facility |
| BLOB | Binary Large Object | DRDA® | Distributed Relational Database Architecture™ |
| BW | Business Information Warehouse (SAP) | DSA | Dynamic Scalable Architecture |
| CCMS | Computing Center Management System | DSN | Data Source Name |
| CFG | Configuration | DSS | Decision Support System |
| CLI | Call Level Interface | EAI | Enterprise Application Integration |
| CLOB | Character Large Object | EBCDIC | Extended Binary Coded Decimal Interchange Code |
| CLP | Command Line Processor | EDA | Enterprise Data Architecture |
| CORBA | Common Object Request Broker Architecture | EDU | Engine Dispatchable Unit |
| CPU | Central Processing Unit | EDW | Enterprise data warehouse |
| CS | Cursor Stability | EGM | Enterprise Gateway Manager |
| DAS | DB2 Administration Server | EJB | Enterprise Java Beans |
| DB | Database | | |
| DB2 | Database 2™ | | |
| DB2 UDB | DB2 Universal DataBase | | |
| DBA | Database Administrator | | |
| DBM | DataBase Manager | | |
| DBMS | DataBase Management System | | |

| | | | |
|-------------|--|---------------|--|
| ER | Enterprise Replication | J2EE | Java 2 Platform Enterprise Edition |
| ERP | Enterprise Resource Planning | JAR | Java Archive |
| ESE | Enterprise Server Edition | JDBC | Java DataBase Connectivity |
| ETL | Extract, Transform, and Load | JDK™ | Java Development Kit |
| ETTL | Extract, Transform/Transport, and Load | JE | Java Edition |
| FP | Fix Pack | JMS | Java Message Service |
| FTP | File Transfer Protocol | JRE™ | Java Runtime Environment |
| Gb | Giga bits | JVM | Java Virtual Machine |
| GB | Giga Bytes | KB | Kilobyte (1024 bytes) |
| GUI | Graphical User Interface | LDAP | Lightweight Directory Access Protocol |
| HADR | High Availability Disaster Recovery | LPAR | Logical Partition |
| HDR | High availability Data Replication | LV | Logical Volume |
| HPL | High Performance Loader | Mb | Mega bits |
| I/O | Input/Output | MB | Mega Bytes |
| IBM | International Business Machines Corporation | MDC | Multidimensional Clustering |
| ID | Identifier | MPP | Massively Parallel Processing |
| IDE | Integrated Development Environment | MQI | Message Queuing Interface |
| IDS | Informix Dynamic Server | MQT | Materialized Query Table |
| II | Information Integrator | MRM | Message Repository Manager |
| IMG | Integrated Implementation Guide (for SAP) | MTK | DB2 Migration ToolKit for Informix |
| IMS™ | Information Management System | NPI | Non-Partitioning Index |
| ISAM | Indexed Sequential Access Method | ODBC | Open DataBase Connectivity |
| ISM | Informix Storage Manager | ODS | Operational Data Store |
| ISV | Independent Software Vendor | OLAP | OnLine Analytical Processing |
| IT | Information Technology | OLE | Object Linking and Embedding |
| ITR | Internal Throughput Rate | OLTP | OnLine Transaction Processing |
| ITSO | International Technical Support Organization | ORDBMS | Object Relational DataBase Management System |
| IX | Index | OS | Operating System |
| | | O/S | Operating System |
| | | PDS | Partitioned Data Set |
| | | PIB | Parallel Index Build |

| | | | |
|--------------|---------------------------------------|-------------|-----------------------------------|
| PSA | Persistent Staging Area | XBSA | X-Open Backup and Restore APIs |
| RBA | Relative Byte Address | XML | eXtensible Markup Language |
| RBW | Red Brick™ Warehouse | XPS | Informix eXtended Parallel Server |
| RDBMS | Relational DataBase Management System | | |
| RID | Record Identifier | | |
| RR | Repeatable Read | | |
| RS | Read Stability | | |
| SCB | Session Control Block | | |
| SDK | Software Developers Kit | | |
| SID | Surrogage Identifier | | |
| SMIT | Systems Management Interface Tool | | |
| SMP | Symmetric MultiProcessing | | |
| SMS | System Managed Space | | |
| SOA | Service Oriented Architecture | | |
| SOAP | Simple Object Access Protocol | | |
| SPL | Stored Procedure Language | | |
| SQL | Structured Query | | |
| TCB | Thread Control Block | | |
| TMU | Table Management Utility | | |
| TS | Tablespace | | |
| UDB | Universal DataBase | | |
| UDF | User Defined Function | | |
| UDR | User Defined Routine | | |
| URL | Uniform Resource Locator | | |
| VG | Volume Group (Raid disk terminology). | | |
| VLDB | Very Large DataBase | | |
| VP | Virtual Processor | | |
| VSAM | Virtual Sequential Access Method | | |
| VTI | Virtual Table Interface | | |
| WSDL | Web Services Definition Language | | |
| WWW | World Wide Web | | |

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 about ordering these publications, see “How to get IBM Redbooks” on page 559. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Mining Your Own Business in Health Care Using DB2 Intelligent Miner for Data*, SG24-6274
- ▶ *Enhance Your Business Applications: Simple Integration of Advanced Data Mining Functions*, SG24-6879
- ▶ *Data Mart Consolidation: Getting Control of Your Enterprise Information*, SG24-6653
- ▶ *DB2 Cube Views: A Primer*, SG24-7002
- ▶ *DB2 UDB's High-Function Business Intelligence in e-business*, SG24-6546
- ▶ *Dimensional Modeling: In a Business Intelligence Environment*, SG24-7138
- ▶ *Preparing for DB2 Near-Realtime Business Intelligence*, SG24-6071
- ▶ *Up and Running with DB2 UDB ESE Partitioning for Performance in an e-Business Intelligence World*, REDP-6917

