Patterns: Information Aggregation and Data Integration with DB2 Information Integrator

- Information Aggregation and Data Integration patterns
- DB2 Information Integration architecture overview
- Customer Insight scenario

Nagraj Alur
YunJung Chang
Barry Devlin
Bill Mathews
John Matthews
Sreeram Potukuchi
Uday Sai Kumar

ibm.com/redbooks
Note: Before using this information and the product it supports, read the information in “Notices” on page xix.

First Edition (September 2004)

This edition applies to Version 8, Release 1, Modification 1 of IBM DB2 Information Integrator (product number 5724-C74).

© Copyright International Business Machines Corporation 2004. 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

Figures ......................................................................................... ix
Tables ......................................................................................... xv
Examples ..................................................................................... xvii
Notices ......................................................................................... xix
Trademarks .................................................................................. xx
Preface ........................................................................................... xxi
The team that wrote this redbook ................................................... xxii
Become a published author ........................................................... xxiv
Comments welcome ................................................................. xxv

Chapter 1. Introduction to Patterns for e-business ......................... 1
1.1 Role of the IT architect ............................................................ 2
1.2 The Patterns for e-business layered asset model ....................... 2
1.3 How to use the Patterns for e-business ..................................... 4
  1.3.1 Select a Business, Integration, or Composite pattern, or a Custom
design ....................................................................................... 4
  1.3.2 Selecting Application patterns ............................................ 9
  1.3.3 Review Runtime patterns .................................................. 11
  1.3.4 Review Product mappings ................................................ 13
  1.3.5 Review guidelines and related links .................................... 14
1.4 Summary .................................................................................. 15

Chapter 2. DB2 Information Integration architecture overview ......... 17
2.1 Introduction ............................................................................. 18
2.2 Current business trends ........................................................... 18
  2.2.1 Grid computing .................................................................. 19
  2.2.2 Data federation ................................................................. 20
  2.2.3 Information integration .................................................... 21
2.3 IBM’s DB2 Information Integration overview ............................ 21
  2.3.1 Data consolidation or placement ....................................... 23
  2.3.2 Distributed access (federation) ......................................... 24
  2.3.3 DB2 Information Integrator portfolio ................................. 24
2.4 DB2 Information Integrator V8.1 .............................................. 26
  2.4.1 DB2 II V8.1 overview ...................................................... 26
  2.4.2 DB2 II components ........................................................ 34
Patterns: Information Aggregation and Data Integration with DB2 Information Integrator

Appendix C. Data models, table/file definitions, et al used in the DFC

Customer Insight business solution

Introduction .................................................. 232
Operational systems’ details .............................. 232
Checkings/Savings ......................................... 232
Credit Card ................................................. 240
Rewards .................................................... 242
Brokerage ................................................... 246
Loans ...................................................... 252
Data warehouse details ..................................... 257
Initial load of the data warehouse ....................... 266
Incremental update of the data warehouse ........................................... 275
Data mart details ...................................................................................... 276
CIIS details .................................................................................................. 277
  Initial load of the CIIS ............................................................................. 280
  Incremental update of CIIS objects ......................................................... 294

Related publications .................................................................................. 299
IBM Redbooks ............................................................................................ 299
Other publications ...................................................................................... 301
Online resources ......................................................................................... 301
How to get IBM Redbooks .......................................................................... 301
Help from IBM ............................................................................................. 301

Index .......................................................................................................... 303
## Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>The Patterns for e-business layered asset model</td>
<td>3</td>
</tr>
<tr>
<td>1-2</td>
<td>The four primary Business patterns</td>
<td>5</td>
</tr>
<tr>
<td>1-3</td>
<td>Integration patterns</td>
<td>6</td>
</tr>
<tr>
<td>1-4</td>
<td>Patterns representing a Custom design</td>
<td>7</td>
</tr>
<tr>
<td>1-5</td>
<td>Self-Service, Information Aggregation, Access &amp; Application Integration</td>
<td>7</td>
</tr>
<tr>
<td>1-6</td>
<td>Composite patterns</td>
<td>8</td>
</tr>
<tr>
<td>1-7</td>
<td>Self-Service::Directly Integrated Single Channel</td>
<td>9</td>
</tr>
<tr>
<td>1-8</td>
<td>Self-Service::Decomposition</td>
<td>10</td>
</tr>
<tr>
<td>1-9</td>
<td>Directly Integrated Single Channel application pattern::Runtime pattern</td>
<td>12</td>
</tr>
<tr>
<td>1-10</td>
<td>Directly Integrated Single Channel application pattern::Runtime pattern</td>
<td>13</td>
</tr>
<tr>
<td>1-11</td>
<td>Directly Integrated Single Channel application pattern</td>
<td>14</td>
</tr>
<tr>
<td>2-1</td>
<td>Data federation concept</td>
<td>20</td>
</tr>
<tr>
<td>2-2</td>
<td>Overview of IBM information products for information integration</td>
<td>25</td>
</tr>
<tr>
<td>2-3</td>
<td>Data federation technology</td>
<td>27</td>
</tr>
<tr>
<td>2-4</td>
<td>DB2 Information Integrator data federation using wrappers</td>
<td>28</td>
</tr>
<tr>
<td>2-5</td>
<td>DB2 Information Integration V8.1 components</td>
<td>36</td>
</tr>
<tr>
<td>2-6</td>
<td>DB2 Information Integrator on a Windows platform</td>
<td>38</td>
</tr>
<tr>
<td>2-7</td>
<td>Basic steps in configuring a federated system</td>
<td>40</td>
</tr>
<tr>
<td>2-8</td>
<td>MQTs</td>
<td>55</td>
</tr>
<tr>
<td>2-9</td>
<td>MQTs on nicknames</td>
<td>56</td>
</tr>
<tr>
<td>3-1</td>
<td>Business, Integration, and related Application patterns</td>
<td>61</td>
</tr>
<tr>
<td>3-2</td>
<td>Data Integration and Information Aggregation patterns</td>
<td>63</td>
</tr>
<tr>
<td>3-3</td>
<td>Federation application pattern</td>
<td>67</td>
</tr>
<tr>
<td>3-4</td>
<td>Federation runtime pattern</td>
<td>68</td>
</tr>
<tr>
<td>3-5</td>
<td>Federation: Cache variation application pattern</td>
<td>70</td>
</tr>
<tr>
<td>3-6</td>
<td>Population application pattern</td>
<td>74</td>
</tr>
<tr>
<td>3-7</td>
<td>Population runtime pattern</td>
<td>75</td>
</tr>
<tr>
<td>3-8</td>
<td>Population: Multi Step variation application pattern</td>
<td>77</td>
</tr>
<tr>
<td>3-9</td>
<td>Population: Multi Step variation runtime pattern</td>
<td>79</td>
</tr>
<tr>
<td>3-10</td>
<td>Population: Multi Step Gather variation application pattern</td>
<td>81</td>
</tr>
<tr>
<td>3-11</td>
<td>Population: Multi Step Gather variation runtime pattern</td>
<td>84</td>
</tr>
<tr>
<td>3-12</td>
<td>Population: Multi Step Process variation application pattern</td>
<td>86</td>
</tr>
<tr>
<td>3-13</td>
<td>Population: Multi Step Process variation runtime pattern</td>
<td>88</td>
</tr>
<tr>
<td>3-14</td>
<td>Population: Multi Step Federated Gather variation application pattern</td>
<td>89</td>
</tr>
<tr>
<td>3-15</td>
<td>Population: Multi Step Federated Gather variation runtime pattern</td>
<td>90</td>
</tr>
<tr>
<td>3-16</td>
<td>Two-way Synchronization application pattern</td>
<td>95</td>
</tr>
<tr>
<td>3-17</td>
<td>Two-way Synchronization: Multi Step variation application pattern</td>
<td>97</td>
</tr>
<tr>
<td>3-18</td>
<td>User Information Access application pattern</td>
<td>101</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>B-3</td>
<td>Server definition for NET8</td>
<td>170</td>
</tr>
<tr>
<td>B-4</td>
<td>Oracle - Create Server dialog</td>
<td>171</td>
</tr>
<tr>
<td>B-5</td>
<td>Oracle - Create Server - Settings tab.</td>
<td>172</td>
</tr>
<tr>
<td>B-6</td>
<td>User mapping for CHKSVG</td>
<td>175</td>
</tr>
<tr>
<td>B-7</td>
<td>Oracle - Create User Mappings</td>
<td>176</td>
</tr>
<tr>
<td>B-8</td>
<td>Oracle - Create User Mappings - Settings</td>
<td>177</td>
</tr>
<tr>
<td>B-9</td>
<td>Nickname creation for CHKSVG</td>
<td>178</td>
</tr>
<tr>
<td>B-10</td>
<td>Oracle - Create Nicknames</td>
<td>179</td>
</tr>
<tr>
<td>B-11</td>
<td>Oracle - Create nicknames - Add.</td>
<td>179</td>
</tr>
<tr>
<td>B-12</td>
<td>Create nickname - List selected table</td>
<td>180</td>
</tr>
<tr>
<td>B-13</td>
<td>Oracle - Sample contents of nickname</td>
<td>181</td>
</tr>
<tr>
<td>B-14</td>
<td>Results of nickname access</td>
<td>182</td>
</tr>
<tr>
<td>B-15</td>
<td>Determining userid of DB2 II</td>
<td>184</td>
</tr>
<tr>
<td>B-16</td>
<td>XML - Create Wrapper</td>
<td>184</td>
</tr>
<tr>
<td>B-17</td>
<td>XML - Create Server</td>
<td>185</td>
</tr>
<tr>
<td>B-18</td>
<td>XML - Create nickname</td>
<td>187</td>
</tr>
<tr>
<td>B-19</td>
<td>XML - Create nickname Add screen</td>
<td>188</td>
</tr>
<tr>
<td>B-20</td>
<td>Add column and data type details</td>
<td>189</td>
</tr>
<tr>
<td>B-21</td>
<td>Add column settings</td>
<td>189</td>
</tr>
<tr>
<td>B-22</td>
<td>Completed list of columns with data types</td>
<td>190</td>
</tr>
<tr>
<td>B-23</td>
<td>Nickname settings</td>
<td>190</td>
</tr>
<tr>
<td>B-24</td>
<td>XML - Create nickname window</td>
<td>191</td>
</tr>
<tr>
<td>B-25</td>
<td>Nickname creation using SQL</td>
<td>191</td>
</tr>
<tr>
<td>B-26</td>
<td>ODBC Data Source Administrator</td>
<td>193</td>
</tr>
<tr>
<td>B-27</td>
<td>Create New Data Source</td>
<td>193</td>
</tr>
<tr>
<td>B-28</td>
<td>ODBC Create New Data Source</td>
<td>194</td>
</tr>
<tr>
<td>B-29</td>
<td>CrossAccess32 Communications Protocol</td>
<td>194</td>
</tr>
<tr>
<td>B-30</td>
<td>CrossAccess32 data source configuration</td>
<td>195</td>
</tr>
<tr>
<td>B-31</td>
<td>CrossAccess32 data source configuration - Advanced tab</td>
<td>196</td>
</tr>
<tr>
<td>B-32</td>
<td>ODBC wrapper - Create Wrapper</td>
<td>197</td>
</tr>
<tr>
<td>B-33</td>
<td>ODBC wrapper - Create wrapper settings</td>
<td>197</td>
</tr>
<tr>
<td>B-34</td>
<td>ODBC wrapper - Create Server</td>
<td>198</td>
</tr>
<tr>
<td>B-35</td>
<td>ODBC wrapper - Create server settings</td>
<td>199</td>
</tr>
<tr>
<td>B-36</td>
<td>ODBC data source - Create User Mapping dialog</td>
<td>201</td>
</tr>
<tr>
<td>B-37</td>
<td>ODBC data source - Create User Mapping settings</td>
<td>202</td>
</tr>
<tr>
<td>B-38</td>
<td>ODBC wrapper - Create nickname - Main window</td>
<td>203</td>
</tr>
<tr>
<td>B-39</td>
<td>ODBC data source - Add Nickname</td>
<td>203</td>
</tr>
<tr>
<td>B-40</td>
<td>ODBC data source - Create Nicknames</td>
<td>204</td>
</tr>
<tr>
<td>B-41</td>
<td>ODBC data source - SQL for creating the nickname</td>
<td>204</td>
</tr>
<tr>
<td>B-42</td>
<td>ODBC wrapper - Create nickname - Sample Contents</td>
<td>205</td>
</tr>
<tr>
<td>B-43</td>
<td>ODBC wrapper - Create nickname - Sample contents results</td>
<td>206</td>
</tr>
<tr>
<td>B-44</td>
<td>DB2 UDB for z/OS - Create wrapper</td>
<td>208</td>
</tr>
<tr>
<td>B-45</td>
<td>DB2 UDB for z/OS - Create server</td>
<td>209</td>
</tr>
<tr>
<td>Page</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>B-46</td>
<td>DB2 UDB or z/OS - Create Server Settings</td>
<td></td>
</tr>
<tr>
<td>B-47</td>
<td>DB2 UDB for z/OS - Create User Mapping dialog</td>
<td></td>
</tr>
<tr>
<td>B-48</td>
<td>DB2 UDB for z/OS - Create user mapping settings</td>
<td></td>
</tr>
<tr>
<td>B-49</td>
<td>DB2 UDB for z/OS - Nickname creation</td>
<td></td>
</tr>
<tr>
<td>B-50</td>
<td>DB2 UDB for z/OS - Add Nickname</td>
<td></td>
</tr>
<tr>
<td>B-51</td>
<td>DB2 UDB for z/OS - Create Nicknames</td>
<td></td>
</tr>
<tr>
<td>B-52</td>
<td>DB2 UDB for z/OS - SQL for creating the nickname</td>
<td></td>
</tr>
<tr>
<td>B-53</td>
<td>DB2 UDB for z/OS - Sample contents of nickname</td>
<td></td>
</tr>
<tr>
<td>B-54</td>
<td>DB2 UDB for z/OS - Sample Contents results of nickname</td>
<td></td>
</tr>
<tr>
<td>B-55</td>
<td>DB2 UDB for Multiplatforms - Create server</td>
<td></td>
</tr>
<tr>
<td>B-56</td>
<td>DB2 UDB for Multiplatforms - Create Server settings</td>
<td></td>
</tr>
<tr>
<td>B-57</td>
<td>DB2 UDB for Multiplatforms - Create User Mapping dialog</td>
<td></td>
</tr>
<tr>
<td>B-58</td>
<td>DB2 UDB for Multiplatforms - Create User Mapping settings</td>
<td></td>
</tr>
<tr>
<td>B-59</td>
<td>DB2 UDB for Multiplatforms - Nickname creation</td>
<td></td>
</tr>
<tr>
<td>B-60</td>
<td>DB2 UDB for Multiplatforms - Add Nickname</td>
<td></td>
</tr>
<tr>
<td>B-61</td>
<td>DB2 UDB for Multiplatforms - Create Nicknames</td>
<td></td>
</tr>
<tr>
<td>B-62</td>
<td>DB2 UDB for Multiplatforms - SQL for creating the nickname</td>
<td></td>
</tr>
<tr>
<td>B-63</td>
<td>DB2 UDB for Multiplatforms - Sample contents of nickname</td>
<td></td>
</tr>
<tr>
<td>B-64</td>
<td>DB2 UDB for Multiplatforms - Sample contents results of nickname</td>
<td></td>
</tr>
<tr>
<td>C-1</td>
<td>Checkings/Savings operational system data model</td>
<td></td>
</tr>
<tr>
<td>C-2</td>
<td>Credit Card operational system data model</td>
<td></td>
</tr>
<tr>
<td>C-3</td>
<td>Rewards Tracking VSAM file field attributes</td>
<td></td>
</tr>
<tr>
<td>C-4</td>
<td>Rewards Offering Partners VSAM file field attributes</td>
<td></td>
</tr>
<tr>
<td>C-5</td>
<td>Rewards Transactions VSAM file field attributes</td>
<td></td>
</tr>
<tr>
<td>C-6</td>
<td>Rewards Journal VSAM file field attributes</td>
<td></td>
</tr>
<tr>
<td>C-7</td>
<td>Brokerage operational system data model</td>
<td></td>
</tr>
<tr>
<td>C-8</td>
<td>Loans operational system data model</td>
<td></td>
</tr>
<tr>
<td>C-9</td>
<td>Data warehouse data model</td>
<td></td>
</tr>
<tr>
<td>C-10</td>
<td>DataStage Designer - Data warehouse tables</td>
<td></td>
</tr>
<tr>
<td>C-11</td>
<td>DataStage Designer Extract ODBC Stage</td>
<td></td>
</tr>
<tr>
<td>C-12</td>
<td>DataStage Designer Transformer Stage</td>
<td></td>
</tr>
<tr>
<td>C-13</td>
<td>DataStage Designer Load_Credit_Data ODBC Stage</td>
<td></td>
</tr>
<tr>
<td>C-14</td>
<td>DataStage Designer Credit Transactions summary</td>
<td></td>
</tr>
<tr>
<td>C-15</td>
<td>DataStage Designer Credit Transactions EXTRACT ODBC Stage</td>
<td></td>
</tr>
<tr>
<td>C-16</td>
<td>DataStage Designer Credit Transactions EXTRACT Aggregator Stage</td>
<td></td>
</tr>
<tr>
<td>C-17</td>
<td>TRANSFORM Aggregator Stage</td>
<td></td>
</tr>
<tr>
<td>C-18</td>
<td>DataStage Designer Credit Transactions Transformer Stage</td>
<td></td>
</tr>
<tr>
<td>C-19</td>
<td>DataStage Designer - Data mart tables</td>
<td></td>
</tr>
<tr>
<td>C-20</td>
<td>DFC Customer Insight data model (Patterns) in CIIS Workbench</td>
<td></td>
</tr>
<tr>
<td>C-21</td>
<td>DFC Customer Insight data model (Patterns)</td>
<td></td>
</tr>
<tr>
<td>C-22</td>
<td>Incremental update of CIIS</td>
<td></td>
</tr>
<tr>
<td>C-23</td>
<td>Update CIIS Extract Stage</td>
<td></td>
</tr>
<tr>
<td>C-24</td>
<td>Update CIIS Transform Stage</td>
<td></td>
</tr>
</tbody>
</table>
C-25  Update CIIS Load Stage - Update existing rows only  . . . . . . . . . . . . . 297
C-26  Update CIIS Load Stage  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 298
## Tables

- **2-1** Data sources, supported versions and access method .................................. 30
- **2-2** Unsupported data types on specific data sources ......................................... 31
- **2-3** Write operation restrictions on data type on specific data sources ............ 32
- **2-4** Excel write operations limitations .............................................................. 34
- **2-5** Global catalog contents for remote data sources ......................................... 36
- **B-1** Oracle server options .................................................................................. 173
- **B-2** Oracle additional server options ................................................................. 174
- **B-3** ODBC additional server options .................................................................. 200
- **B-4** The DB2 UDB for z/OS system .................................................................... 207
- **B-5** DB2 UDB for z/OS server options ............................................................... 210
- **B-6** DB2 UDB for z/OS additional server options ............................................... 212
Examples

4-1 Snippet of CIIS search code .................................................. 149
4-2 Snippet of data warehouse access code .................................. 151
4-3 Snippet of operational systems’ access code ......................... 152
B-1 The tnsnames.ora file .......................................................... 167
B-2 Create wrapper statement for Oracle. .................................... 169
B-3 Oracle - Create server statement .......................................... 172
B-4 Oracle - Create user mapping statement .............................. 177
B-5 Oracle - Create nickname statements .................................... 180
B-6 XML - Create wrapper statement ......................................... 185
B-7 XML - Create server statement .......................................... 186
B-8 Content of the RewardsXML file ......................................... 186
B-9 ODBC wrapper - Create wrapper statement .......................... 198
B-10 ODBC wrapper - Create server statement ........................... 199
B-11 ODBC data source - Create user mapping statements ............ 202
B-12 DB2 UDB for z/OS - Create wrapper statements .................... 208
B-13 DB2 UDB for z/OS - Create server statement ....................... 211
B-14 DB2 UDB for z/OS - Create user mapping statement ............... 214
B-15 DB2 UDB for z/OS - Create server statements ....................... 222
B-16 DB2 UDB for Multiplatforms - Create user mapping statements 224
C-1 Checkings/Savings DDL ....................................................... 233
C-2 Credit Card DDL ................................................................. 241
C-3 Rewards Lookup XML file .................................................. 245
C-4 Brokerage DDL ................................................................. 246
C-5 Loans DDL ................................................................. 252
C-6 Data warehouse DDL .......................................................... 258
C-7 Sample extract and load into temporary tables of a single customer 281
C-8 Java™ program for inserting into the CIIS ............................. 283
Notices

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

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

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

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

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

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

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

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

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

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

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

AIX®, CICS®, DataPropagator™, DB2 Universal Database™, DB2®, DFS™, DRDA®, e-business on demand™, Everyplace®, Informix®, IBM®, ibm.com®, IMS™, iSeries™, Lotus®, MQSeries®, OS/2®, OS/390®, Redbooks™, Redbooks (logo)™, TXSeries®, WebSphere®, z/OS®, zSeries®

The following terms are trademarks of other companies:

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

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

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

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

This IBM Redbook documents and prototypes the role of DB2® Information Integrator technologies and architecture in IBM®'s Patterns for e-business using a typical customer insight e-business scenario. It is aimed at an audience of IT architects and data integration professionals responsible for developing e-business solutions that integrate processes and data from multiple distributed heterogeneous systems.

This redbook documents the step-by-step patterns approach to implementing a customer insight e-business scenario solution. At each level in the patterns hierarchy, each available pattern is evaluated and chosen before drilling down to the next lower layer where the pattern evaluation and selection process is repeated. The final drill down leads to product selection and implementation information.

This book is organized as follows:

- **Chapter 1** provides an introduction to the Patterns for e-business, and describes the process of applying pattern approaches to implement successful e-business solutions through the reuse of components and solution elements from proven successful experiences.

- **Chapter 2** discusses the business requirements driving the need for information integration, and IBM’s response to this demand with its DB2 Information Integrator portfolio of products. It introduces the IBM DB2 Information Integrator portfolio of products and focuses on DB2 Information Integrator since it is used in the customer insight solution.

- **Chapter 3** describes the various application and runtime patterns identified to date for Data Integration and Information Aggregation as defined in “Patterns for e-business”.

- **Chapter 4** describes a hypothetical customer insight scenario involving a financial services company named Druid Financial Corporation (DFC) that provides banking, credit card and brokerage services to its clients. This chapter describes a step by step patterns approach to implementing the DFC customer insight e-business scenario solution.

- **Appendix A** provides an overview of IBM’s Customer Information Integration Solution (CIIS) used to implement the operational data store (ODS) in the DFC Customer Insight scenario.

- **Appendix B** describes DB2 II configuration of data sources used in the DFC customer insight insight scenario.
Appendix C lists the table definitions and file definitions used in the DFC customer insight scenario.

The team that wrote this redbook

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

Nagraj Alur is a Project Leader with the IBM International Technical Support Organization, San Jose Center. He holds a Masters Degree in Computer Science from the Indian Institute of Technology (IIT), Mumbai, India. He has more than 28 years of experience in DBMSs, and has been a programmer, systems analyst, project leader, consultant, and researcher. His areas of expertise include DBMSs, data warehousing, distributed systems management, and database performance, as well as client/server and Internet computing. He has written extensively on these subjects and has taught classes and presented at conferences all around the world. Before joining the ITSO in November 2001, he was on a two-year assignment from the Software Group to the IBM Almaden Research Center, where he worked on Data Links solutions and an eSourcing prototype.

YunJung Chang is a DB2 II Field Technical Sales Support (FTSS) Engineer with IBM Korea. She supports DB2 and Informix®. She has three years of experience in consulting, configuring, and implementing DB2 II, and DB2 for UNIX/NT/Linux; and seven years of experience with Informix. Her areas of expertise include database federation, homogeneous and heterogeneous database replication, and Information Integration. She is an IBM Certified Solutions Expert of DB2 UDB, and holds a Bachelors degree in Computer Science from DongDuk University, Korea.

Barry Devlin is among the foremost authorities in the world on data warehousing. He was responsible for the definition of IBM's warehouse architecture in the mid 1980s and is a widely respected consultant and lecturer on this and related topics, and author of a comprehensive book on the subject "Data Warehouse - from Architecture to Implementation". Barry currently works on DB2 Information Integrator, where he is part of the team defining IBM's information integration architecture, as well as industry solutions and applications of the technology. Barry has been in the IT business for 20 years, mainly with IBM's Software and Solutions Centre in Dublin. He is a Council Member of the IBM Academy of Technology and an IBM Distinguished Engineer.

Bill Mathews is a Consulting IT Architect in the IBM Financial Services Sector for the Americas and is the architectural lead for Information Integration. He has over 25 years of experience in the IT industry and is an IBM Certified Consultant.
Bill has extensive experience in application and systems programming, analysis, consulting, application development management, and architecting large scale complex systems. His areas of expertise are Information Integration, Enterprise Application Integration, and Web application development. Bill holds a Bachelors of Science degree in Computer Science from Hofstra University and a Masters of Business Administration degree from Union College.

John Matthews is a Technical Services Consultant with the IBM Financial Services Solution Centre in Dublin, Ireland. He has more than 19 years of experience in Enterprise systems, and has been a programmer, systems analyst, project leader and consultant. His areas of expertise include Enterprise systems and software, and Enterprise Client Information Files (CIIS) in both the banking and insurance industries. He has installed CIIS, consulted on CIIS, and taught CIIS classes all around the world.

Sreeram Potukuchi is a Data Architect with Werner Enterprises, Omaha, NE. During his seven-year association with DB2, he has been a Database Developer, DBA, and Architect. His areas of expertise include Database Design/Development, Administration, Performance tuning, Data Warehousing, and Distributed Systems Management. He is an IBM Certified Specialist - DB2 V7.1 User, IBM Certified Solutions Expert - DB2 UDB V7.1 Database Administration for UNIX®, Linux, Windows® and OS/2®, IBM Certified Advanced Technical Expert - DB2 for Clusters, IBM Certified Database Administrator - DB2 UDB V8.1 for Linux, UNIX and Windows, IBM Certified Advanced Database Administrator – DB2 Universal Database™ V8.1 for Linux, UNIX, and Windows. He also teaches classes on DB2/UDB.

Uday Sai Kumar has been with IBM for the past four years, and has seven years of IT experience in object-oriented, middleware, and distributed technologies. He delivers technical talks within IBM and outside IBM on recent technological developments like Data Warehousing and OLAP Services, Distributed Internet Application Architecture, SQL Server Accelerator for Business Intelligence, Data Access Technologies, TXSeries® D-CICS, .Net, e-business on demand™, and Autonomic Computing. He has a Masters degree in Computer Applications from Osmania University and a Post Graduate Diploma in Business Administration in Operations Management from Symbiosis University. Uday is a certified IBM DB2 Specialist and Microsoft® Certified Solutions Developer.

We are extremely grateful to Jonathan Adams and David Bryant for their support and constructive comments throughout this project. We would also like to thank Michele Galic and Bill Tworek (SG24-6881) for allowing us to borrow heavily from their redbooks.
Thanks to the following people for their contributions to this project:

Ramani Ranjan Routray  
IBM Almaden Research Center

Raj Datta  
Jacques Labrie  
Robert Montroy  
Micks Purnell  
Cindy Saracco  
Guenter Sauter  
Mel Zimowski  
IBM Silicon Valley Laboratory

Isaac Allotey-Pappoe  
IBM Sweden

Jonathan Adams  
David Bryant  
IBM UK

Chris Delgado  
Delgado Enterprises, Inc.

Jayanti Krishnamurthy  
Interval International, Miami, Florida

Emma Jacobs  
Bart Steegmans  
International Technical Support Organization, San Jose Center

Richard Conway  
Julie Czubik  
International Technical Support Organization, Poughkeepsie Center

Becomes a published author

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

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.
Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

► Use the online Contact us review redbook form found at:
  ibm.com/redbooks

► Send your comments in an Internet note to:
  redbook@us.ibm.com

► Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. QXXE  Building 80-E2
  650 Harry Road
  San Jose, California 95120-6099
Chapter 1. Introduction to Patterns for e-business

This redbook is part of the Patterns for e-business series. In this introductory chapter we provide an overview of how IT architects can work effectively with the Patterns for e-business.

The topics covered include:
- Role of the IT architect
- The Patterns for e-business layered asset model
- How to use the Patterns for e-business
1.1 Role of the IT architect

The role of the IT architect is to evaluate business problems and to build solutions to solve them. To do this, the architect begins by gathering input on the problem, an outline of the desired solution, and any special considerations or requirements that need to be factored into that solution. The architect then takes this input and designs the solution. This solution can include one or more computer applications that address the business problems by supplying the necessary business functions.

To enable the architect to do this better each time, we need to capture and reuse the experience of these IT architects in such a way that future engagements can be made simpler and faster. We do this by taking these experiences and using them to build a repository of assets that provides a source from which architects can reuse this experience to build future solutions, using proven assets. This reuse saves time, money, and effort, and in the process helps ensure delivery of a solid, properly architected solution.

The IBM Patterns for e-business help facilitate this reuse of assets. Their purpose is to capture and publish e-business artifacts that have been used, tested, and proven. The information captured by them is assumed to fit the majority, or 80/20, situation.

The IBM Patterns for e-business are further augmented with guidelines and related links for their better use.

The layers of patterns plus their associated links and guidelines allow the architect to start with a problem and a vision for the solution, and then find a pattern that fits that vision. Then by drilling down using the patterns process, the architect can further define the additional functional pieces that the application will need to succeed. Finally he can build the application using coding techniques outlined in the associated guidelines.

1.2 The Patterns for e-business layered asset model

The Patterns for e-business approach enables architects to implement successful e-business solutions through the reuse of components and solution elements from proven successful experiences. The Patterns approach is based on a set of layered assets that can be exploited by any existing development methodology. These layered assets are structured in a way that each level of detail builds on the last. These assets include:

- Business patterns that identify the interaction between users, businesses, and data.
Integration patterns that tie multiple Business patterns together when a solution cannot be provided based on a single Business pattern.

Composite patterns that represent commonly occurring combinations of Business patterns and Integration patterns.

Application patterns that provide a conceptual layout describing how the application components and data within a Business pattern or Integration pattern interact.

Runtime patterns that define the logical middleware structure supporting an Application pattern. Runtime patterns depict the major middleware nodes, their roles, and the interfaces between these nodes.

Product mappings that identify proven and tested software implementations for each Runtime pattern.

Best-practice guidelines for design, development, deployment, and management of e-business applications.

These assets and their relation to each other are shown in Figure 1-1.
Patterns for e-business Web site
The Patterns Web site provides an easy way of navigating top down through the
layered Patterns’ assets in order to determine the preferred reusable assets for
an engagement.

For easy reference to Patterns for e-business refer to the Patterns for e-business
Web site at:


1.3 How to use the Patterns for e-business

As described in the last section, the Patterns for e-business are a layered
structure where each layer builds detail on the last. At the highest layer are
Business patterns. These describe the entities involved in the e-business
solution.

Composite patterns appear in the hierarchy shown in Figure 1-1 on page 3 above
the Business patterns. However, Composite patterns are made up of a number of
individual Business patterns, and at least one Integration pattern. In this section,
we discuss how to use the layered structure of Patterns for e-business assets.

1.3.1 Select a Business, Integration, or Composite pattern, or a
Custom design

When faced with the challenge of designing a solution for a business problem,
the first step is to take a high-level view of the goals you are trying to achieve. A
proposed business scenario should be described and each element should be
matched to an appropriate IBM Pattern for e-business. You may find, for
example, that the total solution requires multiple Business and Integration
patterns, or that it fits into a Composite pattern or Custom design.

For example, suppose an insurance company wants to reduce the amount of
time and money spent on call centers that handle customer inquiries. By allowing
customers to view their policy information and to request changes online, they
will be able to cut back significantly on the resources spent handling this by
phone. The objective is to allow policyholders to view their policy information
stored in legacy databases.

The Self-Service business pattern fits this scenario perfectly. It is meant to be
used in situations where users need direct access to business applications and
data. Let us take a look at the available Business patterns.
Business patterns
A Business pattern describes the relationship between the users, the business organizations or applications, and the data to be accessed.

There are four primary Business patterns, explained in Figure 1-2.

<table>
<thead>
<tr>
<th>Business Patterns</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Service (User-to-Business)</td>
<td>Applications where users interact with a business via the Internet or intranet</td>
<td>Simple Web site applications</td>
</tr>
<tr>
<td>Information Aggregation (User-to-Data)</td>
<td>Applications where users can extract useful information from large volumes of data, text, images, etc.</td>
<td>Business intelligence, knowledge management, Web crawlers</td>
</tr>
<tr>
<td>Collaboration (User-to-User)</td>
<td>Applications where the Internet supports collaborative work between users</td>
<td>E-mail, community, chat, video conferencing, etc.</td>
</tr>
<tr>
<td>Extended Enterprise (Business-to-Business)</td>
<td>Applications that link two or more business processes across separate enterprises</td>
<td>EDI, supply chain management, etc.</td>
</tr>
</tbody>
</table>

Figure 1-2 The four primary Business patterns

It would be very convenient if all problems fit nicely into these four slots, but reality says that things will often be more complicated. The patterns assume that most problems, when broken down into their most basic components, will fit more than one of these patterns. When a problem requires multiple Business patterns, the Patterns for e-business provide additional patterns in the form of Integration patterns.

Integration patterns
Integration patterns allow us to tie together multiple Business patterns to solve a business problem. The Integration patterns are outlined in Figure 1-3 on page 6.
The Access Integration pattern maps to User Integration. The Application Integration pattern can be divided into two essentially different approaches:

- **Process Integration** - The integration of the functional flow of processing between the applications.

- **Data Integration** - The integration of the information used by applications.

These Business and Integration patterns can be combined to implement installation-specific business solutions. We call this a Custom design.

**Custom design**

We can represent the use of a Custom design to address a business problem through an iconic representation, as shown in Figure 1-4 on page 7.
If any of the Business or Integration patterns are not used in a Custom design, we can show that with the blocks lighter than the others. For example, Figure 1-5 shows a Custom design that does not have a Collaboration business pattern or an Extended Enterprise business pattern for a business problem.

A Custom design may also be a Composite pattern if it recurs many times across domains with similar business problems. For example, the iconic view of a Custom design in Figure 1-5 can also describe a Sell-Side Hub composite pattern.
## Composite patterns

Several common uses of Business and Integration patterns have been identified and formalized into Composite patterns. The identified Composite patterns are shown in Figure 1-6.

<table>
<thead>
<tr>
<th>Composite Patterns</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electronic Commerce</td>
<td>User-to-Online-Buying</td>
<td>• <a href="http://www.macys.com">www.macys.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <a href="http://www.amazon.com">www.amazon.com</a></td>
</tr>
<tr>
<td>Portal</td>
<td>Typically designed to aggregate multiple information sources and applications to provide uniform, seamless, and personalized access for its users.</td>
<td>• Enterprise Intranet portal providing self-service functions such as payroll, benefits, and travel expenses.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Collaboration providers who provide services such as e-mail or instant messaging.</td>
</tr>
<tr>
<td>Account Access</td>
<td>Provide customers with around-the-clock account access to their account information.</td>
<td>• Online brokerage trading apps.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Telephone company account manager functions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Bank, credit card and insurance company online apps.</td>
</tr>
<tr>
<td>Trading Exchange</td>
<td>Allows buyers and sellers to trade goods and services on a public site.</td>
<td>• Buyer's side - interaction between buyer's procurement system and commerce functions of e-Marketplace.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Seller's side - interaction between the procurement functions of the e-Marketplace and its suppliers.</td>
</tr>
<tr>
<td>Sell-Side Hub (Supplier)</td>
<td>The seller owns the e-Marketplace and uses it as a vehicle to sell goods and services on the Web.</td>
<td><a href="http://www.carmax.com">www.carmax.com</a> (car purchase)</td>
</tr>
<tr>
<td>Buy-Side Hub (Purchaser)</td>
<td>The buyer of the goods owns the e-Marketplace and uses it as a vehicle to leverage the buying or procurement budget in soliciting the best deals for goods and services from prospective sellers across the Web.</td>
<td><a href="http://www.wre.org">www.wre.org</a> (WorldWide Retail Exchange)</td>
</tr>
</tbody>
</table>

*Figure 1-6  Composite patterns*
The makeup of these patterns is variable, in that there will be basic patterns present for each type, but the Composite can easily be extended to meet additional criteria. For more information on Composite patterns, refer to *Patterns for e-business: A Strategy for Reuse* by Jonathan Adams, Srinivas Koushik, Guru Vasudeva, and George Galambos.

### 1.3.2 Selecting Application patterns

Once the Business pattern is identified, the next step is to define the high-level logical components that make up the solution and how these components interact. This is known as the Application pattern. A Business pattern will usually have multiple possible Application patterns. An Application pattern may have logical components that describe a presentation tier for interacting with users, an application tier, and a back-end application tier.

Application patterns break the application down into the most basic conceptual components, identifying the goal of the application. In our example, the application falls into the Self-Service business pattern and the goal is to build a simple application that allows users to access back-end information. The Application pattern shown in Figure 1-7 fulfills this requirement.

![Figure 1-7 Self-Service::Directly Integrated Single Channel](image-url)
The Application pattern shown consists of a presentation tier that handles the request/response to the user. The application tier represents the component that handles access to the back-end applications and data. The multiple application boxes on the right represent the back-end applications that contain the business data. The type of communication is specified as synchronous (one request/one response, then next request/response) or asynchronous (multiple requests and responses intermixed).

Suppose that the situation is a little more complicated than that. Let us say that the automobile policies and the homeowner policies are kept in two separate and dissimilar databases. The user request would actually need data from multiple, disparate back-end systems. In this case there is a need to break the request down into multiple requests (decompose the request) to be sent to the two different back-end databases, then to gather the information sent back from the requests, and then put this information into the form of a response (recompose). In this case the Application pattern shown in Figure 1-8 would be more appropriate.

![Diagram](image)

**Figure 1-8  Self-Service::Decomposition**

This Application pattern extends the idea of the application tier that accesses the back-end data by adding decomposition and recomposition capabilities.
1.3.3 Review Runtime patterns

The Application pattern can be further refined with more explicit functions to be performed. Each function is associated with a runtime node. In reality these functions, or nodes, can exist on separate physical machines or may co-exist on the same machine. In the Runtime pattern this is not relevant. The focus is on the logical nodes required and their placement in the overall network structure.

As an example, let us assume that our customer has determined that his solution fits into the Self-Service business pattern and that the Directly Integrated Single Channel pattern is the most descriptive of the situation. The next step is to determine the Runtime pattern that is most appropriate for his situation.

He knows that he will have users on the Internet accessing his business data, and he will therefore require a measure of security. Security can be implemented at various layers of the application, but the first line of defense is almost always one or more firewalls that define who and what can cross the physical network boundaries into his company network.

He also needs to determine the functional nodes required to implement the application and security measures. The Runtime pattern shown in Figure 1-9 on page 12 is one of his options.
By overlaying the Application pattern on the Runtime pattern, you can see the roles that each functional node will fulfill in the application. The presentation and application tiers will be implemented with a Web application server, which combines the functions of an HTTP server and an application server. It handles both static and dynamic Web pages.

Application security is handled by the Web application server through the use of a common central directory and security services node.

A characteristic that makes this Runtime pattern different from others is the placement of the Web application server between the two firewalls. The Runtime pattern shown in Figure 1-10 on page 13 is a variation on this. It splits the Web application server into two functional nodes by separating the HTTP server function from the application server. The HTTP server (Web server redirector) will serve static Web pages and redirect other requests to the application server. It moves the application server function behind the second firewall, adding further security.
These are just two examples of the possible Runtime patterns available. Each Application pattern will have one or more Runtime patterns defined. These can be modified to suit the customer's needs. For example, she may want to add a load-balancing function and multiple application servers.

1.3.4 Review Product mappings

The last step in defining the network structure for the application is to correlate real products with one or more runtime nodes. The Patterns Web site shows each Runtime pattern with products that have been tested in that capacity. The Product mappings are oriented toward a particular platform, though more likely the customer will have a variety of platforms involved in the network. In this case, it is simply a matter of mix and match.