Other publications

These publications are also relevant as further information sources:

- ▶ *IBM DB2 Data Warehouse Edition Administration and Programming for Intelligent Miner Modeling, Version 9.1*, SH12-6838-00
- ▶ *Knowledge Discovery in Databases: An Overview*, W. Frawley, G. Piatetsky-Shapiro, and C. Matheus, *AI Magazine* (Fall 1992), pp. 57–70
- ▶ *Embedded Analytics in IBM DB2 Universal Database for Information on Demand*, Information Integration Software Solutions white paper (Aug. 2003)
- ▶ *Embedded Data Mining: Steps to Success*, R. Hale and J. Rollins, *Computerworld* (March 13, 2006)

- ▶ *Accessible Insight: Embedded Data Mining*, Rollins, J.B. and Hale, R., DB2 Magazine (Quarter 4, 2005)
- ▶ *DB2 Administration Guide: Performance*, SC10-4222
- ▶ *Informix JDBC Driver - Programmer's Guide V1.4*, Part No. 000-5343

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ Cross Industry Standard Process for Data Mining (CRISP-DM)
<http://www.crisp-dm.org/>
- ▶ The Data Mining Group
<http://www.dmg.org/>
<http://www.dmg.org/pmm1-v3-0.html>
- ▶ Pearson's correlation coefficient
<http://www.stat.tamu.edu/stat30x/notes/node39.html>
- ▶ IBM developerWork Web site: DB2 Data Warehouse OLAP Services, Part 1: Starting out with OLAP services
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0606gong/index.html>
- ▶ Referential Integrity Utility for IBM DB2 Cube Views
<http://www.alphaworks.ibm.com/tech/riu4db2cv>
- ▶ DB2 Basics: An introduction to materialized query tables
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0509melnyk/>
- ▶ DB2 Cube Views Version 8.2 Best Practices
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0511pay/>
- ▶ Integrate DB2 Alphablox and DB2 Cube Views to build multidimensional OLAP Web apps
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0602wen/index.html#download>

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

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Archived

Index

A

aggregate 20, 81–82, 86, 440, 490, 516
AIX 25, 27, 376, 466, 501, 508–509, 534
alert 4, 218, 246, 336
alerts 319
algorithms 12, 17, 55, 231, 238, 244, 248, 485, 496
Alphablox xiii, xv, 20–22, 24, 244, 252, 316–324,
326–333, 335–348, 350, 352–355, 357–363,
366–369, 371, 455–458, 460–463
 also see DB2 Alphablox
Alphablox Cube Server 22, 339–341, 345, 457
Alphablox Cubing Engine 22
Alphablox repository 329–330, 333, 338, 461
analytic application 9, 329, 335, 342, 346, 355
analytic applications 8, 10, 21, 230, 317–318, 320,
323, 328–330, 336–338, 343
analytic structures 13
API 34, 80–81, 248, 250, 333, 335, 448
Application Developer 12, 23, 30, 349–350, 352,
355
application developer 30, 329, 334, 458, 508–509
application logic 336
Application Profile 227
application profile 143, 227, 229, 409
Application server 341
application server 12, 22, 25, 317–318, 325–329,
332, 337–338, 340, 343, 355, 381–382, 394, 402,
407–408, 411, 438, 458, 531
apply program 159–160
architecture xiii, xv, 8, 11, 15, 23, 32, 81–82,
137–138, 141, 159, 230, 232, 248–251, 317–318,
325–326, 328, 335, 368, 388, 391, 401, 457,
466–467, 469–470, 508–509
artifacts 48, 147
Ascential Software 230
attributes 58, 84–85, 88, 103–104, 106, 108–110,
133, 171–172, 238, 240–241, 249, 274–275, 298,
306, 368, 413, 419, 432, 445–446, 477, 482
awk 523

B

best practices xiii, 43, 69, 71, 78, 527, 530
BI xiv–xvi, 8–11, 14, 20, 29–31, 38, 40–41, 48, 54,

57–58, 80–82, 87, 113, 132, 138, 230, 232, 234,
237, 240, 245–247, 252, 358, 469, 537
 also see business intelligence
Bivariate 50, 255, 257, 259
block index 478, 481
Blox 22, 327–328, 330–332, 334–336, 343, 352,
358, 361, 363, 458, 462–463
blox 10, 22, 328, 350, 352, 354, 358, 361
BPU 469
buffer pools 59, 499–500
Business Intelligence xiv–xv, 4, 7, 9, 11, 14, 29–30,
38, 54, 82, 138, 248, 293, 474
business intelligence xiii–xv, 1–2, 8, 10, 80, 230,
537
Business Objects 358, 504
business performance 7
business process 315, 336, 338
business processes 1–4, 245, 316, 456
business rules 196, 231

C

capture program 159
catalog statistics 88
cell density 480–481
ClearCase 34, 55
CLI 380, 383, 519
Cloudscape 399
CLP 387, 442, 444, 523
clustering 241–242, 244, 246–248, 251, 254,
269–270, 274, 293, 297, 384, 449, 452, 467,
476–480, 485, 500, 503, 506, 508–509
clustering index 477
Code generation 197, 199, 221, 229, 233
Cognos 358
collocation 370, 403–404, 475–476, 493
Command operator 213, 428
Command operators 210
composite block index 478, 481
Compression 81, 495, 497–498
compression 80, 496–500
concurrency 417
concurrent 80, 332, 346, 454, 484, 511, 516
Concurrent Versions System 34, 55

- conformed dimensions 231
- containers 48, 54, 147, 326–327, 361
- control flow debugger 221
- Control flow editor 41, 209
- control flow editor 17, 140, 209, 222
- Control Flows 55, 235
- Control flows 142, 205, 209
- control flows xiii, 10, 17–18, 29, 35, 48, 55, 137, 139, 141–143, 147–148, 205, 209, 218, 221–222, 226, 228–229, 235–236, 252, 276, 365, 387, 389, 393, 405, 409–410, 412
- Control operators 210, 216
- conversion 243, 503
- coordinator partition 525, 532
- Cube xiii, xv, 20, 22, 25, 70, 79, 81–82, 84–85, 88–89, 96–97, 103–104, 106–107, 114–116, 124, 131, 320, 338–343, 345–348, 445–446, 457, 462–463
- cube xiii, 8, 10, 14, 20, 22, 29, 81, 84–85, 95, 103–104, 106, 108, 111–117, 121–122, 125, 127–129, 131–133, 339, 341–343, 346, 348, 351, 368, 438, 442, 445–446, 456, 458, 461–462
- Cube dimension 70, 85
- Cube facts 70, 85
- Cube hierarchy 70, 85
- Cube level 70, 85
- Cube Views 79, 82, 89, 346
- Custom SQL operator 194
- custom SQL operator 194
- CVS 38, 54–55

D

- dashboards 317, 319, 361, 368, 456
- Data Architect xiii, 9, 14–15, 29–30, 58, 234, 538
- Data Definition Language 490
- data elements 59, 65
- data federation 236
- data flow canvas 160
- Data flow code generation 199
- Data Flow Editor 151
- data flow editor 17, 140, 151, 209
- Data flow operator 212
- data flow operators 148, 152, 154, 206, 219
- Data flow validation 198
- Data Flows 55
- Data flows 10, 17, 55, 148
- data flows xiii, 10, 17, 19, 35, 43, 49, 55, 137–142, 145, 147–148, 151, 195, 200, 205, 209, 211–212,

- 218, 220, 222, 226, 229, 235–236, 252, 276, 284, 290, 365, 371, 388–389, 395, 404–405, 409, 412, 422
- data integration xv, 231–232, 236
- Data Mart 82
- data mart 145, 148, 203, 232, 241, 243, 247, 275, 340, 399
- Data Mining 7, 10, 12–13, 17, 19, 23–25, 54–55, 174, 247, 293, 310, 358, 366, 447
- Data mining 14, 19, 29, 41, 174, 195, 238–240, 245, 252, 310, 312, 447
- data mining xiii–xiv, xvi, 3–14, 16–18, 29, 36, 38, 46, 48, 50, 55, 81, 138, 153, 174, 192, 195, 234, 237–248, 250, 252, 255, 262, 269, 273, 275, 280, 290, 298, 310, 312–313, 358, 365, 379, 447, 450
- data mining components 247
- data mining modeling xiii
- data model 10, 13, 15, 20, 41, 49, 54, 57–60, 65–66, 69, 72, 75, 94, 99, 101, 103, 118–120, 132–133, 142, 147, 156, 162, 171, 190, 192, 194, 196, 241–242, 265, 365, 383–386, 448, 462
- data models 9, 12–13, 15, 17, 35, 39, 48, 55, 57, 59–61, 65–66, 69, 72, 75, 94, 142, 147, 156, 234
- Data movement 486
- data movement xiii, 7, 10, 12–13, 16, 30, 137–139, 141, 145, 148, 174, 229–231, 234, 236
- Data Output View 46–47, 50, 74
- data partitioning 518
- Data Project Explorer 42–43, 46, 59–61, 64, 67, 69, 91–92, 94, 105–106, 108, 110–112, 114, 117, 132, 134, 147, 151, 276
- data quality 3, 243, 258–259, 266, 269, 539
- data replication 160
- Data Station operator 191
- data station operator 191–192
- Data Transformation 12
- data transformation 10, 55, 142, 193, 232
- Data Transforms 13
- data types 58, 154, 156, 241, 363, 476, 493
- data warehouse xiii–xiv, 3, 6–13, 15–17, 19, 25, 29–30, 41, 48–49, 79–83, 87, 90, 129, 138, 145, 147–148, 153, 171–172, 199, 203–204, 224, 227, 231–236, 241–243, 247, 340, 368, 370, 372, 387, 393–394, 396–397, 402, 404–405, 409–410, 413, 438, 448, 450, 455, 465, 468–469, 483, 495, 504, 513, 516–517, 523, 531, 534
- Data Warehouse Edition xiii, 9–11, 79, 96, 112–113, 137, 148, 205, 230, 365, 447, 465, 537
- data warehousing xiv–xv, 31, 43, 47–48, 55, 222,