For example, the runtime variation in Figure 1-10 could be implemented using the product set depicted in Figure 1-11 on page 14.
1.3.5 Review guidelines and related links

The Application patterns, Runtime patterns, and Product mappings are intended to guide you in defining the application requirements and the network layout. The actual application development has not been addressed yet. The Patterns Web site provides guidelines for each Application pattern, including techniques for developing, implementing, and managing the application based on the following:

- **Design guidelines** instruct you on tips and techniques for designing the applications.
- **Development guidelines** take you through the process of building the application, from the requirements phase all the way through the testing and rollout phases.
- **System management guidelines** address the day-to-day operational concerns, including security, backup and recovery, application management, etc.
- **Performance guidelines** give information on how to improve the application and system performance.
1.4 Summary

The IBM Patterns for e-business are a collective set of proven architectures. This repository of assets can be used by companies to facilitate the development of Web-based applications. They help an organization understand and analyze complex business problems and break them down into smaller, more manageable functions that can then be implemented.
In this chapter we briefly describe the business requirements driving the need for information integration, and IBM’s response to this demand with its DB2 Information Integrator portfolio of products. We introduce the IBM DB2 Information Integrator portfolio of products and focus on DB2 Information Integrator since it is used in the customer insight solution.

The topics covered are:
- Current business trends
- IBM’s DB2 Information Integration overview
- DB2 Information Integrator V8.1
2.1 Introduction

A number of business trends are driving the need for integration of data and processes across employees, customers, business partners, and suppliers. The inherent heterogeneity of hardware and software platforms in intranets and extranets presents unique challenges that must be overcome in order to gain a competitive advantage in the global economy.

In this chapter we discuss the current business trends fueling integration demands, IBM’s DB2 Information Integration solution, and IBM’s federated DB2 Information Integrator V8.1 offering.

2.2 Current business trends

To keep up with the evolution of e-business computing, companies in every industry are being challenged to act—and react—on demand. Responding to any customer demand, each market opportunity and every external threat requires integration between people, processes, and information. This integration must extend across the company, and across partners, suppliers, and customers.

Integration, automation, and virtualization are the three key elements of this on-demand operating environment:

- **Integration** is the efficient and flexible combination of data to optimize operations across and beyond the enterprise. It is about people, processes, and information.

- **Automation** is the capability to increasingly automate business processes with the ultimate goal of self-regulation, thereby reducing the complexity of data management to enable better use of assets.

- **Virtualization** provides a single, consolidated view of and easy access to all available resources in a network, no matter where the data resides, or the type of data source.

**Note:** IBM defines an on demand business as an enterprise whose business processes integrate end-to-end across the company with key partners, suppliers, and customers in order to respond with speed to any customer demand, market opportunity, or external threat.

IBM has identified five types of integration that are based on an open services infrastructure. You can use these types of integration together or separately to solve business issues. The following five types of integration represent the
various integration challenges that face businesses today—with information integration being at the core of these integration types.

1. **User interaction**
   A user can work with a single tailored user interface, which is available through virtually any device, with full transactional support. The results of the user's interaction are integrated into multiple business systems.

2. **Process integration**
   A business can change how it operates through modeling, automation, and monitoring of processes across people and heterogeneous systems—both inside and outside the enterprise.

3. **Application connectivity**
   Applications can connect to one another so that they share and use information for better use at the enterprise level.

4. **Build to integrate**
   Users can build and deploy integration-ready applications by using Web services and existing assets. You can integrate new solutions with existing business assets.

5. **Information integration**
   Diverse forms of business information can be integrated across the enterprise. Integration enables coherent search, access, replication, transformation, and analysis over a unified view of information assets to meet business needs.

In the following subsections, we describe how the success of an on demand business enterprise is significantly dependent upon a seamless and scalable information integration infrastructure that is enabled by the following:

- Grid computing
- Data federation
- Information integration

### 2.2.1 Grid computing

Grid computing is distributed computing taken to the next evolutionary level. The grid provides an infrastructure on which to support a large collection of communication resources such as hardware and software.

The standardization of communications between heterogeneous systems created the Internet explosion. The emerging standardization for sharing resources, along with the availability of higher bandwidth, is driving a potentially equally large evolutionary step in grid computing.
One major function of the grid is to better balance resource utilization.

An organization may have occasional unexpected peaks of activity that demand more resources. If the applications are grid enabled, the application workload can be moved to under-utilized machines during such peaks. In general, a grid can provide a consistent way to balance workloads on a wider federation of resources.

2.2.2 Data federation

An increasing number of grid applications manage very large volumes of geographically distributed data. The complexity of data management on a grid is due to the scale, dynamism, autonomy, and distribution of data sources.

One way of accessing diverse business information from a variety of sources and platform is through data federation.

Data federation is the ability to transparently access diverse business data from a variety of sources and platforms as though it were a single resource. A federated server may access data directly, such as accessing a relational database or accessing an application that creates and returns data dynamically such as a Web service. Figure 2-1 shows the federated approach to information integration as providing the ability to synchronize distributed data without requiring that it be moved to a central repository.

![Figure 2-1 Data federation concept](image)
Based on ongoing research investments and proven data management technologies in areas such as relational data, XML, content management, federation, search, and replication, IBM has developed the integrated infrastructure shown in Figure 2-1 on page 20. Data federation uses SQL as the single language to access all data sources. This enables all data sources to be accessed in a standardized format, whether they are an application program, tool, or program product.

### 2.2.3 Information integration

Information integration builds on the solid foundation of existing data management solutions. Information integration provides an end-to-end solution for transparently managing both the volume and diversity of data that exists in enterprises and organizations today.

Increasingly business IT operations involve the need to integrate diverse and unconnected infrastructures.

The following goals are critical to increasing operations efficiency and gaining a competitive advantage:

- Integrate seamlessly with new businesses and link packaged applications with legacy systems.
- Control the accelerating costs of managing disparate systems and integrating across heterogeneous pockets of automation.
- Mitigate shortages of people and skills while quickly reaching new markets.
- Implement solutions that efficiently access and manage information across product and industry boundaries.

### 2.3 IBM’s DB2 Information Integration overview

Today, any but the simplest of business tasks requires the use of information from the variety of data sources that businesses have built over many years. These sources may be local or remote, on the intranet, extranet or internet. The data may be stored in any of a variety of formats such as relational or non-relational databases, flat files, and unstructured content stores. The data may be current or point-in-time copies. Often, the users need both read and write access to these sources.

This complex and dynamic environment presents significant challenges to business users and applications, as well as to the IT people who must maintain and manage it.
The underlying principle of information integration is for users to be able to see all of the data they use as if it resided at a single source. Information integration technology shields the requester from all the complexities associated with accessing data in diverse locations including connectivity, semantics, formats and access methods. Using a standards-based language such as structured query language (SQL), extensible markup language (XML) through SQL/XML, or a standard Web services or content API, information integration middleware enables users (or applications acting on their behalf), to access information transparently without concern for its physical implementation.

The goal of providing an integrated view of information can be achieved in two ways, as follows:

1. **Data consolidation or placement**, which involves moving the data to a more efficient or accessible location

   Consolidating data into a single physical store has been the best way to achieve fast, highly available, and integrated access to related information. Creating a single physical copy lets businesses meet access performance or availability requirements, deliver snapshots that are point-in-time consistent, and provide sophisticated transformation for semantic consistency.

   Consolidated data stores, which are typically managed through extract, transform, load (ETL) or replication processes, are the standard choice for information integration today.

   However, these consolidated stores have some drawbacks, as follows:
   
   – They are expensive—racking up significant additional administration, server, and storage costs.
   
   – The latency between the copy and the source of record can be a problem when you need the most current data.
   
   – Rich content such as documents, images, or audio is typically not included.

2. **Distributed access**, which involves providing distributed access to data through data access or federation

   Distributed access corresponds to the emerging category of technology called enterprise information integration (EII), which addresses some of the shortfalls of data consolidation or placement. EII represents middleware technology that lets applications access diverse and distributed data as if it were a single source, regardless of location, format, or access language.

   Access performance will typically be slower than for consolidated stores

**Important:** IBM’s vision of information integration is to significantly reduce or even eliminate these issues.
because the query may have to gather information from distributed locations across the network rather than access a single, local copy of data.

However, the benefits of EII include:

– Reduced implementation and maintenance costs because you do not have the additional hardware (server and storage), skills, and personnel costs.

– Access to current data from the source of record.

– Combining traditional data with mixed format data.

– Access to copy-prohibited data based on data security, licensing restrictions, or industry regulations that restrict data movement, for example, some European countries prohibit commingling a customer’s personal data with account data in a single database. But you can materialize a single image of the data by federating it at the time of access.

**Note:** Distributed sources must be sufficiently consistent to make joining the data both possible and meaningful. There must be a key on which the data can be joined or correlated, such as a customer identifier, and the joined data must represent related topics.

Both data consolidation or placement, and distributed access data consolidation serve distinct problem domains and are very complementary. They may be used alone or together to form the heart of what is required to integrate information.

Both approaches require extensive and largely common supporting functionality. Neither distributed access nor data consolidation or placement can exist without mapping and transformation functionality that ensure data integrity. Furthermore, depending on the business requirement, the same data may need to be consolidated in some cases and federated in others. Therefore, a common set of transformation and mapping functionality is required in both cases to maintain consistency across the data used by the business.

In the following sections, we briefly describe scenarios where data consolidation and distributed access are appropriate, and then provide an overview of DB2 Information Integration products.

### 2.3.1 Data consolidation or placement

Data consolidation or placement brings together data from a variety of locations into one place, in advance, so that a user query does not always need to be distributed. This approach corresponds to ETL and replication functionality. You can use ETL to build a warehouse, replication to keep it automatically updated
on a scheduled basis, and extend it with federation for queries that require data that did not make sense to put in the warehouse.

Scenarios where ETL or replication approaches are appropriate include the following:

- Access performance or availability requirements demand centralized or local data.
- Complex transformation is required to achieve semantically consistent data.
- Complex, multidimensional queries are involved.
- Currency requirements demand point-in-time consistency such as at the close of business.

2.3.2 Distributed access (federation)

Very simply, federation takes a query in one location and distributes the appropriate parts of it to act upon the data wherever and in whatever form it resides.

Scenarios where distributed access approaches are appropriate include the following:

- Access performance and load on source systems can be traded for an overall lower implementation cost.
- Data currency requirements demand a fresh copy of the data.
- Widely heterogeneous data.
- Rapidly changing data.
- Data security.
- Licensing restrictions or industry regulations restrict data movement.
- Unique functions must be accessed at the data source.
- Queries returning small result sets among federated systems.
- Large volume data that are accessed infrequently.

2.3.3 DB2 Information Integrator portfolio

IBM's Information Integration solution consists of a number of products and technologies that fall under a solution umbrella called DB2 Information Integrator portfolio, as shown in Figure 2-2 on page 25.
There are three main products that fall under the federation approach:

- **DB2 Information Integrator (DB2 II)**
  
  DB2 II is targeted at the application development community familiar with relational database application development. Applications that use SQL, or tools that generate SQL such as integrated development environments, and reporting and analytical tools, can now access and manipulate distributed and diverse data through a federated data server.

- **DB2 Information Integrator for Content (DB2 IIC)**
  
  DB2 IIC is targeted at the content application developer (mainly one who works with DB2 Content Manager) who needs to search for and access text and augment it with other content or relational sources. In addition to federated search, it also offers sophisticated information mining to discover new metadata from text documents and advanced workflow (based on WebSphere® MQ Workflow) to facilitate building content-centric processes. DB2 IIC represents a renaming and repositioning of the Enterprise Information Portal (EIP) offering.
2.4 DB2 Information Integrator V8.1

In this section we provide an overview of DB2 Information Integrator V8.1, describe its main components, discuss the steps involved in configuring a data source, and review some of the performance considerations.

The topics covered are:

- DB2 II V8.1 overview
- DB2 II components
- Configuring the federated system
- Performance considerations

2.4.1 DB2 II V8.1 overview

DB2 II’s federated technology enables customers to abstract a common data model across diverse and distributed data and content sources, and to access and manipulate them as though they were a single source.

As mentioned earlier, with the data federation capability, the federated system acts as a virtual database with remote objects configured similar to local tables, as shown in Figure 2-3 on page 27.
With a federated system, you can send distributed requests to multiple data sources within a single SQL statement; for example, you can join data that is located in a DB2 UDB table, an Informix table, and an XML tagged file in a single SQL statement.

When an application submits a query to the federated system, the federated DB2 identifies the relevant data sources, and develops a query execution plan for obtaining the requested data. The plan typically breaks the original query into fragments that represent work to be delegated to individual data sources, as well as additional processing to be performed by the federated DB2 to further filter, aggregate, or merge the data. The ability of the federated DB2 to further process data received from sources allows applications to take advantage of the full power of the query language, even if some of the information requested comes from data sources with little or no native query processing capability, such as simple text files. The federated DB2 has a local data store to cache query results for further processing.

A DB2 federated system is a special type of DBMS. A federated system consists of the following:

- A DB2 instance that operates as a federated server.
- A database that acts as the federated database for various relational and non-relational data sources.
Clients (users and applications) that access the database and data sources—a nickname is the mechanism used by the clients to reference a remote data source object as if it were a local table.

The federated DBMS communicates with the data sources by means of software modules called wrappers, as shown in Figure 2-4.

Wrappers are mechanisms by which the federated server interacts with the data sources. The federated server uses routines stored in a library called a wrapper module to implement a wrapper. These routines allow the federated server to perform operations such as connecting to a data source and retrieving data from it. The wrapper encapsulates data source information and models data as tables. It is aware of the characteristics of the data source, and it can expose unique functions. A wrapper provides the programming logic to facilitate the following tasks:

- **Federated object registration**
  A wrapper encapsulates the data source characteristics from the federated engine. A wrapper knows what information is needed to register each type of data source.

- **Communication with the data source**
  Communication includes establishing and terminating connections with the data source, and maintaining the connection across statements within an application if possible.
Services and operations

Depending on the capabilities of the type of data sources that a wrapper is meant to access, different operations are supported. The operations can include sending a query to retrieve results, updating remote data, transaction support, large object manipulation, input value binding, compensation\(^1\) and more.

Data modelling

A wrapper is responsible for mapping the data representation of the result of remote queries into the table format as required by the federated engine.

Wrappers are available for each type of data source. For example, if you want to access three DB2 for z/OS database tables, one for DB2 for iSeries™ table, two DB2 UDB for Windows tables, two Informix tables, and one Informix view, you need to define only two wrappers—one for the DB2 data source objects and one for the Informix data source objects. Once these wrappers are registered in the federated database, you can use these wrappers to access other objects from those data sources.

DB2 Information Integrator V8.1 includes the ability to federate, search, cache, transform, and replicate data. As a federated data server, it provides out-of-the box access to DB2 Universal Database™, IBM Informix products, as well as databases from Microsoft, Oracle, Sybase, and Teradata. In addition, it can also access semi-structured data from WebSphere MQ messages, XML documents, Web services, Microsoft Excel, flat files, ODBC or OLE DB sources, plus a variety of formats unique to the life sciences industry. Integrated support for IBM Lotus® Extended Search provides the solution’s broad content access to a variety of content repositories, including DB2 Content Manager, as well as e-mail databases, document repositories, third-party Internet search engines, and LDAP directories.

Note: Applications can insert, update, or delete rows in federated relational databases; however, this is limited to single-site updates with only one-phase commits.

DB2 II V8.1 is supported on the Linux, UNIX and Windows platforms.

---

\(^1\) Compensation is the ability by DB2 to process SQL that is not supported by a data source. DB2 compensates for lack of functionality at the data source in two ways. One way is to ask the data source to use one or more operations that are equivalent to the DB2 function stated in the query, and another way is to return the set of data from the data source to the federated server and perform the function locally.
Table 2-1 on page 30 lists the data sources supported, their corresponding versions and the access method used by IBM DB2 Information Integrator V8.1 to access the supported data sources.

<table>
<thead>
<tr>
<th>Data source</th>
<th>Supported versions</th>
<th>Access method</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2 Universal Database for Linux, UNIX, and Windows</td>
<td>7.1, 7.2, 8.1</td>
<td>DRDA®</td>
</tr>
<tr>
<td>DB2 Universal Database for z/OS and OS/390®</td>
<td>6.1, 7.1 with the following APARs applied: PQ62695 PQ55393 PQ56616 PQ54605 PQ46183 PQ62139</td>
<td>DRDA</td>
</tr>
<tr>
<td>DB2 Universal Database for iSeries</td>
<td>4.5 (or later) with the following APARs applied: SA95719 SE06003 SE06872 SI05990 SI05991</td>
<td>DRDA</td>
</tr>
<tr>
<td>DB2 Server for VM and VSE</td>
<td>7.1 (or later) with fixes for APARs for schema functions applied</td>
<td>DRDA</td>
</tr>
<tr>
<td>Informix</td>
<td>7, 8, 9</td>
<td>Informix Client SDK</td>
</tr>
<tr>
<td>ODBC</td>
<td>3.x</td>
<td>ODBC driver for the data source, such as Redbrick ODBC Driver to access Redbrick</td>
</tr>
<tr>
<td>OLE DB</td>
<td></td>
<td>OLE DB 2.0 (or later)</td>
</tr>
<tr>
<td>Oracle</td>
<td>7.3.4, 8.x, 9.x</td>
<td>SQLNET or NET8 client software</td>
</tr>
</tbody>
</table>
| Microsoft SQL Server | 6.5, 7.0, 2000 | On Windows, the Microsoft SQL Server Client ODBC 3.0 (or later) driver  
On UNIX, the DataDirect Technologies (formerly MERANT) Connect ODBC 3.7 (or later) driver |
Table 2-1  Data sources, supported versions and access method

<table>
<thead>
<tr>
<th>Data source</th>
<th>Supported versions</th>
<th>Access method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sybase</td>
<td>11.x, 12.x</td>
<td>Sybase Open Client</td>
</tr>
<tr>
<td>Teradatas</td>
<td>V2R3, V2R4</td>
<td>Teradata Call-Level Interface Version 2 (CLIv2) Release 04.06 (or later)</td>
</tr>
<tr>
<td>BLAST</td>
<td>2.x</td>
<td>BLAST daemon (supplied with the wrapper)</td>
</tr>
<tr>
<td>Documentum</td>
<td>Documentum server: EDMS 98 (also referred to as version 3) and 4i</td>
<td>Documentum Client API/Library</td>
</tr>
<tr>
<td>Entrez</td>
<td>1.0</td>
<td>None</td>
</tr>
<tr>
<td>HMMER</td>
<td>2.2g</td>
<td>HMMER daemon (supplied with the wrapper)</td>
</tr>
<tr>
<td>IBM Lotus Extended Search</td>
<td>4.0</td>
<td>Extended Search Client Library (supplied with the wrapper)</td>
</tr>
<tr>
<td>Microsoft Excel</td>
<td>97, 2000</td>
<td>Excel 97 or 2000 installed on the federated server</td>
</tr>
<tr>
<td>Table-structured files</td>
<td></td>
<td>None</td>
</tr>
<tr>
<td>XML</td>
<td>1.0 specification</td>
<td>None</td>
</tr>
</tbody>
</table>

**Note:** For specific details on operating system versions and data sources supported, refer to the *IBM DB2 Information Integrator: Installation Guide*, GC18-7036.

Table 2-2, Table 2-3 on page 32, and Table 2-4 on page 34 list some of DB2 II’s restrictions as they relate to data types supported and data source writes.

Table 2-2  Unsupported data types on specific data sources

<table>
<thead>
<tr>
<th>Data</th>
<th>Unsupported Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2 for iSeries</td>
<td>VARG data</td>
</tr>
<tr>
<td>Extended Search</td>
<td>DECIMAL</td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td>SQL_VARIANT</td>
</tr>
<tr>
<td>Data source</td>
<td>Restriction on data types</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Informix</td>
<td>BLOB, CLOB, TEXT</td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td>image, ntext, text, SQL_VARIANT</td>
</tr>
<tr>
<td>ODBC</td>
<td>SQL_LONG_BINARY (length &gt; 255), SQL_LONGVARCHAR (length &gt; 255), SQL_WLONGVARCHAR (length &gt; 255)</td>
</tr>
</tbody>
</table>

**Table 2-3**  *Write operation restrictions on data type on specific data sources*
<table>
<thead>
<tr>
<th>Data source</th>
<th>Restriction on data types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle (NET8 wrapper only)</td>
<td>INTERVAL DAY (day_precision) TO SECOND (fractional_seconds_precision) INTERVAL YEAR (year_precision) TO MONTH LONG LONG RAW NCCHAR NVARCHAR2 TIMESTAMP (fractional_seconds_precision) WITH TIME ZONE TIMESTAMP (fractional_seconds_precision) WITH LOCAL TIME ZONE</td>
</tr>
<tr>
<td>Oracle (SQLNET wrapper only)</td>
<td>BLOB CLOB INTERVAL DAY (day_precision) TO SECOND (fractional_seconds_precision) INTERVAL YEAR (year_precision) TO MONTH NCCHAR NVARCHAR2 TIMESTAMP (fractional_seconds_precision) WITH TIME ZONE TIMESTAMP (fractional_seconds_precision) WITH LOCAL TIME ZONE</td>
</tr>
<tr>
<td>Sybase (CTLIB wrapper only)</td>
<td>image text unichar univarchar</td>
</tr>
<tr>
<td>Sybase (DBLIB wrapper only)</td>
<td>All data types. Write operations are not supported by the DBLIB wrapper.</td>
</tr>
<tr>
<td>Teradata</td>
<td>char (length 32673-64000) varchar(length 32673-64000)</td>
</tr>
</tbody>
</table>
Table 2-4  Excel write operations limitations

<table>
<thead>
<tr>
<th>Type</th>
<th>Read</th>
<th>Insert</th>
<th>Update</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessing Excel through ODBC Wrapper</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Accessing Excel through Excel Wrapper</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**Note:** A wrapper development kit is included in DB2 II that allows you to develop your own wrapper modules for non-relational data sources. The wrapper module is a shared library with specific entry points that provide access to a class of data sources. DB2 UDB loads it dynamically. The wrapper module is what you will be developing using the specific classes supplied with DB2—it will contain specific building blocks that allow it to act as a translator between your data source and the federated system.

The power of DB2 II is in its ability to:

- Join data from local tables and remote data sources, as if all the data is stored locally in the federated database.
- Update data in relational data sources, as if the data is stored in the federated database.
- Replicate data to and from relational data sources.
- Take advantage of the data source processing strengths, by sending distributed requests to the data sources for processing.
- Compensate for SQL limitations at the data source by processing parts of a distributed request at the federated server.

### 2.4.2 DB2 II components

DB2 II contains the following components, as shown in Figure 2-6 on page 38:

- The *relational wrappers* are used for non-IBM relational databases. In DB2 UDB Enterprise Server Edition (ESE) V8 for Linux, UNIX, and Windows, relational wrappers are required if you want to access data that is stored in Oracle, Sybase, Microsoft SQL Server, ODBC, and Teradata data sources.
- **Non-relational wrappers** are used by the DB2 federated system to integrate non-relational data sources, such as flat files and XML files, and genetic, chemical, biological, and other research data from distributed sources.

- **Global catalog** is the catalog in the federated database that holds information about the entire federated system. The global catalog holds information about the objects (tables, indexes, functions, etc.) in the federated database, as well as information about objects (wrappers, remote servers, nicknames and their relationships) at the data sources. The information stored is about local and remote column names, column data types, column default values, and index information.

- **DB2 Net Search Extender** is used to perform SQL-based searches on full-text documents across your enterprise. DB2 Net Search Extender performs searches efficiently by using text indexes, which Net Search Extender updates dynamically, and stores in-memory, reducing scans and physical read operations.

DB2 Information Integrator V8.1 *extends* the data federation technology already available in DB2 UDB for Linux, UNIX, and Windows, as shown in Figure 2-5 on page 36.
The global catalog contains statistical information for nicknames, information on remote indexes for nicknames, and information on some attributes of each remote source, as shown in Table 2-5. It also contains type and function mappings.

<table>
<thead>
<tr>
<th>Federated objects</th>
<th>Catalog views</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrappers</td>
<td>SYSCAT.WRAPPERS</td>
<td>Registered wrappers and their specific options.</td>
</tr>
<tr>
<td></td>
<td>SYSCAT.WRAPOPTIONS</td>
<td>(wraptype='R'/N' for Relational/Non-relational wrapper)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Registered remote data sources and their specific options.</td>
</tr>
<tr>
<td>Servers</td>
<td>SYSCAT.SERVERS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SYSCAT.SERVEROPTIONS</td>
<td></td>
</tr>
<tr>
<td>Federated objects</td>
<td>Catalog views</td>
<td>Descriptions</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>User mappings</td>
<td>SYSCAT.USEROPTIONS</td>
<td>Registered user authentications for specific servers for a DB2 user. The password setting is stored encrypted.</td>
</tr>
<tr>
<td>Nicknames</td>
<td>SYSCAT.TABLES, SYSCAT.TABOPTIONS,</td>
<td>Registered nicknames are identified with TYPE='N' in SYSCAT.TABLES. SYSCAT.TABOPTIONS stores specific options about nicknames.</td>
</tr>
<tr>
<td></td>
<td>SYSCAT.COLUMNS, SYSCAT.COLUMNS,</td>
<td>SYSCAT.COLOPTIONS stores specific options about nicknames; for instance, the servername, remote schema, and remote table name.</td>
</tr>
<tr>
<td></td>
<td>SYSCAT.INDEXES, SYSCAT.INDEXES,</td>
<td>SYSCAT.KEYCOLUSE stores information about primary key.</td>
</tr>
<tr>
<td></td>
<td>SYSCAT.INDEXOPTIONS,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SYSCAT.KEYCOLUSE</td>
<td></td>
</tr>
<tr>
<td>Index specifications</td>
<td>SYSCAT.INDEXES, SYSCAT.INDEXES</td>
<td>Index specifications created for nicknames.</td>
</tr>
<tr>
<td>Type mappings</td>
<td>SYSCAT.TYPEMAPPINGS</td>
<td>User-defined type mappings used in nickname registration and transparent DDL. Default built-in type mappings are not stored in these catalog views</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mapping direction = 'F'/R'.</td>
</tr>
<tr>
<td>Function templates</td>
<td>SYSCAT.FUNCTIONS, SYSCAT.ROUTINES</td>
<td>Registered user-defined functions. In V8, SYSCAT.ROUTINES supersedes SYSCAT.FUNCTIONS in V8, but SYSCAT.FUNCTIONS still exists, not documented.</td>
</tr>
<tr>
<td>Function mappings</td>
<td>SYSCAT.FUNCMAPPINGS,</td>
<td>User-defined function mappings to map a local function to a remote function.</td>
</tr>
<tr>
<td></td>
<td>SYSCAT.FUNCMAPOPTIONS,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SYSCAT.FUNCMAPPARAMOPTIONS</td>
<td></td>
</tr>
<tr>
<td>Passthru privileges</td>
<td>SYSCAT.PASSTHRUAUTH</td>
<td>Authorization to allow users to query a specific server using PASSTHRU.</td>
</tr>
</tbody>
</table>

This information is collected when the federated system is configured as discussed in 2.4.3, “Configuring the federated system” on page 38. This information can be queried by issuing queries against the catalog.
The DB2 query optimizer uses the information in the global catalog and the data source wrapper to plan the optimal way to process SQL statements. Execution plans for federated queries are chosen by the same DB2 optimizer that optimizes regular queries. The difference is that the federated engine uses the native client interface to each target data source, and sends queries to it in its own dialect.

Figure 2-6 summarizes some of the DB2 II components on a Windows platform.

2.4.3 Configuring the federated system

The DB2 federated server allows you to access and join data from relational and non-relational data sources. By setting the database manager configuration parameter FEDERATED to YES, the DB2 instance (without DB2 II) allows federated access to other DB2 sources, Informix, and any OLE DB source, as shown in Figure 2-6.
Attention: If you need access to other non-relational or non-IBM relational sources such as Oracle, Sybase, or Microsoft SQL databases as well as generic ODBC access, and Teradata, then you need to install DB2 II.

Figure 2-7 on page 40 highlights the basic steps involved in configuring the federated system. Some of these steps may be optional depending upon the data source being configured. Most of the steps to configure access to a data source can be accomplished through the DB2 Control Center. Use the DB2 Command Center for the steps that require a command line.

Attention: Before configuring access to a data source, ensure that the federated server has been set up properly. It is especially important to:

- Link DB2 to the client software. This creates the data source wrapper libraries on the federated server.
- Set up the data source environment variables.

For further details, refer to the IBM DB2 Information Integrator: Installation Guide, GC18-7036.
Each of these steps is described briefly:

1. **Step 1** involves preparing the federated server for the data source. For the DB2 family, this involves cataloging the node and the remote database. For Informix, Sybase, and Microsoft SQL Server data sources, it involves setting up and testing the client configuration file.

2. **Step 2** involves creating the wrappers in the federated server. One wrapper is created for each type of data source to be accessed. When a wrapper is created, it is registered in the federated database and the wrappers can now be used to access objects from these data sources.

3. **Step 3** involves creating the server definition that defines the data source to be accessed by the federated database. The name of the data source and other information is part of the server definition.
   - For a relational DBMS (RDBMS), it includes the type and version of the RDBMS, the database name for the data source on the RDBMS, and metadata that is specific to the RDBMS. A DB2 data source can have multiple databases, and therefore a database name is required to identify it as the target. An Oracle data source, on the other hand, can only have a
single database, and a database name is therefore not included in the federated server definition of an Oracle data source.

- For non-relational data sources, you must register a server object because the hierarchy of federated objects requires that specific files that you want to access must be associated with a server object.

During the creation of a server definition of a relational data source, server options can be used to set server attributes that contain information about the data source location, connection security, and some server characteristics that affect performance. These characteristics and restrictions are used by the query compiler in planning the query.

COLLATING_SEQUENCE, PUSHDOWN and DB2_MAXIMAL_PUSHDOWN are a few of the server attributes discussed here. For more details about server options refer to *IBM DB2 Information Integrator Federated Systems Guide*, SC18-7364:

- The COLLATING_SEQUENCE attribute can be used to adjust the collating sequence of a data source.

Data sources, such as DB2 for z/OS, use a collating sequence based on the EBCDIC encoding scheme. The default setting for the server option COLLATING_SEQUENCE is ‘N’ for such sources because it is not usual for DB2 for z/OS to have a database with similar encoding. DB2 UDB for Linux, UNIX, and Windows uses mostly ASCII encoding, and the default sorting order is dictionary sort. Although some data sources like Oracle use ASCII encoding, their sorting order is different from that of DB2 UDB for Linux, UNIX and Windows.

It is possible to create a DB2 II database with a sorting order to match that of sources like Oracle. You need to specify COLLATE USING IDENTITY in the CREATE DATABASE statement at the time the federated database is created. In this case, the server option COLLATING_SEQUENCE can be set to ‘Y’ for sources with ASCII encoding and identity sorting order. This server option allows range comparison such as “string_col > string_constant ) and “LIKE” predicates to be executed remotely.

If the remote data source is case insensitive and the DB2 federated database is set to use a case-sensitive search collating sequence, the equality comparison operations will also be executed locally.

- Pushdown is a very important aspect of federated query processing. If the PUSHDOWN server option is set to ‘Y’, the DB2 optimizer will consider generating a plan that “pushes down” certain parts of the query execution to the remote source. The intent of pushdown is to reduce network transport (trips) and exploit the intelligence of the relational remote sources. Pushdown analysis (PDA) is the component of the DB2 optimizer that decides which parts of a query can be pushed down and processed.
remotely at the data sources. The decision of actually pushing down is cost-based, and influenced by information about the hardware at the remote sources, the network characteristics, and the estimated number of rows processed and returned from the remote sources. Pushdown is discussed in 2.4.4, “Performance considerations” on page 45.

- DB2_MAXIMAL_PUSHDOWN is another server option for relational nicknames that you might want to consider setting to ‘Y’. For queries that contain nicknames, the federated server identifies which operations in the query can be pushed down to the data sources during the PDA phase. During the subsequent cost optimization phase, this option influences whether the DB2 optimizer determines the execution plan based on cost (default behavior), or favors pushing down the maximum number of operations identified during the PDA phase regardless of cost. Setting DB2_MAXIMAL_PUSHDOWN to ‘Y’ directs the query optimizer to favor access plans that tend to allow the remote relational data sources to evaluate as much of the query as possible. Because this setting will change the cost-based decision process in the query optimizer, it is not recommended as the first phase of customization. If your remote data source is as powerful as the federated server, it might make sense to set this option. Setting this option to ‘Y’ is also useful to compare the performance of your queries. Setting this option to ‘Y’ affects all queries that reference data from this remote data source.

**Attention:** Server options are generally set to persist over successive connections to the data source; however, they can be set or overridden for the duration of a single connection.

The federated system provides the SET SERVER OPTION statement for you to use when you want a server option setting to remain in effect while your application is connected to the federated server. When the connection ends, the previous server option setting is reinstated.

4. **Step 4** involves establishing a mapping between the federated server userid/password and the userid/password of the data source. This association is called a user mapping and is required so that the federated server can successfully connect to the target data source. This association must be created for each userid that will be using the federated system to send distributed requests.

**Note:** Each userid accessing this nickname on DB2 II will need to be mapped to the remote data source userid.
5. **Step 5** involves checking to see whether the federated system can connect to the target data source. A passthru session allows you to send SQL statements directly to a data source. Ensure proper privileges are granted to those users who can use the passthru session for this new data source. For example, with DB2 UDB for z/OS and OS/390, you can establish a passthru session and issue an SQL SELECT statement on the DB2 system table as follows:

```
SET PASSTHRU servername
SELECT count(*) FROM sysibm.systables
SET PASSTHRU RESET
```

6. **Step 6** involves creating a nickname, which is an identifier that is used to reference an object located at the data source that you want to access. The objects that nicknames identify are referred to as data source objects. Nicknames are *not* alternative names for data source objects in the same way that aliases are alternative names. They are pointers by which the federated server references these objects. Nicknames are typically defined with the CREATE NICKNAME statement.

Additional metadata information can be supplied about the nicknamed object via column options. Data mappings and function mappings may also be required between the target data source and DB2 data types in the federated server, if the default mappings provided in the wrappers are inadequate.

Determine whether additional data type mappings need to be defined if you are connecting to a relational data source. Specifying additional data type mappings is necessary if you want to change the default mapping between a DB2 data type and a remote data type. If you are accessing a relational data source, determine whether you need to register additional mappings between the DB2 functions and the remote data source functions. These mappings allow data source functions to be used by the federated system.

- **Data type mappings**

  Data types of remote data sources must correspond to DB2 data types. An appropriate mapping enables the federated server to retrieve data from the data source. These default data mappings are implemented in the wrappers. DB2 Information Integrator supplies a set of default data type mappings such as the following:

  - Oracle type FLOAT maps by default to the DB2 type DOUBLE.
  - Oracle type DATE maps by default to the DB2 type TIMESTAMP.
  - DB2 UDB for z/OS type DATE maps by default to the DB2 type DATE.

  If you want to customize the default mapping provided by DB2 II, then you need to create alternative data type mappings.
In order to use an alternative data type mapping for a nickname, you must create this mapping prior to creating the nickname. If you create the nickname first, you may set the appropriate mapping later as follows:

- Altering the nickname
- Changing default mapping types and recreating the nickname

For further details on data mappings, refer to the *IBM DB2 Information Integrator Data Source Configuration Guide Version 8*, available as softcopy from the Web site:

http://www.ibm.com/software/data/integration/solution

**Function mappings**

DB2 Information Integrator supplies default mappings between existing built-in relational data source functions and their built-in DB2 counterpart functions. These default function mappings are implemented in the wrappers.

You can create a function mapping if there is no default mapping available. There are several reasons for creating function mappings, as follows:

- No DB2 function corresponding to a remote data source function is available.
- A corresponding DB2 function is available, but with a specification that is different from that of its remote counterpart.
- A new built-in function becomes available at the data source.
- A new user-defined function becomes available at the data source.

The DB2 catalog view for function mappings is `SYSCAT.FUNCMAPPINGS`.

Function mappings are one of several inputs to the pushdown analysis performed by the query optimizer. If your query includes a function or operation, the optimizer evaluates if this function can be sent to the data source for processing. If the data source has the corresponding function available, then the processing of this function can be pushed down to help improve performance.

A DB2 function template can be used to force the federated server to invoke a data source function. Function templates do not have executable code, but they can be the object of a function mapping. After creating a DB2 function template, you need to create the actual function mapping between the template and the corresponding data source function.
The CREATE FUNCTION MAPPING statement gives considerable control over the scope of the mapping. For example, you can:

- Create a function mapping for all data sources of a specific type such as all Informix data sources.
- Create a function mapping for all data sources of a specific type and version, such as all Oracle 9 data sources.
- Create a function mapping for all data source objects located on a specific server.
- Disable a default function mapping. Default function mappings cannot be dropped.

For further details on function mappings, refer to the *IBM DB2 Information Integrator Data Source Configuration Guide Version 8*, available as softcopy from the Web site:

http://www.ibm.com/software/data/integration/solution

For further details on column functions, data mapping, and function mapping, refer to *IBM DB2 Information Integrator Federated Systems Guide*, SC18-7364.

7. **Step 7** involves checking to ensure that the nickname has been configured correctly by issuing an SQL statement against it as follows:

```sql
SELECT count(*) FROM nickname
```

**Note:** This query may not be appropriate for some data sources, for example, Lotus Extended Search.

**Attention:** Appendix B, “Configuring data sources in DB2 Information Integrator” on page 165, provides examples of configuring a number of the data sources used in the CFS portal described in Chapter 4, “The Druid Financial Corporation (DFC) Customer Insight solution” on page 115.

### 2.4.4 Performance considerations

Probably the most significant concern about federated technology is the issue of acceptable performance. IBM invests heavily in query optimization research and development.

The DB2 Information Integrator optimizer takes into account standard statistics from source data (such as cardinality or indexes), data server capability (such as join features or built-in functions), data server capacity, I/O capacity, and network...
speed. The following capabilities of the DB2 optimizer have a significant impact on the quality of the access plan generated:

- **Query rewrite** logic rewrites queries for more efficient processing. For example, it can convert a join of unions that drives a tremendous amount of data traffic, into a union of joins that leverages query power at the data server and thereby minimizes data traffic back to the federated server. The database administrator (DBA) can define materialized query tables (MQTs), which the DB2 optimizer can transparently leverage via query rewrite to satisfy user queries.

- **Pushdown analysis** (PDA) capability identifies which operations can be executed at the data server prior to returning results to the federated server. The DB2 optimizer can perform a nested loop join that queries a small table on one server, and uses the results as query predicates to a large table on another.

This section describes performance factors influencing federated queries as follows:

- Performance factors
- Pushdown concept
- Federated server options for best performance
- Nickname column options for best performance
- Indexes and statistics

### Performance factors

Factors that influence federated query performance include:

1. The processing power of local and remote machines, as well as the bandwidth of the intervening communication network.

2. Quality of the generated query execution plans at the federated server and the remote sources. The query execution plans influence the number of interactions required between the federated server and the remote sources, and the amount of data that is moved.

Data movement between the federated server and the remote source is a key performance factor.

The amount of data moved mainly depends upon two factors:

a. The amount of processing and filtering that can be pushed down (see “Pushdown concept” on page 47) to the remote data sources

   If there are some filtering predicates in the WHERE-clause, and the remote source is able to apply those predicates, then the federated server
pushes down these predicates to the remote server to reduce the amount of data that needs to be shipped back.

b. Data placement among multiple sources

If you join two tables and they are both on the same data source so that the join can be done at that data source without moving the tables out, then that usually results in better performance than if the two tables resided at two different sources.

In a join between tables that are not co-located, data from both tables must be moved to the federated server, which will then do the join.

**Note:** The federated server *never* moves data between remote data sources—only between each remote data source and itself.

DB2 Information Integrator has some efficient techniques for performing this data movement as follows:

a. Nested loop join in which the results of SQL sent to one data source are supplied as values for host variables sent in SQL to the second data source

b. Use of hash-joins to obtain a join result from two data sources

**Pushdown concept**

Pushdown is an important aspect of federated query processing. As mentioned earlier, if the PUSHDOWN server option is set to ‘Y’, the DB2 optimizer will consider generating a plan that “pushes down” certain parts of the query execution to the remote source.

The pushdown analysis (PDA) component (described earlier in 2.4.3, “Configuring the federated system” on page 38) decides which parts of a query can be pushed down to the data sources and processed remotely at the data sources.

The decision to push down certain parts and operators of a query depends on several factors as follows:

- The availability of required functionality at the remote source
  - If the remote source is simply a file system with a flat file, then it is probably not possible to push down any filtering predicates.
- The options specified in the server definition of a remote source as discussed in “Federated server options for best performance” on page 48
  - For instance, if the collating sequence at the federated server is different from the one at the remote server, then operations on string data like sorting and
some predicates involved in the query have to occur at the federated server, and cannot be pushed down.