- 230, 233, 236, 243, 259, 368, 370, 378, 386, 438, 465, 468–470, 483, 511, 537
- Database 11, 14, 27, 43, 45–46, 49–51, 60, 62, 64, 75, 82, 88, 90, 92, 95–96, 101, 121, 132, 134, 145, 148, 152, 157, 162, 209, 253, 255, 269–270, 293, 310, 312–313, 326, 341, 379–380, 382–384, 398, 400–402, 406, 466, 470, 484–485, 500, 524, 531, 540
- database xiv–xv, 4, 9, 12–13, 15–16, 18–19, 22–25, 27, 29–30, 43, 46–52, 54–55, 58–64, 69, 72, 74–75, 78, 80–82, 84–85, 87, 89–90, 92, 94–95, 97, 100–101, 103, 117–120, 123, 129–130, 132–133, 138, 140, 142, 144–146, 148, 152, 156–157, 159–160, 162, 176, 192–194, 200, 215, 221–222, 224, 228, 230–231, 233–234, 236–239, 244–247, 252–253, 269, 277, 288, 293, 310, 320, 326, 330, 335, 338, 340–341, 343, 368, 370–372, 374, 378–380, 382–384, 386–388, 390, 392–397, 399, 401–402, 404–407, 409–410, 412, 424, 430, 435, 437–444, 446–447, 450, 453–455, 457, 461–462, 466–469, 471–474, 476–477, 480–481, 483–487, 489–496, 498, 500–504, 506, 508–509, 511, 514–515, 517–520, 523–525, 527–529, 531–533, 540
 - connect 95
 - Runtime 371
 - set up 95
- Database Explorer 43, 49–50, 62, 64, 75, 90, 92, 121, 269–270, 293, 311
- database partition 194, 470–473, 476, 484, 493–495
- database partitioning 470, 484, 503
- database schema 90, 92, 156
- Datamart 82
- DataStage 14, 16, 24, 29, 139–140, 210, 219, 224, 228–232, 234–235, 387, 405, 408, 411, 415, 538
 - also see WebSphere DataStage
- DataStage operators 210
- DB2 Alphablox xiii, 21, 317–320, 322–330, 332–333, 335–341, 343, 345–346, 348, 350, 352–355, 357–359, 361–363, 455–458, 461–462
 - also see Alphablox
 - application building Blox 328, 332
 - application instance 334
 - applications 317–318, 456
 - collaboration 324
 - components 318, 320, 327, 330, 336, 456
 - Cube Manager 342
 - deploying 317, 328
 - Lightweight Directory Access Protocol 333
 - repository 329, 333, 338
 - security models 337
 - services 317, 329, 332, 457, 462
 - Session Manager 332, 457
 - User Manager 333
- DB2 Alphablox server 318, 328–329, 334, 457
- DB2 CLI driver 383
- DB2 Command Center 387
- DB2 command line 407
- DB2 Control Center 89, 482
- DB2 coordinator partition 525
- DB2 Cube Views 79, 89–90
 - stored procedure 95
- DB2 Data Warehouse Edition 9, 79
- DB2 Data Warehouse Enterprise Edition 79, 252, 466
- DB2 database 25, 49–50, 59, 82, 118, 141, 146, 148, 370–371, 390, 395, 399, 406, 439, 444, 453, 468, 503, 518, 525
- DB2 Design Advisor 130, 474, 480, 503
- DB2 Design Studio 39, 58
- DB2 DWE xiii, xvi, 11, 29, 31–35, 37, 40, 46, 368, 504, 511
- DB2 ESE 466
- DB2 for z/OS 496
- DB2 Governor 512, 531–532, 534–535
- DB2 High Performance Unload 540
- DB2 Load Utility 169
- DB2 Load utility 159
- DB2 Optimizer xvi, 340, 403, 491
- DB2 optimizer 20, 25, 440, 483, 513, 531
- DB2 Query Patroller xvi, 10, 13, 25, 130, 480, 511–513, 518, 524–525, 527, 531
- DB2 shell scripts 215
- DB2 SQL script 213–214, 393, 419
- DB2 staging table 149
- DB2 table function operator 193
- DB2 UDB xiv, 9, 25, 79, 81–83, 85–86, 89, 95, 100, 118–120, 129–130, 132–133, 145, 339, 402, 465–466, 470, 473–474, 476, 481–484, 489, 508–509
 - also see DB2 Universal Database
 - High availability 508
 - partitioning 470, 473
- DB2 UDB ESE 474
- db2mdapi.sql script 96
- DDL 46, 54, 60, 62, 64, 72–74, 77, 118–120, 368, 384–387, 410, 442, 470, 474, 478, 490, 508

DDL script 64, 73–74, 118, 120
 DDL Wizard 72
 Debug Control Flow 222
 debugging a control flow 221
 decision making 2, 4, 6, 80, 248, 315, 319
 declared global temporary tables 402
 deferred refresh 123, 491, 506
 delete data flow 167
 deployment xiii, 10, 72, 74, 83, 119, 137–139, 141, 143–144, 156, 160, 199, 225–226, 228–230, 232, 244, 247, 249, 252, 320, 325, 330, 349, 368, 382–388, 390–391, 396, 402, 405, 407, 409–413, 419–420, 422, 432, 438, 445–446, 450, 457, 511
 Deployment EPG 199
 Design Advisor 130, 474, 480, 503–506, 508
 design phase 409, 412
 design project 49, 54, 74
 Design Studio xiii, 9, 12–14, 16, 20, 25, 29, 31–37, 39–40, 42–43, 46, 48–55, 58–61, 65, 69, 72, 75, 77, 79, 89, 94–95, 99, 112–113, 117, 133, 138–139, 141–142, 145, 147–148, 151, 157, 160, 162, 169, 198, 200, 203, 205, 209, 212, 219–221, 225, 228, 235, 237, 248, 251–252, 255–256, 262–263, 277–278, 291, 293, 310, 346, 349–350, 352, 355–358, 368, 370–371, 384–387, 389–390, 392–396, 399, 402, 405, 409, 411, 424, 438–439, 441, 444–448, 543
 DIFFERENCE 183–184
 Dimension 70, 85, 107–109, 115, 171–172, 339, 341–342, 445
 dimension 10, 19, 70, 81, 84–85, 103, 107–109, 111–112, 114–117, 122, 171–172, 185, 190, 205, 388, 445–446, 461, 477–478, 480–481, 484–486, 488, 493–494, 504
 Dimensions 99, 106–107, 115, 445–446
 dimensions 17, 20, 54, 80, 84–85, 88, 103, 113, 115, 117, 122, 129, 161, 231, 335, 339, 343, 346, 352, 445, 462, 476–480, 484–488, 491, 503, 507
 DISTINCT operator 181–182
 Distinct operator 181
 DPF xiv, 25, 233, 403, 466, 470, 472–475, 481, 484–488, 493, 506, 508–509, 518, 525, 531
 drag 43, 45, 64, 67, 95, 151, 209, 352, 358
 drill-down 321, 339, 348, 438, 446, 456
 drop 41, 43, 60, 64–65, 87, 95, 151–152, 176, 224, 320, 346, 352, 358, 475, 479, 484, 491
 DWE xiii, xv–xvi, 10–13, 16, 18–19, 21–25, 27, 29, 31–37, 40, 46, 48–51, 53–55, 58–60, 65, 69, 72, 74–75, 77, 79, 82, 84, 87, 89, 91–92, 94–95, 97, 99, 103–104, 112–113, 121, 129, 131–132, 137–142, 144, 147–148, 151, 157, 159–160, 162, 169, 198, 200, 203, 205, 209, 212, 219, 221, 225, 228, 230, 233, 235–237, 243–244, 247–248, 250–251, 278, 293, 310, 317, 319–320, 325, 338, 340, 346, 349–350, 352–353, 355–358, 362, 365–380, 383–396, 399, 404–405, 407–409, 411, 413–415, 421, 423–424, 426, 431, 435, 438–439, 442, 444, 447–448, 450, 454–457, 459, 463, 504, 511, 518, 523, 531, 543, 545
 also see DB2 Data Warehouse Edition
 DWE Admin Console xiii, 90, 129, 369, 371, 375–377, 391
 DWE Administration Console 23–25, 139, 144, 225, 365–367, 373, 375–376, 378–379, 381, 384–387, 390, 406, 424, 427, 447, 456, 523
 DWE Application Server 27, 144, 157, 162
 DWE Clients 27
 DWE Data Integration Service 390
 DWE Database Server 27
 DWE Design Studio xiii, 14, 29, 31–33, 37–38, 53, 55, 59, 65, 75, 77, 79, 90, 138–139, 141–142, 147–148, 151, 205, 209, 226, 237, 252, 346, 356, 372, 384, 389, 409, 411
 DWE Installation 26
 DWE installation 369
 DWE Intelligent Miner 25
 DWE Mining 138, 359, 368, 455
 DWE OLAP 20, 22, 24, 79, 82–83, 87, 91, 94–95, 97, 121, 129, 131–132, 338, 358, 368, 371, 386, 438–439, 455
 DWE runtime server 142
 Dynamic HTML 320, 330, 458

E

EAI 539
 Easy Mining Procedures 248
 Eclipse xiii, 15, 29, 31–33, 53, 55, 140, 151
 Eclipse-based 33
 EII 539
 Email operator 219
 embedded SQL 194
 Enterprise Application Integration 539
 enterprise data warehouse 6–7, 9
 Enterprise Information Systems 326
 Enterprise JavaBeans 333
 EPG viewer 201, 221
 ETL 14, 16, 29–30, 132, 139, 230, 233–236, 387,

396, 506, 538
 also see extract, transform and load
ETL tool 139, 234
Event Monitor 480
events 270, 328, 382
Executables 210, 216
execution database 24, 141–142, 145, 149, 156,
160, 163, 194, 200, 235, 370, 392, 394–395, 397,
399, 404
Execution Plan Graph 138, 199, 221, 390, 432
Explain 86, 432, 442
Expression Builder 110, 175

F

FACT KEY REPLACE operator 185
fact table 19, 85–86, 103, 106, 171, 185, 203, 205,
395, 445, 479–480, 488, 493–494, 504, 511
facts 10, 19, 54, 70, 80, 84–85, 103–104, 106, 113,
117, 129, 445
federation 157, 236
File export operator 161
file export operator 161–162, 194
file import 155, 158, 192
File import operator 158
File wait operator 216
flat file 51, 146, 149, 192, 392–393, 424, 428, 453,
530
flat files 17, 138, 148, 161, 193, 233
Framework 9, 140
framework xiii, 1, 7–11, 14, 32, 53, 55, 230,
326–327
FTP 210, 216, 389, 392–393, 403, 405, 408, 411,
422

G

Generating code 229
GROUP BY operator 177
Group by operator 177
GUI 32, 503, 506, 517
guided analysis 319

H

hash partitioning 470, 493
hashing algorithm 472–473
hierarchies 10, 20, 54, 85, 88, 103, 110, 112–113,
116, 241, 338, 445
Hierarchy 70, 85, 112, 116, 445–446

hierarchy 41, 70, 85, 111–112, 114, 116–117,
242–243, 387, 444–446
high availability 232, 467–468, 508
HOLAP 80–81, 83, 338
HP-UX 466, 508
Hybrid OLAP 81