- The following issues influence pushdown:
  - You can specify that a remote column always contains numeric strings so that differences in the collating sequence do not matter. Sometimes remote source functionality is dependent on data type issues, for example, a remote function may accept only certain argument types.
  - Attributes within the DB2 Information Integrator wrappers that indicate which operations and functions are supported by the type and version of the data source.
  - Function mappings in the DB2 Information Integrator catalog.
    The function mappings in the catalog are created by the DB2 Information Integrator administrator, and are additions and overrides to the default function mappings that are in the wrappers.

For further details, refer to the *Using the federated database technology of IBM DB2 Information Integrator* white paper by Anjali Grover, Eileen Lin, and Ioana Ursu, available from the Web site:


**Federated server options for best performance**

Each remote data source has an entry in the federated server's catalog. Using special DDL, you can add more entries to the catalog that describe attributes of the server, called server options.

This section discusses the most common federated server options that can significantly impact the decisions made by the DB2 optimizer, and thereby on query performance:

- **COMM_RATE, CPU_RATIO, IO_RATIO**
  These attributes describe the communication links to the remote source, and the relative speed of the remote system's CPU and I/O. By default, the federated server assumes that the remote machine is equal in power to the local machine, and that there is a 2 MB/sec link to it. Setting these options to indicate a more powerful remote machine or a faster link will tend to encourage query pushdown. These knobs are not perfect, but they are a way to indicate to the DB2 optimizer that a remote machine is fast or slow.

- **COLLATING_SEQUENCE**
  If you set this attribute to 'Y', you are telling the PDA that the remote source sorts characters the same way that DB2 does. This means that the federated server can consider pushing down operations involving sorting, grouping, or
inequality comparisons on characters and VARCHAR columns. For instance, setting COLLATING_SEQUENCE to ‘Y’ allows the DB2 optimizer to push down ORDER BY clauses that reference character and VARCHAR columns. Pushdown of these operations on numeric, date, time, and date/time columns is not affected by this server option.

**Attention:** If the remote source’s collating sequence does not match DB2’s after you set COLLATING_SEQUENCE to ‘Y’, you could get incorrect results.

- **VARCHAR_NO_TRAILING_BLANKS**

  This attribute is used for databases like Oracle that do not pad VARCHAR fields with trailing blanks. The query compiler uses this information while checking any character comparison operations to decide the pushdown strategy to evaluate the operations.

  DB2 uses blank padded comparison semantics while comparing character strings of unequal lengths. The comparison is made by using a copy of the shorter string, which is padded on the right with blanks so that its length is equal to that of the longer string. This means that the string “A” is considered equivalent to “A “ in DB2 UDB.

  However, this behavior does not apply to all character data types across all data sources, such as the VARCHAR2 data type in Oracle.

  In general, comparison operations on string columns without blank padding comparison semantics need to be evaluated locally unless the query compiler is able to find functions to enforce similar logic remotely. For certain operations such as predicates, the federated system maintains performance by rewriting the predicates to ensure the same semantics when these predicates are sent to an Oracle server. Performance of operations such as DISTINCT, ORDER BY, GROUP BY, UNION, column functions (MIN()/MAX()) evaluation, relational comparison and IN predicates might be affected if this column option is set.

  This attribute is used for databases like Oracle that do not pad VARCHAR fields with trailing blanks. If you are sure that your VARCHAR columns do not contain trailing blanks to begin with, then setting this option at the server level will allow the use of the remote source’s non-blank-padded comparison operations that return the same results as DB2.

  Here again, setting VARCHAR_NO_TRAILING_BLANKS to ‘Y’ when trailing blanks do exist at the remote data source can return erroneous results.
Nickname column options for best performance
The NUMERIC_STRING and VARCHAR_NO_TRAILING_BLANKS nickname column options impact the decisions made by the DB2 optimizer and thereby query performance.

- **NUMERIC_STRING**
  
  This nickname column option applies to character data types and is applicable to those data sources for which the COLLATING_SEQUENCE server option is set to ‘N’.

  The federated system does not push down any operations that can produce different results due to differences in collating sequences between the federated database and the remote data source. Suppose that a data source has a collating sequence that differs from the federated database collating sequence—in this case, the federated server typically does not sort any columns containing character data at the data source. It returns the retrieved data to the federated database, and performs the sort locally.

  However, suppose that the column is a character data type (CHAR or VARCHAR) and contains only numeric characters (0 through 9). This fact can be indicated to the DB2 optimizer by setting the NUMERIC_STRING column option to ‘Y’. This gives the DB2 query optimizer the option of performing the sort at the data source. If the sort is performed remotely, you can avoid the overhead of porting the data to the federated server and performing the sort locally.

- **VARCHAR_NO_TRAILING_BLANKS**
  
  As discussed in “VARCHAR_NO_TRAILING_BLANKS” on page 49, this is also a server option. However, specifying this option at the column level provides greater flexibility and granularity if there are multiple tables in a database that do not all have missing trailing blanks.

Indexes and statistics
For the DB2 optimizer to make superior access path decisions, it needs knowledge about available indexes on it, and to have accurate statistics about a remote object. The federated server relies on the remote source for its index and
statistics information about each remote object. This information is retrieved when a nickname is created and stored in the federated server’s global catalog.

Attention: The information in the global catalog is not automatically maintained if statistics on the remote object are refreshed or indexes are added or dropped.

It is the federated DBA responsibility to ensure that the statistics and metadata information in the global catalog are kept in sync with the corresponding statistics and index information of the remote data objects.

If you believe that the federated server’s information is out of sync with that of the remote data source, drop and re-create the nickname to retrieve fresh information from the remote data source. This may be disruptive, since the action invalidates views and packages that reference the nickname, and also clears any authorizations (GRANTS) on the nickname.

Manually adding statistics or using an “index specification”\(^2\) may be a non-disruptive alternative to dropping and recreating a nickname, as discussed in “Indexes” on page 51 and “Statistics” on page 52.

Index information for a nickname will not be retrieved if:

- The nickname for a table has no indexes.
- The nickname is for a remote view, Informix synonym, table structured file, Excel spreadsheet, or XML tagged file.

Views and Informix synonyms do not have index information in the data source catalog, but the tables referred to by the view or synonym may have indexes.

- The remote index has a column with more than 255 bytes, or a total index key length with more than 1024 bytes.
- The remote index is on LOB columns.

Another possible case in which index information for a nickname will be missing from the catalog is when you create a new index on a remote object after creating the nickname for the object. DB2 II is not notified of the change,

\(^2\) This is a set of metadata catalog information about a data source index.
and has no way of knowing that it needs to update its index information in the global catalog to include the new index.

To notify DB2 II of the existence of a missing index for a nickname, you can create an index specification to record information that includes the keys that comprise the index, but does not include any statistical information. Thus, creating an index specification does not record as much data about the remote index as would be obtained by dropping and recreating the nickname, but it is less disruptive. If you want to ensure that index information for nicknames includes all available statistical data, you need to follow the steps described in “Statistics” on page 52.

Similarly, when a nickname is created for a remote view, the federated server is unaware of the underlying tables (and their indexes) from which the view was generated. An index specification can be used to tell the DB2 optimizer about indexes on the underlying tables of a remote view, which may help it choose better access paths for queries involving the nickname to the remote view.

In either case, you supply the necessary index information to the global catalog using the CREATE INDEX... SPECIFICATION ONLY statement. No physical index is built on behalf of this nickname—only an entry is added to the system catalog to indicate to the query optimizer that such a remote index exists. This helps the query optimizer in generating remote plans for relational nicknames.

An index specification that defines a unique index also conveys the information about the uniqueness of the index columns to the federated system. Just like a regular unique index definition registered during relational nickname registration, such uniqueness information can help the query optimizer to generate a more optimal plan with strategies such as eliminating unnecessary DISTINCT operations.

Statistics

DB2 stores statistical information on objects stored in the database including tables, table columns, and indexes. These statistics help the DB2 optimizer work out the best access plan for queries. In order to help the DB2 optimizer do its job, it is necessary to keep the statistics for each object in the database up to date. DB2 stores statistical information for nicknames as well. As nicknames are really just local references for remote tables, they look much like local tables to the DB2 optimizer. In fact, statistics for both local tables and nicknames are stored in the same way, and are accessible through DB2 system catalog views in the schema SYSSTAT.

DB2 stores the following types of nickname statistics:
- Table cardinality (row count) and page counts (SYSSTAT.TABLES)
– Column cardinality (number of unique values) and column minima and maxima (SYSSTAT.COLUMNS)
– Information on remote indexes for nicknames (SYSSTAT.INDEXES)

The amount of statistical information stored for nicknames varies depending on the type of remote data source involved, for example, while table cardinality is available for nicknames on most sources, column minima and maxima are only available for some sources.

As mentioned earlier, nickname statistics and index information are retrieved from available information on the remote data source at the time that the nickname is created. Therefore, nickname statistics can only be as good as available remote statistics at nickname creation time. In particular, if no statistics have been collected on a remote object before a nickname is created, the nickname itself will not have any statistics. Similarly, if statistics are updated for an object on a remote data source, the new information is not automatically propagated to the corresponding DB2 nickname. Again, as discussed earlier, the same principle applies to indexes—DB2 is only aware of remote indexes for an object that are in existence at the time of nickname creation.

To make sure that DB2 nicknames have the best possible statistics and index data:

– Update statistics for objects on remote sources and create remote indexes before defining DB2 nicknames to them, so that DB2 can retrieve and store the current statistics information for the nickname.
– If updated statistics are collected for a remote object, or a new remote index is created, the DB2 statistics and index information for the corresponding nickname will be out of date. There is no runstats for nicknames. You can drop and re-create the DB2 nickname for the object to re-start the remote statistics and index discovery process, and retrieve the updated information. As mentioned earlier, this can be quite disruptive due to side effects such as view and package invalidation, and loss of GRANTs for the nickname.

Some remote data sources store little or no statistical data for their objects, or the information is not available to an external client. In this case, DB2 II is not able to retrieve statistical data at nickname creation time. An alternative in this situation is to:

a. Use a special tool called get_stats that populates DB2 nickname statistics by issuing queries directly against the nickname. This tool is available at:
   

b. Manually update the statistics and create the index information using SQL. Manually supplying the statistics for important columns of frequently used
remote tables may help the DB2 optimizer build better query plans. Basic properties about a column, such as minima and maxima values and the number of unique values it has, can have a significant impact on the quality of the query plan, and thereby query performance.

**MQT performance**

A materialized query table (MQT) is a table that contains pre-computed aggregate values, or a filtered subset of base tables. An MQT’s definition is based on a query that accesses one or more tables. The target of the query may also be a nickname. An MQT is refreshed using a deferred approach or an immediate approach. However, an MQT based on a nickname can only be refreshed via a deferred refresh approach.

When a query is written against base tables/views or nicknames, the DB2 optimizer will automatically rewrite the query to target the MQT instead (if appropriate) in order to satisfy the original query. This can result in significant performance gains.

**Attention:** MQT functionality is somewhat similar to the role of a DB2 index, which provides an efficient access path that the query user is typically unaware of. However, unlike an index, a user may directly query the MQT, but this is not generally recommended since it would detract from the appeal of an MQT being a black box that a DBA creates and destroys as required to deliver superior query performance.

MQT support for nicknames can help improve the performance of distributed queries involving local and remote data. Figure 2-8 on page 55 shows a possible four-way join across two local tables, and two nicknames referencing a remote data source.
If the four-way join query is executed frequently and the query can tolerate remote data (accessed through nicknames) being slightly out-of-date, then an MQT created on nicknames would provide a local copy of the remote data as shown in Figure 2-9 on page 56, and provide significant performance benefits. As mentioned earlier, an MQT based on nicknames is not automatically updated, and must be manually refreshed.
The advantage of using MQTs rather than replication is that the DB2 optimizer may actually choose to use the nickname (and not the local MQT) if the query joins with other remote data on the same data source. But if the nickname is joined with local data, then the DB2 optimizer will choose the MQT instead of the nickname. MQTs are like other tables and must be tuned with indexes and current statistics to ensure that the DB2 optimizer makes the correct decision about choosing the MQT.
Data Integration and Information Aggregation patterns

In this chapter we describe the various Business, Integration, Application and Runtime patterns that relate to data management.

The topics covered are:
- Business and Integration patterns overview
- Data Integration:: Federation
- Data Integration:: Population
- Data Integration:: Two-way Synchronization
- Information Aggregation:: User Information Access
- Self Service:: Agent pattern overview
3.1 Introduction

From a data management point of view, the patterns of interest are the Data Integration (which was previously known as data-focused Application Integration) patterns and the Information Aggregation business patterns, since they relate to the creation and management of data stores, and the general access to and manipulation of that data.

Briefly:

- **Business patterns** identify the interaction between users, businesses, and data. Business patterns are used to create simple, end-to-end e-business applications.

- **Integration patterns** connect other Business patterns together to create applications with advanced functionality. Integration patterns are used to combine Business patterns in advanced e-business applications. Integration patterns may also be used within a single Business pattern to integrate business applications and databases.

- **Application patterns** use logical tiers to illustrate the various ways to configure the interaction between users, applications, and data. The focus of these tiers is on the application layout, shape, and application logic for the associated data. In some cases though, multiple Application patterns may be required to define a complete interaction between users, applications, and data. In such cases, the results of one Application pattern feed into another Application pattern, so that the combination of Application patterns results in a functioning e-business solution.

- **Runtime patterns** define functional nodes (logical) that underpin an Application pattern. The Application pattern exists as an abstract representation of application functions, whereas the Runtime pattern is a middleware representation of the functions that must be performed, the network structure to be used, and systems management features such as load balancing and security. In reality, these functions, or nodes may exist on separate physical machines or co-exist on the same machine. In the Runtime pattern, this is not relevant. The focus is on the logical nodes required and their placement in the overall network structure. Typical logical nodes include the following:
  - User/Internal User node
  - Application Server node
  - Presentation Server node
  - Collaboration node
  - Content Management node
  - Data Integration node
  - Search and Indexing node
– Data Server/Services node

For an up-to-date list and a detailed description of runtime nodes, please refer to the Patterns Web site:


**Important:** The patterns described here are a significant rework of those in the Information Aggregation (User-to-data) and data-focused Application Integration areas on the Patterns for e-business Web site (http://www.ibm.com/developerWorks/patterns/). They also represent an evolution of the thinking described in the redbook *Patterns: Portal Search Custom Design*, SG24-6881-00.

The major change has been to consolidate the Application patterns for both structured and unstructured into a single, self-consistent set. In this redbook, the focus in the Runtime patterns is on the structured data area. When dealing with unstructured data, and particularly Web-based content, the redbook *Patterns: Portal Search Custom Design*, SG24-6881-00, should be read in conjunction with this chapter.

This chapter introduces the Business and Integration patterns and describes the different Application patterns and corresponding Runtime patterns for the Data Integration and Information Aggregation patterns.

### 3.2 Business and Integration patterns overview

Prior to examining specific Application patterns in more detail, it is important to understand the relationship between the Business and Integration patterns covered in this redbook, in particular the relationship between the Data Integration patterns and Information Aggregation. Solutions based on these two patterns tend to adopt the following model:

- **Data Integration** patterns serve to integrate the information (or data) used by multiple applications. Such integration occurs in advance of the actual access to the data by an end user.

  The basic premise is that existing data is available in both structured and unstructured forms in application data repositories managed by other applications.

  A proven repeatable pattern is therefore needed for combining this data into a single optimized form and place (or perhaps multiple forms and places). Data Integration patterns provide this functionality.
Information Aggregation patterns allow users to access and manipulate data that is aggregated from multiple sources. Conceptually, these patterns take the data that is available from the multiple sources and applications via Data Integration, and provide tools to extract useful information and value from such large volumes of data.

A modern solution, however, is likely to require more than these two high-level patterns. Applications that require both analysis activities and updates to the base data are becoming the norm. For example, an Internet customer of a financial institution may wish to do what-if analysis of investment options, consult with a service representative via instant messaging, browse some assorted Web sites, and then place an order based on the results of that activity. In such a scenario, a combination of patterns will be required as follows:

- Self Service and Collaboration business patterns
- Data Integration and Information Aggregation patterns
- Access Integration pattern to provide a consolidated portal for the user

Figure 3-1 on page 61 depicts this relationship between these categories of Business and Integration patterns, and positions the key Application patterns of interest within them.
In Figure 3-1, the Data Integration patterns are used to move, copy, and transform data and information between different stores and representations in advance of such data being required by the user. Arrows are used to show this prior data movement.

Three types of data store are characterized by how the data is managed and whether the data is structured or unstructured:

- **Systems of record** (or operational systems) contain structured data and are managed by applications that are responsible for the transactional consistency of such data.

- **Unstructured data stores** hold data such as e-mail, voice, or images managed through collaborative and other means.

- **Derived data stores** contain structured data, unstructured data, or both, and are managed through Data Integration processes that create and maintain
consistent, conglomerated, or summarized information needed for decision support, mining, analysis, content search, and similar (largely) read-only needs.

Note: We see that each type of data store is associated only with a particular Business pattern in Figure 3-1. This is a slight generalization—the distinctions between different types of data stores and different Business patterns is becoming somewhat blurred today, and may disappear in the future. For the moment, however, the distinction remains valid and useful.

Data Integration is a subset of the Application Integration patterns (also known as Enterprise Application Integration) that serve to integrate multiple Business patterns, or to integrate applications and data within an individual Business pattern. Application Integration has two approaches for providing such integration:

- Process Integration (aka process-focused Application Integration), which involves the integration of the functional flow of processing between the applications
- Data Integration (aka data-focused Application Integration), which involves the integration of the information used by applications

Since our focus is on data management, we concentrate only on the Data Integration and Information Aggregation application patterns as shown in Figure 3-2 on page 63. Also, because Data Integration and Information Aggregation are so interdependent, their business and IT drivers are closely related, as described in 3.2.3, “Business and IT drivers” on page 64.
3.2.1 Data Integration application patterns

Figure 3-2 shows that Data Integration has three Application patterns as follows:

1. **Federation**
   Federation is a fundamental Application pattern that can be used in a number of ways, typically in support of another pattern. Its basis is a federated query that hides the fact that its target data may be local, remote, or fully distributed.

2. **Population**
   Population involves data (or information) being copied and, if necessary, transformed, from one place to another "off-line", in advance of when it is required by a user or application. Population is a one-way process where one data store is the source of the process and the other the target.

3. **Two-way Synchronization**
   Two-way Synchronization has in common with Population the activity of data (or information) being copied and, if necessary, transformed, from one place
It differs from Population in that with Synchronization, data can flow in both directions. Population is clearly the simpler of the two patterns. Conceptually, Two-way Synchronization may be thought of as a combination of two population patterns operating in opposite directions with additional function needed to ensure that conflicting updates are not applied to either data store.

### 3.2.2 Information Aggregation application patterns

Figure 3-2 on page 63 shows that Information Aggregation contains only one Application pattern, called User Information Access.

User Information Access is the basic pattern through which users interact directly with information through the generic functionality (typically) provided by a database management system (DBMS). Allowing such direct interaction provides users with the ability to access and analyze the information in any way they choose. It is thus optimized for decision support, search, analysis, mining, and other investigative activities, where the way the data will be used cannot be easily predicted in advance.

However, with the inherent flexibility of a DBMS, users may also modify the data if required. Information Aggregation can therefore compromise the integrity of the underlying data. Contrast it with the Self Service business pattern, where access to the data is under the control of the owning applications, which limit the ways in which the user can access or change the underlying data.

In order to ensure that users do not make unintended or disallowed changes to critical business data, we recommend that User Information Access acts against derived copies of the actual business data. When users change such derived data via the Information Aggregation pattern, these changes can either be totally ignored (the data is considered to be the user's personal data) or special procedures can be put in place to integrate the changes back into the original source systems using Data Integration patterns when changes made by the user have wider business relevance.

### 3.2.3 Business and IT drivers

As mentioned earlier, the close interdependence of Information Aggregation and Data Integration means that their business and IT drivers are also closely related. In general, business requirements drive Information Aggregation to a greater extent than Data Integration, whereas the opposite is the case for the IT drivers.
The key business drivers, particularly for Information Aggregation, are:

- Distill meaningful information from a vast amount of structured data, both in summary and in detail.
- Provide easier access to vast amounts of unstructured data through indexing, categorization, and other advanced forms of summarization.
- Improve organizational efficiency by extensive reconciliation, transformation, restructuring and combination of disparate data.
- Reduce the latency of business decisions.
- Provide a unified customer view across Lines of Business (LOB).
- Support effective cross selling.
- Provide mass customization.
- Enable easy adaptation during mergers and acquisitions.

The key IT drivers are:

- Promote consistency of operational, informational, and content data stores.
- Leverage value of and existing skills in the legacy investment.
- Integrate the back-end applications and data stores.
- Minimize the total cost of ownership (TCO).
- Improve availability.
- Provide scalability.
- Improve maintainability.

### 3.3 Data Integration:: Federation

We describe the basic Federation application pattern and its variations along with their corresponding Runtime patterns. The key business and IT drivers for this pattern are described briefly along with guidelines for their usage.

The Federation application pattern is a basic Data Integration application pattern that provides access to many diverse data sources and provides the appearance that these sources are a single logical data store. This appearance is delivered as follows:

1. Exposing a single consistent interface to the user (or application) that invokes the function
2. Translating that interface to whatever interface is needed for the underlying data
3. Compensating for any differences in function between the different sources
4. Allowing data from different sources to be combined into a single result set that is returned to the user

This section is organized as follows:

- Business and IT drivers
- Federation pattern
- Federation: Cache variation pattern
- Guidelines for usage and scenario

### 3.3.1 Business and IT drivers

Federation may be required in any business process where the data needed exists in a number of different locations. Such diversity may be the result of historical, technical or organizational factors. Federation is preferred over other data integration methods, such as Population, when the access required meets one or more of the following criteria:

- (Near) real-time access is needed to rapidly changing data.
- Making a consolidated copy of the data is not possible for technical, legal or other reasons.
- Read/write access to the data is required, rather than read-only.
- Reducing or limiting the number of copies of the data is a goal.

The Federation application pattern's connector/adaptor design allows for improved maintainability, minimized TCO, leveraging of existing technology investments, and reduced deployment and implementation costs.

### 3.3.2 Federation pattern

The Application and Runtime patterns for the basic Federation pattern are described here.

**Federation application pattern**

Figure 3-3 on page 67 represents the Federation application pattern.

---

1 Connectors/adapters contain the logic required to access the data, perform the necessary function, and then send the results back to the requestor tier.
Figure 3-3  Federation application pattern

When called by an application, Federation uses its metadata store to determine where and in what format the required data is stored. Metadata mapping also enables the decomposition of the unified query into requests to each individual repository. The information model thus appears as one unified virtual repository to users. Using adapters for each target repository, data is accessed and retrieved. Based on its knowledge of functionality, performance, and other factors, Federation determines the optimal plan for performing the incoming query, pushing down function to the remote data stores or compensating for missing function locally, and storing intermediate results in the local temporary store. Federation then returns a single result to the calling application, thus integrating the multiple disjoint formats into a common federated schema.
Federation supports both structured and unstructured data, as well as read-only and read/write access to the underlying data stores. Read-write access is best limited to single remote sources, in part because of fundamental theoretical limitations in support for two-phase commit in a fully distributed environment.

**Federation runtime pattern**

Figure 3-4 shows the basic Runtime pattern corresponding to the Federation application pattern shown in Figure 3-3 on page 67.

The Data Integration node is at the core of the Federation runtime pattern. It contains the knowledge and functionality that enables real-time access to and manipulation of data in disparate locations and formats across multiple Data Server/Services nodes. The function of the Data Integration node is invoked by an Application Server/Services node, which may be entirely separate, or integrated with the Data Integration functionality.
A Data Server/Services node is a generic data storage node that provides managed, persistent storage of any type of data and a means to directly access and manipulate that data. The data may be stored in files and accessed through file I/O routines or may be stored in a database with more structured and managed access methods.

**Note:** Although omitted for simplicity of representation, an Application Server/Services node can be substituted for the Data Server/Services node where access to the data is provided through an application API rather than directly to the database management system.

The flow is as follows.

1. A requesting application makes a query of data from the "federated" data source, for example, a simple SQL Select request.
2. The Data Integration node processes the request, and utilizing its metadata (which defines the data sources) passes on the requests to the appropriate data sources.
   
   In many cases, the data integration/federation logic within the Data Integration node may be logically separate from the data connector logic. This data connector logic spreads out the overhead of making the query to multiple data sources, allowing the queries to run in parallel against each database. When performance is of major concern, multiple logical data connectors may exist to process queries against a single data source—the idea here being to prevent any single node in the process from becoming a bottleneck if too many requests run against one data source.

3. In all cases, the results that are returned from each individual data source must then be aggregated and normalized by the data integration layer so that these results appear to be from one "virtual" data source.

4. The results are then sent back to the requesting application, which has no idea that multiple data sources were involved.

### 3.3.3 Federation: Cache variation pattern

Figure 3-5 on page 70 represents the Federation: Cache variation pattern.
Local temporary storage can be used to cache data returned from read-only queries to remote data sources. Under defined circumstances, this cache can be used to speed up query response time or to compensate for a data source that is temporarily off line. Such function must be used carefully, however, as the cached data and its underlying source may no longer be in sync (there may be a latency involved).
It is also possible and often necessary to maintain the contents of the cache. This involves the use of the Population application pattern described in 3.4, “Data Integration:: Population” on page 72.

**Note:** This is an emerging variation pattern where significant research work is currently underway, and will not be discussed further here.

### 3.3.4 Guidelines for usage and scenario

The Federation application pattern is appropriate in the following situations:

- When an infrastructure for integrating data sources is needed, avoiding propagation and/or additional repositories.
- When there is a need for unified information access by portal projects. It is useful where relational data and text data need to be accessible through one common Web search interface.
- For structured data-only/business intelligence solutions where the frequency of change of application data would prohibit an Operational Data Store (ODS) type of solution, or where more transient access to the source data meets the business need.

While this solution eliminates most duplication of data that exists in other data integration patterns, it requires metadata mapping at setup and during the processing of the federated query. Such federated real-time query environments need to be tuned for optimum performance.

**Usage scenario**

A customer support representative requires information about a certain product to answer a customer's support call. Documents exist in multiple locations, a file system, a knowledge base, a Web site, etc., from which an answer could be found. Additionally, search interfaces to some of these locations have already been created in the past. Rather than requiring the customer support representative to use multiple search engines to locate the needed information, a single “federated search” application could be created that would perform a unified query against all of the existing search engines, and return one set of normalized results, so that the customer support representative can quickly find the relevant information to solve the customer's problem.
3.4 Data Integration:: Population

We describe the basic Population application pattern and its variations along with their corresponding Runtime patterns. The key business and IT drivers for this pattern are described briefly along with guidelines for their usage.

The Population application pattern has a very simple model. It gathers data from one or more sources, processes that data in an appropriate way, and applies it to some data target. The primary business driver for population is to gather and reconcile data from multiple data sources in advance of a user's need to use this information.

In some cases, the reconciliation is sufficiently simple that it can be conceived as a single (integrated) function. In many cases, however, the transformation and restructuring is rather complex or the gathering phase has unique characteristics. This leads to four variations on the basic Application pattern as follows:

- Population: Multi Step variation pattern
- Population: Multi Step Gather variation pattern
- Population: Multi Step Process variation pattern
- Population: Multi Step Federated Gather variation pattern

These population patterns are often applied towards business intelligence-related business problems. They can also be utilized to provide content feeds into an e-business portal of more unstructured data. This "content" can then be accessed via the portal, or even searched via basic portal search capabilities.

This section is organized as follows:

- Business and IT drivers
- Population pattern
- Population: Multi Step variation pattern
- Population: Multi Step Gather variation pattern
- Population: Multi Step Process variation pattern
- Population: Multi Step Federated Gather variation pattern
- Guidelines for usage and scenario

3.4.1 Business and IT drivers

Any business need that requires a specialized copy of data (derived data) from a pre-existing source may indicate the use of the Population application pattern or one of its variation patterns. These needs are most often seen in business intelligence and content search and related applications. However, some cases are also seen in a pure operational environment, where a dedicated copy of data is needed. A key indicator is that the use of the derived data is read-only or a
close approximation to it. If there are significant amounts of read/write usage of the derived data, the Two-way Synchronization pattern is indicated.

Such specialized, derived data copies may be:

- Subsets of existing data sources: Limiting access for ease of use or understanding, for security or privacy, or for the needs of a particular business process
- Modified versions of existing data sources: Creating point-in-time (stable) or historical versions of the source data, cleansing data of errors or changing structures
- Combinations of existing data sources: Joining or reconciling data from multiple sources
- Creation of a more usable and relevant organization of documents or unstructured data, built from a vast set of original documents, and based on specified selection criteria

The business objective can often be summarized as providing the user with quick access to useful information instead of bombarding the user with too much, irrelevant, incorrect, or otherwise useless misinformation.

In many cases, it is the IT drivers rather than the business drivers that dictate the use of the Population set of patterns, because in many cases one can envisage that the business need can be equally well satisfied either by direct access to the original sources or to a copy of those sources. These IT drivers include, among others:

- Improved performance of user access
- Protection of performance of data source systems
- Reliability of access to and extended availability of the required data
- Load distribution across systems

### 3.4.2 Population pattern

The Application and Runtime patterns for the basic Population pattern are described here.

**Population application pattern**

Figure 3-6 on page 74 represents the basic Population application pattern.
Figure 3-6 represents the basic population functionality as a "read dataset — process — write dataset" model.

- There can be one or more source data stores that are read by the population application. These source data stores are created and maintained by other processes. The target data stores are the output of the population application. These can be the final output from the process, or can be an intermediate data store used as a source for another step in the process.

- The extraction rules may range from a simple rule such as including all data, to a more complex rule, prescribing the extraction of only specific fields from
specific records under varying conditions. Similarly, the load rules for the target data can range from a simple process of overwriting the target data store to a complex process of inserting new records and updating existing records.

- The metadata contains the rules describing which records from the source are read, how they are modified (if needed) on their way to the target, and how they are applied to the target. The rules are depicted in this way to emphasize the best practice of having a rules-driven application, rather than hard-coding the rules in the application; this facilitates easier maintenance.

The metadata also describes the output that the population application produces, such as statistics, timing information, and so on. In general, both source and target can contain any type of data, including structured and unstructured data. However, in the majority of the cases, this application pattern is used for moving/copying structured data from one data store to another with relatively simple manipulation of the data.

**Population runtime pattern**

Figure 3-7 represents the basic Runtime pattern corresponding to the Population application pattern shown in Figure 3-6 on page 74.

A Data Server/Services node is a generic data storage node that provides managed, persistent storage of any type of data and a means to directly access
and manipulate that data. The data may be stored in files and accessed through file I/O routines or may be stored in a database with more structured and managed access methods. Although omitted for simplicity of representation, an Application Server/Services node can be substituted for the Data Server/Services node where access to the data is provided through an application API rather than directly to the database management system.

The Population node is a specialized processing node designed and optimized for reading and writing data from/to data stores and transforming the data, often in sophisticated ways, as it passes through. Some Population nodes are further specialized for handling the data under different circumstances, such as efficient throughput of large batches of records that require extensive transformation, or for fast throughput of individual records in near real-time.

Multiple data sources may be involved in the base Population runtime pattern process; and reasonably sophisticated filtering, cleansing, and transformations may occur within the Population function. The main point is that this process can occur in a single step.

### 3.4.3 Population: Multi Step variation pattern

The Application and Runtime patterns for the Population: Multi Step variation pattern are described here.

**Population: Multi Step variation application pattern**

Figure 3-8 on page 77 represents the Population: Multi Step variation application pattern.
In the Multi Step variation of the Population application pattern, the basic population function of the Population application pattern is decomposed into its three primary constituents or steps:

- Gather
- Process
- Apply

Note: We have deliberately avoided using the traditional extract, transform, and load terminology in order to accommodate the emerging functionality requirements and variations of population patterns.

The intermediate target data created by one step acts as the source data for the subsequent step. In some cases, the temporary stores may be physically
instantiated files; in more modern implementations, the data may be "pipéd" from one step of the population process to the next.

Figure 3-8 on page 77 shows the three logical steps: Gather, Process and Apply. In most best practice implementations, these functional steps contain additional sub tasks.

- The Gather step extracts data according to some defined rules from the source data store. This data store is typically owned by another application and used in a read/write fashion by that application. This data source may also be a special kind of data source created by system or user processes. The extraction rules may range from a simple rule such as including all data, to a more complex rule, prescribing the extraction of only specific fields from specific records under varying conditions.

Breaking out this step recognizes that Gather may have very specific function or placement depending on the particular implementation. Gather may have to read unusual data structures or may need to take into account very specific conditions in the data source, such as "read this field as character if a particular flag is set in some other field; otherwise read as decimal". Furthermore, particular instances of Gather may have to be co-located with their corresponding data sources.

- The Process step transforms data from an input to an output structure according to supplied rules. Processing covers a wide variety of activities, including reconciling data from many inputs, transforming data in individual fields based on predefined rules or based on the content of other fields, and so on. When two or more inputs are involved, there is generally no guarantee that all inputs will be present when required. The Process step must be able to handle this situation.

- The Apply step loads the processed data into the target data store. Applying the target data can range from a simple process of overwriting the target data store to a complex process of inserting new records and updating existing records.

In any population process, different Apply functions may need to be invoked under different circumstances. For example, the first time a target is loaded is a straightforward write, but later updates may require logic to determine whether data should be overwritten, appended to, or some other custom operation.

- A common metadata store links the three steps. This store contains the metadata that describes the data to be gathered, the rules for processing it and the way to apply the resulting data to the target. It also serves as a store for information about the success or failure of each step and as a means for inter-step communication.
When Population consists of multiple steps, there clearly must exist an entity that controls and orchestrates the entire set of function. This is not shown explicitly in the diagram simply because this controlling function seldom exists as a separate entity. It may be considered to be a function of the Process step in this case.

The actual implementation of Population: Multi Step can involve a fewer or greater number of steps than the three shown here. In such cases, the steps in Figure 3-8 on page 77 must be adjusted accordingly, and consideration must be given to the placement of any additional tiers. A number of special cases are treated in the variations below. It is also important to note that this application pattern has been generalized to cover any source data store and target data store.

**Population: Multi Step variation runtime pattern**

Figure 3-9 represents the Population: Multi Step variation runtime pattern corresponding to the Population: Multi Step variation application pattern shown in Figure 3-8 on page 77.

![Figure 3-9 Population: Multi Step variation runtime pattern](image)

The major difference in this variation is that the Population node in the basic Population runtime pattern shown in Figure 3-7 on page 75 is now split into three steps, representing a common reality that population may often be sufficiently complex to require separate nodes to handle the three different steps of Gather, Process, and Apply.

Figure 3-9 also shows how all functional elements may occur one or multiple times. Thus, a single Gather node may access data on multiple Data
Server/Services nodes. This is similar for the Apply node. A single Process node may invoke multiple Gather or Apply nodes as required (not shown in Figure 3-9 on page 79). In addition, there may be multiple Process nodes involved. However, a real-life implementation is unlikely to be as complex as this; a suitable subset of these nodes will usually suffice.

Grouping of different population functions also varies by product and implementation. In some case, Process and Apply functions may be collocated on a single node. In other cases, Process and Gather may reside together. These product-driven variations need to be considered in the final design.

3.4.4 Population: Multi Step Gather variation pattern

The Application and Runtime patterns for the Population: Multi Step Gather variation pattern are described here.

Population: Multi Step Gather variation application pattern

Figure 3-10 on page 81 represents the Population: Multi Step Gather variation application pattern.
The Multi Step Gather shown in Figure 3-10 is an extension of the Population: Multi Step variation shown in Figure 3-8 on page 77 that recognizes that the Gather function itself may need to occur in multiple steps.

In Figure 3-10 an independent Gather step (Gather 1) extracts a specialized subset of the data and stores it in a temporary or persistent store. This data store is read, perhaps in conjunction with the original data store, by the Gather step (Gather 2) of the Population: Multi Step variation that completes the overall population process.
There are a number of circumstances where Multi Step Gather is found for structured and unstructured data, as follows:

- **Structured data - Gathering changed data**

  When used with structured data, the primary driver for the Multi Step Gather variation is to reduce the latency of updating an existing target with changes that are occurring in the source system. Without such an ability to collect only the changes occurring in the source system(s), one would need to constantly rebuild the target data by scanning the entire contents of the data source(s) resulting in a high latency of content between the source and the target. The Multi Step Gather variation is therefore applied towards all data warehouses, as well as real-time or near real-time business intelligence related business problems and operational data stores (ODS) where very low latencies between the source and target data are critical.

  In Figure 3-10 on page 81:

  - Gather 1 identifies the changes that have occurred in the data source and writes them out to a target data store containing only changed data—either every occurrence or a consolidation of multiple occurrences. The Gather method involved varies by data source. For example, with relational data sources, the data replication features of most products provide a change capture facility that collects changes as they occur and writes them out to another table. For non-relational data sources, a more complex mechanism may be required, such as having an application create an "audit" journal of all changes, or having a general purpose program compare different versions of a data source and perform a DIFF operation to identify the changes and then write them out to another relational or non-relational target. In some cases, the Gather function is specific to a single type of data source; in other cases, it can handle a number of (usually related) types.

  - The Temporary/Persistent Store can be the final output from the process (for example, in the generation of an audit journal), or can be an intermediate store used as input to the remainder of the steps as shown for updating data warehouses, data marts, or an ODS.

  - The separate metadata contains the rules describing the specific objects of interest, the frequency of collecting changes, the collection of every change occurrence or a consolidation of the changes over a given interval, and the pruning of the target data store. Here too, the rules are depicted in this way to emphasize the best practice of having a rules-driven application, rather than hard-coding the rules in the application; this facilitates easier maintenance.
Unstructured data - Creating indices and taxonomies

For unstructured data, this approach is often required to create indices or taxonomies of the source documents.

The Population: Multi Step Gather variation also replaces the "Population Crawl and Discovery" application pattern previously used for unstructured data. It provides a structure for applications that retrieve and parse documents and data, and create resulting indices, taxonomies, and other summarizations of the original data. The Multi Step Process variation described in 3.4.5, “Population: Multi Step Process variation pattern” on page 85, may also be required.

These result sets may include:

- A basic index of relevant documents that match a specified selection criteria
- A categorization or clustering of common documents from the original data
- An automatically built taxonomy of the original data, to allow for easy browsing
- Locating expertise by automatically mapping the authors of the original data to topics of "experts"—based on the contents of the documents and the categories discovered.

The primary driver here is to provide a more usable and relevant organization of documents or unstructured data, built from a vast set of original documents, and based on a specified selection criteria. The objective is to provide quick access to useful information instead of bombarding the user with too much information.

Search engines that crawl the World Wide Web/file systems implement this variation, as well as the more advanced "discovery" search engines that perform document clustering/categorization, expertise location (that is, identify experts), and intelligent analysis of the document contents.

This approach is best suited for selecting useful information from a huge collection of unstructured textual data. A variation of this can be used for working with other forms of unstructured data such as images, audio, and video files. In such cases additional transformation and translation services are required to parse and analyze the data.

In Figure 3-10 on page 81:

- Gather 1 crawls through multiple data stores, retrieving documents, parsing them, and building a result set of all documents that match the selection criteria. Alternatively, this initial step may parse the original data from multiple sources and build a single interim "index" that contains key pieces of document data and metadata.
– This initial step then allows additional steps to summarize, categorize, create taxonomies, or locate experts from this single normalized index. In some cases, such as World Wide Web search engines, the contents of documents in one data source (that is, URL links) may actually be used to determine additional data sources to crawl. This is shown by Gather 2 using the results of Gather 1 in combination with the same or other data sources to build a more complete set of input data.

– When the unstructured data recovered by these activities must be transformed, cleansed, or manipulated before it can be purposefully used, this is the responsibility of the Process and Apply steps. In more advanced search applications that perform document clustering and expertise identification, Multi Step Process described in 3.4.5, “Population: Multi Step Process variation pattern” on page 85, may also be invoked.

Further variations can be envisaged, for example, a two-step process where one step performs "search" types of activities, and the other actually populates the index from these searches. Such approaches are discussed in the redbook Patterns: Portal Search Custom Design, SG24-6881.

Population: Multi Step Gather variation runtime pattern

Figure 3-11 represents the Population: Multi Step Gather variation runtime pattern corresponding to the Population: Multi Step Gather variation application pattern shown in Figure 3-10 on page 81.
Figure 3-11 on page 84 shows the Runtime pattern for this Multi Step Gather variation, specifically aimed at gathering changes in a structured data source.

In this instance, the main Population pathway (Gather 2, Process, and Apply) handles both the initial load and the update of the target Data Server.

- For the initial load, Gather 2 obtains the data from the source directly.
- For update, however, a separate step is required (Gather 1), which reads the changes from the source and passes them to the main pathway.

Using the same main pathway for both initial load and updates is an important feature to help maintain data consistency and integrity.

Figure 3-11 on page 84 allows for the possibility that some of the nodes may be combined—Process and Apply nodes in this case.

For examples of runtimes for unstructured data, please refer to the redbook: *Patterns: Portal Search Custom Design*, SG24-6881.

### 3.4.5 Population: Multi Step Process variation pattern

The Application and Runtime patterns for the Population: Multi Step Process variation patterns are described here.

**Population: Multi Step Process variation application pattern**

Figure 3-12 on page 86 represents the Population: Multi Step Process variation application pattern.
Figure 3-12  Population: Multi Step Process variation application pattern

Like the Multi Step Gather variation described in 3.4.4, “Population: Multi Step Gather variation pattern” on page 80, the Multi Step Process variation is also an extension of the Multi Step Population variation described in 3.4.2, “Population pattern” on page 73. In this case, the focus is on supporting population instances where the processing of the received data is rather complex and cannot be performed in a single pass as shown in Figure 3-12.

Note: One may also envisage a Multi Step Apply variation; however, no population instances requiring it have been described at this stage.

In this Multi Step Process variation, the Process step is replaced by a more powerful Multi Step Process approach. Within this, the individual Process stages
are more likely to be linked directly, as shown by the line connecting them, rather than through intermediate temporary stores, although this possibility is also depicted. Clearly, there may also be more than two stages.

As mentioned earlier, this Multi Step Process approach may be required when building summaries or categorizations of unstructured data, often in conjunction with the Multi Step Gather variation. The Multi Step Process variation may also be required with structured data, for example, when populating a multidimensional cube or snowflake schema from an enterprise data warehouse.

Another use of this variation is in data cleansing implementations. Data cleansing often requires multiple passes of the data to gather statistics, perform analyses, propose changes, obtain human approval, and so on. In many cases, the cleansed data may be partially written back directly to the source, as shown in Figure 3-12 on page 86.

**Important:** This write-back process is not proposed as a general approach. It is provided solely to support data cleansing under closely controlled circumstances. See also the discussion under 3.6.4, “User Information Access: Write-back variation pattern” on page 106, and 3.5, “Data Integration:: Two-way Synchronization” on page 93 for more details of the considerations.

**Population: Multi Step Process variation runtime pattern**

Figure 3-13 on page 88 represents the Population: Multi Step Process variation runtime pattern corresponding to the Population: Multi Step Process variation application pattern shown in Figure 3-12 on page 86.
Figure 3-13 shows the Runtime pattern for this Multi Step Process variation for cleansing of structured data.

The main pathway in this Multi Step Process variation is through the Population nodes from the source on the right to the target on the left, cleansing data through a complex Processing stage on its way, for example, from an operational system to the data warehouse. However, an additional pathway is provided to apply the corrections to the data back to the source in order to avoid cleansing the same data again on subsequent runs. An alternative approach is to write the cleansed data only back to the source and to use a standard Population pattern to later update the target data store.

For examples of runtimes for unstructured data, please refer to the redbook *Patterns: Portal Search Custom Design*, SG24-6881.

### 3.4.6 Population: Multi Step Federated Gather variation pattern

The Application and Runtime patterns for the Population: Multi Step Federated Gather variation pattern are described here.
Population: Multi Step Federated Gather variation application pattern

Figure 3-14 represents the Population: Multi Step Federated Gather variation application pattern.

LEGEND:
Data sources are represented by disks in three different colors / shades:
- Blue / plain: Read/write
- Yellow / diagonal hatching: Read-only
- Green / vertical hatching: Temporary

Read/write and read-only refer only to the interaction between the overall pattern and that data source as also indicated in most cases by annotation on the linkages. In general we may assume that the application associated with a particular data source has read/write access.

A dotted box around an application and source data indicates that the source data may need to be accessed through the owning application via its API, or may be accessed directly via a database API. In general, a dotted box around a number of components indicates that we are not specifying which of those components we are interacting with.

A dashed line, arrow or component indicates an optional component.

Figure 3-14  Population: Multi Step Federated Gather variation application pattern

Figure 3-14 shows how the Population application pattern can be composed with the Federation application pattern as a means to gather data from one or more sources, by providing a unified query that accesses data in separated or remote structured and unstructured repositories in real-time.

Use of this variation pattern is indicated by a number of key requirements, such as reduced latency of population, reuse or extension of existing population investments, and reduced implementation or maintenance costs.

Figure 3-8 on page 77 shows how the Gather step in Population: Multi Step variation described in 3.4.3, “Population: Multi Step variation pattern” on page 76, is replaced by a potentially synchronous “Federated Gather” step that
directly accesses remote data stores, structured or unstructured. This access is mediated through wrappers (aka adapters) that contain the logic to access the data, either directly or through an application API, and send the results back to the requestor tier. These requests may simultaneously access multiple data stores.

Metadata mapping enables the decomposition of a unified query into requests to each individual data store. The multiple data sources thus appear as one unified virtual data store to the requestor. In some cases, there may be separate metadata stores for the Population and Federation components, although this may lead to data consistency issues.

The Multi Step Federated Gather variation application pattern contains its own temporary/persistent store. This store can be used to cache results data obtained from remote sources, allowing continued access to remote data when the actual source is unavailable. Clearly, use of such a cache may have implications for data currency in the target.

**Population: Multi Step Federated Gather variation runtime pattern**

Figure 3-15 represents the Population: Multi Step Federated Gather variation runtime pattern corresponding to the Population: Multi Step Federated Gather variation application pattern shown in Figure 3-14 on page 89.
The flow in Figure 3-15 on page 90 is that the Process tier makes a query for data from the "federated" data source such as a simple SQL SELECT request. The Data Integration node processes the request by using the metadata (which defines the data sources) to pass on the requests to the appropriate data sources. Usually, the Process node and the Data Integration node are collocated and tightly integrated.

In many cases, the data integration/federation logic within the Data Integration node is logically separate from the data connector logic. This data connector logic spreads out the overhead of making the query to multiple data sources, thereby allowing the queries to run in parallel against each database. When performance is of major concern, multiple logical data connectors may exist to process queries against a single data source—the idea here being to eliminate any single node in the process from becoming a bottleneck, if too many requests run against one data source.

In all cases, the results that are returned from each individual data source must then be aggregated and normalized by the Data Integration node so that these results appear to be from one "virtual" data source. The results are then sent back to the Population (Process) node, which has no idea that multiple data sources were involved.

### 3.4.7 Guidelines for usage and scenario

We strongly recommend that the logic that governs the transformation of source data into target data (including any transformation or cleansing) be implemented using rules-driven metadata rather than hard coding these rules. This approach enhances the maintainability of the application and thereby reduces the total cost of ownership.

- The Population application pattern is appropriate when simple filtering and transformation is required, as in the case of a dependent data mart created from an enterprise data warehouse. While not necessarily the case, often only a single data source is required in situations where the basic pattern applies.

- The Multi Step variation involves multiple data sources with complex extraction criteria, joins, transformations, reconciliations, restructuring and possible data cleansing requirements. This is probably the norm when creating an enterprise data warehouse or an independent data mart from data sources that happen to be existing stovepipe operational systems. Reconciling data from multiple sources is often a complex undertaking and requires a considerable amount of effort, time, and resources. This is especially true when different systems use different semantics.

- The Multi Step Gather variation is appropriate when an existing target data store such as a data warehouse, data mart, or ODS needs to be maintained...
with a latency that cannot be achieved by rebuilding it from scratch from the
data sources.

- The **Multi Step Process variation** is used when complex processing is
  required, as in the creation of multi-dimensional cubes, cleansing of data, and
  so on.

- The **Multi Step Gather and Process variations** are prevalent in the population
  of unstructured data stores, as the processing required may need to be widely
  distributed and the resulting data also distributed across multiple target data
  stores. For example, an indexing engine that produces a basic document
  index, summary, and taxonomy may create the index during an initial step,
  the summary in a secondary step, and then the taxonomy in a final third step.
  The taxonomy may be stored in a second target data store to improve
  performance when accessing the index or walking the taxonomy.

- The **Multi Step Federated Gather variation** is used to extend the capabilities
  of the Multi Step Gather and Process variations to transparently access
  remote structured and unstructured data with very low latency.

**Usage scenario**

Consider a Financial Services Company that provides various services, including
checking account, savings account, brokerage account, insurance, and so on.
The company has built this impressive portfolio of services primarily through
mergers and acquisitions. As a result, the company has inherited a number of
product-specific operational systems. The company would like to create a
Enterprise Data Warehouse (EDW) that provides a consolidated view of
customer information. It would like to use this consolidated information for
sophisticated pattern analysis and fraud detection purposes.

- Populating such an EDW would require reconciling customer records from
different operational systems that use different identification mechanisms to
identify the same customer. Further, other operational systems record
transactions with different time dependencies. The reconciliation process
must resolve these semantic and time differences and must check for any
inconsistencies and irregularities. Due to the complexity involved, the
Financial Services Company chooses the Population: Multi Step variation
pattern and the Population: Multi Step Gather variation pattern.

- The company also requires its EDW to have near real-time latency for
effective customer service. This is achieved through the use of the
Population: Multi Step Gather variation to update the EDW.

- Providing personalized data marts to internal and external users is also a high
priority to drive customer loyalty and cross-selling. Creating such specialized,
summarized, or subsetted derived data stores requires complex processing
and requires the use of the Population: Multi Step Process variation.
As another example, consider a large software company with a huge array of software products. The company develops vast amounts of technical documentation to support these products. Each product line publishes its own documentation on its own department Web site. As products change, so does the technical support documentation. Locating a particular piece of information in this sea of ever-changing data can be quite challenging and time consuming.

In order to improve efficiency of information access, the company wants to create a categorized and federated index of all documents that can then be searched or browsed by users as needed to find the required information. Such an index must be refreshed on a periodic basis to keep it current. To meet these requirements, the software company chooses to implement the Population: Multi Step Process variation.

### 3.5 Data Integration:: Two-way Synchronization

We describe the basic Two-way Synchronization application pattern and its variations. The key business and IT drivers for this pattern are described briefly along with guidelines for their usage.

This Two-way Synchronization application pattern was previously known as the Replication pattern. It enables a coordinated bidirectional update flow of data in a multi-copy database environment. It is important to highlight the “two-way” synchronization aspect of this Application pattern, as it is what distinguishes it from the "one-way" capabilities provided by the Population application patterns discussed in 3.4, “Data Integration:: Population” on page 72.

**Attention:** Two-way Synchronization is a special case of the more general multi-way synchronization need.

We focus on the two-way case here because it is of more interest in business intelligence and similar applications where the relationship between replicas is usually limited to pairs of replicas operating as true master/slaves or where the distributed read/write function is limited to a small percentage of the shared data.

**Note:** Patterns for multi-way synchronization with more complex relationships between the replicas are a future need.

This section is organized as follows:

- Business and IT drivers
- Two-way Synchronization pattern
- Two-way Synchronization: Multi Step variation pattern
3.5.1 Business and IT drives

As in the case of Population, any business need that requires a specialized copy of data—derived data—from a pre-existing source may indicate the need for the Two-way Synchronization application pattern. These needs are most often seen in business intelligence and content search and related applications. However, some cases are also seen in a pure operational environment, where a dedicated copy of data is needed. The key indicator for Synchronization is that the use of the derived data has some strong read-write characteristics.

The business and IT drivers for Two-way Synchronization are partially the same as those listed for Population in 3.4.1, “Business and IT drivers” on page 72. However, modern and more sophisticated business intelligence and combined operational/informational needs such as customer relationship management (CRM), call centers, customer portals, etc. place added requirements for updating the derived data. These modern business processes often require that the source and derived data are more closely synchronized than "pure" business intelligence applications, and thus need Two-way Synchronization.

As the need for synchronization increases, the differences between the source and derived data that can be handled decreases, because some transformations are fundamentally unidirectional, or are time-dependent. In the limit, the IT drivers for creating and managing a copy of the source have to be traded off against those for having a single copy of data and accessing that distributed data through the Federation application pattern.

3.5.2 Two-way Synchronization pattern

The Application and Runtime patterns for the Two-way Synchronization pattern are described here.

Two-way Synchronization application pattern

Figure 3-16 on page 95 represents the Two-way Synchronization application pattern.
Figure 3-16 Two-way Synchronization application pattern

Figure 3-16 shows a basic two-way synchronization of data between two separate data stores. At a simplistic level, it can be compared to the basic Population application pattern described in 3.4.2, “Population pattern” on page 73, with the only difference being that data now flows in both directions. Depending on the relationship between the data flowing in either direction, this similarity with Population may be more apparent than real. If the data elements flowing in both directions are fully independent, then Two-way Synchronization is no more than two separate instances of Population. However, it is more common to find some overlap between the data sets flowing in either direction. In this case, the need to reconcile data updates on both source/target systems means that the Two-way Synchronization pattern is rather more than two separate Population instances. A significant issue in this case is conflict detection and resolution when updates occur independently in the different data stores.
As indicated by the dotted boxes enclosing the source/target data stores and their controlling applications in Figure 3-16 on page 95, the Two-way Synchronization pattern may act directly at the data level or at the application level. However, from the viewpoint of Data Integration, the interactions are more likely to be at the data level, while in Process Integration the interactions are more often at the application level.

Applications in this solution design do not necessarily have to be identical.

**Two-way Synchronization runtime pattern**

As the Synchronization application patterns are evolving rapidly, typical Runtime patterns are in the process of being identified and will be documented on the Patterns Web site (http://www.ibm.com/developerWorks/patterns/) when they are finalized.

### 3.5.3 Two-way Synchronization: Multi Step variation pattern

The Application and Runtime patterns for Two-way Synchronization: Multi Step pattern are described here.

**Two-way Synchronization: Multi Step variation application pattern**

Figure 3-17 on page 97 represents the Two-way Synchronization: Multi Step variation application pattern.
Figure 3-17 shows how the Population application pattern can be composed to implement both directions of the synchronization data flow. An additional function “Reconcile” appears between the two data flows, and it is here that the complex process of ensuring that data updates do not conflict, cancel out, or get otherwise corrupted is handled. If the opportunities for conflict are minimal (when there are few overlaps between data flowing in either direction), this pattern can be effectively constructed from existing Population components. However, for more complex situations, a specialized product solution will be more appropriate.
Two-way Synchronization: Multi Step variation runtime pattern
As the Synchronization application patterns are evolving rapidly, typical Runtime patterns are in the process of being identified and will be documented on the Patterns Web site (http://www.ibm.com/developerWorks/patterns/) when they are finalized.

3.5.4 Guidelines for usage and scenario
The topology and simultaneous update requirements of the business application will dictate the complexity of the conflict detection and resolution needed in the general case of multi-way Synchronization. Detailed guidelines are still to be developed for this pattern, and will be documented on the Patterns Web site (http://www.ibm.com/developerWorks/patterns/) when they are finalized.

Two-way Synchronization, while simpler, will be treated in this more general context. However, the key considerations for limiting risk are:

► The derived data store should have a similar schema to its source.
► Data elements that are updated in the derived store should overlap to the smallest extent possible with those updated in the source.
► Synchronization should not be required at very short intervals.

Where these considerations are not met, direct access to the source data, using the Federation application pattern described in 3.3, “Data Integration:: Federation” on page 65, should be considered as an option.

Usage scenario
A Financial Institution creates a customer relationship management (CRM) system fed from a combination of sources, including operational systems and the data warehouse. In the CRM system, the agent can update certain elements of the customer record based on interactions with the customer. These include customer preferences and contact details. Some of these elements exist only in the CRM system. However, the contact details overlap with data stored in the operational Customer Information File. The Two-way Synchronization pattern is chosen to implement a method for keeping these contact details aligned between the two systems.

3.6 Information Aggregation:: User Information Access
We describe the basic Information Aggregation application pattern and its variations along with their corresponding Runtime patterns. The key business and IT drivers for this pattern are described briefly along with guidelines for their usage.
As mentioned earlier, the Information Aggregation business pattern is also known as User-to-Data, and it exists in e-business solutions that allow users to access and manipulate data that is aggregated from multiple sources. This Business pattern captures the process of taking large volumes of data, text, images, video, and so on, and using various user-controlled tools to extract useful information from them.

User Information Access (UIA, previously known as Information Access), is the only Application pattern in Information Aggregation. It helps structure a system design that provides mainly read-only access to aggregated information. It is most often used in conjunction with one of the Data Integration patterns discussed earlier, to provide users access to an aggregated repository created by these data movement-related Application patterns. For many data-oriented applications, UIA might also be called "query".

There are two variation patterns based on this Application pattern, as follows:

- **UIA: Federation**, which uses the Federation application pattern to extend the reach of UIA to additional sources
- **UIA: Write-back**, which uses the Population or Two-way Synchronization pattern to support managed update of data sources when the user writes to his copy rather than simply reading from it

This section is organized as follows:

- Business and IT drivers
- User Information Access pattern
- User Information Access: Federation variation pattern
- User Information Access: Write-back variation pattern
- Guidelines for usage and scenario

### 3.6.1 Business and IT drivers

The primary business driver for choosing the User Information Access application pattern and its variations is to provide efficient access to information that has been derived or aggregated from one or multiple sources. This mechanism can access both structured and unstructured data populated by any of the Data Integration patterns. Internal and/or external users may use this information for decision-making purposes.

For example, an Executive Information System (EIS) might generate a summary report on a periodic basis that compares the sales performance of various divisions of a company with the sales targets of those divisions. In this example, the UIA application pattern is used for accessing information from structured derived data. In addition, the application may provide drill-through capability
allowing the user to track the performance of individual sales representatives against their individual targets.

The IT drivers relate to the following:

- Mode of access—read-only versus read/write
- Direct access and manipulation of the data versus application-mediated access

While these distinctions are not hard and fast rules, Information Aggregation applications, in general, access data directly, allowing the users freedom to manipulate the data as they please. However, because of the dangers inherent in such an approach, this access is limited to read-only actions. This limitation may be through the UIA tooling itself, but more often is enforced by making available a copy (derived data) of the real source (operational system) and if the user makes changes to the copy, no harm is done. The IT drivers, therefore, are direct, read-only data access.

### 3.6.2 User Information Access pattern

The Application and Runtime patterns for the basic User Information Access pattern are described here.

**User Information Access application pattern**

Figure 3-18 on page 101 represents the User Information Access application pattern.
Figure 3-18 shows the basic User Information Access application pattern consisting of three logical tiers, as follows:

1. The Presentation tier is responsible for all the user interface related logic that includes data formatting and screen navigation. In some cases the presentation might be as simple as a printout.

2. The Query, Analyze and Search tier is responsible for accessing the associated data stores and providing the function to manipulate the information therein. Such function is primarily read-only and examples include simple query, data mining and other complex analysis functions, search and other means of investigating unstructured content.
3. The back end data sources contain the data to be accessed via the DBMS, content management system, or other direct data access methods.

As in other patterns, a temporary data store and metadata store serve similar purposes.

An additional "drill-through" capability may be provided in this Application pattern. Such a facility is needed when the data store has multiple levels. For example, a data mart may contain summarized multi-dimensional data that is regularly used, while the data warehouse contains the details. This function provides the user with the ability to drill-through to detailed data when required. This drill-through capability is implied in the Application pattern diagram as an inherent function of the Source data DBMS or access method.

As mentioned, the vast majority of access to data in the UIA pattern is read-only. However, this is really a convention, since UIA products and the data access methods they use are fully open to read/write access as well. As shown in Figure 3-18 on page 101, read/write access, when allowed, should be against data sources that are not owned or managed by applications. This reduces the risk to data integrity somewhat, but does not eliminate it entirely, depending on how the data source is maintained (using the Population pattern described in 3.4, "Data Integration:: Population" on page 72, for example). This is elaborated in the following sections.

**User Information Access runtime pattern**

User Information Access runtime patterns are more conveniently divided into those that access structured data and those that go after unstructured data.

Figure 3-19 on page 103 represents the basic Runtime pattern for structured data including a representation of drill-through corresponding to the User Information Access application pattern shown in Figure 3-18 on page 101.
In Figure 3-19, the Presentation/Web Application Server node handles the user interaction originating from a Browser. This component or a fat client operating via a product API (represented by the Win32 node in this Runtime pattern) passes requests to the Query and Analysis Server node, which then processes them and returns the results to the user on the Browser or the client. The underlying interaction between two Data Servers to support the drill-through function can be seen in this diagram.

Figure 3-20 on page 104 represents the basic Runtime pattern for unstructured data corresponding to the User Information Access application pattern shown in Figure 3-18 on page 101.
In Figure 3-20, we illustrate the case of the Search Server accessing data from two independent Data Servers.

For further examples of Runtime patterns for unstructured data, please refer to the redbook *Patterns: Portal Search Custom Design*, SG24-6881.

### 3.6.3 User Information Access: Federation variation pattern

The Application and Runtime patterns for the basic User Information Access: Federation variation pattern are described here.

**UIA: Federation variation application pattern**

Figure 3-21 on page 105 represents the User Information Access: Federation variation application pattern.
Figure 3-21 shows how the Federation application pattern can be composed with the User Information Access pattern, allowing access to additional data sources.

Use of this variation pattern is indicated when there exist multiple, diverse data sources that must be accessed in the same process. Federation provides both read-only and read/write access and also allows access either directly to the data store or via an application API. The value provided by Federation here is in hiding the diverse access methods and data structures behind the single access method provided by the UIA application pattern. Federation may also cache data as described earlier; this is omitted from the figure for simplicity.

Federation also adds the possibility of writing to its data sources, and clearly, the UIA: Federation variation can take advantage of this. While potential data integrity issues arise here as well as in the basic UIA pattern, one advantage that
Federation offers is the potential to do the write through the owning application API rather than directly to the source data.

**UIA: Federation variation runtime pattern**

Figure 3-22 represents the User Information Access: Federation variation runtime pattern corresponding to the User Information Access: Federation variation application pattern shown in Figure 3-21 on page 105.

The Runtime pattern for this variation pattern is a simple combination of the Runtimes for UIA and Federation.

**3.6.4 User Information Access: Write-back variation pattern**

The Application and Runtime patterns for the User Information Access: Write-back variation pattern are described here.

**UIA: Write-back variation application pattern**

Figure 3-23 on page 107 represents the User Information Access: Write-back variation application pattern.
Figure 3-23 provides a way to address some of the data consistency issues encountered when a user invokes the read/write functionality of the UIA pattern.

This variation pattern represents situations such as the following. The user uses a read-only source to perform analysis and creates an updated version of the data in the Temporary Store. This new data might be called a "what-if" or a "forecast". As part of the business process, some or all of the data in the Temporary Store needs to be reflected in a managed way back into the live environment. This is achieved via a Population (for a derived source) or
Synchronization (for an operational source) function between the Temporary Store and the original source. Although not shown in Figure 3-23 on page 107, the Population or Synchronization might actually occur in multiple stages, from the Temporary Store back to the Operational System first, followed by another stage from the Operational System to the Derived data.

**UIA: Write-back variation runtime pattern**

Figure 3-24 represents the User Information Access: Write-back variation runtime pattern corresponding to the User Information Access: Write-back variation application pattern shown in Figure 3-23 on page 107.

![Figure 3-24 User Information Access: Write-back variation runtime pattern](image)

The Runtime pattern for this variation pattern is a simple combination of the runtimes for UIA, Population and Synchronization.

### 3.6.5 Guidelines for usage and scenario

A clear separation of the presentation logic and the information access logic increases the maintainability of the application and decreases the total cost of ownership. This allows the same information to be accessed using various user interfaces.
The basic UIA application pattern provides maximum consistency in a multi-user analysis or reporting environment. This simple yet powerful Application pattern meets the majority of the information aggregation and distillation needs. The simplicity of this Application pattern reduces implementation risk.

The pattern implicitly supports drill-through approaches to local or homogeneous data stores. It can also be extended to remote or heterogeneous data stores, as shown by the UIA: Federation variation pattern.

In most cases, the data sources utilized in this Application pattern will be single consolidated data sources (such as a data mart) used in a read-only mode and taken from multiple original data sources. In such cases, any updates to the consolidated data would not be propagated back to the originating data/source applications. However, where it is required to propagate changes back to the sources, the UIA: Write-back variation pattern provides for the use of the Population or Synchronization patterns to do this.

User Information Access has been defined in principle to be a read-only pattern in order to allow users the freedom to access and manipulate the data in any way they please while preventing unmanaged changes to the underlying business data sources. As business needs for hybrid informational/operational applications increase, there is a growing need to relax this read-only restriction. All of the read/write methods described above—directly from UIA, through Federation, and in combination with Population or Synchronization—introduce the danger of corruption of the source systems to varying degrees. Additional work in this emerging area remains to be done in order to determine best practices.

Usage scenario
Consider a Personal Portal such as my.yahoo.com that aggregates information from disparate data sources and allows users to personalize this information to their unique preferences. These portals aggregate both structured data such as weather information and stock quotes, and unstructured data such as news and links to other sources of information. Based on the type of the data and the amount of transformation required, the portal developers may choose one or more of the Population application patterns. Once the data has been stored in the optimal format, the portal developers may offer the UIA application pattern to search this information, access this information in a personalized style, and/or to provide drill-through capabilities.

Another example is where an Insurance Company has created a data warehouse and dependent data mart to service claims analysis and reporting. The basic User Information Access pattern provides the ability to access and analyze the summarized, structured data in the mart as well as the possibility to drill-down to the detailed data in the warehouse. Combining UIA with the
Federation pattern, users can access images of claims documentation and include them in their reports. Where problems are discovered in the reports, claimants records in the operational systems can be flagged or even updated using the UIA: Write-back variation pattern.

### 3.7 Self Service:: Agent pattern overview

The Self Service business pattern is the means by which fully managed, transaction-consistent read/write function is provided to users of business systems. One important Application pattern in Self Service is the Agent, shown in Figure 3-25.

**Figure 3-25  Agent application pattern**

The Agent application pattern structures an application design to provide a unified customer-centric view that can be exploited for mass customization of services, and for cross-selling purposes. The unified customer-centric view across Lines of Businesses (LOB) in this case is either dynamically developed or supported by an Operational Data Store (ODS) that collects near real-time data about the user from multiple systems.
This pattern is defined on http://www.ibm.com/developerWorks/patterns/ as well as in a number of redbooks, as referenced in “IBM Redbooks” on page 299, and most recently in Self-service solutions with Process Choreographer, SG24-6322-00, and will not be described further here.

However, as previously mentioned, applications that combine analysis and update function are becoming more common. With this in mind, here we explore use of the Agent application pattern in combination with Federation as described in “Agent: Federation variation pattern” on page 111. This variation pattern forms an important component of our overall solution, in addition to the 3.4, “Data Integration:: Population” on page 72, and 3.6, “Information Aggregation:: User Information Access” on page 98, patterns described earlier.

This section is organized as follows:

- Business and IT drivers
- Agent: Federation variation pattern
- Guidelines for usage and scenario

### 3.7.1 Business and IT drivers

Business and IT drivers for the Self Service:: Agent pattern are described in the prior references. However, there are specific IT indicators for the use of the Agent: Federation variation pattern.

Like the Agent pattern itself, the Agent: Federation variation meets the business need for an application design that provides a unified customer-centric view that can be exploited for mass customization of services and for cross-selling purposes. This business need almost inevitably leads to a requirement to access and/or integrate disparate data from distributed sources. As the heterogeneity of these sources increases, it is worth considering the use of the Federation pattern to simplify this distributed access, rather than arbitrarily increasing the complexity of the Agent pattern to handle all possible data sources and combinations.

The specific IT drivers for the Agent: Federation variation pattern are therefore:

- Leveraging existing skills and investment
- Simplifying back-end integration
- Reducing coding complexity and maintenance costs

### 3.7.2 Agent: Federation variation pattern

The Application and Runtime patterns for the Agent: Federation variation pattern are described here.
Agent: Federation variation application pattern
Figure 3-26 represents the Agent: Federation variation application pattern.

```
LEGEND:
Data sources are represented by disks in three different colors / shades:
• Blue / plain: Read/write
• Yellow / diagonal hatching: Read-only
• Green / vertical hatching: Temporary

Read/write and read-only refer only to the interaction between the overall pattern and that data source as also indicated in most cases by annotation on the linkages. In general we may assume that the application associated with a particular data source has read/write access.

A dotted box around an application and source data indicates that the source data may need to be accessed through the owning application via its API, or may be accessed directly via a database API. In general, a dotted box around a number of components indicates that we are not specifying which of those components we are interacting with.

A dashed line, arrow or component indicates an optional component.
```

Figure 3-26  Agent: Federation variation application pattern

Figure 3-26 shows how the Agent function can invoke the Federation pattern to gain access to diverse and distributed data sources.

Use of the Federation pattern in support of the Agent tier is indicated when there exist multiple, diverse data sources that must be accessed from the Agent tier. Federation provides both read-only and read/write access and also allows access either directly to the data store or via an application API. The value provided by Federation here is in hiding the diverse access method and data structures behind the single access method provided by the Agent pattern. This can significantly simplify the programming of the Agent tier, as Federation takes care of the many diverse access methods. Federation may also cache data as described earlier; this is omitted from the figure for simplicity.

The standard Agent pattern emphasizes the role of an ODS as a supporting data store for the Agent. However, with the introduction of the Federation pattern, we
have the alternative possibility to now access the ODS through this path. This provides the same benefits as described for other data sources.

**Agent: Federation variation runtime pattern**

Figure 3-27 represents the Agent: Federation variation runtime pattern corresponding to the Agent: Federation variation application pattern shown in Figure 3-26 on page 112.

---

**3.7.3 Guidelines for usage and scenario**

When a portal needs to access data in disparate and distributed data sources, the Agent: Federation variation pattern provides a good approach. The Federation component can then take over much of the complexity of the data access, translating multiple source formats to/from a single interface in the portal, handling connectivity to the remote sources, and so on. This allows the portal, and indeed the portal programmer, to focus on the logic of the interactions between the different data sources rather than the details of the APIs needed. Using Federation in this way can significantly decrease the complexity of the portal coding, leading to faster coding and reduced maintenance.

In emerging hybrid informational/operational solutions, the Federation component can be shared with the User Information Access and even Population
patterns to create a fully integrated environment. This leads to increased reuse of components as well as the metadata required to define the integration environment.

**Usage scenario**
A bank wants to create a fully integrated CRM system, where call center agents and customers have access to the same, current set of base data to do both analytical, "what-if" tasks, as well as make transactions as a result of these analyses. The Agent: Federation pattern allows a portal to be developed where users can access and make transactions on a wide variety of operational systems as well as on the Operational Data Store. The same Federation pattern also plays behind the User Information Access pattern to allow users to analyze data from across the data marts, data warehouse, and some limited current data from the operational environment.
The Druid Financial Corporation (DFC) Customer Insight solution

In this chapter, we describe the design and implementation of a Customer Insight solution at the Druid Financial Corporation using the Patterns for e-business approach. The emphasis is on Business patterns, Application patterns, Runtime patterns, and Product Mapping relating to Information Aggregation and Data Integration.

The topics covered are:

- Solution definition process.
- Develop a high-level business description.
- Develop the Solution Overview Diagram.
- Select the Business patterns.
- Select the Integration patterns.
- Select the Composite patterns.
- Select the Application patterns.
- Select the Runtime patterns.
- Select the Product mappings.
- Review guidelines.
- Typical CSR scenario.
4.1 Solution definition process

The following steps are involved in arriving at the final solution to the business requirements:

- Develop a high-level business description.
- Develop a Solution Overview Diagram (SOD).
- Select the Business patterns.
- Select the Integration patterns.
- Select the Composite pattern.
- Select the Application patterns.
- Select the Runtime patterns.
- Select the Product Mappings.
- Review Guidelines.

These steps are covered in the following sections for the Druid Financial Corporation (DFC) Customer Insight solution.

**Attention:** Our objective in this chapter is to expose the reader to the process involved in reusing assets per the Patterns for e-business approach, rather than conduct a rigorous discussion of the pros and cons of the various pattern alternatives available at each step that led to the final solution for DFC's Customer Insight business requirements.

In a real-world environment, a more detailed description of the business requirements, and business and IT drivers needs to be defined before a proper evaluation of the available alternatives can be performed to arrive at the solution that most closely addresses the business needs of the organization.

This chapter references some Application patterns unrelated to Information Aggregation and Data Integration. For further details on such patterns, please refer to the Patterns Web site (http://www.ibm.com/developerWorks/patterns/).

4.2 Develop a high-level business description

In the first step of the solution definition process, the business owner should develop a high-level business description that illustrates the core business functions of the proposed solution. It should describe the actors who participate in the solution, and the high-level interactions between these actors that explain the core business functions.
Actors represent entities that exist outside the scope of the solution, but are critical for its completeness. For example, actors can be people, devices, external institutions, legacy applications that will not be modified, and packaged applications with which this solution will interact.

**Note:** In the following high-level business description of the Druid Financial Corporation (DFC) (a fictional company), underlined items identify actors in the solution, while items in **bold** represent the high-level business functions that need to be provided by the solution.

The Druid Financial Corporation is a financial services company offering products ranging from checkings and savings, loans, brokerages, and credit cards. These products to date have been provided to their customers through stovepipe systems that are isolated from each other, and therefore did not provide DFC with a holistic view of customers receiving multiple product services from the individual stovepipe systems.

DFC recognized the need to develop a Customer Insight business solution that provided a holistic view of their customers across each product category in order to identify:

- **Most profitable customers in order that they may be provided superior customer service, and offered incentives and rewards for their continued patronage.**
- **High potential customers that could be targeted to receive additional products with suitable marketing campaigns**
- **Low or unprofitable customers that should either be targeted for attrition or an aggressive campaign to make them more profitable**

Such a solution was essential to gain a competitive edge, increase operational efficiency through lowering the total cost of ownership (TCO), and increase profitability by identifying and exploiting emerging business opportunities in a timely manner.

---

1 Customer Insight helps develop strategies to promote customer loyalty, which in the generic sense describes the tendency of a customer to choose services from a particular vendor or a particular brand of product(s) when the need arises. In our context, it involves building an infrastructure and implementing business processes that enable an organization to support activities such as superior customer service and targeted marketing campaigns to promote customer loyalty within their clientele. One necessary ingredient of a Customer Insight scenario is the need for a holistic view of all customer interactions with the organization, sometimes also called customer360. From a customer's perspective, loyalty may be engendered with consistent and superior service over all channels such as the Web, telephone, and face-to-face, as well as personalized information and attention. Customer Insight programs apply to many different industries such as banking, financial services, retail, telco, and insurance.
DFC concluded that these objectives could be met with a solution that involved the following:

1. **Data warehouses** and **data marts** that collect historical information about customers’ activities, and industry trends and events. This data can then be **analyzed and mined** by **knowledge users** to gather personalized and targeted information in real time if necessary, in order to make more effective strategic as well as tactical business decisions.

2. A **customer service representative (CSR)** browser-based portal that provides current and holistic customer information from operational systems; a **customer information file; data warehouses** and **data marts**; and a complete history of all customer contacts regardless of channel or medium including face-to-face contact, letters, e-mail, voice mail, telephone conversations and fax. This would not only help provide superior customer service and enhance operational efficiency, but also facilitate cross sell and up sell of products to prospects.

The data warehouse contains transaction history as of the end of business the previous day, and is populated from the **checkings and savings**, **brokerage**, **loans**, **credit card**, and **rewards** operational systems. The data mart is populated from the data warehouse on an on demand basis, as and when marketing campaigns are planned and potential target demographics need to be identified. Data marts could also be generated on an on demand basis for data mining against specific target groups. The **customer information file** maintains cross operational systems’ relationship information as well as a holistic profile for delivering appropriate levels of service. The holistic view of a customer would be obtained from the customer information file, operational systems (provides the current state and recent detailed transaction information), and the data warehouses (provides historical information regarding the accounts). This information would be maintained to have near real-time currency for events such as loss of credit cards or account closures.

The CSR would be able to **view account** and any personal relationship details and transaction history, as well as initiate transactions on individual accounts such as **stock purchases and sales, account transfers** and **contact information updates**. In addition, the CSR would have **search** capabilities against the database to identify a caller over the telephone based on name and city, for example; access to internal ratings about the customer such as Platinum

---

2 Traditional retail banking products/accounts that also include the capability of having a Debit card.
3 Ability to purchase and sell stocks, bonds, and mutual funds with the support of a margin account.
4 Traditional retail products such as automobile, mortgage, and personal loans.
5 Bank-owned credit card products.
6 Incentives or points program associated with the use of specific high-profit products such as the credit cards.
7 It was decided to defer to a subsequent phase the ability of a CSR to initiate transactions such as stock purchases and sales, and account transfers against backend stovepipe systems.
or Gold customer; and also have the ability to **record** details of the customer interaction in the system. The holistic view of a customer would be obtained from the operational systems, a customer information file, and data warehouses.

**Important:** In developing this solution, DFC considered it necessary to leverage the organization’s existing infrastructure as well as any existing third-party solutions as far as possible. The use of third-party solutions would facilitate speed-to-market, ease of maintenance, and rapid access to ongoing product enhancements. Another important consideration was to minimize disruption and impact on existing applications and systems. Another continuing requirement was to host all machines in a single data center, with a backup remote data center for disaster recovery purposes.

Figure 4-1 partially describes DFC’s operational environment with its stovepipe applications of checkings and savings, loans, brokerage, credit card, and rewards. The Web Server and WebSphere Application Server V5 are part of DFC’s infrastructure that will be leveraged in the development of the Customer Insight solution.

![Diagram of DFC's operational environment](image)

**Figure 4-1  DFC's operational environment**
In the following sections, we illustrate how the Patterns for e-business are used to reduce risk and improve the time to market of the Customer Insight solution at DFC.

Attention: The DFC Customer Insight business requirement defined here is only an approximation of a “real world” Customer Insight solution, and therefore does not reflect elements such as e-mail collaboration interactions and possible business-to-business interactions with external systems such as news feeds or market events affecting stock purchases and sales.

4.3 Develop a Solution Overview Diagram

The Solution Overview Diagram (SOD) translates the business requirements of the Customer Insight business solution into a pictorial representation containing business functions and actors. The objective is to keep the SOD simple and informative. The SOD shown in Figure 4-2 provides a concise and comprehensive way of representing the key aspects of the proposed solution where the connectors represent process flows. The connectors in Figure 4-3 on page 121 show the data flows. It provides the foundation for the process of identifying and applying the Patterns for e-business.

Figure 4-2 Customer Insight Solution Overview Diagram - Process flows
Figure 4-3  Customer Insight Solution Overview Diagram - Data flows

Note: In the subsequent SOD figures, we will only represent the data flows version.

The actors are:
- Customer Service Representative
- Knowledge users
- Checkings and Savings application
- Loans application
- Brokerage application
- Credit Card application
- Rewards application
- Browser
- Internet

The high-level business functions are:
- Customer Information File (CIF).
- Data warehouse.
- Data mart.
- Search on last name, city and zipcode.
- View customer metadata and transaction history.
- Stock purchase/sale.
- Transfer assets between accounts.
- Update customer metadata and relationships.
- Record customer interaction.
- Analyze and mine.
- Update CIF.

The connectors link the individual symbols based on the business processes involved.

Developing the SOD is usually an iterative process.

**Important:** In Figure 4-2, the Analyze and Mine function is typically performed by knowledge users through local connect or over the Internet, and not by CSRs.

### 4.4 Select the Business patterns

The SOD in Figure 4-2 on page 120 identifies the following Business patterns:

- CSRs search and review existing account details in the CIF, data warehouse, and operational systems; and initiate transactions against the various operational systems. These functions involve direct interaction between the CSRs and the various data sources and applications. These interactions indicate a Self-Service pattern, as described on the Patterns Web site:
  

- Once the data warehouse and data marts have been populated, knowledge users can analyze and mine them to extract meaningful information from them that can then be used to update the CIF. An example of such information is the determination that certain customers qualify for superior service by virtue of their transactional history and asset holdings. These interactions indicate an Information Aggregation business pattern.

Figure 4-4 on page 123 shows the Business patterns occurring in the Customer Insight solution; the grayed out patterns are not reflected in the solution.
Figure 4-4  Business pattern diagram

Figure 4-5 reflects the Business patterns in the SOD, where the hashed shaded box identifies the Information Aggregation pattern, while the simple shaded box depicts the Self-Service pattern.

Figure 4-5  SOD with Business patterns
4.5 Select the Integration patterns

When multiple Business patterns are needed to solve a business problem, one or more Integration patterns must also be included so that the solution can be simplified or made seamless to the user or application requiring the solution.

The connectors between the functions in the SOD in Figure 4-5 on page 123 should be examined to determine how the integration will be accomplished.

The Application Integration pattern accomplishes the integration between the various functions, while the Access Integration pattern provides a consistent user experience to the CSR and knowledge user.