I

IBM AIX 508
IBM WebSphere DataStage 230
immediate refresh 123
Impact analysis 77
impact analysis 15, 77
Import 49, 132–133, 148, 346, 451
importing 49, 104, 133, 235, 453, 462
independent data marts 7
indexes 25, 54, 58, 60, 70, 86, 121, 130, 385, 441,
470, 475, 477–479, 481, 484, 486–487, 489, 492,
494, 500, 503, 505–506, 517, 522, 526, 531
Information as a service 2, 6
information as a service 3, 6
Information Engineering 43, 59
Information Integration 230, 236, 539
information integration 3, 539
Information Management xiv–xvi
Information On Demand 3
information on demand 31
information warehouse 6–7
InLine Analytics xiii, 21, 316
Inline Analytics 12–13, 20
innovation 485
insert data flow 166
Integrated Development Environment 14, 349
Integrating DataStage 235
INTERSECTION 183
IT xiv–xv, 3, 80, 82, 316, 360
Iterator operator 217
IXF 158

J

J2EE 10, 12, 22, 140, 317–318, 325–327, 329,
333, 337, 348, 355, 369, 457–458, 462
Java 23, 145, 193, 199, 233, 250–251, 326–327,
330–331, 348, 352, 355–357, 369, 374, 381–383,
397–398, 400–401, 405, 448, 458
Java editor 349, 356
Java Server Faces 369
JavaBean 328

JavaBeans 333
JDBC 43, 50, 145, 199, 233–234, 368, 370, 380,
383, 390, 396, 399, 406, 418
Join xvii, 70, 85, 108, 177, 445, 475, 491
join 20, 85, 105, 108–109, 178, 204, 445, 474–475,
480, 488, 493

K

KEY LOOKUP operator 180
Key Lookup operator 149, 180
key performance indicators 319, 321
KPI 4, 319

L

latency 492
Lifecycle 30
lifecycle 29–31, 48, 54, 57–58, 137–138, 222, 224,
413
Linux 12, 25, 27, 112–113, 376, 382–383, 442,
466–467, 496, 500, 508–509, 535
logging 212, 381–382, 403, 417, 435, 438, 479
logical model 58

M

maintenance costs 232
Master Data Management 541
master data management 8, 541
Materialized Query Tables 20, 81, 83, 368, 489,
503
materialized query tables 489, 503
MDC 130, 384, 476–481, 484–486, 488, 503,
505–507
 also see Multidimensional Clustering
MDC table 477–478, 481
Measure 70, 85, 106
measure 106, 294, 517
Metadata 92, 94, 132–134, 330, 339, 342, 463
metadata xiii, 9, 13–15, 20, 22–24, 27, 29–30, 34,
49, 54, 82, 84, 87, 89–90, 94, 96, 104, 119, 129,
132, 134–135, 140, 142, 147, 154, 197, 199, 219,
221, 229, 231, 234, 241, 330, 336, 339, 341, 346,
368, 371, 374, 389–390, 394, 402, 410, 414, 417,
439, 444, 446, 455–456, 458, 519
metadata bridge 339
metadata catalog tables
 create 95
metadata objects 84, 95, 132, 346, 446

Microsoft 466, 508
Mining flow operator 213
Mining Flows 277
Mining flows 18, 138, 252, 262
mining flows 10, 19, 29, 35, 48–49, 55, 138–139,
141, 147, 205, 211–212, 220, 222, 234–235, 237,
252, 263, 268, 276–277, 279, 291, 310
model-based approach 129
Modeling 12–13, 15, 18, 43, 54, 59, 89–90, 140,
243, 248–251, 254, 310, 538
MOLAP 80–83, 113, 122, 338, 446
monotonic 480
MPP 466, 509
MQ Series 234
MQT 24, 83, 86–88, 123, 130, 339, 440–441,
489–492, 494, 505–507, 530
 also see Materialized Query Tables
MQTs 14, 20, 24, 29, 59, 83, 85–88, 121–124,
129–130, 340, 343, 346, 368, 438, 443, 489,
491–493, 495, 503, 506–507, 513, 530–531
Multidimensional 80, 122, 130, 485, 500
multidimensional 16, 21, 80, 84, 231, 240, 317,
319–320, 326, 335, 338–340, 345, 455–456, 458,
462, 485, 493
multidimensional analysis 318, 335
Multidimensional Clustering 130, 476, 480,
485–486, 500, 503
Multidimensional clustering 476
multidimensional clustering 480, 503
multidimensional data 340, 462
Multivariate 50, 255, 257, 260

N

natural key 172, 185
near real-time 82
nickname 157
Notification operators 210, 218

O

ODBC 383
ODBC/JDBC 383
OLAP xiii–xiv, 3–5, 7–14, 19, 21–23, 25, 29–30, 35,
38, 48–49, 54, 59, 70, 79–82, 84–85, 87, 89, 91–92,
94–99, 103–104, 110, 113, 117–118, 121–122, 129,
131–133, 135, 172, 234, 238, 251, 338–341,
345–346, 348, 358, 365–369, 371–372, 378–380,
386, 438–441, 443–444, 446, 455–457, 462–463
OLAP Center 89

- OLAP databases 80, 340
- OLAP objects 29, 35, 90, 92, 94, 103, 117–118, 133, 444
- on demand 31
- Online Analytical Processing 3, 79, 172
- operational data 241, 290, 315, 389
- operational systems 5
- Operator ports 150
- operator ports 54, 208
- Optimization 24, 70, 82, 87–88, 117, 121–123, 129, 131, 339, 368, 386, 438–441, 443, 446
- optimization 10, 13, 20, 69, 89, 117, 122–123, 129, 346, 440–441, 446, 483, 509
- Optimization Advisor 87, 121, 123, 129, 131, 441, 446
- optimize 3, 10, 12, 20, 54, 439, 474, 500, 503–504
- ORDER BY operator 183
- Outline View 44–45

P

- parallelism 82, 233, 470, 484, 493
- partitioned database 470–471, 532
- partitioned storage model 471
- partitioning key 472–476, 486, 488–489, 493, 507
- Perl 523
- persistent table 192
- Perspective 38
- Perspectives 37
- perspectives 37–39, 41, 53, 152, 209
- physical data model 13, 15, 21, 54, 59, 65, 72, 75, 94, 118–119, 134, 147, 156, 191, 384
- physical data modeling xiii, 14, 29, 38, 54, 58
- physical data models 15, 57, 59, 69, 75, 94, 148
- Physical Model 59–61, 64
- physical model 16, 57–59, 64, 69, 72, 74, 78, 94, 366, 384
- PIVOT operator 187
- Platform 29, 32, 138
- Portal 23, 353, 361, 364, 540
- Portlet applications 363
- Portlets 361, 363
- Predictive Model Markup Language 244, 247, 447
- Problems View 47, 72
- process management 327
- ProfileStage 234, 539
- Project Explorer 14, 42–43, 46, 49, 59–61, 64, 66, 69, 72, 75, 91–92, 94, 105–106, 108, 110–112, 114, 117, 132, 134, 147, 151, 223, 276

- Properties View 34, 46–47, 50, 68–69, 76, 92, 108–109, 111, 113, 115–117
- Property Compare 76

Q

- quality of data 3, 7
- queries 12, 20, 23, 25, 81, 83, 85–86, 88, 121–122, 131, 193, 238, 240, 335, 340–341, 343, 438–439, 441–442, 445, 463, 470, 474–475, 477, 479–481, 483, 486–488, 490–491, 499–500, 504–505, 509, 511, 513–518, 520, 522–532, 534–535
- Query xiii, xvi, 10, 13, 20, 25, 81, 83, 85–86, 88, 130, 350, 368, 398, 400–401, 463, 480, 486, 489, 503–504, 511–531, 534–535, 540
- query 10, 12–13, 20, 22, 25, 80–81, 83, 85–86, 114, 122, 129–130, 238, 247, 315, 317, 320, 335, 339, 341–343, 351, 358, 438, 440, 442, 446, 455–456, 463, 466–467, 469, 476, 478, 480, 483, 489–491, 493–495, 498, 502–504, 509, 511, 513–517, 520–522, 525–531, 533–535
- Query Patroller
 - also see DB2 Query Patroller
- Query Patroller administrators 523
- Query Patroller Center 25, 517, 519–523, 528, 530, 534
- Query Patroller server 518, 520, 523–525, 527
- Query Re-write 81

R

- Range Partitioning 81, 481
- Rational Application Developer 349, 352
- Rational Software Architect 354
- Real-time 82, 234, 246, 319–320, 325
- real-time 18, 21, 24, 31, 82, 231, 240, 246, 315, 320–321, 325, 454, 456, 492
- real-time analytics 82
- Redbooks Web site 559
 - Contact us xvii
- referential integrity
 - informational 103
- REFRESH DEFERRED 125–126, 128, 494
- REFRESH IMMEDIATE 507
- Relational OLAP 80
- relational OLAP 54
- relationship diagrams 65
- replication 155, 159–160, 233
 - SQL 160
- reporting tools 246, 248, 317, 456

reports 80, 273, 309, 319, 343, 358, 456, 490, 492, 513, 517
Resource Navigator 42
reverse engineering 60, 62, 64, 66, 75, 100, 118–119, 234
REXX 523
ROLAP 80–81
runstats utility 131
Runtime EPG 199

S

SAS 251, 269, 358, 504
scalability 80–81, 231, 233, 467–469, 509
schedules 145, 372, 421–426
Schema 62, 444, 498
schema 10, 13, 19, 43, 54, 59, 62, 65–66, 69, 72, 80, 84, 90, 92, 94, 98, 103, 106, 109, 114, 121, 124, 145, 156, 160, 162, 171, 196, 202, 205, 224, 244, 254, 287, 312, 371, 412, 424, 444, 480, 488, 493, 505, 518, 525, 530
Schemas 62, 96
schemas 13, 30, 43, 48, 50, 54, 59, 62–63, 69, 72, 84, 98, 101, 145, 152, 194, 222, 232
scorecards 368
Scoring 246, 248, 250–251, 254, 310, 447–448, 539
scoring 8, 10, 12–13, 17–18, 24, 55, 153, 246–247, 251, 268–269, 273, 285–287, 293, 310–311, 448–449, 454
Select list operator 176
select list operator 176, 204
Sequences 248, 251, 270, 274, 304
services
 infrastructure 9, 32, 369
SMP 466–467, 470, 509
snowflake schema 84, 103
SOA 230
Solaris 466, 508
Solution 9
solution 9, 30–31, 54, 57, 81, 159, 229, 236, 240, 245, 263, 323, 333, 365–368, 370, 372, 431, 438, 468–469, 484, 495, 505, 535
Solutions 30, 48, 541
solutions xiii, xv, xvii, 2, 8–11, 14, 29–31, 55, 57, 81, 237, 247, 317, 319, 325–328, 330, 333, 343, 365–366, 537, 541
Source operators 149, 155
source operators 149, 152, 155–156