4.5.1 Application Integration patterns

The SOD in Figure 4-5 on page 123 shows the need for Application Integration between the Self Service and Information Aggregation business patterns, as follows:

- View customer data and transaction history function and the data warehouse function.
- Update CIF function and the CIF function.
- Populate the data warehouse function from the operational systems.

These and other Application patterns are discussed in more detail in 4.7, “Select the Application patterns” on page 126.

4.5.2 Access Integration pattern

This is required to implement the following:

- CSR portal functionality of the DFC Customer Insight solution. It includes accessing the CIF and the data warehouse using native APIs, and the Application Integration:: Federation application pattern, as shown in Figure 4-14 on page 133, for drill-down to the operational systems using federation.
- Knowledge workers access the data warehouse and data mart, which may involve using a portal similar to that of the CSR.

4.5.3 IT drivers

The main IT driver for these patterns is promoting consistency of operational data.
The final solution can be composed by adding these Integration patterns to the Business patterns identified, earlier as shown in Figure 4-6 on page 125.

Figure 4-6  Final solution with Business and Integration patterns

The SOD can be extended by drawing ellipses to show the Access and Application Integration patterns, as shown in Figure 4-7. Note that it does not include the Application Integration patterns within the constructs of the Self-Service and Information Aggregation business patterns.
4.6 Select the Composite patterns

There does not appear to be any matching Composite pattern for the specific requirements of the DFC Customer Insight solution. Therefore, further refinement is recommended using the subsequent steps before a final product decision is made.

4.7 Select the Application patterns

As identified earlier, the two Business patterns in the DFC Customer Insight solution are Self Service and Information Aggregation. The Application patterns in each of these Business patterns, and for Access Integration and Application Integration, are discussed in the following subsections.

4.7.1 Select Application pattern for Self Service

The CSR portal of the DFC Customer Insight solution provides a unified view of a customer’s accounts in operational systems, CIF, and the data warehouse.
Based on the DFC Customer Insight solution requirements, the IT drivers of the various Application patterns for Self Service, the Application pattern required to automate this Business pattern is the Agent: Federation variation application pattern, as shown in Figure 4-8. The Agent IT drivers include minimizing the total cost of ownership (TCO), leveraging legacy investment, providing back-end application integration, minimizing the enterprise’s complexity, and providing scalability that are critical to the DFC Customer Insight solution.

**Figure 4-8  Self Service:: Agent: Federation variation application pattern**

In Figure 4-8 on page 127:

- The presentation tiers can support different presentation styles including the Internet, call centers, kiosks, and voice recognition units. For the CSR only a Browser interface is appropriate.

- The agent tier supports a consolidated view of the customer’s relationship with the organization, and uses a CIF implemented via an ODS to provide this capability. The CIF provides a current or near-real-time integrated view of all the services subscribed by a customer, aggregated from multiple operational systems. In addition, it stores additional demographics information about customers that has been collected from various sources.

- The Federation tier supports the Agent tier since it needs to drill down to multiple, diverse back-end data sources.

The links between these tiers may be either synchronous or asynchronous.

Since the CSR has to respond to the customer while (s)he is on the phone, a fast connection between the various tiers is required in order to respond promptly to customer requests, as well as cross-sell different products to the customer.

At this time, there are few off-the-shelf middleware products that can provide the end-to-end functionality required by this Application pattern. Therefore, we need to pursue a custom development solution of the components identified by the
Agent: Federation variation application pattern, which involves building the ODS. Note that the data warehouse and data marts are built in the Information Aggregation business pattern.

4.7.2 Select Application patterns for Information Aggregation

The query, analysis, and mining of information in the data warehouse and the data mart(s) represent the Information Aggregation business pattern.

The data warehouse is populated from the various operational systems on an end-of-business-day frequency. It then serves to populate the data mart, and both the data warehouse and data mart are accessed in read-only mode for analysis and mining by knowledge users.

Therefore, the best-fit Application pattern for Information Aggregation is the User Information Access pattern for knowledge users as shown in Figure 4-9 on page 128.

![Figure 4-9](image)

*Figure 4-9  Information Aggregation:: User Information Access application pattern*

The Query, Analyze, and Search tier analyzes the query/request and, based on its metadata definitions, passes parts of the request on to the appropriate data servers (data warehouse and data mart).
4.7.3 Select Application pattern for Access Integration

The DFC Customer Insight solution supports Browser access for the CSR via an intranet. Therefore, the Single Sign-on and Role-Based Access application pattern, as shown in Figure 4-10, is the best fit for our DFC Customer Insight solution.

![Access Integration Single Sign-on and Role-Based application pattern](image)

In Figure 4-10 on page 129:
- The client tier represents the user interface client such as a Browser, mobile phone, or PDA.
- The single sign-on and role-based access tier implements the Security and Administration service, which provides a seamless sign-on and access capability across multiple applications. This tier uses a user-profile data store, which is mostly read only, but may be read-write to keep track of such things as the last sign-on and the number of invalid sign-on attempts.
- The application tier may represent new applications, modified existing applications, or unmodified existing applications.

4.7.4 Select Application patterns for Application Integration

Based on the DFC Customer Insight solution requirements, and the IT drivers of the various Application patterns for Self Service and Information Aggregation, the following Application patterns are appropriate for the DFC Customer Insight solution:
- Agent: Federation variation application pattern, as shown in Figure 4-8 on page 127, for the unified access by the CSR to the data warehouse, CIF, and operational systems (through drill-down).
- Population: Multi Step variation application pattern, as shown in Figure 4-11 on page 130, for the initial load of the data warehouse and CIF from
information in the operational systems. The Federation pattern, as shown in Figure 4-14 on page 133, is required for accessing non-DB2 UDB relational and non-relational data sources in the operational systems.

Figure 4-11  Population: Multi Step variation application pattern

- Population: Multi Step Gather variation application pattern as shown in Figure 4-12 on page 131, for the incremental update of the data warehouse (end-of-business-day frequency) and CIF (near real-time for certain events such as loss of a credit card) from information in the operational systems or other external sources in the future. The Federation pattern, as shown in Figure 4-14 on page 133, is required for accessing non-DB2 UDB relational and non-relational data sources in the operational systems.
The independent Gather step (Gather 1) extracts a specialized subset (such as changes only) of the data and stores it in a temporary or persistent store, for subsequent processing by the process and apply functions to accomplish the incremental update function.

**Note:** The apply function of the basic Population and Multi Step variation patterns include the semantics of inserting new records and updating existing records, which is required for the incremental update functionality.

Other optional steps could involve data cleansing and extraction from multiple data sources using a federated server, as discussed in 3.4.5, “Population: Multi Step Process variation pattern” on page 85.

- Basic Population application pattern, as shown in Figure 4-13 on page 132, for populating the data mart from the data warehouse on an on demand basis, as well as updating the CIF based on analysis and mining of the data warehouse and/or data mart. An example of such an update may be the changing of the status of a customer from Gold to Platinum based on transaction history and asset holdings.
User Information Access application pattern for knowledge users, as shown in Figure 4-9 on page 128, is appropriate for accessing both the data warehouse and the data mart for analysis and mining.

Federation application pattern (described in “Federation application pattern” on page 66), as shown in Figure 4-14 on page 133, for accessing relational and non-relational data sources for the initial load and incremental update of the data warehouse.
4.7.5 Summarize Application patterns in DFC solution

Figure 4-15 on page 134 summarizes the various best-fit Application patterns discussed in 4.7.1, “Select Application pattern for Self Service” on page 126; 4.7.2, “Select Application patterns for Information Aggregation” on page 128; 4.7.3, “Select Application pattern for Access Integration” on page 129; “Population: Multi Step Gather variation application pattern” on page 80; and 4.7.4, “Select Application patterns for Application Integration” on page 129.
Figure 4-15  DFC Customer Insight SOD - Summary of Application patterns

Figure 4-16 on page 135 consolidates all the selected Application patterns in the DFC Customer Insight solution.
The dotted arrows in Figure 4-16 indicate data movement, while the regular arrows indicate data access.

### 4.8 Select the Runtime patterns

A Runtime pattern is used to define the logical middleware structure required to support a particular Application pattern. Runtime patterns define functional (logical) nodes that underpin an Application pattern. Like other types of patterns, multiple Runtime patterns might be appropriate for a single Application pattern and the best fit alternative must be selected by evaluating specific operational requirements such as security, scalability, availability, and ease of maintenance.

The following Application patterns were selected for the DFC Customer Insight solution:

- Agent: Federation variation application pattern
- User Information Access (UIA) application pattern
- Federation application pattern
- Population application pattern
- Population: Multi Step variation application pattern
4.8.1 Agent: Federation variation application pattern

The Runtime pattern in Figure 4-17 can be used in our DFC Customer Insight solution, which involves CSR access to a unified view of customer information via an intranet. For our environment, however, we only implemented local access to the Web Application Server.

4.8.2 User Information Access (UIA) application pattern

This pattern relates to knowledge users' (not the CSR) interactive access to the data warehouse and data mart to perform analysis and mining activity for tactical and strategic decision making. The consequence of these analyses and mining activities may include the update of the CIF database to reflect customer status changes such as an upgrade from Gold to Platinum status due to a review of recent transaction history.
Note: In reality, the analysis and mining of the data warehouse and data mart will also occur in batch mode as well.

The Runtime pattern shown in Figure 4-18 is the best-fit for knowledge user access to information stored in the data warehouse and data mart.

4.8.3 Federation application pattern

The Runtime pattern for the Federation application pattern shown in Figure 4-19 on page 138 is the best-fit for the DFC Customer Insight solution.
4.8.4 Population application pattern

The DFC Customer Insight solution requires the creation of a data mart from the data warehouse on an on demand basis, as well as the update of the CIF from the results of the analyses and mining of the data warehouse and/or data mart.

The best-fit Runtime pattern for this activity is shown in Figure 4-20 on page 139.
4.8.5 Population: Multi Step variation application pattern

The DFC Customer Insight solution requires a multi-step initial load from multiple operational systems using federation.

The Runtime pattern for the initial load of the data warehouse and the CIF is shown in Figure 4-21 on page 140.
Figure 4-21  Population: Multi Step variation runtime pattern

Figure 4-21 is represents the initial load of the CIF and some of the tables in the data warehouse where a transform and merge is required. It also represents the independent load of data warehouse tables, such as the transaction tables where no merge is necessary.

Note: To avoid clutter, we have deliberately not included the federation function and Data Integration node in Figure 4-21.

4.8.6 Population: Multi Step Gather variation application pattern

The DFC Customer Insight solution requires a multi-step incremental update of the data warehouse and CIF using changes (accessed using federation) occurring in the operational systems.

The Runtime pattern for the incremental update of the data warehouse and the CIF is shown in Figure 4-22 on page 141.
Figure 4-22  Population: Multi Step Gather variation runtime pattern

Figure 4-22 represents the incremental update of the CIF and data warehouse with changes originating in the operational systems.

**Note:** To avoid clutter, we have deliberately *not* included the federation function and Data Integration node in Figure 4-22.

### 4.8.7 Integrated Runtime environment

Figure 4-23 on page 142 reflects the consolidated view of the DFC Customer Insight solution's selected Runtime patterns.
Figure 4-23: DFC Customer Insight - Consolidated Runtime patterns

Figure 4-23 highlights the interactions of the various runtime patterns. By combining this high-level consolidation of runtime patterns with the constraints of business drivers, security, availability, and scalability requirements, and the existing IT infrastructure, the final DFC Customer Insight system environment shown in Figure 4-24 on page 144 can be derived.

4.9 Select the Product Mappings

After choosing the Runtime patterns, one needs to map the products to implement them. A Product Mapping maps the logical nodes defined in the Runtime patterns to specific products that implement the Runtime solution design on a selected platform. The Product Mapping identifies the platform, software product name, and often version numbers as well.
The following factors need to be considered in determining a product and technology mix.

- Business drivers
- Existing systems and platform investments
- Customer and developer skills available
- Customer choice
- Future functional enhancement direction

In the DFC Customer Insight business solution described in 4.2, “Develop a high-level business description” on page 116, gaining a competitive edge, lowering TCO, and identifying emerging business opportunities were identified as being critical business drivers for the business solution. Additionally, these benefits had to be delivered speedily to the market with minimal disruption to the existing IT infrastructure, and in-line with strategic vendor relationships, which happen to be IBM products and services.

Figure 4-1 on page 119 describes the existing applications and IT infrastructure of DFC, showing a heterogeneous mix of machines (names in parentheses in Figure 4-1 on page 119) and platforms (AIX®, Windows, and z/OS) involving stovepipe applications and using Web Application Server WebSphere Application Server V5.

The single data center requirement also meant that there was no wide area network connectivity required between these machines, and that any additional machines would need to only require local connectivity.

Based on these aforementioned considerations, we settled on the systems environment shown in Figure 4-24 on page 144 for the initial phase of the DFC Customer Insight solution.

The design assumption is that the CIF would maintain the global view of a customer and the cross relationships between the various accounts; the data warehouse would maintain historical transaction information with a latency of one day; and the operational systems have up-to-the-minute transactions. The CSR portal will access these different data sources depending upon the information desired. “Typical CSR scenario” on page 147 describes a typical CSR customer interaction scenario.
Figure 4-24 highlights DFC’s strategic vendor partnership with IBM (products, services, and business partner relationships) in the selection of Client Information Integration Solution (CIIS), DB2, and Ascential’s DataStage as follows:

- The IBM CIIS services offering that provides the customer information file that maps the relationships between accounts in the various operational systems such as loans, checkings/savings, credit card, brokerage, and rewards.

This product was chosen because of speed-to-market considerations, current functionality, and plans for future enhancements, customer references, as well as scalability considerations. CIIS uses DB2 as its metadata store.

**Note:** In real-world situations where critical operational systems are typically deployed on z/OS systems, a z/OS platform for CIIS would most likely be appropriate for its scalability characteristics. In our pseudo-production environment, we opted for a Windows platform as a matter of expediency.
Change capture is provided by a combination of the replication features of DB2 UDB and Oracle, and an application-supported audit trail in the case of VSAM.

The data warehouse is a custom design using DB2 UDB ESE on an AIX platform, given AIX’s superior price/performance, availability, and scalability characteristics over a Windows environment.

The data mart(s) is also a custom design that is appropriate for DB2 UDB ESE on Windows platforms, since its availability and scalability characteristics are less stringent and the Windows platform has a lower TCO.

The heterogeneous mix of operational systems need to be accessed by the portlets in the CSR portal for up-to-the-minute transaction information, as well as by Ascential DataStage for both the initial load and incremental update of the data warehouse and CIIS.

- Portlets in CSR portal
  The portlets in the CSR portal access the CIIS using its native API, and the data warehouse using DB2 SQL. With federation, the portlets can use the same DB2 SQL API to access all the operational systems without having to acquire the native API skills of the individual relational and non-relational data sources.

Since the existing Web Application Server resides on an AIX platform, it is appropriate to collocate DB2 Information Integrator on the same platform, assuming adequate capacity. An additional consideration is the ease of management of the DB2 Information Integrator platform when collocated.

In our contrived environment, we also had our Rewards Lookup operational system XML file on the same AIX machine (JAMESBAY), as shown in Figure 4-24 on page 144. Since such flat files need to be network accessible by the federated server, it is another reason to collocate DB2 Information Integrator on the same AIX machine.

Note: We artificially had the XML Rewards Lookup file on an AIX machine when the main Rewards operational system application was deployed on the z/OS platform in order to showcase DB2 Information Integrator's non-relational data source support capability.

By creating nicknames of the data warehouse tables in the federated server in addition to the operational system objects, it would be possible for the portlet developer to join tables across the operational system tables and data warehouse tables transparently as if they were all part of the same (federated) database.
Ascential DataStage

The Ascential DataStage tool provides rich functionality for extraction from a number of relational and non-relational data sources; powerful transform capabilities including joins and aggregations; and options to load, update, as well as append target databases. Ascential is an IBM business partner and the DataStage product supports a number of platforms including AIX, Windows and Linux.

Ascential DataStage has native support for a number of relational and non-relational data sources including all those used in DFC’s various operational systems. Therefore, it would appear to make sense not to include federation functionality to access DFC’s operational systems. However, we still chose to implement Ascential DataStage access to these operational systems through federation for the following reasons:

- As a performance consideration by taking advantage of DB2 Information Integrator’s superior optimization strategies when joining data from multiple operational systems. While such joins were not planned for the initial phase of DFC’s Customer Insight deployment, subsequent phased deployments in immediate succession are expected to demand such capabilities.
- Simplifying configuration of population jobs when all the data originates from the same (federated) database, rather than different heterogeneous data sources.
- Potential cost savings of deploying DB2 Information Integrator with specific data sources such as the z/OS platform.

We chose to isolate Ascential DataStage on a separate Windows machine and serviced the population of the CIIS, data warehouse and data mart from this machine. We also collocated DB2 Information Integrator on the same machine. Such an environment provides better manageability of the population environment and a lower TCO.
4.10 Review guidelines

The Application patterns, Runtime patterns, and Product Mappings are intended to guide you in defining the application requirements and the network layout. However, the next step involves designing, developing, and implementing the DFC Customer Insight application.

The Patterns Web site (http://www.ibm.com/developerWorks/patterns/) provides guidelines for each Application pattern, including techniques for developing, implementing, and managing the application based on design guidelines, development guidelines, systems management guidelines, and performance guidelines, as described in 1.3.5, “Review guidelines and related links” on page 14.

Note: The Patterns Web site (http://www.ibm.com/developerWorks/patterns/) is a work-in-progress, and will be updated as new patterns are identified and existing patterns evolve.

This Web site may sometimes lag behind patterns discussed in various IBM presentations, articles, and redbooks, but every effort is being made to minimize this window.

Attention: Since the focus of this book is on Information Aggregation and Data Integration patterns, we will not be discussing these review guidelines here. For details on this topic, please refer to:

4.11 Typical CSR scenario

In this section, we describe a typical CSR scenario involving a customer’s inquiry disputing recent transactions on one of his accounts.

The general processing involved is expected to be as follows:

1. The CSR logs in to an intranet portal, as shown in Figure 4-25 on page 148, that then presents the Welcome screen shown in Figure 4-26 on page 149.
Figure 4-25  DFC CSR portal login screen
2. When the CSR selects the Customer Overview page, it displays the panel shown in Figure 4-27 on page 150 with a CustSearch portlet window that enables the CSR to determine the unique global identifier of a customer calling in for service. The search criteria supported are last name, city, and zipcode; and the portlet accesses the CIIS database to retrieve all customers meeting this criteria.

Example 4-1 shows the string nature of the input in the CIIS API and the invocation of a DB2 stored procedure RELSPROC (provided by CIIS) with this input.

Example 4-1  Snippet of CIIS search code

```
final String CIISStoreProcedure = "\{call RELSPROC(?');\}"
// String strSearchPerson
String strSearchPerson
="0000001000aaaaaaasessionid00000000000000000000000300030000001000SS000100000"
```
// Create a Store procedure Statement object so we can execute
CallableStatement cs2 = con2.prepareCall(CIISStoreProcedure);
cs2.setString(1, strSearchPerson);
cs2.registerOutParameter(1,Types.VARCHAR);

// Submit a query, creating a ResultSet object
ResultSet rs=cs2.executeQuery();
String answer = cs2.getString(1);

---

Figure 4-27 DFC CSR portal Customer Overview screen

- The global identifier (not shown here) is then used to obtain a holistic view of the customer, as shown in Figure 4-28 on page 151. This screen provides a summary overview of all the accounts held by this customer, a recent history of interactions with this customer, and details of any...
targeted marketing campaign for this specific customer. This consolidated information is retrieved from the CIIS.

3. Depending upon the customer’s inquiry, appropriate details may be retrieved from the relevant application, as follows:
   
   – For information about transactions as of the end of the previous day, information is retrieved from the data warehouse, which has a latency of one day. Access to the data warehouse is direct via the DB2 SQL API as SQL with no federation involved. A snippet of the DB2 SQL is shown in Example 4-2.

   **Example 4-2  Snippet of data warehouse access code**

   ..................
   String sb3 = new String();
   sb3="select type, transaction_ts, amount, vendor from patrnsdw.crdt_trans_his where card_holder_id = 12003
$\text{order by transaction\_ts desc}$

Vector cpv = new Vector();
CustomerProfile cp;
try {
    Connection conn = null;
    String dbURL = "jdbc:db2:patrnda";
    conn = DriverManager.getConnection(dbURL,"db2inst1","db2inst1");
    Statement st = conn.createStatement();
    ResultSet rs = st.executeQuery(sb3.toString());
    while (rs.next()) {
        cp = new CustomerProfile();
        cp.setType(""+rs.getObject(1));
        cp.setTxnDate(""+rs.getObject(2));
        cp.setAmount(""+rs.getObject(3));
        cp.setVendor(""+rs.getObject(4));
        cpv.add(cp);
    }
    conn.close();
}

---

– For information about up-to-the-minute transactions, information is retrieved directly from the operational systems. Access to the operational systems is via a federated server using the DB2 SQL API as SQL. A snippet of the DB2 SQL is shown in Example 4-3, where CC\_TRANSACTIONS is a nickname on the federated server referencing the TRANSACTIONS table on DB2 for z/OS of the Credit Card operational system.

Example 4-3  Snippet of operational systems’ access code

.........

sb1="select type, transaction\_ts, amount\_charged, vendor from cc\_transactions
where cardholder\_id = 12003 and date(transaction\_ts) = current date
order by transaction\_ts desc";
.........

Figure 4-29 on page 153 shows the credit card transaction information as of the end of the previous business day, which is retrieved from the data warehouse, while Figure 4-30 on page 154 shows today’s up-to-the-minute credit card transaction information by accessing the operational system.
Figure 4-29  DFC CSR portal Credit Card end of previous day transactions screen
4. After the customer interaction has successfully concluded, details about the customer interaction may be entered into the CIIS by the CSR as an audit trail to assist with subsequent service calls from the same customer.
IBM Client Information Integration Solution (CIIS)

In this appendix we provide a brief overview of IBM’s Client Information Integration Solution (CIIS) used to implement the operational data store (ODS) in DFC’s Customer Insight solution.

The topics covered are:

- CIIS overview
- CIIS technical components
- CIIS benefits
- Deployment and contact details
CIIS overview

In financial services, the building of customer loyalty is one of the principal business goals of the enterprise. A single view of the customer is fundamental to such a strategy and the cornerstone of building a customer-centric organization.

The IBM Client Information Integration Solution (CIIS) consolidates operational\(^1\) customer information into a single, enterprise-wide view, and makes it available through all line-of-business (LOB) applications and across all customer contact points. CIIS provides an integrated view of a customer across all customer channels.

Business issue addressed by CIIS

CIIS addresses the business issue of coping with customer information that is incomplete, inconsistent, or out of date. Many companies have developed IT infrastructures organically over decades, primarily from a product-centric viewpoint. The result is that operational customer information is fragmented throughout the company's different business systems. Without accurate and complete customer data available to online business applications, companies experience:

- Increasing costs to service customers
- Higher costs in introducing new products and services
- Higher cost and risk in implementing new CRM initiatives that require a single view of the customer

Target audience for CIIS

CIIS addresses the aforementioned issue, and is targeted at Banking, Insurance or Financial Services companies that need to:

- Consolidate operational line-of-business systems that contain incomplete or inconsistent customer information.
- Need to provide a single customer view across all the their customer channels.
- Harmonize customer data management after mergers and acquisitions.
- Replace inadequate customer information systems with one enterprise solution.

\(^1\) The term “operational” data describes information primarily used by business systems that manage the interface with the customer. By their nature such systems require fast and reliable access to accurate customer information.
Main features of CIIS

CIIS provides an operational customer information management solution that:

- Consolidates existing sources of customer information into a single customer view.
- Provides this integrated view to all business applications and customer channels.
- Provides the performance, availability, and reliability required to provide online business applications with their customer information needs.
- Manages enterprise volumes of customer data. CIIS benchmarking has proven its capability as a high-end and upwardly scalable customer information solution.
- Enables new business initiatives and CRM applications that require a single view of customer.
- Ensures the validation and integrity of customer information as it is used by applications throughout the enterprise.
- Is designed to integrate with the company's existing business applications.
- Provides a powerful business model to capture all customer information and customer-related information.
- Is designed to be customized to meet the company's specific needs.
- Supports conformance to data directives such as those relating to privacy.
- Provides a proven solution to implementing a single enterprise view of the customer that has been successfully deployed in companies worldwide.

Kinds of information managed by CIIS

The CIIS Business Model implements the concept of "objects types" to refer to instances of business data on which actions can be performed. Essentially this model defines the services that CIIS provides to external applications. This
model simplifies how other business applications integrate with the single view of the customer maintained by CIIS.

CIIS is based on proven enterprise models—the Information Framework for banking (IFW) and Insurance Application Architecture (IAA) developed by IBM in conjunction with the banking and insurance industries.

The CIIS Business Model resolves many of the data-modeling and analysis challenges in implementing a single view of the customer. CIIS captures for each Involved Party\(^2\) or Party the following information:

- Personal information
- Demographics and psychological graphics (buying behavior)
- Multiple roles and hierarchies
- Multiple names
- Households
- Multiple addresses, telephone numbers, e-mail IDs (contact points)
- All communications and contacts
- A history of information changes

In addition, the CIIS Business Model is enhanced with business objects that represent key banking and insurance information that include:

- Accounts
- Account balances
- Payment schedules
- Agreements
- Registrations
- Policies
- Claims
- Resource item

**Important:** A key feature of the CIIS is that its business model can be readily modified and customized to meet the enterprise's specific requirements. This is achieved through the CIIS Customization Workbench described in the following section.

**CIIS technical components**

The core components of CIIS are as follows:

- **CIIS Data Server**
  
  The CIIS Data Server is a scalable and high-performance host-tier application that physically consolidates customer information from many sources and

\(^2\) A Party is any person or organization that relates to the business.
provides this information to the organization's business applications through a set of functional services defined by the CIIS Business Model. The CIIS Data Server provides an open and flexible implementation model that can be rapidly customized to meet the specific business needs of the company.

- **CIIS Customization Workbench**
  This component provides a GUI-based development environment that allows the company to customize the data content and functional services provided by the CIIS Data Server.

- **CIIS XML Adapter**
  This additional component provides an XML-based integration component that allows an external application to link to the CIIS Data Server using XML.

Each of these components is covered in greater detail in the following sections.

**CIIS Data Server**

CIIS is configured as a host-tier data server, as shown in Figure A-1 on page 160, and is designed to interface with companies' front office and legacy systems that may exist on many different platforms.
The CIIS Data Server provides functional services to other business applications through a well-defined application programming interface (API) that isolates business applications from the complexity of the underlying database access mechanisms.

Within the CIIS Data Server, there are three layers, as follows:

1. The Interface function layer provides the interfaces to external client-request systems. This layer can accept a variety of interface scenarios, including XML over MQSeries®, COM through ODBC, EJB through JDBC, Stored Procedure, Batch, or CICS®.

2. The Business function layer enforces the business model rules that govern the data accepted by the CIIS Data Server.

3. The Data function layer provides the logic for manipulation of CIIS data, that is, it governs the way in which data is stored, updated, or retrieved.
CIIS Customization Workbench

The CIIS Customization Workbench is only available on the Windows platform, and provides a GUI-based development environment that allows the company to customize the data content and functional services provided by the CIIS Data Server for all its target platforms.

The Customization Workbench provides the company with an integrated environment to customize a CIIS Data Model to meet their requirements, as shown in Figure A-2.

![CIIS Customization Workbench](image)

**Figure A-2  CIIS Customization Workbench**

The workbench brings all aspects of creating and deploying the CIIS Data Server into a common environment where data modelers, solution gap analysts, programmers, and database administrators can work together to build a solution.

**Note:** The CIIS Data Server in its standard configuration is provided on z/OS, Windows NT®, or Linux platforms. The underlying database management system is IBM DB2.
The workbench provides development capabilities that include the generation of all Business Objects together with technical documentation and sample interface programs. These capabilities together with source compilation and platform deployment functionality provide a seamless environment for customizing CIIS.

The availability of the CIIS Customization Workbench reduces the level of technical skills required to customize CIIS and reduces the reliance on traditional database analyst and development professionals.

**CIIS XML Adapter**

The CIIS XML Adapter provides an additional software component that can be optionally used to link the CIIS Data Server to other business applications. The CIIS XML Adapter is aimed at those banking and insurance companies with the following requirements:

- A need to share and exchange customer-related information to other applications in a standard and well-structured fashion
- A need to integrate CIIS with their business systems using MQSeries and a robust XML message architecture

**CIIS benefits**

These can be broadly classified into the two categories of business benefits and technical benefits, as follows.

**Business benefits**

CIIS delivers a comprehensive set of business benefits that ultimately derive from a robust and scalable customer information management solution and the availability of an accurate single view of customer information to all business applications and customer touch points.

At a high level, CIIS:

- Reduces costs. CIIS reduces the costs resulting from inefficiencies due to a combination of fragmented channel management and incompatible technology platforms.
- Provides a consistent customer experience. CIIS provides a means for companies to provide consistent customer experience across multiple channels and so retain customers and improve market share.
- Accelerates the introduction of new products and services. CIIS facilitates this through a consistent view of the customer across all channels and by
providing an infrastructure that can be used to support business initiatives that require a single view of the customer.

**Technical benefits**

Other companies have chosen CIIS as their solution to integrating operational customer information for the following reasons:

- Its proven scalability and availability in a range of environments. CIIS is based on DB2 and can support medium to large companies with customer numbers ranging from 500,000 to over 100 million customer profiles.

- Its proven flexibility. CIIS is designed to meet the specific requirements of the organization. Following a standard analysis and modeling approach, CIIS can be customized to meet the specific needs of the company.

- Its proven maintainability. CIIS is designed as an open solution that can be maintained by the implementing organization, with little ongoing external maintenance requirement. This is a critical requirement when the organization does not want to be dependent on a third party for functional changes or extensions to its customer information solution.

- Its lower total cost of ownership. The CIIS Data Server is completely model-generated and is designed for each company to become self-sufficient in maintenance and extension. In addition, CIIS is sold using an enterprise/country site license approach. Therefore the price of CIIS in a single site is the same regardless of the number of customers managed or applications supported.

**Deployment and contact details**

CIIS provides much more than just a technical implementation of a customer data management system. CIIS provides a comprehensive methodology that:

- Defines the changes required to provide your business with a single enterprise customer view

- Guides you on the best strategy for integrating customer management with your existing applications

- Ensures that CIIS deployment stays on schedule and within budget

- Is supported by the skills and resources of IBM

CIIS is IBM's strategic solution offering to the Financial Services industry and is fully aligned with IBM solutions for the financial services sector.
Attention: To learn more about how you can put IBM solutions to work for you today, please contact your local IBM representative, or contact 3IFWHelpdesk@ie.ibm.com.
Configuring data sources in DB2 Information Integrator

In this appendix we describe the DB2 Information Integrator Version 8.1 configuration of data sources used in the DFS™ portal discussed in Chapter 4, “The Druid Financial Corporation (DFC) Customer Insight solution” on page 115.

The topics covered are:
- Oracle 8i data source
- XML data source
- ODBC data source (VSAM using IICF)
- DB2 UDB for z/OS data source
- DB2 UDB for Multiplatforms data source
Introduction

DB2 II Version 8.1 supports a wide variety of data sources with many options to customize each data source configuration. Full details on all the supported data sources and the options available are documented in *IBM DB2 Information Integrator Federated Systems Guide*, SC18-7364, and refer to the *IBM DB2 Information Integrator Data Source Configuration Guide Version 8*, available as softcopy from the Web site:

http://www.ibm.com/software/data/integration/solution

In this appendix, we focus only on the configuration of data sources used in the DFS portal discussed in Chapter 4, “The Druid Financial Corporation (DFC) Customer Insight solution” on page 115.

Refer to Figure 2-7 on page 40, which highlights the basic steps involved in configuring a data source.

In the following subsections, we briefly describe the steps involved in configuring the following data sources used in the DFS portal:

- Oracle 8i data source
- XML data source
- ODBC data source (VSAM data using IICF)
- DB2 UDB for z/OS data source
- DB2 UDB for Multiplatforms data source

**Note:** The following configurations assume a DB2 II Windows install.

Oracle 8i data source

This section describes all the steps involved in configuring an Oracle 8i data source, and creating a nickname for a database object in it.

The Checking and Savings operational system resides on an Oracle 8i platform, and includes a number of tables. This section only describes the configuration of one of these tables; the reader can extrapolate the process described to the remaining tables.

The basic steps for configuring a data source are as shown in Figure 2-7 on page 40, and the following steps describe the configuration of an Oracle 8i data source:

1. Install and test the Oracle Client connection.
2. Create the Oracle wrapper.
3. Create the Oracle server definition.
4. Create the Oracle user mappings.
5. Create the Oracle nickname.
6. Test the Oracle nickname.

We used the DB2 Control Center on the Windows platform to configure the Oracle 8i data source.

**Install and test the Oracle Client connection**

Verify that the following actions have been completed successfully:

1. Oracle Client is installed on the federated server (DIOMEDE on Windows 2000), and it has been successfully configured and tested to connect to the Oracle server (MANSEL on AIX) using Oracle Client utilities like SQLPlus or the Oracle Enterprise Management Console.

   For more information on using the Oracle Client please refer to the documentation that comes with the Oracle Client.

2. Ensure that there is an entry in the Oracle tnsnames.ora file defining the parameters of the target Oracle Server that the federated server will be accessing. Example B-1 shows the contents of our tnsnames.ora file.

   **Example: B-1  The tnsnames.ora file**

   ```
   ORADB =
   (DESCRIPTION =
   (ADDRESS_LIST =
   (ADDRESS = (PROTOCOL = TCP)(HOST = 9.1.39.196)(PORT = 1521))
   )
   (CONNECT_DATA =
   (SERVICE_NAME = DEDICATED)
   )
   )
   
   The ORADB entry at the beginning of the tnsnames.ora file in Example B-1 is called the Oracle Network Service Name. This is the value that will be used as the NODE in our server definition in “Create the Oracle server definition” on page 169.

   Test the connection to Oracle as follows:

   c:\>SQLPLUS user/password@ORADB

   Where the user ID password is valid for the Oracle system.

---

1 This is a configuration file that contains the information needed by the Oracle Client to connect to the Oracle Server. This file is usually located in the `network/admin` sub-directory of the installed directory of the Oracle Client on both Windows and UNIX.
Create the Oracle wrapper

Navigate to the PATTERN federated database that was created during the DB2 II install, right click the Federated Database Objects, and click Create Wrapper, as shown in Figure B-1.

![DB2 Control Center navigation to Federated Database Objects](image)

*Figure B-1  DB2 Control Center navigation to Federated Database Objects*

This action displays the panel shown in Figure B-2 on page 169.
Figure B-2  Oracle - Create Wrapper

Select **Oracle using OCI 8** as the target data source, and choose **NET8** for the Wrapper name, as shown in Figure B-2.

Click **OK** to create the NET8 wrapper.

Example B-2 shows the command line version for creating the NET8 wrapper.

**Example: B-2  Create wrapper statement for Oracle**

```
CONNECT TO PATTERN;
CREATE WRAPPER "NET8" LIBRARY 'db2net8.dll';
CONNECT RESET;
```

**Create the Oracle server definition**

After creating the NET8 wrapper, which just specifies the type and version of the Oracle server, create the server definition for NET8 from the screen shown in Figure B-3 on page 170.
Select and expand NET8, right-click the Servers folder, and click Create to display Figure B-5 on page 172.
Choose CHKSVG for the Name field, select ORACLE for the Type field, and select 8 for the Version field, as shown in Figure B-4.

Click the **Settings** tab to complete the server definition, as shown in Figure B-5 on page 172.
Choose ORADB (value from our tnsnames.ora file in Example B-1 on page 167) for the NODE field, and set Password to Y, as shown in Figure B-5. Let the other values default.

A number of server options are available to describe a data source server; the set of options applicable to Oracle servers are listed in Table B-1 on page 173. These server options may be set at creation time or modified later.

**Note:** There can be multiple server definitions for each wrapper.

Example B-3 shows the command line version of creating the CHKSVG server definition.

**Example: B-3  Oracle - Create server statement**

```sql
CONNECT TO PATTERN;
CREATE SERVER CHKSVG TYPE ORACLE VERSION '8' WRAPPER "NET8" OPTIONS( ADD NODE 'ORADB', PASSWORD 'Y');
```
### Table B-1  Oracle server options

<table>
<thead>
<tr>
<th>Connectivity</th>
<th>Node</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>It identifies the entry name (Network Service Name) in Oracle <code>tnsnames.ora</code> file, and is case sensitive.</td>
</tr>
<tr>
<td>Password</td>
<td></td>
<td>Default=Y: Include the password on connections to Oracle</td>
</tr>
<tr>
<td>Fold_ID / Fold_PW</td>
<td></td>
<td>Default is wrapper dependent. Y: Connect four times with user ID/password with all combinations of lower and uppercase. N (recommended): Connect once with user ID/password exactly as specified. U: Connect once with user ID/password in uppercase. L: Connect once with user ID/password in lower case.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pushdown</th>
<th>Collating_Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>It specifies whether the data source uses the same default collating sequence as the federated database. This affects the pushdown of operations on character columns. Default=N Y: Both use same collating sequence. ORDER BY can be pushed down without compromising integrity of result. N: Both use different collation sequence. ORDER BY cannot be pushed down. I: Case-insensitive. ORDER BY, DISTINCT, WHERE= cannot be pushed down.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optimization</th>
<th>CPU_RATIO</th>
<th>Default: 1.0: Specifies the ratio of the DB2 Information Integrator server CPU capacity against the data source CPU capacity.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IO_RATIO</td>
<td>Default: 1.0: Specifies the ratio of the DB2 Information Integrator server I/O rate against the data source I/O rate.</td>
</tr>
<tr>
<td></td>
<td>COMM_RATE</td>
<td>Default: 2: Specifies the effective data rate of network to data source in MB per second.</td>
</tr>
</tbody>
</table>
### Table B-2  Oracle additional server options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DB2_MAXIMAL_PUSHDOWN</strong></td>
<td>It specifies the primary criteria for the optimizer in choosing the access plan. The optimizer can choose between cost optimization and the user requirement to perform query processing by the remote data source as much as possible. ‘<strong>Y</strong>’: Choose the plan with most query operations to be pushed down to the data sources. ‘<strong>N</strong>’: Choose the plan with minimum cost.</td>
</tr>
<tr>
<td><strong>PLAN_HINTS</strong></td>
<td>Hints are statement fragments that provide extra information for the Oracle optimizer. ‘<strong>Y</strong>’: Enabled ‘<strong>N</strong>’: Disabled</td>
</tr>
</tbody>
</table>

**Create the Oracle user mappings**

As mentioned earlier, the user mapping defines an association between a user ID on the federated server and a user ID on the Oracle Server. This user mapping is used by the federated database server whenever it connects to the Oracle server on behalf of the calling federated database user. An association must be created for each user that would be using the federated system.
Select and expand CHKSVG, right-click the User Mappings folder, and click Create to display Figure B-7 on page 176.
Figure B-7  Oracle - Create User Mappings

Figure B-7 lists all the user IDs available on our federated system. Select the user that sends the federated requests to the Oracle data source.

We selected the NALUR1 user, and clicked the Settings tab to display Figure B-8 on page 177.
Choose CHKSVG for the REMOTE_AUTHID field, and set the appropriate value for REMOTE_PASSWORD as shown in Figure B-8 to connect to the Oracle data source to complete the user mapping. Click OK to complete the user mapping.

User mappings may be modified after creation time.

Example B-4 shows the command line version of creating our user mapping for our Oracle instance.

Example: B-4  Oracle - Create user mapping statement

CONNECT TO PATTERN;
CREATE USER MAPPING FOR "NALUR1" SERVER "CHKSVG" OPTIONS ( ADD REMOTE_AUTHID 'CHKSVG', ADD REMOTE_PASSWORD '******');
CONNECT RESET;

Create the Oracle nickname

Once the Oracle wrapper, the server definition, and the user mapping is complete, test the connection to the data source.

After successfully testing the connection to the data source (not shown here), you may create nicknames for the database objects in the data source.
You may choose to define data mappings and/or function mappings during the creation of a nickname. Data mappings are described in “Data type mappings” on page 43, while function mappings are described in “Function mappings” on page 44.

**Note:** We did not define any data mappings or function mappings for our nicknames.

When you create a nickname for an Oracle table, catalog data from the remote server is retrieved and stored in the federated global catalog.

Select and expand CHKS VG, right click Nicknames, and click Create from the screen shown in Figure B-9.

![Figure B-9 Nickname creation for CHKS VG](image)

This action opens up a dialog window as displayed in Figure B-10 on page 179.
Figure B-10  Oracle - Create Nicknames

Figure B-10 shows two options for adding a nickname.

1. Manually add a nickname (Add) by specifying local and remote schema and table identification.

2. Use the discover functionality (Discover) to be prompted to provide all the required information, as follows.

We chose to click Add to open up a dialog window, shown in Figure B-11.

Figure B-11  Oracle - Create nicknames - Add

Enter the appropriate details, as shown in Figure B-11, and click OK to display Figure B-12 on page 180.
Click **OK** to create the nickname.

**Attention:** Once a nickname is created, its name and schema cannot be modified without dropping and recreating it. However, some other changes can be made.

Example B-5 shows the command line version of creating the S_ACCOUNT nickname on the CHKSVG.ACCOUNT table.

**Example: B-5  Oracle - Create nickname statements**

```
CONNECT TO PATTERN;
CREATE NICKNAME NALUR1.S_ACCOUNT FOR CHKSVG.CHKSVG.ACCOUNT;
CONNECT RESET;
```

**Test the Oracle nickname**

To test the nickname right-click it, and click **Sample Contents**, as shown in Figure B-13 on page 181.
This action should display the contents of the database object referenced by the nickname, if nickname creation was successful, as shown in Figure B-14 on page 182.
This section describes all the steps involved in configuring an XML data source, and creating a nickname on it.

The Rewards description is stored as an XML file.

The basic steps for configuring a data source are as shown in Figure 2-7 on page 40, and the following steps describe the configuration of an XML data source:

1. Create the XML wrapper.
2. Create the XML server definition.
3. Create the XML nickname.
4. Test the XML nickname.

We used the DB2 Control Center on the Windows platform to configure the XML data source.

**XML data source**

This section describes all the steps involved in configuring an XML data source, and creating a nickname on it.

The Rewards description is stored as an XML file.

The basic steps for configuring a data source are as shown in Figure 2-7 on page 40, and the following steps describe the configuration of an XML data source:

1. Create the XML wrapper.
2. Create the XML server definition.
3. Create the XML nickname.
4. Test the XML nickname.

We used the DB2 Control Center on the Windows platform to configure the XML data source.
Create the XML wrapper

The XML wrapper can be used in the following cases:

- The user wants to join XML data with other nicknames or other relational data.
- The user wants to keep the original XML intact (to avoid replicating data or data that may change often or is composed on the fly).

The XML document used in a nickname may come from any of the following sources:

- A text file, for example, c:\myfiles\mydata.xml
- A directory of text files, for example, c:\myfiles
- An XML document stored in a column of a table in a database
- From a URI, for example, http://www.mysite.com/mydata.xml

The XML wrapper supports the following data types: INTEGER, SMALLINT, FLOAT, REAL, DECIMAL, CHAR, VARCHAR, and DATE.

**Note:** The XML wrapper does not support the INSERT, UPDATE, and DELETE functions.

Ensure that the following prerequisites are in place before deciding to use the XML wrapper:

- Flat files (table structured files), XML, and Microsoft Excel files *must* be located on a local or network mapped drive of the Windows server where DB2 II is installed.
- Since it is DB2 that needs to access the files on the local or network drive, you need to ensure that the *userid under which DB2 is running* (default is db2admin) is authorized to the domain containing the network drives. In our case, the userid is ‘NALUR1’.

To determine the userid of DB2, navigate to the Services screen as follows: **Start → Settings → Control Panel → Administrative Tools → Services.**

This displays Figure B-15 on page 184. The Log On As field identifies the DB2 userid (db2admin in Figure B-15 on page 184).
Figure B-15  Determining userid of DB2 II

Navigate to the PATTERN federated database that was created during the DB2 II install, right-click Federated Database Objects, and click Create Wrapper, as shown in Figure B-1 on page 168, to display Figure B-16.

Figure B-16  XML - Create Wrapper
Select **XML** in the Data source field, and type **XML** in the Wrapper name field, as shown in Figure B-16 on page 184.

Click **OK** to create the XML wrapper using the default settings.

Example B-6 shows the command line version of creating the XML wrapper.

**Example: B-6  XML - Create wrapper statement**

```
CONNECT TO PATTERN;
CREATE WRAPPER "XML" LIBRARY 'db2lsxml.dll';
CONNECT RESET;
```

### Create the XML server definition

For XML, the DB2 Information Integrator server definition is required even though it does not require information such as the version and connection information needed in the server definition for relational data sources.

Select and expand **XML**, right-click the **Servers** folder, and click **Create**. This is similar to the actions taken for **NET8** from Figure B-3 on page 170.

This action will display Figure B-17.

![Create Server](image)

**Figure B-17  XML - Create Server**

The server definition requires the name of the server definition to be unique within the federated database. In our case we entered **RWDMKT**, as shown in Figure B-17.

Click **OK** to complete the RWDMKT server definition.

For the XML server, the server options cannot be updated, added, or dropped.
**Note:** There can be multiple server definitions for each defined XML wrapper

Example B-7 shows the command line version of creating the RWDMKT server definition.

**Example: B-7  XML - Create server statement**

```sql
CONNECT TO PATTERN;
CREATE SERVER RWDMKT WRAPPER "XML";
CONNECT RESET;
```

**Create the XML nickname**

After setting up the XML wrapper and the XML server definition, we can create the actual link to an XML file, a directory containing an XML file, or a URI.

**Attention:** Unlike some of the other data sources, there are no user mappings associated with XML data sources since the federated server user ID must have access to the XML file or directory.

Example B-8 shows content of the RewardsXML file used in the DFS portal.

**Example: B-8  Content of the RewardsXML file**

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!-- edited with XMLSpy v2004 rel. 3 U (http://www.xmlspy.com) by Bill Mathews (IBM) -->
<!-- Druid Bank Credit Card Rewards -->
<Rewards xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\MyShares\RewardsXML\Rewards.xsd">
  <id>100001</id>
  <Short_Name>Travel2004</Short_Name>
  <Description>Druid Bank Credit Card Travel Purchase Awards</Description>
  <Sponsor>DruidBankTravel.com</Sponsor>
  <Card_Type>VISA</Card_Type>
  <Affiliation>
    <Name>We Get You There Airlines</Name>
    <Points>3</Points>
    <per_unit>1</per_unit>
    <unit_type>Dollar</unit_type>
    <Marketing_Msg>Use your Druid Bank Credit Card to purchase your "We Get You There Airlines" tickets at DruidBankTravel.com and receive 3 points for every dollar spent!</Marketing_Msg>
```
Select and expand **RWDMKT**, right-click **Nicknames**, and click **Create**. This is similar to the actions taken for the **CHKSVG** in Figure B-9 on page 178.

This action opens up a dialog window, as displayed in Figure B-18.

![Figure B-18 XML - Create nickname](image)

Figure B-18 is similar to Figure B-10 on page 179 and provides two options (Add and Discover) for adding a nickname.

We chose to click **Add** to open up the dialog window shown in Figure B-19 on page 188.
Click the **Add** button to add the various columns, data types, and settings, as shown in Figure B-20 on page 189, Figure B-21 on page 189, Figure B-22 on page 190, Figure B-24 on page 191, and Figure B-24 on page 191.
Appendix B. Configuring data sources in DB2 Information Integrator

Figure B-20   Add column and data type details

Figure B-21   Add column settings
Figure B-22  Completed list of columns with data types

Figure B-23  Nickname settings
Figure B-24  XML - Create nickname window

Select the nickname and click OK to complete the definition of the nickname.

Attention: Once a nickname is created, its name and schema cannot be modified without dropping and recreating it. However, some other changes can be made.

Figure B-25 shows the equivalent SQL for creating this nickname.

```sql
CREATE NICKNAME NALUR1 M_REWARDS (ID INTEGER OPTIONS(XPATH '/id'), NAME VARCHAR (16) OPTIONS(XPATH '/Short_Name'), DESC VARCHAR (80) OPTIONS(XPATH '/Description'), SPONSER VARCHAR (50) OPTIONS(XPATH '/Sponsor'), TYPE VARCHAR (50) OPTIONS(XPATH '/Card_Type'), CID VARCHAR (16) OPTIONS(PRIMARY_KEY "YES") FOR SERVER "RWDMKT" OPTIONS(XPATH "Reward", FILE_PATH 'D:\Documentation\Data\RewardsXML\Rewards.xml');
```

Figure B-25  Nickname creation using SQL
Test the XML nickname

To test the M_REWARDS nickname, right click it and click Sample Contents, similar to Figure B-13 on page 181.

This action should display the contents of the database object referenced by the nickname if nickname creation was successful. We have not displayed the contents here.

ODBC data source (VSAM using IICF)

This section describes the steps involved in configuring a VSAM data set (that appears as an ODBC data source through the IICF product), and creating a nickname on it.

The Rewards operational system resides on a VSAM file system on z/OS, and includes three VSAM KSDS files (Tracking, Offering Partners, and Transactions). We used the DB2 IICF product to provide a relational view of these three VSAM KSDS files, and then used the ODBC wrapper of DB2 II to create nicknames for these relational table representations of the VSAM files.

This section will only describe the configuration of one of these tables. The reader can extrapolate the process described to the remaining tables.

The basic steps for configuring a data source are as shown in Figure 2-7 on page 40.

In the following subsections, we describe the configuration of an ODBC data source that includes the following steps.

1. Set up the VSAM ODBC data source name on Windows.
2. Create the ODBC wrapper.
3. Create the ODBC server definition.
4. Create the ODBC user mapings.
5. Create the ODBC nickname.
6. Test the ODBC nickname.

Set up the VSAM ODBC data source name on Windows

Perform the following steps to set up the IICF VSAM ODBC Data Source Name on Windows 2000:

1. Select Start → Settings → Control Panel → Administrative Tools → Data Sources (ODBC).
This opens the ODBC Data Source Administrator screen, as shown in Figure B-26.

**Figure B-26  ODBC Data Source Administrator**

2. Click the **System DSN** tab, and click **Add** to add a new data source name, which displays Figure B-27.

**Figure B-27  Create New Data Source**
3. Select the driver **CrossAccess32**, as shown in Figure B-28.

![Create New Data Source](image)

**Figure B-28  ODBC Create New Data Source**

4. Click the **Finish** button to view the communications protocol setup screen shown in Figure B-29.

![Communications Protocol](image)

**Figure B-29  CrossAccess32 Communications Protocol**

5. Select **TCP/IP** and click **OK** to display the Database Integrator ODBC Data Source Configuration screen shown in Figure B-30 on page 195.
6. Provide details for the Data Source Name (CACSAMP), Host IP Address (wtsc63.itso.ibm.com), Host Port Number (5002), check OS Login ID Required, and click the Advanced tab to display Figure B-31 on page 196.
7. Ensure that the Catalog Owner Name is SYSIBM, and click **OK** to complete the data source name configuration.

**Create the ODBC wrapper**

Navigate to the **PATTERN** federated database that was created during the DB2 II install, right-click the **Federated Database Objects**, and click **Create Wrapper**, as shown in Figure B-1 on page 168.

This action displays Figure B-32 on page 197.
Select **ODBC** in the **Data source** field, and type **ODBC** in the **Wrapper name** field, as shown in Figure B-32.

Click the **Settings** tab to see Figure B-33.

Ensure that the **DB2_FENCED** option is set to **N** (the default value), and set the **MODULE** option with the full library path for the ODBC driver as shown in Figure B-33.
Click **OK** to create the ODBC wrapper.

Example B-9 shows the command line version of creating the ODBC wrapper.

**Example: B-9   ODBC wrapper - Create wrapper statement**

```
CONNECT TO PATTERN;
CREATE WRAPPER "ODBC" LIBRARY 'db2rcodbc.dll'
  OPTIONS( ADD MODULE 'odbc32.dll');
CONNECT RESET;
```

### Create the ODBC server definition

Select and expand the **ODBC**, right-click the **Servers** folder, and click **Create**, similar to the actions taken for the **NET8** wrapper in Figure B-3 on page 170.

This action will display Figure B-34.

![Create Server dialog](image)

**Figure B-34   ODBC wrapper - Create Server**

Choose **REWARDS** for the Name field, select **ODBC** for the Type field, and select **3.0** for the Version field, as shown in Figure B-34.

Click the **Settings** tab to open the dialog window shown in Figure B-35 on page 199.
Set the following server options in Figure B-35.

For the NODE option, type CACSAMP in the Value column. This is the name of the ODBC data source name that we configured earlier.

Click OK to create the server definition.

Example B-10 shows the command line version of the ODBC server definition.

Example: B-10   ODBC wrapper - Create server statement

CONNECT TO PATTERN;
CREATE SERVER REWARDS TYPE ODBC VERSION '3.0' WRAPPER "ODBC"
  OPTIONS( ADD NODE 'CACSAMP' );
CONNECT RESET;

The following options may be set for increased performance using the CREATE SERVER statement, or by using the ALTER SERVER statement:

- PUSHDOWN 'Y'
- DB2_BASIC_PRED 'Y'
- DB2_ORDER_BY 'Y'
- DB2_GROUP_BY 'Y'
- DB2_COLFUNC 'Y'
- DB2_SELECT_DISTINCT 'Y'
Create the ODBC user mappings

As mentioned earlier, user mapping is used by the federated database server whenever it connects to the ODBC data source server on behalf of the calling federated database user. An association must be created for each user that would be using the federated system.

In our case, we define a single user mapping for our NALUR1 user since that is the only user ID used in our DFS portal.

Select and expand REWARDS, right-click the User Mappings folder, and click on Create, similar to what was done for CHKSVG from Figure B-6 on page 175.

This action displays the screen shown in Figure B-36 on page 201.
Figure B-36 lists all the user IDs available on the federated system. Select the user that sends the federated requests to the ODBC data source. We selected the NALUR1 user, and switched to the Settings menu, as shown in Figure B-37 on page 202, to complete the user mapping.
Choose NAGRAJ1 for the REMOTE_AUTHID field, and set the appropriate value for REMOTE_PASSWORD as shown in Figure B-37 to connect to the ODBC data source to complete the user mapping. Click **OK** to complete the user mapping.

User mappings may be modified after creation time.

Example B-14 on page 214 shows the command line version of creating the user mapping for our ODBC data source.

**Example: B-11  ODBC data source - Create user mapping statements**

```
CONNECT TO PATTERN;
CREATE USER MAPPING FOR "NALUR1" SERVER "REWARDS" OPTIONS ( ADD REMOTE_AUTHID 'NAGRAJ1', ADD REMOTE_PASSWORD '*****') ;
```

**Create the ODBC nickname**

After setting up the ODBC wrapper and the ODBC server definition, we can create the actual link to the data source.

Select and expand **REWARDS**, right-click **Nicknames**, and click **Create**, similar to the actions taken for the **CHKSVG** server in Figure B-9 on page 178.
This action opens up the dialog window displayed in Figure B-38.

![Figure B-38 ODBC wrapper - Create nickname - Main window](image)

Figure B-38 is similar to Figure B-10 on page 179 and provides two options (Add and Discover) for adding a nickname.

We chose to click **Add** to open up a dialog window, as shown in Figure B-39, to create a nickname.

![Figure B-39 ODBC data source - Add Nickname](image)

Fill in the details as shown, and click **OK** to open up the dialog window shown in Figure B-40 on page 204.
Select the R_TRACKING nickname and click OK to create the nickname with the defaults. Clicking Show SQL displays the SQL used to create the nickname, as shown in Figure B-41.

The default schema for the nicknames is the user ID that is creating it. In our case that is NALUR1.

Test the ODBC nickname

To test the R_TRACKING nickname, right-click it, and click Sample Contents, as shown in Figure B-42 on page 205.
The results of the sample contents request is displayed in Figure B-43 on page 206, showing that the nickname creation was successful.
This section describes the steps involved in configuring an DB2 UDB for z/OS data source, and creating a nickname for a database object in it.

The Credit Card operational system resides on a DB2 UDB for z/OS platform, and includes a number of tables. This section only describes the configuration of one of these tables; the reader can extrapolate the process described to the remaining tables.

The basic steps for configuring a data source are as shown in Figure 2-7 on page 40, and the following steps describe the configuration of an DB2 UDB for z/OS data source:

1. Catalog DB2 UDB for z/OS.
2. Create the DB2 UDB for z/OS wrapper.

**DB2 UDB for z/OS data source**

**Figure B-43  ODBC wrapper - Create nickname - Sample contents results**
3. Create the DB2 UDB for z/OS server definition.
4. Create the DB2 UDB for z/OS user mappings.
5. Create the DB2 UDB for z/OS nickname.
6. Test the DB2 UDB for z/OS nickname.

We used the DB2 Control Center on the Windows platform to configure the DB2 UDB for z/OS data source.

**Catalog DB2 UDB for z/OS**

Table B-4 lists the information needed to configure a DB2 UDB for z/OS data source.

<table>
<thead>
<tr>
<th>Table B-4</th>
<th>The DB2 UDB for z/OS system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>Host name</td>
<td>9.12.6.8</td>
</tr>
<tr>
<td>TCP Port</td>
<td>33378</td>
</tr>
<tr>
<td>User</td>
<td>&lt;user&gt;</td>
</tr>
<tr>
<td>Password</td>
<td>&lt;password&gt;</td>
</tr>
<tr>
<td>Location</td>
<td>DB2G</td>
</tr>
<tr>
<td>Creator</td>
<td>NAGRAJ1</td>
</tr>
</tbody>
</table>

The following steps catalog a database from the DB2 command line:
1. Log in with user administrator or db2admin to the Windows system.
2. Catalog the DB2 UDB UDB for z/OS node as follows:
   
   ```
   db2 catalog tcpip node DB2ZSRV remote 9.12.6.8 server 33378
   ```
3. Store information about the remote host in the Database Connection Services (DCS) directory as follows:
   
   ```
   db2 catalog dcs database DCSDB2G as DB2G with “Comment on DB2 for z/OS”
   ```
4. Catalog the database as follows:
   
   ```
   db2 catalog database DCSDB2G at node DB2ZSRV authentication dcs
   ```
5. Test the connection to the database as follows:
   
   ```
   db2 connect to DCSDB2G user <user> using <password>
   ```

**Note:** DCSDB2G is the database name that will be used in the DBNAME server option in our federated server definition for this DB2 UDB for z/OS data source.
Create the DB2 UDB for z/OS wrapper

**Note:** If there is an existing wrapper for DB2, it can be reused for the DB2 UDB for z/OS subsystem and a new wrapper is not necessary.

Navigate to the PATTERN federated database that was created during the DB2 II install, right-click the **Federated Database Objects**, and click **Create Wrapper**, as shown in Figure B-1 on page 168.

This action displays Figure B-44.

![Create Wrapper](image)

*Figure B-44  DB2 UDB for z/OS - Create wrapper*

Select **DB2** in the Data Source field, and enter the unique name **DRDA** as the Wrapper name. The Settings tab option is allowed to default. Click **OK** to create the DB2 wrapper.

Example B-12 shows the command line version for creating the DRDA wrapper for the DB2 UDB for z/OS instance.

*Example: B-12  DB2 UDB for z/OS - Create wrapper statements*

```sql
CONNECT TO PATTERN;
CREATE WRAPPER "DRDA" LIBRARY 'libdb2drda.a';
CONNECT RESET;
```
Create the DB2 UDB for z/OS server definition

A server definition identifies a data source to the federated database. A server definition consists of a local name and other information about that data source server.

Select and expand the DRDA, right-click the Servers folder, and click Create, similar to the actions taken for NET8 from Figure B-3 on page 170.

This action will display Figure B-45.

![Create Server](image)

**Figure B-45   DB2 UDB for z/OS - Create server**

Choose CREDIT for the Name field, select DB2/390 for the Type field, select 7 for the Version field; supply the User ID (NAGRAJ1) and Password values to connect to this server, as shown in Figure B-45.

Click the Settings tab to complete the server definition, as shown in Figure B-46 on page 210.
In Figure B-46, only the DBNAME and PASSWORD fields are required values; the rest are optional. Provide CREDIT in the DBNAME field and set PASSWORD to Y. Server options are used to describe a data source server. The DB2 UDB for z/OS server has a number of options, as listed in Table B-5. These options may be set at server creation time, or modified later.

**Table B-5  DB2 UDB for z/OS server options**

<table>
<thead>
<tr>
<th>Connectivity</th>
<th>DB Alias in DB Directory on DB2 Information Integrator server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password</td>
<td>Default=Y, Include the password on connections to DB2 UDB for z/OS.</td>
</tr>
<tr>
<td>Fold_ID / Fold_PW</td>
<td>Option to fold or not for the DRDA wrapper.  \n</td>
</tr>
</tbody>
</table>
Example B-13 shows the command line version for creating the CREDIT server definition.

**Example: B-13  DB2 UDB for z/OS - Create server statement**

```sql
CONNECT TO PATTERN;
CREATE SERVER CREDIT TYPE DB2/390 VERSION '7' WRAPPER "DRDA" AUTHID "NAGRAJ1"
PASSWORD "*****" OPTIONS( ADD DBNAME 'DB2G', PASSWORD 'Y');
```

**Note:** The server option shown in Table B-6 is *not* available through the DB2 Control Center. It must be set through the command line.
Create the DB2 UDB for z/OS user mappings

As mentioned earlier, the user mapping defines an association between a user ID on the federated server and a user ID on the DB2 UDB for z/OS server. This user mapping is used by the federated database server whenever it connects to the DB2 UDB for z/OS server on behalf of the calling federated database user. An association must be created for each user that would be using the federated system.

In our case, we define a single user mapping for our NALUR1 user since that is the only user ID used in our DFS portal.

Select and expand CREDIT, right-click the User Mappings folder, and click Create, similar to what was done for CHKSVG from Figure B-6 on page 175.

This action displays the screen shown in Figure B-47 on page 213.
Figure B-47 lists all the user IDs available on the federated system. Select the user that sends the federated requests to the DB2 UDB for z/OS data source. We selected the NALUR1 user, and switched to the Settings menu, as shown in Figure B-48 on page 214, to complete the user mapping.
Choose NAGRAJ1 for the REMOTE_AUTHID field, set the appropriate value for REMOTE_PASSWORD, and let ACCOUNTING_STRING default as shown in Figure B-48 to connect to the DB2 UDB for z/OS data source to complete the user mapping.

**Attention:** You might also consider adding the user mapping option ACCOUNTING_STRING; DB2 UDB for z/OS is the only data source that uses it.

Click **OK** to complete the user mapping.

User mappings may be modified after creation time.

Example B-14 shows the command line version of creating the user mapping for our DB2 UDB for z/OS instance.

**Example: B-14  DB2 UDB for z/OS - Create user mapping statement**

```sql
CONNECT TO PATTERN;
CREATE USER MAPPING FOR "NALUR1" SERVER "CREDIT" OPTIONS (ADD REMOTE_AUTHID 'NAGRAJ1', ADD REMOTE_PASSWORD '*****');
```
Create the DB2 UDB for z/OS nickname

Once the DB2 wrapper, server definition, and user mapping is complete, one needs to test the connection to the data source.

After successfully testing the connection to the data source (not shown here), we can create the actual link to a table located on our remote database as a nickname.

You may choose to define data mappings and/or function mappings during the creation of a nickname. Data mappings are described in “Data type mappings” on page 43, while function mappings are described in “Function mappings” on page 44.

**Note:** We did not define any data mappings or function mappings for our nicknames.

When you create a nickname for a DB2 UDB for z/OS table, catalog data from the remote server is retrieved and stored in the federated global catalog.

Select and expand **CREDIT**, right-click **Nicknames**, and click **Create**, similar to the actions taken for **CHKSVG** from Figure B-9 on page 178.

This action opens up the dialog window shown in Figure B-49 on page 216, similar to the one shown in Figure B-10 on page 179, which shows two options (Add and Discover) for adding a nickname.
Click **Add** to open up a dialog window, as shown in Figure B-49, to create a nickname.

**Figure B-49  DB2 UDB for z/OS - Nickname creation**

Click **Add** to open up a dialog window, as shown in Figure B-49, to create a nickname.

**Figure B-50  DB2 UDB for z/OS - Add Nickname**

Fill in the details as shown, and click **OK** to open up the dialog window shown in Figure B-51 on page 217.
Appendix B. Configuring data sources in DB2 Information Integrator

Figure B-51   DB2 UDB for z/OS - Create Nicknames

Select the **C_CARD_HOLDER** nickname and click **OK** to create the nickname with the defaults. Clicking **Show SQL** displays the SQL used to create the nickname, as shown in Figure B-52.

Figure B-52   DB2 UDB for z/OS - SQL for creating the nickname

The default schema for the nicknames is the user ID that is creating it. In our case that is NALUR1.

**Test the DB2 UDB for z/OS nickname**

To test the **C_CARD_HOLDER** nickname, right-click it, and click **Sample Contents**, as shown in Figure B-53 on page 218.
The results of the sample contents request is displayed in Figure B-54 on page 219, showing that the nickname creation was successful.
DB2 UDB for Multiplatforms data source

This section describes the steps involved in configuring a DB2 UDB for Multiplatforms data source, and creating a nickname for a database object in it.

The Loans and Brokerage operational systems reside on a DB2 UDB for Multiplatforms platform, and includes a number of tables. This section only describes the configuration of one of these tables; the reader can extrapolate the process described to the remaining tables.

The basic steps for configuring a data source are as shown in Figure 2-7 on page 40, and the following steps describe the configuration of a DB2 UDB for Multiplatforms data source:

1. Catalog the DB2 UDB for Multiplatforms database.
2. Create the DB2 UDB for Multiplatforms wrapper.
3. Create the DB2 UDB for Multiplatforms server definition.
4. Create the DB2 UDB for Multiplatforms user mappings.
5. Create the DB2 UDB for Multiplatforms nickname.
6. Test the DB2 UDB for Multiplatforms nickname.

We used the DB2 Control Center on the Windows platform to configure the DB2 UDB for Multiplatforms data source.

Catalog DB2 UDB for Multiplatforms

The remote database must first be cataloged using either the Client Configuration Assistant (CCA) or through the command line.

The following steps catalog a database from the DB2 command line:
1. Log in with user administrator or db2admin to the Windows system.
2. Catalog the DB2 UDB for Multiplatforms node as follows:
   `db2 catalog tcpip node malmo remote 9.1.39.175 server DB2_db2loan`
3. Catalog the database as follows:
   `db2 catalog database DB2LOANS as LOANS at node malmo`
4. Test the connection to the database as follows:
   `db2 connect to LOANS user <user> using <password>`

Create the DB2 UDB for Multiplatforms wrapper

Since we had already created a wrapper named DRDA for DB2 UDB for z/OS, we can reuse the same wrapper definition for the DB2 UDB for Multiplatforms data sources.

Create the DB2 UDB for Multiplatforms server definition

Select and expand the DRDA wrapper, right-click the Servers folder, and click Create, similar to the actions taken for the NET8 wrapper from Figure B-3 on page 170.

This action will display Figure B-55 on page 221.
Appendix B. Configuring data sources in DB2 Information Integrator

Figure B-55  DB2 UDB for Multiplatforms - Create server

Choose LOANS for the Name field, select DB2/UDB for the Type field, select 8.1 for the Version field, and supply the User ID (db2loan) and Password values to connect to this server as shown in Figure B-55.

Click the Settings tab to complete the server definition as shown in Figure B-56 on page 222.
Figure B-56  DB2 UDB for Multiplatforms - Create Server settings

In Figure B-56, only the DBNAME and PASSWORD fields are required values; the rest are optional. Provide LOANS in the DBNAME field and set PASSWORD to Y. Server options are used to describe a data source server. The DB2 UDB for Multiplatforms server has a number of options similar to those listed in Table B-5 on page 210. These options may be set at server creation time, or modified later.

Note: There can be multiple server definitions for each wrapper.

Example B-15 shows the command line version for creating the LOANS server definition.

Example: B-15  DB2 UDB for z/OS - Create server statements

CONNECT TO PATTERN;
CREATE SERVER LOANS TYPE DB2/UDB VERSION '8.1' WRAPPER "DRDA" AUTHID "db2loan"
PASSWORD "*****" OPTIONS( ADD DBNAME 'LOANS', PASSWORD 'Y');

Create the DB2 UDB for Multiplatforms user mappings

As mentioned earlier, the user mapping defines an association between a user ID on the federated server and a user ID on the DB2 UDB for Multiplatforms server. This user mapping is used by the federated database server whenever it
connects to the DB2 UDB for Multiplatforms server on behalf of the calling federated database user. An association must be created for each user that would be using the federated system.

In our case, we define a single user mapping for our NALUR1 user since that is the only user ID used in our DFS portal.

Select and expand LOANS, right-click the User Mappings folder, and click Create, similar to what was done for CHKSVG from Figure B-6 on page 175.

This action displays the screen shown in Figure B-57.

Figure B-57   DB2 UDB for Multiplatforms - Create User Mapping dialog

Figure B-57 lists all the user IDs available on the federated system. Select the user that sends the federated requests to the DB2 UDB for z/OS data source. We selected the NALUR1 user, and switched to the Settings menu, as shown in Figure B-58 on page 224, to complete the user mapping.
Choose `db2loan` for the `REMOTE_AUTHID` field, set the appropriate value for `REMOTE_PASSWORD`, and let `ACCOUNTING_STRING` default as shown in Figure B-58 to connect to the DB2 UDB for Multiplatforms data source to complete the user mapping.

User mappings may be modified after creation time.

Example B-16 shows the command line version of creating the user mapping for our DB2 UDB for Multiplatforms instance.

**Example: B-16  DB2 UDB for Multiplatforms - Create user mapping statements**

```sql
CONNECT TO PATTERN;
CREATE USER MAPPING FOR "NALUR1" SERVER "LOANS" OPTIONS ( ADD REMOTE_AUTHID 'db2loan', ADD REMOTE_PASSWORD '*****');
```

**Create the DB2 UDB for Multiplatforms nickname**

Once the DB2 wrapper, server definition, and user mapping is complete, you need to test the connection to the data source.

After successfully testing the connection to the data source (not shown here), we can create the actual link to a table located on our remote database as a nickname.
You may choose to define data mappings and/or function mappings during the creation of a nickname. Data mappings are described in “Data type mappings” on page 43, while function mappings are described in “Function mappings” on page 44.

**Note:** We did not define any data mappings or function mappings for our nicknames.

When you create a nickname for a DB2 UDB for Multiplatforms table, catalog data from the remote server is retrieved and stored in the federated global catalog.

Select and expand **LOANS**, right-click **Nicknames**, and click **Create**, similar to the actions taken for **CHKSVG** from Figure B-9 on page 178.

This action opens up the dialog window shown in Figure B-59, which shows two options (Add and Discover) for adding a nickname.

![Create Nicknames - LOANS](image)

*Figure B-59  DB2 UDB for Multiplatforms - Nickname creation*

Click **Add** to open up a dialog window, as shown in Figure B-60 on page 226, to create a nickname.
Fill in the details as shown, and click **OK** to open up a dialog window, as shown in Figure B-60.

Select the **L_BORROWER** nickname and click **OK** to create the nickname with the defaults. Clicking **Show SQL** displays the SQL used to create the nickname, as shown in Figure B-62 on page 227.
The default schema for the nicknames is the user ID that is creating it. In our case that is NALUR1.

**Test the DB2 UDB for Multiplatforms nickname**

To test the L_BORROWER nickname, right-click it, and click Sample Contents, as shown in Figure B-63 on page 228.
Figure B-63  DB2 UDB for Multiplatforms - Sample contents of nickname

The results of the sample contents request is displayed in Figure B-64 on page 229, showing that the nickname creation was successful.
<table>
<thead>
<tr>
<th>BORROWER</th>
<th>NAME</th>
<th>STREET_</th>
<th>CITY</th>
<th>ST</th>
<th>PHONE</th>
<th>BIRTH_DATE</th>
<th>DOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>9273506122</td>
<td>SHARPTO</td>
<td>7517 Cam</td>
<td>Dallas</td>
<td>TX</td>
<td>972-267-6</td>
<td>Feb 1, 1949</td>
<td>8624</td>
</tr>
<tr>
<td>9273535611</td>
<td>ROSE</td>
<td>7517 Cam</td>
<td>Dallas</td>
<td>TX</td>
<td>972-267-6</td>
<td>Jun 25, 1952</td>
<td>724</td>
</tr>
<tr>
<td>2643764531</td>
<td>DONALD</td>
<td>397 Las C.</td>
<td>Irving</td>
<td>TX</td>
<td>972-810-3</td>
<td>May 26, 1941</td>
<td>625</td>
</tr>
<tr>
<td>9013452012</td>
<td>VINCENZO</td>
<td>333 Market</td>
<td>San Francisco</td>
<td>CA</td>
<td>415-817-5</td>
<td>Nov 5, 1923</td>
<td>620</td>
</tr>
<tr>
<td>9013452012</td>
<td>SEAN</td>
<td>60 State St</td>
<td>Boston</td>
<td>MA</td>
<td>617-973-5</td>
<td>Oct 18, 1942</td>
<td>683</td>
</tr>
<tr>
<td>9013453017</td>
<td>DICREGES</td>
<td>305 2nd Ave</td>
<td>Waltham</td>
<td>MA</td>
<td>781-469-8</td>
<td>Oct 15, 1919</td>
<td>734</td>
</tr>
<tr>
<td>9013453040</td>
<td>HEATHER</td>
<td>716 Main St</td>
<td>Waltham</td>
<td>MA</td>
<td>781-647-8</td>
<td>Jan 10, 1946</td>
<td>737</td>
</tr>
<tr>
<td>9013453093</td>
<td>BRUCE</td>
<td>445 Main St</td>
<td>Wakefield</td>
<td>MA</td>
<td>781-248-2</td>
<td>May 17, 1947</td>
<td>733</td>
</tr>
<tr>
<td>9013404138</td>
<td>ELIZABET</td>
<td>699 North</td>
<td>Wakefield</td>
<td>MA</td>
<td>781-245-9</td>
<td>Apr 12, 1956</td>
<td>974</td>
</tr>
<tr>
<td>9013404180</td>
<td>MARITOS</td>
<td>611 Congr.</td>
<td>Portland</td>
<td>ME</td>
<td>207-777-3</td>
<td>Jun 5, 1948</td>
<td>679</td>
</tr>
<tr>
<td>9013404173</td>
<td>MARILYN</td>
<td>2 Portland</td>
<td>Portland</td>
<td>ME</td>
<td>207-761-8</td>
<td>Feb 21, 1949</td>
<td>656</td>
</tr>
<tr>
<td>9013404130</td>
<td>JAMES</td>
<td>52 S BRO.</td>
<td>Norwich</td>
<td>NY</td>
<td>603-337-2</td>
<td>Jun 26, 1952</td>
<td>820</td>
</tr>
<tr>
<td>9013404130</td>
<td>DAVID</td>
<td>520 S Sed.</td>
<td>Chicago</td>
<td>NE</td>
<td>402-555-1</td>
<td>May 26, 1941</td>
<td>850</td>
</tr>
<tr>
<td>9013515148</td>
<td>WILLIAMS</td>
<td>7110 S 149.</td>
<td>Omaha</td>
<td>NE</td>
<td>402-537-7</td>
<td>Feb 25, 1953</td>
<td>962</td>
</tr>
<tr>
<td>9013604369</td>
<td>JENNIFER</td>
<td>34 Camp St.</td>
<td>Concord</td>
<td>NH</td>
<td>603-229-3</td>
<td>Mar 19, 1948</td>
<td>972</td>
</tr>
<tr>
<td>9013635532</td>
<td>PEIBERG</td>
<td>1540 Hamil</td>
<td>San Jose</td>
<td>CA</td>
<td>408-448-3</td>
<td>Aug 11, 1932</td>
<td>621</td>
</tr>
<tr>
<td>9013670874</td>
<td>WEIL</td>
<td>7 Kenney Ave</td>
<td>San Jose</td>
<td>CA</td>
<td>408-812-1</td>
<td>Feb 21, 1948</td>
<td>681</td>
</tr>
<tr>
<td>9014001372</td>
<td>FERO</td>
<td>5055 Alma</td>
<td>San Jose</td>
<td>CA</td>
<td>408-761-1</td>
<td>Jun 26, 1952</td>
<td>862</td>
</tr>
<tr>
<td>9014006237</td>
<td>JANOSKOK</td>
<td>167 W San</td>
<td>San Jose</td>
<td>CA</td>
<td>406-971-2</td>
<td>Sep 11, 1947</td>
<td>972</td>
</tr>
<tr>
<td>9014044802</td>
<td>JERRY</td>
<td>216 Castro</td>
<td>Mountain,</td>
<td>CA</td>
<td>650-869-3</td>
<td>Feb 4, 1954</td>
<td>662</td>
</tr>
<tr>
<td>9014062927</td>
<td>KERRY</td>
<td>233 State St</td>
<td>Los Allos</td>
<td>CA</td>
<td>650-847-3</td>
<td>Dec 12, 1960</td>
<td>652</td>
</tr>
<tr>
<td>9014066622</td>
<td>DEAN</td>
<td>4913 Paseo</td>
<td>4913 Paseo</td>
<td>CA</td>
<td>510-797-5</td>
<td>May 26, 1952</td>
<td>622</td>
</tr>
<tr>
<td>9014050580</td>
<td>EDWARDS</td>
<td>2550 El Ca.</td>
<td>Redwood</td>
<td>CA</td>
<td>650-369-8</td>
<td>Mar 26, 1952</td>
<td>813</td>
</tr>
<tr>
<td>9014050589</td>
<td>GILBERT</td>
<td>300 Penins.</td>
<td>San Mateo</td>
<td>CA</td>
<td>650-344-2</td>
<td>Jul 15, 1948</td>
<td>823</td>
</tr>
<tr>
<td>9014050590</td>
<td>WESLEY</td>
<td>1206 Newe</td>
<td>Walnut Creek</td>
<td>CA</td>
<td>925-635-8</td>
<td>Oct 27, 1936</td>
<td>853</td>
</tr>
<tr>
<td>9014008600</td>
<td>KUCINTCH</td>
<td>1631 Wbaby</td>
<td>Atascadero</td>
<td>CA</td>
<td>510-823-5</td>
<td>Apr 21, 1931</td>
<td>863</td>
</tr>
<tr>
<td>7714008603</td>
<td>LEBERMAN</td>
<td>3721 Gray</td>
<td>San Francisco</td>
<td>CA</td>
<td>415-668-9</td>
<td>Aug 11, 1932</td>
<td>872</td>
</tr>
<tr>
<td>6239000009</td>
<td>KIRCHN</td>
<td>1901 Sam</td>
<td>Concord</td>
<td>CA</td>
<td>925-685-2</td>
<td>Jul 15, 1941</td>
<td>624</td>
</tr>
</tbody>
</table>

Figure B-64: DB2 UDB for Multiplatforms - Sample contents results of nickname
Data models, table/file definitions, et al used in the DFC Customer Insight business solution

In this appendix, we document the data models of the operational systems’, data warehouse and CIIS, and table/file definitions, and population solutions used in the DFC business solution discussed in Chapter 4, “The Druid Financial Corporation (DFC) Customer Insight solution” on page 115.

The topics covered are:

- Operational systems data model and table/file definitions
- Data warehouse data model, table definitions, and population
- Data mart data model and table definitions
- CIIS data model, table definitions, and population
Introduction

Our DFC Customer Insight solution was meant to be represent a typical financial services organization offering multiple products via stovepipe operational systems. The objective was to apply IBM’s Patterns for e-business in implementing this solution with particular emphasis on Information Aggregation and Integration patterns.

Towards the end of showcasing IBM’s Patterns for e-business, we designed somewhat simple operational systems, a data warehouse, a data mart, and an operational data store (ODS). However, our aim was to faithfully represent the various processes that an organization would typically require to populate and access all the data repositories in the customer insight solution in order to highlight some of the commonly used Information Aggregation and Integration patterns.

In the following sections, we will briefly describe each of these systems and the processes used to populate them.

Operational systems’ details

We assumed the existence of four stovepipe operational systems as follows:

- Checkings/Savings
- Credit Card
- Rewards
- Brokerage
- Loans

A brief description of each of these systems follows.

Checkings/Savings

This is a typical banking checking and savings application hosted on an IBM AIX platform using an Oracle database. It supports the concept of multiple account holders within a master account, with each individual having different addresses as well as holding individual debit cards.

Figure C-1 on page 233 describes the data model of our checkings/savings operational system.
Example C-1 shows the DDL used to create the Oracle tables shown in Figure C-1.