SPLITTER operator 179, 182
Splitter operator 179, 264
spreadsheets 358
SQL Expression Builder 110, 175
SQL Merge operator 170
SQL Merge target operator 170
SQL Replication 159
SQL replication 160
SQL Warehousing xiii, 10, 14, 16–17, 23, 29, 137–138, 174, 230, 232, 365–372, 378, 385, 387–394, 402, 405, 407–408, 413–415, 421, 423–424, 427, 431, 456
SQL Warehousing Tool 16, 138
SQW 10, 16, 18, 24, 137–141, 143–146, 148, 160, 205, 210, 212, 214, 219, 226, 230, 232, 234–235, 252, 262, 365, 367–370, 372–374, 389, 392, 405, 412–413, 430, 543, 545
 also see DWE
SQW flow operators 212
staging area 148
staging table 149, 159, 170–171, 173
staging tables 15, 159, 492
star schema 20, 84, 103
Static HTML 330
statistics 24, 51, 88, 130, 141, 238, 250, 259–260, 268, 298, 304, 368, 387, 390, 414, 417, 429–430, 432–434, 513, 522, 531
stored procedure 69, 95, 118, 339, 448, 529
stored procedures 199, 234, 248, 253, 359, 380
Structural Compare 75–76
subflow 195–197, 199, 235
Subflows 195–196
summary tables 124, 439–443, 446, 489
Sun Cluster 467, 508
surrogate keys 185

T

TABLE JOIN operator 178
Table source operator 156
table source operator 154, 156, 165
tablespaces 54, 69, 384, 518, 520, 525, 530
Target operators 161, 268
target operators 148–149, 152, 155, 158, 161–162
temporary table 192, 399, 404
terabytes 470, 509
Tivoli 508, 540
total cost of ownership 11, 468
training 1, 238–239, 243, 245, 249, 264–265,

271–272, 281–282, 291
Transform operators 174
Transformation 12
transformation xiii, 7, 10–13, 16, 55, 137–139, 141,
145, 148, 174–176, 193, 229–232, 235–236, 252
triggers 371, 518

U

Undeployment EPG 199
UNION 183
UNION operator 183
unique indexes 494
Univariate 50, 255, 257
UNIX 112–113, 466–467, 496, 500, 535
Unload 475, 540
unpivot operator 188–189
unstructured data 2, 8
update data flow 166
upsert 161, 170
User-defined functions 176
user-defined functions 248, 253

V

Validation 197, 219
validation 54, 78, 121, 148, 178, 197–198, 200,
205, 209, 219–221, 231, 243, 265, 268, 284, 287
Variables 222, 224, 411, 415, 420
Variables Manager 223–224
Views xiii, xv, 25, 41, 59, 79, 82, 89, 91, 94,
345–346
Virtual Table 154
virtual table 149, 154–155, 158
virtualization 3
Visualization 25, 248, 250, 447

W

Web Browsers 4
Web-based xiii, 4, 6, 8–10, 12, 21, 23, 140, 144,
326, 358, 367, 523, 531
WebSphere Application Server 11–12, 22, 25, 140,
233, 353, 355, 365, 368–370, 375, 381–382,
392–393, 457, 541
Websphere Application Server 10
WebSphere DataStage 230, 232, 236
WebSphere Information Integration 230
WebSphere Information Integrator 157, 236, 539,
541

WebSphere MQ 324, 539
WebSphere Portal 361, 364, 541
WebSphere Portal Server 362
WebSphere Studio 355
WHERE CONDITION operator 182
Windows 12, 25, 27, 94–95, 112–113, 215, 344,
356, 376, 382–383, 438, 442, 466–467, 496, 500,
508–509, 524, 527, 535
Windows Server 2003 25, 27
Wizard 54, 61, 71–72, 103, 133, 175, 227, 525
wizard 20, 54, 60–63, 66, 69, 72, 90, 94, 96, 103,
117–118, 120, 142–143, 196, 228–229, 248, 276,
524, 527
wizards 10, 53–54, 90, 193, 226, 248

X

XML 75, 90, 92, 94, 96, 133, 193, 235, 244, 247,
251, 330, 339, 390, 411, 417, 419, 439, 453–454,
543, 545
XML rendering 331



Redbooks

Leveraging DB2 Data Warehouse Edition for Business Intelligence



Redbooks

Leveraging DB2 Data Warehouse Edition for Business Intelligence

Building complete and robust business intelligence solutions

Integrated tools for fast and easy deployment

Standards based for flexibility and change

In this IBM Redbook we describe and discuss DB2 Data Warehouse Edition (DWE) Version 9.1, a comprehensive platform offering with functionality to build a business intelligence infrastructure for analytics and Web-based applications, and best practices for deployment. DB2 DWE integrates core components for data warehouse construction and administration, data mining, OLAP, and InLine Analytics and reporting. It extends the DB2 data warehouse with design-side tooling and runtime infrastructure for OLAP, data mining, InLine Analytics, and intra-warehouse data movement and transformation, on a common platform based on DB2 and WebSphere. The platform pillars are based on the technology of DB2, Rational Data Architect (for physical data modeling only), the SQL Warehousing Tool, Intelligent Miner, DB2 Cube Views, and Alphablox. DWE includes an Eclipse-based design environment, DWE Design Studio, that integrates the DWE products (with the exception of Alphablox and Query Patroller) with a common framework and user interface. The new SQL Warehousing Tool enables visual design of intra-warehouse, table-to-table data flows and control flows using generated SQL. DB2 Alphablox is the tool for developing custom applications with embedded analytics-based visual components. DWE enables faster time-to-value for enterprise analytics, while limiting the number of vendors, tools, skill sets and licenses required.

**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