Example C-1  Checkings/Savings DDL

```
CREATE TABLE ACCOUNT_MEMBER  (
  MASTER_ACT_NBR INTEGER NOT NULL,
  MEMBER_ACT_NBR INTEGER NOT NULL ,
  FIRST_NAME CHAR (40) NOT NULL ,
  MIDDLE_NAME CHAR (40) ,
  LAST_NAME CHAR (40) NOT NULL ,
  TITLE CHAR(5) ,
  DATE_OF_BIRTH DATE NOT NULL ,
  MAIL_ADR_ID INTEGER NOT NULL ,
  SSN CHAR(11) NOT NULL ,
  CREATED_BY CHAR(10) NOT NULL ,
  CREATED_DT DATE NOT NULL )
);

COMMENT ON COLUMN "ACCOUNT_MEMBER"."MASTER_ACT_NBR" IS 'MASTER ACCOUNT NUMBER';
COMMENT ON COLUMN "ACCOUNT_MEMBER"."MEMBER.ACT.NBR" IS 'MEMBER ACCOUNT NUMBER';
COMMENT ON COLUMN "ACCOUNT_MEMBER"."CREATED_BY" IS 'ID OF EMPLOYEE WHO CREATED ACCT';
COMMENT ON COLUMN "ACCOUNT_MEMBER"."CREATED_DT" IS 'DATE ACCOUNT CREATED';
```
COMMENT ON COLUMN "ACCOUNT_MEMBER"."TITLE" IS 'MR. MRS. DR.';
COMMENT ON COLUMN "ACCOUNT_MEMBER"."MAIL_ADR_ID" IS 'FOREIGN KEY TO ADDRESS TABLE';

-- DDL Statements for primary key on Table ACCOUNT_MEMBER

ALTER TABLE ACCOUNT_MEMBER
  ADD CONSTRAINT ACCOUNT_MEMBER_PK PRIMARY KEY
  (MEMBER_ACT_NBR);

-------------------------------------------------------------------------
CREATE TABLE ACCOUNT_MASTER
  (MASTER_ACT_NBR INTEGER NOT NULL ,
   PRIMARY_ACT_MBR INTEGER NOT NULL ,
   PRIMARY_ACT_NO INTEGER NOT NULL ,
   MAIL_ADR_ID INTEGER NOT NULL ,
   STATUS CHAR(1) NOT NULL ,
   CREATED_DT DATE NOT NULL ,
   CREATED_BY CHAR(10) NOT NULL )
;

COMMENT ON COLUMN "ACCOUNT_MASTER"."MASTER_ACT_NBR" IS 'MASTER ACCOUNT NUMBER';
COMMENT ON COLUMN "ACCOUNT_MASTER"."CREATED_BY" IS 'ID OF EMPLOYEE WHO CREATED ACCT';
COMMENT ON COLUMN "ACCOUNT_MASTER"."CREATED_DT" IS 'DATE ACCOUNT CREATED';
COMMENT ON COLUMN "ACCOUNT_MASTER"."STATUS" IS 'ACCOUNT ACTIVE/INACTIVE INDICATOR';
COMMENT ON COLUMN "ACCOUNT_MASTER"."PRIMARY_ACT_MBR" IS 'FOREIGN KEY TO MEMBER TABLE';
COMMENT ON COLUMN "ACCOUNT_MASTER"."PRIMARY_ACT_NO" IS 'FOREIGN KEY TO SAVINGS TABLE';
COMMENT ON COLUMN "ACCOUNT_MASTER"."MAIL_ADR_ID" IS 'FOREIGN KEY TO ADDRESS TABLE';

-- DDL Statements for primary key on Table ACCOUNT_MASTER

ALTER TABLE ACCOUNT_MASTER
  ADD CONSTRAINT ACCOUNT_MASTER_PK PRIMARY KEY
  (MASTER_ACT_NBR);

-----------------------------------------------------------------------
CREATE TABLE ACCOUNT
  (MASTER_ACT_NBR INTEGER NOT NULL ,
   ACT_NBR INTEGER NOT NULL ,
   PRODUCT_ID INTEGER NOT NULL ,
   BALANCE NUMBER(15,2) DEFAULT 0 NOT NULL ,
   LAST_TRANS_TS DATE NOT NULL ,
   CREATED_DT DATE NOT NULL ,
   CREATED_BY CHAR(10) NOT NULL)
COMMENT ON COLUMN "ACCOUNT"."PRODUCT_ID" IS 'FOREIGN KEY TO PRODUCT TAB FOR ACCOUNTS';
COMMENT ON COLUMN "ACCOUNT"."ACT_NBR" IS 'ACCOUNT NUMBER';
COMMENT ON COLUMN "ACCOUNT"."BALANCE" IS 'ACCOUNT BALANCE';
COMMENT ON COLUMN "ACCOUNT"."CREATED_BY" IS 'ID OF EMPLOYEE WHO CREATED ACCT';
COMMENT ON COLUMN "ACCOUNT"."CREATED_DT" IS 'DATE ACCOUNT CREATED';
COMMENT ON COLUMN "ACCOUNT"."LAST_TRANS_TS" IS 'DATE FOR LAST TRANSACTION';
COMMENT ON COLUMN "ACCOUNT"."MASTER_ACT_NBR" IS 'FOREIGN KEY TO ACCOUNT MASTER';

-- DDL Statements for primary key on Table "ACCOUNT"

ALTER TABLE ACCOUNT
ADD CONSTRAINT ACCOUNT_PK PRIMARY KEY
(ACT_NBR);

CREATE TABLE DEBIT
(
  DEBIT_CARD_NBR INTEGER NOT NULL ,
  MASTER_ACT_NBR INTEGER NOT NULL ,
  ASSOC_ACT_NBR INTEGER NOT NULL ,
  PRODUCT_ID INTEGER NOT NULL,
  STATUS CHAR(1) NOT NULL ,
  LAST_TRANS_TS DATE NOT NULL ,
  ISSUED_DT DATE NOT NULL ,
  PIN CHAR(12) NOT NULL ,
  CREATED_BY CHAR(10) NOT NULL ,
  CREATED_DT DATE NOT NULL)
;

COMMENT ON COLUMN "DEBIT"."DEBIT_CARD_NBR" IS 'DEBIT ACCOUNT NUMBER';
COMMENT ON COLUMN "DEBIT"."ASSOC_ACT_NBR" IS 'SUBACCOUNT NUMBER ASSOCIATED WITH THIS CARD';
COMMENT ON COLUMN "DEBIT"."CREATED_BY" IS 'ID OF EMPLOYEE WHO CREATED ACCT';
COMMENT ON COLUMN "DEBIT"."CREATED_DT" IS 'DATE ACCOUNT CREATED';
COMMENT ON COLUMN "DEBIT"."LAST_TRANS_TS" IS 'DATE FOR LAST TRANSACTION';
COMMENT ON COLUMN "DEBIT"."MASTER_ACT_NBR" IS 'FOREIGN KEY TO ACCOUNT MASTER';

-- DDL Statements for primary key on Table DEBIT

ALTER TABLE DEBIT
ADD CONSTRAINT DEBIT_PK PRIMARY KEY
(DEBIT_CARD_NBR);

CREATE TABLE PRODUCT
(
  PRODUCT_ID INTEGER NOT NULL ,
  DESCRIPTION CHAR (254) NOT NULL ,
...
MINIMUM_BAL NUMBER(15,2) DEFAULT 0 NOT NULL,
INTEREST_RATE NUMBER(7,4) DEFAULT 0 NOT NULL,
DURATION INTEGER,
TYPE CHAR(1))
;

COMMENT ON COLUMN "PRODUCT"."PRODUCT_ID" IS 'UNIQUE PRODUCT IDENTIFIER';
COMMENT ON COLUMN "PRODUCT"."DESCRIPTION" IS 'DESCRIPTIVE PRODUCT NAME';
COMMENT ON COLUMN "PRODUCT"."MINIMUM_BAL" IS 'MINIMUM BALANCE REQUIRED FOR THIS PRODUCT';
COMMENT ON COLUMN "PRODUCT"."INTEREST_RATE" IS 'INTEREST RATE FOR THIS PRODUCT';
COMMENT ON COLUMN "PRODUCT"."DURATION" IS 'LENGTH OF TIME FOR THIS PRODUCT IN MONTHS E.G. 12 MONTH CERTIFICATE';
COMMENT ON COLUMN "PRODUCT"."TYPE" IS 'CHECKING OR SAVING';

-- DDL Statements for primary key on Table CS_PRODS
ALTER TABLE PRODUCT
  ADD CONSTRAINT PRODUCT_PK PRIMARY KEY
  (PRODUCT_ID);

--------------------------------------------------------------------------------------------------------
CREATE TABLE ADDRESS
  (MAIL_ADR_ID INTEGER NOT NULL,
   ADDR_LINE1 CHAR(40) NOT NULL,
   ADDR_LINE2 CHAR(40),
   CITY CHAR(20) NOT NULL,
   STATE CHAR(2) NOT NULL,
   ZIP CHAR(10) NOT NULL,
   CREATED_DT DATE NOT NULL,
   CREATED_BY CHAR(10) NOT NULL)
;

COMMENT ON COLUMN "ADDRESS"."MAIL_ADR_ID" IS 'ADDRESS IDENTIFIER';
COMMENT ON COLUMN "ADDRESS"."ADDR_LINE1" IS 'STREET ADDRESS LINE 1';
COMMENT ON COLUMN "ADDRESS"."ADDR_LINE2" IS 'OPTIONAL STREET ADDRESS LINE 2';
COMMENT ON COLUMN "ADDRESS"."CITY" IS 'CITY';
COMMENT ON COLUMN "ADDRESS"."STATE" IS 'STATE CODE';
COMMENT ON COLUMN "ADDRESS"."ZIP" IS '9 DIGIT ZIP CODE (XXXXX-YYYY)';
COMMENT ON COLUMN "ADDRESS"."CREATED_BY" IS 'ID OF EMPLOYEE WHO CREATED ACCT';
COMMENT ON COLUMN "ADDRESS"."CREATED_DT" IS 'DATE ACCOUNT CREATED';

-- DDL Statements for primary key on Table ADDRESS
ALTER TABLE ADDRESS
  ADD CONSTRAINT ADDRESS_PK PRIMARY KEY
  (MAIL_ADR_ID);
CREATE TABLE TRANSACTION(
    MASTER_ACC_NBR INTEGER NOT NULL,
    ACT_NBR INTEGER NOT NULL,
    DEBIT_CARD_NBR INTEGER NOT NULL,
    MEMBER_NUMBER INTEGER,
    TYPE_OF_TRANS CHAR(10) NOT NULL,
    VENDOR CHAR(30),
    LOCATION CHAR(40) NOT NULL,
    CHECK_NO INTEGER,
    AMOUNT NUMBER(15,2) NOT NULL,
    TRANS_CHANNEL CHAR(10) NOT NULL,
    TRANSACTION_TS DATE NOT NULL
);

COMMENT ON COLUMN "TRANSACTION"."MASTER_ACC_NBR" IS 'ACCOUNT FOR TRANSACTION';
COMMENT ON COLUMN "TRANSACTION"."MEMBER_NUMBER" IS 'MEMBER WHO INITIATED TRANSACTION';
COMMENT ON COLUMN "TRANSACTION"."TYPE_OF_TRANS" IS 'CREDIT/DEBIT';
COMMENT ON COLUMN "TRANSACTION"."VENDOR" IS 'VENDOR NAME WHERE TRANSACTION OCCURRED';
COMMENT ON COLUMN "TRANSACTION"."LOCATION" IS 'WHERE TRANSACTION OCCURRED';
COMMENT ON COLUMN "TRANSACTION"."CHECK_NO" IS 'CHECK NUMBER IF A CHECK';
COMMENT ON COLUMN "TRANSACTION"."AMOUNT" IS 'VALUE OF TRANSACTION';
COMMENT ON COLUMN "TRANSACTION"."TRANS_CHANNEL" IS 'CHECK, ATM, BANK BRANCH';

-- DDL Statements for primary key on Table TRANSACTION

ALTER TABLE TRANSACTION
    ADD CONSTRAINT TRANSACTION_PK PRIMARY KEY
    (MASTER_ACC_NBR , TRANSACTION_TS);

-- Add Foreign key

-- DDL Statements for foreign key on Table ACCOUNT_MEMBER

--ALTER TABLE ACCOUNT_MEMBER
--ADD CONSTRAINT AC_MEM_ACT_FK FOREIGN KEY (MASTER_ACT_NBR)
--REFERENCES ACCOUNT_MASTER (MASTER_Act_NBR);
--ALTER TABLE ACCOUNT_MEMBER
--ADD CONSTRAINT AC_MEM_ADR_FK FOREIGN KEY (MAIL_ADR_ID)
--REFERENCES ADDRESS (MAIL_ADR_ID);
--
--DDL Statements for foreign key on Table ACCOUNT_MASTER

--ALTER TABLE ACCOUNT_MASTER
--ADD CONSTRAINT AC_MAS_MEM_FK FOREIGN KEY (PRIMARY_ACT_MBR)
--REFERENCES ACCOUNT_MEMBER (MEMBER_ACT_NBR);
--ALTER TABLE ACCOUNT_MASTER

Appendix C. Data models, table/file definitions, et al used in the DFC Customer Insight business solution
-- ADD CONSTRAINT AC_MAS_ACT_FK FOREIGN KEY (PRIMARY_ACT_NO)
-- REFERENCES ACCOUNT (ACT_NBR);
--ALTER TABLE ACCOUNT_MASTER
-- ADD CONSTRAINT AC_MAS_ADR_FK FOREIGN KEY (MAIL_ADR_ID)
-- REFERENCES ADDRESS (MAIL_ADR_ID);

-- DDL Statements for foreign key on Table "ACCOUNT"

--ALTER TABLE ACCOUNT
-- ADD CONSTRAINT AC_ACT_NBR_FK FOREIGN KEY (MASTER_ACT_NBR)
-- REFERENCES ACCOUNT_MASTER (MASTER_ACT_NBR);
--ALTER TABLE ACCOUNT
-- ADD CONSTRAINT AC_PRD_FK FOREIGN KEY (PRODUCT_ID)
-- REFERENCES PRODUCT (PRODUCT_ID);

-- DDL Statements for foreign key on Table DEBIT

--ALTER TABLE DEBIT
-- ADD CONSTRAINT DEB_ACT_NBR_FK FOREIGN KEY (ASSOC_ACT_NBR)
-- REFERENCES ACCOUNT (ACT_NBR);
--ALTER TABLE DEBIT
-- ADD CONSTRAINT DEB_MAST_ACT_FK FOREIGN KEY (MASTER_ACT_NBR)
-- REFERENCES ACCOUNT_MASTER (MASTER_ACT_NBR);
--ALTER TABLE DEBIT
-- ADD CONSTRAINT DEB_PRD_FK FOREIGN KEY (PRODUCT_ID)
-- REFERENCES PRODUCT (PRODUCT_ID);

-- DDL Statements for foreign key on Table TRANSACTION

--ALTER TABLE TRANSACTION
-- ADD CONSTRAINT TRAN_MAST_ACT_FK FOREIGN KEY (MASTER_ACC_NBR)
-- REFERENCES ACCOUNT_MASTER (MASTER_ACT_NBR);
--ALTER TABLE TRANSACTION
-- ADD CONSTRAINT TRAN_ACT_FK FOREIGN KEY (ACT_NBR)
-- REFERENCES ACCOUNT (ACT_NBR);
--ALTER TABLE TRANSACTION
-- ADD CONSTRAINT TRAN_MEM_FK FOREIGN KEY (MEMBER_NUMBER)
-- REFERENCES ACCOUNT_MEMBER (MEMBER_ACT_NBR);
--ALTER TABLE TRANSACTION
-- ADD CONSTRAINT TRAN_DEBIT_FK FOREIGN KEY (DEBIT_CARD_NBR)
-- REFERENCES DEBIT (DEBIT_CARD_NBR);

-- DDL Statements for foreign key on Table ACCOUNT_MEMBER_RELATIONSHIP

-- Add Foreign key

-- DDL Statements for foreign key on Table ACCOUNT_MEMBER

ALTER TABLE ACCOUNT_MEMBER
ADD CONSTRAINT AC_MEM_ACT_FK FOREIGN KEY (MASTER_ACT_NBR) REFERENCES ACCOUNT_MASTER (MASTER_ACT_NBR);
ALTER TABLE ACCOUNT_MEMBER
ADD CONSTRAINT AC_MEM_ADR_FK FOREIGN KEY (MAIL_ADR_ID) REFERENCES ADDRESS (MAIL_ADR_ID);

-- DDL Statements for foreign key on Table ACCOUNT_MASTER
ALTER TABLE ACCOUNT_MASTER
ADD CONSTRAINT AC_MAS_MEM_FK FOREIGN KEY (PRIMARY_ACT_MBR) REFERENCES ACCOUNT_MEMBER (MEMBER_ACT_NBR);
ALTER TABLE ACCOUNT_MASTER
ADD CONSTRAINT AC_MAS_ACT_FK FOREIGN KEY (PRIMARY_ACT_NO) REFERENCES ACCOUNT (ACT_NBR);
ALTER TABLE ACCOUNT_MASTER
ADD CONSTRAINT AC_MAS_ADR_FK FOREIGN KEY (MAIL_ADR_ID) REFERENCES ADDRESS (MAIL_ADR_ID);

-- DDL Statements for foreign key on Table "ACCOUNT"
ALTER TABLE ACCOUNT
ADD CONSTRAINT AC_ACT_NBR_FK FOREIGN KEY (MASTER_ACT_NBR) REFERENCES ACCOUNT_MASTER (MASTER_ACT_NBR);
ALTER TABLE ACCOUNT
ADD CONSTRAINT AC_PRD_FK FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT (PRODUCT_ID);

-- DDL Statements for foreign key on Table DEBIT
ALTER TABLE DEBIT
ADD CONSTRAINT DEB_ACT_NBR_FK FOREIGN KEY (ASSOC_ACT_NBR) REFERENCES ACCOUNT (ACT_NBR);
ALTER TABLE DEBIT
ADD CONSTRAINT DEB_MAST_ACT_FK FOREIGN KEY (MASTER_ACT_NBR) REFERENCES ACCOUNT_MASTER (MASTER_ACT_NBR);
ALTER TABLE DEBIT
ADD CONSTRAINT DEB_PRD_FK FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT (PRODUCT_ID);

-- DDL Statements for foreign key on Table TRANSACTION
ALTER TABLE TRANSACTION
ADD CONSTRAINT TRAN_MAST_ACT_FK FOREIGN KEY (MASTER_ACC_NBR) REFERENCES ACCOUNT_MASTER (MASTER_ACT_NBR);
ALTER TABLE TRANSACTION
ADD CONSTRAINT TRAN_ACT_FK FOREIGN KEY (ACT_NBR) REFERENCES ACCOUNT (ACT_NBR);
ALTER TABLE TRANSACTION
ADD CONSTRAINT TRAN_MEM_FK FOREIGN KEY (MEMBER_NUMBER)
We loaded some representative data in these tables, which is not included here. Note that the data in operational systems is maintained by its business transactions.

**Credit Card**

This is a typical credit card application hosted on an IBM z/OS platform using a DB2 for z/OS database. Here again, it supports the concept of multiple credit card holders within a master credit account. Each credit card corresponds to a particular product type such as VISA or MASTERCARD and its associated details such as interest rate and annual fee.

Figure C-2 describes the data model of our credit card operational system.

*Figure C-2  Credit Card operational system data model*
Example C-2 shows the DDL used to create the DB2 for z/OS tables shown in Figure C-2 on page 240.

**Example: C-2  Credit Card DDL**

```sql
create table card_holder (
    cardholder_id integer not null,
    first_name char(30) not null,
    last_name char(30) not null,
    street_adr varchar(70),
    city varchar(30),
    state char(2),
    phone char(12) not null,
    created_dt date not null,
    created_by char(10) not null,
    primary key (cardholder_id)
) in userspace1;
```

```sql
create table credit_card (
    cc_nbr  bigint not null,
    cardholder_id integer not null
    references card_holder(cardholder_id),
    prod_id integer not null
    references product (prod_id),
    status char(10),
    activation_dt date,
    expiry_dt date,
    cvv2 smallint,
    created_dt date not null,
    created_by char(10) not null,
    primary key (cc_nbr, cardholder_id)
) in userspace1;
```

```sql
create table credit_account (
    cc_nbr bigint not null,
    p_cardholder_id integer not null,
    s_cardholder_id integer,
    category char(10) not null,
    credit_limit decimal(10,2) not null,
    status char(1),
    credit_his_ind char(1),
    bal_owed decimal(5,2) not null,
    last_trans_ts timestamp not null,
    created_dt date not null,
    create_by char(10) not null,
    primary key (cc_nbr)
) in userspace1;
```
alter table credit_account
  add constraint cc_acc_fk1 foreign key (cc_nbr, p_cardholder_id)
    references credit_card (cc_nbr, cardholder_id);
alter table credit_account
  add constraint cc_acc_fk2 foreign key (cc_nbr, s_cardholder_id)
    references credit_card (cc_nbr, cardholder_id);

create table product (  
  prod_id integer not null,
  prod_desc char(30),
  type char(20) not null,
  purchase_rate decimal(3,1),
  cash_adv_rate decimal(3,1),
  rewards char(1),
  annual_feestatus decimal(3),
  created_dt date not null,
  created_by char(10) not null,
  primary key (prod_id)
) in userspace1;

create table transaction (  
  cc_nbr bigint not null,
  cardholder_id integer not null,
  trans_category char(10),
  trans_type char(10),
  vendor char(10),
  amount decimal(15,2) not null,
  reference char(20),
  last_trans_ts timestamp not null,
  created_by char(10),
  primary key (cc_nbr, cardholder_id, last_trans_ts)
) in userspace1;

alter table transaction
  ADD CONSTRAINT CC_TRAN_FK FOREIGN KEY (cc_nbr, cardholder_id)
    REFERENCES credit_card (cc_nbr, cardholder_id);

We loaded some representative data in these tables, which is not included here.  
Note that the data in operational systems is maintained by its business 
transactions.

Rewards

This system is associated with the credit card system and supports the 
accumulation of hotel points and airline miles with partner companies for
purchases made with the credit card. This is a rewards application hosted on an IBM z/OS platform using VSAM files. There are four VSAM KSDS files associated with the rewards system, namely, Tracking, Offering Partners, Transactions, and a Journal that maintained an audit trail of all changes occurring in the application.

Note: There is also a rewards lookup XML file stored on an IBM AIX platform that is part of this rewards application. This scenario was contrived to demonstrate DB2 Information Integrator’s XML access capabilities, and does not reflect a real-world environment.

Figure C-3 through Figure C-6 on page 245 describe the fields defined in the various VSAM files, while Example C-3 on page 245 describes the rewards lookup XML file.

<table>
<thead>
<tr>
<th>FIELD</th>
<th>OFFSET</th>
<th>LENGTH</th>
<th>DATA TYPE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCNUM</td>
<td>0</td>
<td>16</td>
<td>PIC</td>
<td>CREDIT CARD NUMBER</td>
</tr>
<tr>
<td>LAST_NAME</td>
<td>16</td>
<td>30</td>
<td>CHAR</td>
<td>LAST NAME</td>
</tr>
<tr>
<td>FIRST_NAME</td>
<td>46</td>
<td>30</td>
<td>CHAR</td>
<td>FIRST NAME</td>
</tr>
<tr>
<td>PARTNER_ID_NUM</td>
<td>76</td>
<td>12</td>
<td>CHAR</td>
<td>AIRLINE FREQUENT FLYER NUMBER</td>
</tr>
<tr>
<td>REWARDID</td>
<td>88</td>
<td>10</td>
<td>PIC</td>
<td>REWARD PRODUCT ID CODE</td>
</tr>
<tr>
<td>POINT_BALANCE</td>
<td>98</td>
<td>8</td>
<td>PIC</td>
<td>POINTS BALANCE AS OF LAST MONTHLY PROCESSING</td>
</tr>
<tr>
<td>PROCESS_DATE</td>
<td>106</td>
<td>10</td>
<td>CHAR</td>
<td>DATE OF LAST MONTHLY PROCESSING</td>
</tr>
<tr>
<td>CREATED_DATE</td>
<td>116</td>
<td>10</td>
<td>CHAR</td>
<td>DATE MM-DD-YYYY</td>
</tr>
<tr>
<td>CREATED_BY</td>
<td>126</td>
<td>10</td>
<td>CHAR</td>
<td>USERID OF PERSON CREATING ENTRY</td>
</tr>
</tbody>
</table>

Figure C-3   Rewards Tracking VSAM file field attributes
### Figure C-4  Rewards Offering Partners VSAM file field attributes

<table>
<thead>
<tr>
<th>FIELD</th>
<th>OFFSET</th>
<th>LENGTH</th>
<th>DATA TYPE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REWARDID</td>
<td>0</td>
<td>10</td>
<td>PIC</td>
<td>REWARD PRODUCT ID CODE</td>
</tr>
<tr>
<td>PARTNER_NAME</td>
<td>10</td>
<td>30</td>
<td>CHAR</td>
<td>NAME OF REWARDS PARTNER</td>
</tr>
<tr>
<td>CLASSIFICATION</td>
<td>40</td>
<td>10</td>
<td>CHAR</td>
<td>AIRLINE, HOTEL</td>
</tr>
<tr>
<td>ACCUMULATION_TYPE</td>
<td>50</td>
<td>6</td>
<td>CHAR</td>
<td>POINTS, MILES</td>
</tr>
<tr>
<td>EFFECTIVE_FROM</td>
<td>56</td>
<td>10</td>
<td>CHAR</td>
<td>DATE THIS REWARD IS AVAILABLE FOR USE (MM-DD-YYYY)</td>
</tr>
<tr>
<td>EFFECTIVE_TO</td>
<td>66</td>
<td>10</td>
<td>CHAR</td>
<td>DATE THIS REWARD IS WITHDRAWN - 99-99-9999 IS DEFAULT</td>
</tr>
<tr>
<td>CREATED_DATE</td>
<td>76</td>
<td>10</td>
<td>CHAR</td>
<td>DATE MM-DD-YYYY</td>
</tr>
<tr>
<td>CREATED_BY</td>
<td>86</td>
<td>10</td>
<td>N</td>
<td>USERID OF PERSON CREATING ENTRY</td>
</tr>
</tbody>
</table>

### Figure C-5  Rewards Transactions VSAM file field attributes

<table>
<thead>
<tr>
<th>FIELD</th>
<th>OFFSET</th>
<th>LENGTH</th>
<th>DATA TYPE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCNUM</td>
<td>0</td>
<td>16</td>
<td>PIC</td>
<td>CREDIT CARD NUMBER</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>16</td>
<td>26</td>
<td>CHAR</td>
<td>DB2 FORMAT TIMESTAMP YYYYY-MM-DD HH:MM:SS.TTTTTT</td>
</tr>
<tr>
<td>AMOUNT</td>
<td>42</td>
<td>10</td>
<td>PIC</td>
<td>POINTS ASSOCIATED WITH TRANSACTION</td>
</tr>
<tr>
<td>TRAN_TYPE</td>
<td>52</td>
<td>1</td>
<td>CHAR</td>
<td>TRANSACTION TYPE - A (ADD), D (DELETE)</td>
</tr>
<tr>
<td>REASON_CODE</td>
<td>53</td>
<td>4</td>
<td>CHAR</td>
<td>PURC (PURCHASE), RETN (RETURN), REST (RESTORE - ADD BACK)</td>
</tr>
<tr>
<td>REWARD_ID</td>
<td>57</td>
<td>10</td>
<td>PIC</td>
<td>REWARD IDENTIFIER</td>
</tr>
<tr>
<td>SEQ</td>
<td>67</td>
<td>8</td>
<td>PIC</td>
<td>PROCESSING SEQUENCE NUMBER</td>
</tr>
</tbody>
</table>
**Example: C-3  Rewards Lookup XML file**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com) by Bill Mathews (IBM) -->
<!--Druid Bank Credit Card Rewards-->
<Reward xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\MyShares\RewardsXML\Rewards.xsd">
  <id>100001</id>
  <Short_Name>Travel2004</Short_Name>
  <Description>Druid Bank Credit Card Travel Purchase Awards</Description>
  <Sponsor>DruidBankTravel.com</Sponsor>
  <Card_Type>VISA</Card_Type>
  <Affiliation>
    <Name>We Get You There Airlines</Name>
    <Points>3</Points>
    <per_unit>1</per_unit>
    <unit_type>Dollar</unit_type>
    <Marketing_Msg>Use your Druid Bank Credit Card to purchase your "We Get You There Airlines" tickets at DruidBankTravel.com and receive 3 points for every dollar spent!</Marketing_Msg>
  </Affiliation>
</Reward>
```

---

**Figure C-6  Rewards Journal VSAM file field attributes**

<table>
<thead>
<tr>
<th>FIELD</th>
<th>OFFSET</th>
<th>LENGTH</th>
<th>DATA TYPE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECORD_TYPE</td>
<td>0</td>
<td>1</td>
<td>CHAR</td>
<td>&quot;C&quot; (CREATE/NEW), &quot;U&quot; (UPDATE), &quot;D&quot; (DELETE), &quot;H&quot; (HISTORICAL/PRIOR VIEW)</td>
</tr>
<tr>
<td>CHANGE_SEQ</td>
<td>1</td>
<td>2</td>
<td>PIC</td>
<td>SEQUENCE NUMBER FOR GROUPED CHANGES, E.G.</td>
</tr>
<tr>
<td>CHANGE_TS</td>
<td>3</td>
<td>26</td>
<td>CHAR</td>
<td>TIMESTAMP FOR CHANGE</td>
</tr>
<tr>
<td>CHANGE_SOURCE</td>
<td>29</td>
<td>1</td>
<td>CHAR</td>
<td>&quot;S&quot; (SYSTEM GENERATED), &quot;R&quot; (CSR), &quot;C&quot; (CUSTOMER)</td>
</tr>
<tr>
<td>INITIATOR</td>
<td>30</td>
<td>10</td>
<td>PIC</td>
<td>IDENTIFICATION NUMBER (CSR SERIAL NUMBER OR CUSTOMER ID) FOR PERSON MAKING CHANGE</td>
</tr>
<tr>
<td>CCNUM</td>
<td>40</td>
<td>16</td>
<td>PIC</td>
<td>CREDIT CARD NUMBER</td>
</tr>
<tr>
<td>LAST_NAME</td>
<td>56</td>
<td>30</td>
<td>CHAR</td>
<td>LAST NAME</td>
</tr>
<tr>
<td>FIRST_NAME</td>
<td>86</td>
<td>30</td>
<td>CHAR</td>
<td>FIRST NAME</td>
</tr>
<tr>
<td>PARTNER_ID_NUM</td>
<td>116</td>
<td>12</td>
<td>CHAR</td>
<td>AIRLINE FREQUENT FLYER NUMBER</td>
</tr>
<tr>
<td>REWARDID</td>
<td>128</td>
<td>10</td>
<td>PIC</td>
<td>REWARD PRODUCT ID CODE</td>
</tr>
<tr>
<td>POINTBALANCE</td>
<td>138</td>
<td>8</td>
<td>PIC</td>
<td>POINTS BALANCE AS OF LAST MONTHLY PROCESSING</td>
</tr>
<tr>
<td>PROCESS_DATE</td>
<td>148</td>
<td>10</td>
<td>CHAR</td>
<td>DATE OF LAST MONTHLY PROCESSING</td>
</tr>
<tr>
<td>CREATED_DATE</td>
<td>156</td>
<td>10</td>
<td>CHAR</td>
<td>DATE MM-DD-YYYY</td>
</tr>
<tr>
<td>CREATED_BY</td>
<td>166</td>
<td>10</td>
<td>CHAR</td>
<td>USERID OF PERSON CREATING ENTRY</td>
</tr>
</tbody>
</table>

Appendix C. Data models, table/file definitions, et al used in the DFC Customer Insight business solution
We loaded some representative data in these files, which is not included here. Note that the data in operational systems is maintained by its business transactions.

**Brokerage**

This is a brokerage application hosted on an IBM AIX platform using a DB2 UDB for Multiplatforms database. It supports an organization offering multiple products, accounts with multiple owners, and portfolios with multiple securities.

Figure C-7 on page 246 describes the data model of our brokerage operational system.

![Brokerage operational system data model](image)

Example C-4 shows the DDL used to create the DB2 UDB for Multiplatforms tables shown in Figure C-7.

**Example: C-4  Brokerage DDL**

```sql
CREATE TABLE "DB2BRK"."ACCOUNT" (
  "ACCOUNT_NBR" BIGINT NOT NULL,
  "OWNER_ID" BIGINT NOT NULL,
  "PRODUCT_ID" BIGINT NOT NULL,
  "ACT_ACTIVE_DATE" DATE NOT NULL,
  "BUYING_POWER" DECIMAL(15,2),
  "STATUS" CHAR(1) NOT NULL,
  "CREATED_DT" DATE NOT NULL,
) ENGINE=INNODB;
```
"CREATED_BY" CHAR(10) NOT NULL,
"LASTTRANS_TS" TIMESTAMP NOT NULL)

DATA CAPTURE CHANGES IN "USERSPACE1";

COMMENT ON COLUMN "DB2BRK "."ACCOUNT"."ACCOUNT_NBR" IS 'ACCOUNT NUMBER';

COMMENT ON COLUMN "DB2BRK "."ACCOUNT"."BUYING_POWER" IS 'AMOUNT AVAILABLE TO BUY ON MARGIN';

COMMENT ON COLUMN "DB2BRK "."ACCOUNT"."CREATED_BY" IS 'ID OF EMPLOYEE WHO CREATED ACCT';

COMMENT ON COLUMN "DB2BRK "."ACCOUNT"."CREATED_DT" IS 'DATE ACCOUNT CREATED';

COMMENT ON COLUMN "DB2BRK "."ACCOUNT"."OWNER_ID" IS 'FOREIGN KEY TO OWNER TABLE';

COMMENT ON COLUMN "DB2BRK "."ACCOUNT"."PRODUCT_ID" IS 'FOREIGN KEY TO PRODUCT TABLE';

COMMENT ON COLUMN "DB2BRK "."ACCOUNT"."STATUS" IS 'ACCOUNT ACTIVE/INACTIVE INDICATOR';

-- DDL Statements for primary key on Table "DB2BRK "."ACCOUNT"

ALTER TABLE "DB2BRK "."ACCOUNT"
ADD CONSTRAINT "ACCOUNT_PK" PRIMARY KEY
("ACCOUNT_NBR");

CREATE TABLE "DB2BRK "."PRODUCT" (  
"PRODUCT_ID" BIGINT NOT NULL,  
"PRODUCT_DESC" CHAR(254) NOT NULL,  
"PRODUCT_STATUS" CHAR(1) NOT NULL,  
"ANNUAL_MAINT_FEE" DECIMAL(15,2) NOT NULL,  
"PER_TRADE_FEE" DECIMAL(15,2) NOT NULL,  
"MINIMUM_UNITS" INTEGER NOT NULL WITH DEFAULT 0,  
"CREATED_DT" DATE NOT NULL,  
"CREATED_BY" CHAR(10) NOT NULL )

DATA CAPTURE CHANGES IN "USERSPACE1";

COMMENT ON COLUMN "DB2BRK "."PRODUCT"."ANNUAL_MAINT_FEE" IS 'ANNUAL FEE FOR THIS ACCOUNT TYPE';

COMMENT ON COLUMN "DB2BRK "."PRODUCT"."CREATED_BY" IS 'ID OF EMPLOYEE WHO CREATED PROD';

COMMENT ON COLUMN "DB2BRK "."PRODUCT"."CREATED_DT" IS 'DATE PROD CREATED';
COMMENT ON COLUMN "DB2BRK "."PRODUCT"."MINIMUM_UNITS" IS 'MINIMUM QUANTITY THAT MUST BE PURCHASED, E.G. 100 SHARES';

COMMENT ON COLUMN "DB2BRK "."PRODUCT"."PER_TRADE_FEE" IS 'FEE FOR EACH TRADE MADE';

COMMENT ON COLUMN "DB2BRK "."PRODUCT"."PRODUCT_DESC" IS 'PRODUCT DESCRIPTION';

COMMENT ON COLUMN "DB2BRK "."PRODUCT"."PRODUCT_ID" IS 'PRODUCT ID NUMBER';

-- DDL Statements for primary key on Table "DB2BRK "."PRODUCT"
ALTER TABLE "DB2BRK "."PRODUCT"
ADD CONSTRAINT "PRODUCT_PK" PRIMARY KEY
("PRODUCT_ID");

-------------------------------------------------------------------------
CREATE TABLE "DB2BRK "."OWNER"  (
"OWNER_ID" BIGINT NOT NULL ,
"FIRST_NAME" CHAR(20) NOT NULL ,
"MIDDLE_INITIAL" CHAR(1) ,
"LAST_NAME" CHAR(30) NOT NULL ,
"ADDRESS_LINE1" CHAR(30) NOT NULL ,
"ADDRESS_LINE2" CHAR(30) ,
"CITY" CHAR(30) NOT NULL ,
"STATE" CHAR(2) NOT NULL ,
"ZIPCODE" CHAR(5) NOT NULL ,
"DOB" DATE NOT NULL ,
"SSN" CHAR(9) NOT NULL ,
"PHOME" CHAR(12) ,
"CITIZEN" CHAR(1) NOT NULL ,
"CREATE_DT" DATE NOT NULL ,
"CREATED_BY" CHAR(10) NOT NULL ,
"REGION_ID" SMALLINT NOT NULL WITH DEFAULT -1 )
DATA CAPTURE CHANGES IN "USERSPACE1" ;

-- DDL Statements for primary key on Table "DB2BRK "."OWNER"
ALTER TABLE "DB2BRK "."OWNER"
ADD CONSTRAINT "OWNER_PK" PRIMARY KEY
("OWNER_ID");

-------------------------------------------------------------------------
CREATE TABLE "DB2BRK "."PORTFOLIO" ( 
"PORTFOLIO_ID" BIGINT NOT NULL ,
"ACCOUNT_ID" BIGINT NOT NULL ,
"SECURITY_ID" BIGINT NOT NULL ,
"NUM_OF_UNITS" DECIMAL(15,2),
"CREATED_DT" DATE NOT NULL,
"CREATED_BY" CHAR(10) NOT NULL,
"PRICE_PER_UNIT" DECIMAL(10,2)
)

DATA CAPTURE CHANGES
IN "USERSPACE1";

COMMENT ON COLUMN "DB2BRK "."PORTFOLIO"."ACCOUNT_ID" IS 'ACCOUNT NUMBER THAT OWNS THIS PORTFOLIO (FOREIGN KEY TO ACCOUNT TABLE)';

COMMENT ON COLUMN "DB2BRK "."PORTFOLIO"."CREATED_BY" IS 'ID OF EMPLOYEE WHO CREATED ACCT';

COMMENT ON COLUMN "DB2BRK "."PORTFOLIO"."CREATED_DT" IS 'DATE ACCOUNT CREATED';

COMMENT ON COLUMN "DB2BRK "."PORTFOLIO"."NUM_OF_UNITS" IS 'NUMBER OF UNITS OF THE SECURITY HELD IN PORTFOLIO';

COMMENT ON COLUMN "DB2BRK "."PORTFOLIO"."PORTFOLIO_ID" IS 'ID FOR THIS SECURITY HOLDING (PORTFOLIO)';

COMMENT ON COLUMN "DB2BRK "."PORTFOLIO"."SECURITY_ID" IS 'UNIQUE SECURITY HELD IN THIS PORTFOLIO (FOREIGN KEY TO SECURITIES TABLE)';

-- DDL Statements for primary key on Table "DB2BRK "."PORTFOLIO"

ALTER TABLE "DB2BRK "."PORTFOLIO"
ADD CONSTRAINT "PORTFOLIO_PK" PRIMARY KEY
("PORTFOLIO_ID");

CREATE TABLE "DB2BRK "."SECURITIES" (
"SECURITY_ID" BIGINT NOT NULL,
"SYMBOL" CHAR(8) NOT NULL,
"TYPE" CHAR(8) NOT NULL,
"STATUS" CHAR(1) NOT NULL,
"CREATED_DT" DATE NOT NULL,
"CREATED_BY" CHAR(10) NOT NULL)

DATA CAPTURE CHANGES
IN "USERSPACE1";

COMMENT ON COLUMN "DB2BRK "."SECURITIES"."CREATED_BY" IS 'ID OF EMPLOYEE WHO CREATED ACCT';

COMMENT ON COLUMN "DB2BRK "."SECURITIES"."CREATED_DT" IS 'DATE ACCOUNT CREATED';

COMMENT ON COLUMN "DB2BRK "."SECURITIES"."SECURITY_ID" IS 'SECURITY IDENTIFIER';
COMMENT ON COLUMN "DB2BRK "."SECURITIES"."STATUS" IS 'AVAILABLE FOR PURCHASE';

COMMENT ON COLUMN "DB2BRK "."SECURITIES"."SYMBOL" IS 'MARKET SYMBOL FOR SECURITY E.G. IBM, VZ, MSFT';

COMMENT ON COLUMN "DB2BRK "."SECURITIES"."TYPE" IS 'STOCK, BOND, FUND';

-- DDL Statements for primary key on Table "DB2BRK "."SECURITIES"

ALTER TABLE "DB2BRK "."SECURITIES"
ADD CONSTRAINT "SECURITIES_PK" PRIMARY KEY
("SECURITY_ID")
;

CREATE TABLE "DB2BRK "."TRANSACTION" (  
"ACCOUNT_ID" BIGINT NOT NULL ,  
"SECURITY_ID" BIGINT NOT NULL ,  
"TYPE_OF_TRANS" CHAR(10) NOT NULL ,  
"PRICE" DECIMAL(15,3) ,  
"UNITS" DECIMAL(15,3) NOT NULL ,  
"TRANSACTION_FEE" DECIMAL(15,2) NOT NULL ,  
"MARGIN" CHAR(1) ,  
"TRANSACTION_TS" TIMESTAMP NOT NULL )
DATA CAPTURE CHANGES
IN "USERSPACE1" ;

COMMENT ON COLUMN "DB2BRK "."TRANSACTION"."ACCOUNT_ID" IS 'ACCOUNT FOR TRANSACTION';

COMMENT ON COLUMN "DB2BRK "."TRANSACTION"."MARGIN" IS 'PURCHASE IS ON MARGIN';

COMMENT ON COLUMN "DB2BRK "."TRANSACTION"."PRICE" IS 'TOTAL VALUE/COST OF SALE/BUY' ;

COMMENT ON COLUMN "DB2BRK "."TRANSACTION"."SECURITY_ID" IS 'SECURITY PURCHASED/SOLD';

COMMENT ON COLUMN "DB2BRK "."TRANSACTION"."TRANSACTION_FEE" IS 'TRANSACTION FEE';

COMMENT ON COLUMN "DB2BRK "."TRANSACTION"."TYPE_OF_TRANS" IS 'BUY/SELL';

COMMENT ON COLUMN "DB2BRK "."TRANSACTION"."UNITS" IS 'QUANTITY BOUGHT/SOLD';

-- DDL Statements for primary key on Table "DB2BRK "."TRANSACTION"

ALTER TABLE "DB2BRK "."TRANSACTION"
ADD CONSTRAINT "TRANSACTION_PK" PRIMARY KEY
("ACCOUNT_ID" ,
"SECURITY_ID",
"TRANSACTION_TS";

-- DDL Statements for foreign keys on Table "DB2BRK "."ACCOUNT"

ALTER TABLE "DB2BRK "."ACCOUNT"
ADD CONSTRAINT "CC1079732465401" FOREIGN KEY
("OWNER_ID"
REFERENCES "DB2BRK "."OWNER"
("OWNER_ID")
ON DELETE NO ACTION
ON UPDATE NO ACTION
ENFORCED
ENABLE QUERY OPTIMIZATION;

-- DDL Statements for foreign keys on Table "DB2BRK "."PORTFOLIO"

ALTER TABLE "DB2BRK "."PORTFOLIO"
ADD CONSTRAINT "SQL040225102410080" FOREIGN KEY
("ACCOUNT_ID"
REFERENCES "DB2BRK "."ACCOUNT"
("ACCOUNT_NBR")
ON DELETE NO ACTION
ON UPDATE NO ACTION
ENFORCED
ENABLE QUERY OPTIMIZATION;

-- DDL Statements for foreign keys on Table "DB2BRK "."TRANSACTION"

ALTER TABLE "DB2BRK "."TRANSACTION"
ADD CONSTRAINT "SQL040225102411010" FOREIGN KEY
("ACCOUNT_ID"
REFERENCES "DB2BRK "."ACCOUNT"
("ACCOUNT_NBR")
ON DELETE NO ACTION
ON UPDATE NO ACTION
ENFORCED
ENABLE QUERY OPTIMIZATION;

ALTER TABLE "DB2BRK "."TRANSACTION"
ADD CONSTRAINT "SQL040225102411011" FOREIGN KEY
("SECURITY_ID"
REFERENCES "DB2BRK "."SECURITIES"
("SECURITY_ID")
ON DELETE NO ACTION
ON UPDATE NO ACTION
ENFORCED
We loaded some representative data in these tables, which is not included here. Note that the data in operational systems is maintained by its business transactions.

Loans

This is a loans application hosted on an IBM AIX platform using a DB2 UDB for Multiplatforms database. It supports the concept of different types of loans for a borrower and records their transaction history.

Figure C-8 describes the data model of our loans operational system.

---

**Example C-5** Loans DDL

```sql
CREATE TABLE "DB2LOAN "."LOAN" (  "ID" BIGINT NOT NULL ,  "STATE" CHAR(12) NOT NULL ,  "AMOUNT" DECIMAL(15,2) NOT NULL ,  "INT_RATE" DECIMAL(6,3) NOT NULL ,  "NUM_INSTALLMENT" INTEGER NOT NULL ,  "BORROWED_DT" DATE ,  "MONTHLY_PAYMENT" DECIMAL(15,2) ,
```
"BILLING_CYCLE_DAY" INTEGER ,
"APPLICATION_DT" DATE NOT NULL ,
"STATUS" CHAR(10) NOT NULL ,
"TYPE" CHAR(10) NOT NULL ,
"DESCRIPTION" CHAR(254) ,
"BORROWER_ID1" BIGINT NOT NULL ,
"BORROWER_ID2" BIGINT ,
"CREATED_DT" DATE NOT NULL ,
"CREATED_BY" CHAR(10) NOT NULL ,
"LOAN_TERM_DT" DATE ,
"OUTSTANDING_BAL" DECIMAL(15,2) ,
"LAST_TRANS_TS" TIMESTAMP )

DATA CAPTURE CHANGES
IN "USERSPACE1" ;

COMMENT ON COLUMN "DB2LOAN ."LOAN"."AMOUNT" IS 'AMOUNT OF LOAN';

COMMENT ON COLUMN "DB2LOAN ."LOAN"."APPLICATION_DT" IS 'DATE APPLICATION RECEIVED';

COMMENT ON COLUMN "DB2LOAN ."LOAN"."BILLING_CYCLE_DAY" IS 'DAY OF MONTH PAYMENT DUE';

COMMENT ON COLUMN "DB2LOAN ."LOAN"."BORROWED_DT" IS 'DATE CHECK ISSUED FOR LOAN';

COMMENT ON COLUMN "DB2LOAN ."LOAN"."BORROWER_ID1" IS 'FOREIGN KEY TO IDENTIFY BORROWER 1';

COMMENT ON COLUMN "DB2LOAN ."LOAN"."BORROWER_ID2" IS 'FOREIGN KEY TO IDENTIFY BORROWER 2';

COMMENT ON COLUMN "DB2LOAN ."LOAN"."ID" IS 'UNIQUE ID FOR LOAN';

COMMENT ON COLUMN "DB2LOAN ."LOAN"."INT_RATE" IS 'INTEREST RATE FOR LOAN';

COMMENT ON COLUMN "DB2LOAN ."LOAN"."MONTHLY_PAYMENT" IS 'MONTHLY PAYMENT';

COMMENT ON COLUMN "DB2LOAN ."LOAN"."NUM_INSTALLMENT" IS 'NUMBER OF INSTALLMENTS FOR LOAN';

COMMENT ON COLUMN "DB2LOAN ."LOAN"."STATE" IS 'A=APPLICATION, L=LOAN';

COMMENT ON COLUMN "DB2LOAN ."LOAN"."STATUS" IS 'ACTIVE OR INACTIVE';

COMMENT ON COLUMN "DB2LOAN ."LOAN"."TYPE" IS 'HOUSE, AUTO, PERSONAL';

-- DDL Statements for primary key on Table "DB2LOAN ."LOAN"
ALTER TABLE "DB2LOAN "."LOAN"
  ADD CONSTRAINT "LOAN_PK" PRIMARY KEY
  ("ID");

-------------------------------------------------------------------------
CREATE TABLE "DB2LOAN "."BORROWER"  (
  "BORROWER_ID" BIGINT NOT NULL ,
  "NAME" CHAR(30) NOT NULL ,
  "STREET_ADDR" CHAR(40) NOT NULL ,
  "CITY" CHAR(40) NOT NULL ,
  "ST" CHAR(5) NOT NULL ,
  "PHONE" CHAR(15) NOT NULL ,
  "BIRTH_DATE" DATE NOT NULL ,
  "SOC_SEC_NUM" CHAR(11) NOT NULL ,
  "CITIZEN" CHAR(1) NOT NULL ,
  "EMPLOYER" CHAR(30) ,
  "EMPLOYER_CONTACT" CHAR(30) ,
  "NUM_YRS_EMP" INTEGER ,
  "SALARY" DECIMAL(15,2) ,
  "GUARANTEE" CHAR(1) ,
  "CREATED_DT" DATE NOT NULL ,
  "CREATED_BY" CHAR(12) NOT NULL ,
  "FNAME" CHAR(40) NOT NULL WITH DEFAULT ,
  "M_INITIAL" CHAR(2) NOT NULL WITH DEFAULT ,
  "STATUS" CHAR(1) NOT NULL WITH DEFAULT ,
  "REGION_ID" SMALLINT NOT NULL WITH DEFAULT -1 )
DATA CAPTURE CHANGES
  IN "USERSPACE1" ;
COMMENT ON COLUMN "DB2LOAN "."BORROWER"."BORROWER_ID" IS 'UNIQUE BORROWER ID
  NUMBER';
COMMENT ON COLUMN "DB2LOAN "."BORROWER"."CITIZEN" IS 'Y/N FLAG';
COMMENT ON COLUMN "DB2LOAN "."BORROWER"."GUARANTEE" IS 'Y/N FLAG IF CO-SIGNER
  ON LOAN';

-- DDL Statements for primary key on Table "DB2LOAN "."BORROWER"

ALTER TABLE "DB2LOAN "."BORROWER"
  ADD CONSTRAINT "BORROWER_PK" PRIMARY KEY
  ("BORROWER_ID");

-------------------------------------------------------------------------
CREATE TABLE "DB2LOAN "."PERSONAL"  (
  "LOAN_ID" BIGINT NOT NULL ,
  "COLLATERAL_DESC" CHAR(254) ,
  "CREATE_DT" DATE NOT NULL ,
  "CREATED_BY" CHAR(10) NOT NULL )
DATA CAPTURE CHANGES
  IN "USERSPACE1" ;
COMMENT ON COLUMN "DB2LOAN "."PERSONAL"."COLLATERAL_DESC" IS 'DESCRIPTION OF COLLATERAL FOR THIS LOAN';

-- DDL Statements for primary key on Table "DB2LOAN "."PERSONAL"

ALTER TABLE "DB2LOAN "."PERSONAL"
    ADD CONSTRAINT "PERSONAL_PK" PRIMARY KEY
    ("LOAN_ID")

CREATE TABLE "DB2LOAN "."HOUSE"  
    ("LOAN_ID" BIGINT NOT NULL ,
    "STREET_ADDR" CHAR(30) NOT NULL ,
    "CITY" CHAR(30) NOT NULL ,
    "ST" CHAR(2) NOT NULL ,
    "DATE_BUILT" DATE NOT NULL ,
    "CURRENT_VALUE" DECIMAL(15,2) ,
    "CREATED_DT" DATE NOT NULL ,
    "CREATED_BY" CHAR(10) NOT NULL )
DATA CAPTURE CHANGES
IN "USERSPACE1" ;

-- DDL Statements for primary key on Table "DB2LOAN "."HOUSE"

ALTER TABLE "DB2LOAN "."HOUSE"
    ADD CONSTRAINT "HOUSE_PK" PRIMARY KEY
    ("LOAN_ID")

CREATE TABLE "DB2LOAN "."AUTO"  
    ("LOAN_ID" BIGINT NOT NULL ,
    "TYPE" CHAR(10) NOT NULL ,
    "MODEL" CHAR(10) NOT NULL ,
    "YEAR" SMALLINT NOT NULL ,
    "VIN" CHAR(20) NOT NULL ,
    "PURCH_DT" DATE NOT NULL ,
    "REG_ST" CHAR(5) NOT NULL ,
    "NEW_FLG" CHAR(10) NOT NULL ,
    "CREATED_DT" DATE NOT NULL ,
    "CREATED_BY" CHAR(10) NOT NULL )
DATA CAPTURE CHANGES
IN "USERSPACE1" ;

COMMENT ON COLUMN "DB2LOAN "."AUTO"."CREATED_BY" IS 'ID OF EMPLOYEE WHO CREATED LOAN';

COMMENT ON COLUMN "DB2LOAN "."AUTO"."CREATED_DT" IS 'DATE LOAN CREATED';

COMMENT ON COLUMN "DB2LOAN "."AUTO"."LOAN_ID" IS 'IDENTIFIER';

COMMENT ON COLUMN "DB2LOAN "."AUTO"."MODEL" IS 'MODEL E.G. MUSTANG';
COMMENT ON COLUMN "DB2LOAN "."AUTO"."NEW_FLG" IS 'Y/N FLAG TO INDICATE IF AUTO IS NEW OR USED';

COMMENT ON COLUMN "DB2LOAN "."AUTO"."PURCH_DT" IS 'PURCHASE DATE';

COMMENT ON COLUMN "DB2LOAN "."AUTO"."REG_ST" IS 'STATE AUTO IS REGISTERED IN, E.G. CA';

COMMENT ON COLUMN "DB2LOAN "."AUTO"."TYPE" IS 'MANUFACTURER E.G. FORD';

COMMENT ON COLUMN "DB2LOAN "."AUTO"."VIN" IS 'VEHICLE ID NUMBER';

COMMENT ON COLUMN "DB2LOAN "."AUTO"."YEAR" IS 'MODEL YEAR E.G. 1967';

-- DDL Statements for primary key on Table "DB2LOAN "."AUTO"
ALTER TABLE "DB2LOAN "."AUTO"
   ADD CONSTRAINT "AUTO_PK" PRIMARY KEY
   ("LOAN_ID");

CREATE TABLE "DB2LOAN "."COMMENTS"  (
   "COMMENT_ID" BIGINT NOT NULL ,
   "LOAN_ID" BIGINT NOT NULL ,
   "COMMENT" CHAR(254) ,
   "CREATED_DT" DATE NOT NULL ,
   "CREATED_BY" CHAR(10) NOT NULL )
DATA CAPTURE CHANGES
IN "USERSPACE1" ;

COMMENT ON COLUMN "DB2LOAN "."COMMENTS"."COMMENT" IS 'ACTUAL COMMENT';

COMMENT ON COLUMN "DB2LOAN "."COMMENTS"."COMMENT_ID" IS 'UNIQUE ID FOR COMMENT';

-- DDL Statements for primary key on Table "DB2LOAN "."COMMENTS"
ALTER TABLE "DB2LOAN "."COMMENTS"
   ADD CONSTRAINT "COMMENTS_PK" PRIMARY KEY
   ("COMMENT_ID");

CREATE TABLE "DB2LOAN "."TRANSACTION" (  
   "LOAN_ID" BIGINT NOT NULL ,
   "TYPE_OF_TRANS" CHAR(10) NOT NULL ,
   "AMOUNT" DECIMAL(15,3) ,
   "TRAN_DATE" DATE NOT NULL ,
   "CHANNEL" CHAR(20) NOT NULL ,
   "PAYMENT_FORM" CHAR(10) NOT NULL ,

256   Patterns: Information Aggregation and Data Integration with DB2 Information Integrator
“APPLY_EXCESS_TO” CHAR(1),
"TRANSACTION_TS" TIMESTAMP NOT NULL )
DATA CAPTURE CHANGES
IN "USERSPACE1" ;
COMMENT ON COLUMN "DB2LOAN "."TRANSACTION"."AMOUNT" IS 'AMOUNT OF
PAYMENT/CHARGE';

COMMENT ON COLUMN "DB2LOAN "."TRANSACTION"."APPLY_EXCESS_TO" IS 'APPLY EXCESS
PAYMENT TO P=PRINCIPLE OR I=INTEREST';

COMMENT ON COLUMN "DB2LOAN "."TRANSACTION"."CHANNEL" IS 'WHERE TRANSACTION
OCCURED E.G. MAIL, BRANCH OFFICE';

COMMENT ON COLUMN "DB2LOAN "."TRANSACTION"."LOAN_ID" IS 'ACCOUNT FOR
TRANSACTION';

COMMENT ON COLUMN "DB2LOAN "."TRANSACTION"."PAYMENT_FORM" IS 'CHECK, CASH,
DEBIT ACCOUNT';

COMMENT ON COLUMN "DB2LOAN "."TRANSACTION"."TRAN_DATE" IS 'TRANSACTION DATE';

COMMENT ON COLUMN "DB2LOAN "."TRANSACTION"."TYPE_OF_TRANS" IS 'PAYMENT,
INTEREST CHARGE, LATE FEE';

-- DDL Statements for primary key on Table "DB2LOAN "."TRANSACTION"

ALTER TABLE "DB2LOAN "."TRANSACTION"
ADD CONSTRAINT "TRANSACTION_PK" PRIMARY KEY
("LOAN_ID",
"TRANSACTION_TS");

We loaded some representative data in these tables, which is not included here. Note that the data in operational systems is maintained by its business transactions.

Data warehouse details

The data warehouse contains transaction history from each operational system, as well as maintains daily and monthly summaries of transactions for some of the operational systems when appropriate. The data warehouse has a latency of end-of-business-day, and is therefore updated from the operational systems on a scheduled basis corresponding to the end-of-business-day.
In our DFC Customer Insight solution, the relationship information between the various customer accounts in the different operational systems is maintained in the CIF. Therefore, the data warehouse tables corresponding to each operational system are independent of each other and can be loaded/updated in parallel.

Figure C-9 on page 258 shows the data model of the data warehouse.

Example C-6 shows the DDL used to create the DB2 UDB for Multiplatforms data warehouse tables shown in Figure C-9.

Example: C-6  Data warehouse DDL

```sql
CREATE TABLE "PATRNSDW"."CRDT_TRANS_HIS"  
("CC_NBR" BIGINT NOT NULL ,
"TRANSACTION_TS" TIMESTAMP NOT NULL ,
"CARD_HOLDER_ID" INTEGER NOT NULL ,
"TYPE" CHAR(5) NOT NULL ,
"VENDOR" CHAR(10) NOT NULL ,
"AMOUNT" DECIMAL(15,3) NOT NULL ,
"REFERENCE" CHAR(20) NOT NULL ,
"CATEGORY" CHAR(15) NOT NULL ,
"CREATED_BY" CHAR(15) NOT NULL )
IN "USERSPACE1" ;

-- DDL Statements for primary key on Table "PATRNSDW"."CRDT_TRANS_HIS"

ALTER TABLE "PATRNSDW"."CRDT_TRANS_HIS"
ADD CONSTRAINT "CC1078958022705" PRIMARY KEY
("CC_NBR",
"CARD_HOLDER_ID",
"TRANSACTION_TS") ;

CREATE TABLE "PATRNSDW"."CHK_SAV_TRANS_HIS"  
("TRANSACTION_TS" TIMESTAMP NOT NULL ,
"MASTER_ACCOUNT_NUMBER" BIGINT NOT NULL ,
"MEMBER_ACCOUNT_NUMBER" BIGINT NOT NULL ,
"TYPE" CHAR(5) NOT NULL ,
"VENDOR" CHAR(10) NOT NULL ,
"AMOUNT" DECIMAL(15,3) NOT NULL ,
"REFERENCE" CHAR(20) NOT NULL ,
"CATEGORY" CHAR(15) NOT NULL ,
"CREATED_BY" CHAR(15) NOT NULL )
IN "USERSPACE1" ;
```
"LOCATION" CHAR(20) NOT NULL ,
"CHECK_NO" BIGINT NOT NULL ,
"AMOUNT" DECIMAL(10,3) NOT NULL ,
"CHANNEL" CHAR(15) NOT NULL )
IN "USERSPACE1" ;

ALTER TABLE "PATRNSDW"."CHCK_SAV_TRANS_HIS"
ADD CONSTRAINT "CC9278958122705" PRIMARY KEY
("TRANSACTION_TS",
"MASTER_ACCOUNT_NUMBER",
"MEMBER_ACCOUNT_NUMBER");

CREATE TABLE "PATRNSDW"."LOAN_TRANS_HIS"  (
"TRANSACTION_TS" TIMESTAMP NOT NULL ,
"LOAN_ID" INTEGER NOT NULL ,
"TRANSACTION_TYPE" CHAR(20) NOT NULL ,
"AMOUNT" DECIMAL(10,3) NOT NULL ,
"DATE" DATE NOT NULL ,
"CHANNEL" CHAR(10) NOT NULL ,
"FORM" CHAR(10) NOT NULL ,
"APPLY_EXCESS_PAY" CHAR(1) NOT NULL ,
"CREATED_BY" CHAR(15) NOT NULL )
IN "USERSPACE1" ;

ALTER TABLE "PATRNSDW"."LOAN_TRANS_HIS"
ADD CONSTRAINT "CC9678958122795" PRIMARY KEY
("TRANSACTION_TS",
"LOAN_ID");

CREATE TABLE "PATRNSDW"."CHKSVG_SUM_DAY"  (
"ACCOUNT_NBR" INTEGER NOT NULL ,
"REPORTING_PERIOD DATE NOT NULL ,
"MEMBER_NUMBER" INTEGER NOT NULL ,
"MASTER_ACCOUNT_NBR" INTEGER NOT NULL ,
"DEBIT_CARD_NBR" CHAR(16) ,
"ACCOUNT_TYPE" CHAR(10) NOT NULL ,
"ACCOUNT_STATUS" CHAR(1) NOT NULL ,
"ACCOUNT_CREATION_DATE" DATE NOT NULL ,
"PRODUCT_ID" INTEGER NOT NULL ,
"NUMBER_OF_DEBIT_TRANS" INTEGER NOT NULL ,
"NUMBER_OF_DEPOSITS SMALLINT NOT NULL ,
"LAST_TRANSACTION_TS TIMESTAMP ,
"TOTAL_MONEY_IN_DEPOSITS DECIMAL(15,2) ,
"TOTAL_MONEY_IN_WITHDRAWALS DECIMAL(15,2))
IN "USERSPACE1" ;

-- DDL Statements for primary key on Table "PATRNSDW"."CHKSVG_SUM_DAY"
ALTER TABLE "PATRNSDW"."CHKSVG_SUM_DAY"
ADD PRIMARY KEY
("ACCOUNT_NBR",
"REPORTING_PERIOD");

CREATE TABLE "PATRNSDW"."CHKSVG_SUM_MONTH"  (
"ACCOUNT_NBR" INTEGER NOT NULL,
"REPORTING_PERIOD" DATE NOT NULL,
"MEMBER_NUMBER" INTEGER NOT NULL,
"MASTER_ACCOUNT_NBR" INTEGER NOT NULL,
"DEBIT_CARD_NBR" CHAR(16),
"ACCOUNT_TYPE" CHAR(10) NOT NULL,
"ACCOUNT_STATUS" CHAR(1) NOT NULL,
"ACCOUNT_CREATION_DATE" DATE NOT NULL,
"PRODUCT_ID" INTEGER NOT NULL,
"NUMBER_OF_DEBIT_TRANS" INTEGER NOT NULL,
"NUMBER_OF_DEPOSITS" SMALLINT NOT NULL,
"LAST_TRANSACTION_TS" TIMESTAMP,
"TOTAL_MONEY_IN_DEPOSITS" DECIMAL(15,2),
"TOTAL_MONEY_IN_WITHDRAWALS" DECIMAL(15,2))
IN "USERSPACE1";

-- DDL Statements for primary key on Table "PATRNSDW"."CHKSVG_SUM_MONTH"
ALTER TABLE "PATRNSDW"."CHKSVG_SUM_MONTH"
ADD PRIMARY KEY
("ACCOUNT_NBR",
"REPORTING_PERIOD");

CREATE TABLE "PATRNSDW"."BRKRG_SUM_DAY"  (
"ACCOUNT_ID" INTEGER NOT NULL,
"REPORTING_PERIOD" DATE NOT NULL,
"OWNER_ID" INTEGER NOT NULL,
"LAST_TRANSACTION_TS" TIMESTAMP NOT NULL,
"BUYING_POWER" DECIMAL(15,3) NOT NULL,
"CURRENT_ACCOUNT_WORTH" DECIMAL(15,3) NOT NULL,
"UNITS_SOLD" DECIMAL(15,3) NOT NULL,
"UNITS_BOUGHT" DECIMAL(15,3) NOT NULL,
"TOTAL_NUMBER_OF_TRANS" SMALLINT NOT NULL,
"TOTAL_TRANSACTION_FEE_PAID" DECIMAL(10,1) NOT NULL)
IN "USERSPACE1";

-- DDL Statements for primary key on Table "PATRNSDW"."BRKRG_SUM_DAY"
ALTER TABLE "PATRNSDW"."BRKRG_SUM_DAY"
ADD PRIMARY KEY
("ACCOUNT_ID",
"REPORTING_PERIOD");
CREATE TABLE "PATRNSDW"."BRKRG_SUM_MONTH"  (  "ACCOUNT_ID" INTEGER NOT NULL ,  "REPORTING_PERIOD" DATE NOT NULL ,  "OWNER_ID" INTEGER NOT NULL ,  "LAST_TRANSACTION_TS" TIMESTAMP NOT NULL ,  "BUYING_POWER" DECIMAL(15,3) NOT NULL ,  "CURRENT_ACCOUNT_WORTH" DECIMAL(15,3) NOT NULL ,  "UNITS_SOLD" DECIMAL(15,3) NOT NULL ,  "UNITS_BOUGHT" DECIMAL(15,3) NOT NULL ,  "TOTAL_NUMBER_OF_TRANS" SMALLINT NOT NULL ,  "TOTAL_TRANSACTION_FEE_PAID" DECIMAL(10,1) NOT NULL ) IN "USERSPACE1" ;

-- DDL Statements for primary key on Table "PATRNSDW"."BRKRG_SUM_MONTH"
ALTER TABLE "PATRNSDW"."BRKRG_SUM_MONTH"
ADD PRIMARY KEY
("ACCOUNT_ID",
 "REPORTING_PERIOD");

CREATE TABLE "PATRNSDW"."LOAN_SUM_MONTH"  (  "LOAN_ID" INTEGER NOT NULL ,  "REPORTING_PERIOD" DATE NOT NULL ,  "BORROWER_ID" INTEGER NOT NULL ,  "NUMBER_OF_PENDING_INSTALLMENTS" SMALLINT NOT NULL ,  "NUMBER_OF_INSTALLMENTS_MADE" SMALLINT NOT NULL ,  "NEXT_PAYMENT_DATE" DATE NOT NULL ,  "PAYOFF_AMOUNT" DECIMAL(10,3) NOT NULL ,  "INTEREST_RATE" DECIMAL(10,3) NOT NULL ) IN "USERSPACE1" ;

-- DDL Statements for primary key on Table "PATRNSDW"."LOAN_SUM_MONTH"
ALTER TABLE "PATRNSDW"."LOAN_SUM_MONTH"
ADD PRIMARY KEY
("LOAN_ID",
 "REPORTING_PERIOD");

CREATE TABLE "PATRNSDW"."DWCREDIT"  (  "CC_NBR" BIGINT NOT NULL ,  "CARDHOLDER_ID" INTEGER NOT NULL ,  "CATEGORY" CHAR(10) NOT NULL ,  "CREDIT_HISTORY_IND" CHAR(1) ,  "PROD_ID" INTEGER NOT NULL ,
...
"ACTIVATION_DATE" DATE NOT NULL ,
"EXPIRY_DATE" DATE NOT NULL ,
"CREATED_DATE" DATE NOT NULL ,
"ANNUAL_FEE" DECIMAL(4,2) NOT NULL ,
"CREDIT_LIMIT" DECIMAL(12,2) NOT NULL ,
"CREDIT_CARD_STATUS" CHAR(1) NOT NULL ,
"PURCHASE_RATE" DECIMAL(3,1) NOT NULL ,
"NEXT_PAYMENT_DUE_DATE" DATE NOT NULL ,
"LAST_TRANSACTION_TS" TIMESTAMP NOT NULL ,
"CASH_ADVANCE_RATE" DECIMAL(3,1) NOT NULL ,
"OUTSTANDING_BALANCE" DECIMAL(12,2) NOT NULL ,
"TYPE" CHAR(10) ,
"REGION_ID" SMALLINT NOT NULL WITH DEFAULT -1 )
IN "USERSPACE1" ;

-- DDL Statements for primary key on Table "PATRNSDW"."DWCREC"

ALTER TABLE "PATRNSDW"."DWCREC"
ADD PRIMARY KEY
("CC_NBR",
 "CARDHOLDER_ID");

CREATE TABLE "PATRNSDW"."DWCHKSVG" (  
 "ACCOUNT_NBR" INTEGER NOT NULL ,
 "MEMBER_NUMBER" INTEGER NOT NULL ,
 "MASTER_ACCOUNT_NBR" INTEGER NOT NULL ,
 "DEBIT_CARD_NBR" CHAR(16) ,
 "ACCOUNT_TYPE" CHAR(10) NOT NULL ,
 "ACCOUNT_STATUS" CHAR(1) NOT NULL ,
 "ACCOUNT_CREATION_DATE" DATE NOT NULL ,
 "PRODUCT_ID" INTEGER NOT NULL ,
 "BALANCE" DECIMAL(15,2) NOT NULL ,
 "LAST_TRANS_TS" TIMESTAMP NOT NULL ,
 "CREATED_BY" CHAR(10) NOT NULL ,
 "CREATED_DATE" DATE NOT NULL )
IN "USERSPACE1" ;

-- THIS table has no primary key (Nagraj March 26th 2004)
-- DDL Statements for primary key on Table "PATRNSDW"."DWCHKSVG"

-- ALTER TABLE "PATRNSDW"."DWCHKSVG"  
-- ADD PRIMARY KEY  
-- ("ACCOUNT_NBR",  
-- "MEMBER_NUMBER",  
-- "MASTER_ACCOUNT_NBR");  
--
CREATE TABLE "PATRNSDW"."DWLOANS"  (
  "LOAN_ID" BIGINT NOT NULL ,
  "BORROWER_ID" BIGINT NOT NULL ,
  "LOAN_TYPE" CHAR(10) NOT NULL ,
  "LOAN_TYPE_DESC" CHAR(250) ,
  "INTEREST_RATE" DECIMAL(6,3) NOT NULL ,
  "AMOUNT_BORROWED" DECIMAL(15,2) ,
  "OUTSTANDING_BALANCE" DECIMAL(15,2) ,
  "NO_OF_INSTALMENTS" INTEGER ,
  "CO_BORROWER_ID" BIGINT ,
  "BORROWED_DATE" DATE NOT NULL ,
  "BORROWER_STATUS" CHAR(10) NOT NULL ,
  "MONTHLY_PAYMENT" DECIMAL(15,2) ,
  "CREATED_DATE" DATE NOT NULL ,
  "PAYMENT_DUE_DAY" INTEGER ,
  "LOAN_TERMINATION_DT" DATE ,
  "LAST_UPDATED_TS" TIMESTAMP ) IN "USERSPACE1" ;

-- DDL Statements for primary key on Table "PATRNSDW"."DWLOANS"
ALTER TABLE "PATRNSDW"."DWLOANS"
  ADD PRIMARY KEY
  ("LOAN_ID",
   "BORROWER_ID");

CREATE TABLE "PATRNSDW"."DWBRKRG"  (
  "ACT_NBR" BIGINT NOT NULL ,
  "PORTFOLIO_ID" BIGINT NOT NULL ,
  "CUST_ID" BIGINT NOT NULL ,
  "PRODUCT_ID" BIGINT NOT NULL ,
  "SECURITY_ID" BIGINT NOT NULL ,
  "SYMBOL" CHAR(8) NOT NULL ,
  "SECURITY_TYPE" CHAR(8) NOT NULL ,
  "ACCOUNT_ACTIVE_DATE" DATE NOT NULL ,
  "NBR_OF_UNITS" DECIMAL(15,2) NOT NULL ,
  "PRICE_PER_UNIT" DECIMAL(10,2) NOT NULL ,
  "STATUS" CHAR(1) NOT NULL ,
  "LAST_TRANS_TS" TIMESTAMP NOT NULL ) IN "USERSPACE1" ;

-- DDL Statements for primary key on Table "PATRNSDW"."DWBRKRG"
ALTER TABLE "PATRNSDW"."DWBRKRG"
  ADD PRIMARY KEY
  ("ACT_NBR",
   "PORTFOLIO_ID",};

Appendix C. Data models, table/file definitions, et al used in the DFC Customer Insight business solution
CREATE TABLE "PATRNSDW"."BRKR_TRANS_HIS" (  "TRANSACTION_TS" TIMESTAMP NOT NULL ,  "ACCOUNT_ID" BIGINT NOT NULL ,  "SECURITY_ID" BIGINT ,  "CATEGORY" CHAR(15) ,  "TYPE" CHAR(5) ,  "PRICE_PER_UNIT" DECIMAL(10,2) ,  "UNITS" DECIMAL(10,2) ,  "TRANSACTION_FEE" DECIMAL(5,2) ,  "LONG_MARGIN" CHAR(10) ,  "CREATED_BY" CHAR(15) ) IN "USERSPACE1" ;

-- DDL Statements for primary key on Table "PATRNSDW"."BRKR_TRANS_HIS"
ALTER TABLE "PATRNSDW"."BRKR_TRANS_HIS"  
ADD PRIMARY KEY  
("TRANSACTION_TS",  "ACCOUNT_ID");

CREATE TABLE "PATRNSDW"."CRDT_SUM_DAY" (  "CARDHOLDER_ID" INTEGER NOT NULL ,  "LAST_TRANSACTION_TS" TIMESTAMP ,  "CC_NBR" BIGINT ,  "TOTAL_NUMBER_OF_TRANS" INTEGER ,  "TOTAL_CREDITS" INTEGER ,  "TOTAL_DEBITS" INTEGER ,  "TOTAL_MONEY_SPENT" DECIMAL(15,2) ,  "MONEY_SPENT_ON_GROCERY" DECIMAL(10,2) ,  "MONEY_SPENT_ON_GAS" DECIMAL(10,2) ,  "MONEY_SPENT_ON_OTHERS" DECIMAL(10,2) ,  "TOTAL_AVAILABLE_CREDIT" DECIMAL(15,2) ,  "REPORTING_PERIOD" DATE NOT NULL ,  "TOTAL_PAYMENT_MADE" DECIMAL(15,2) ) IN "USERSPACE1" ;

-- DDL Statements for primary key on Table "PATRNSDW"."CRDT_SUM_DAY"
ALTER TABLE "PATRNSDW"."CRDT_SUM_DAY"  
ADD PRIMARY KEY  
("CARDHOLDER_ID",  "REPORTING_PERIOD");

CREATE TABLE "PATRNSDW"."CRDT_SUM_MONTH" (  "CARDHOLDER_ID" INTEGER NOT NULL ,  "LAST_TRANSACTION_TS" TIMESTAMP ,  "REPORTING_PERIOD" DATE NOT NULL ,  "TOTAL_MONEY_SPENT" DECIMAL(15,2) ,  "MONEY_SPENT_ON_GROCERY" DECIMAL(10,2) ,  "MONEY_SPENT_ON_GAS" DECIMAL(10,2) ,  "MONEY_SPENT_ON_OTHERS" DECIMAL(10,2) ,  "TOTAL_AVAILABLE_CREDIT" DECIMAL(15,2) ,  "TOTAL_PAYMENT_MADE" DECIMAL(15,2) ) IN "USERSPACE1" ;
"CARDHOLDER_ID" INTEGER NOT NULL,
"REPORTING_PERIOD" DATE NOT NULL,
"LAST_TRANSACTION_TS" TIMESTAMP,
"TOTAL_NUMBER_OF_TRANS" INTEGER,
"TOTAL_CREDITS" INTEGER,
"TOTAL_DEBITS" INTEGER,
"TOTAL_MONEY_SPENT" DECIMAL(10,2),
"MONEY_SPENT_ON_GROCERY" DECIMAL(10,2),
"MONEY_SPENT_ON_GAS" DECIMAL(10,2),
"MONEY_SPENT_ON_OTHERS" DECIMAL(10,2),
"TOTAL_AVAILABLE_CREDIT" DECIMAL(15,2),
"TOTAL_PAYMENT_MADE" DECIMAL(15,2)
) IN "USERSPACE1";

-- DDL Statements for primary key on Table "PATRNSDW"."CRDT_SUM_MONTH"

ALTER TABLE "PATRNSDW"."CRDT_SUM_MONTH"
ADD PRIMARY KEY
("CARDHOLDER_ID",
"REPORTING_PERIOD");

CREATE TABLE "DWPATRNS"."REWARD_TRANS_HIS" (
"CCNUM" CHAR(16) NOT NULL,
"TRANS_TS" TIMESTAMP NOT NULL,
"AMOUNT" DECIMAL(12,2),
"TRAN_TYPE" CHAR(1),
"REASON_CODE" CHAR(4),
"REWARD_ID" CHAR(10),
"SEQ" CHAR(8)
) IN "USERSPACE1";

CREATE TABLE "DWPATRNS"."REWARD_PARTNER" (
"REWARDID" CHAR(10),
"PARTNER_NAME" CHAR(30),
"CLASSIFICATION" CHAR(10),
"ACCUMULATION_TYPE" CHAR(6),
"EFFECTIVE_FROM" DATE,
"EFFECTIVE_TO" DATE,
"CREATED_DATE" DATE,
"CREATED_BY" CHAR(10)
) IN "USERSPACE1";

CREATE TABLE "DWPATRNS"."REWARD_TRACKING" (
"CCNUM" CHAR(16) NOT NULL,
"LAST_NAME" CHAR(30),
"FIRST_NAME" CHAR(30),
"PARTNER_ID_NUM" CHAR(12),
"TOTAL_MONEY_SPENT" DECIMAL(10,2),
"MONEY_SPENT_ON_GROCERY" DECIMAL(10,2),
"MONEY_SPENT_ON_GAS" DECIMAL(10,2),
"MONEY_SPENT_ON_OTHERS" DECIMAL(10,2),
"TOTAL_AVAILABLE_CREDIT" DECIMAL(15,2),
"TOTAL_PAYMENT_MADE" DECIMAL(15,2)
) IN "USERSPACE1";
"REWARDID" CHAR(10) ,
"POINTBALANCE" DECIMAL(10,2) ,
"PROCESS_DATE" DATE ,
"CREATED_DATE" DATE ,
"CREATED_BY" CHAR(10) )
IN "USERSPACE1" ;

---------------------------------------------------------------------------

CREATE TABLE "DWPATRNS"."REWARD_JOURNAL" (  
"RECORD_TYPE" CHAR(1) ,
"CHANGE_SEQ" SMALLINT ,
"CHANGE_TS" TIMESTAMP ,
"CHANGE_SOURCE" CHAR(1) ,
"INITIATOR" INTEGER ,
"CCNUM" CHAR(16) ,
"LAST_NAME" CHAR(30) ,
"FIRST_NAME" CHAR(30) ,
"PARTNER_ID_NUM" CHAR(12) ,
"REWARDID" CHAR(10) ,
"POINTBALANCE" DECIMAL(10,2) ,
"PROCESS_DATE" DATE ,
"CREATED_DATE" DATE ,
"CREATED_BY" CHAR(10) )
IN "USERSPACE1" ;

The data warehouse is both initially populated as well as incrementally updated using the Ascential DataStage product. Each of these processes is described briefly in the following sections.

Note: As mentioned earlier, we chose to access all the operational systems’ data sources through the federated server using nicknames by Ascential DataStage for both the one-time initial population of the data warehouse as well as its recurring incremental updates.

Initial load of the data warehouse

This is a one-time effort to populate the data warehouse from the operational systems.

Figure C-10 on page 267 through Figure C-17 on page 274 show some of the DataStage screens used to define the processes for performing this initial population.
Figure C-10 shows the simple model for loading the four data warehouse tables corresponding to the checkings/savings, credit card, loans and brokerage operational systems. The rewards system load is not shown here. It shows a data source (CREDIT, for example) being linked (named Extract_Credit) to a transformer stage (named Transform_Credit_Data), which is then linked (named Load_Credit_Data) to the target table (CREDIT_DW) for each system.
Figure C-11 displays the contents of the **Columns** tab for the Extract_Credit link, and identifies the data source columns in the **Derivation** field. The **Column name** field lists the user-defined names of the extracted columns. The ODBC driver is used to perform this extraction and is called the **ODBC Stage**.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Group</th>
<th>Derivation</th>
<th>Key</th>
<th>SQL type</th>
<th>Length</th>
<th>Scale</th>
<th>Null</th>
<th>Display</th>
<th>Data element</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC_NBR</td>
<td></td>
<td></td>
<td></td>
<td>Char</td>
<td>10</td>
<td></td>
<td>No</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CARDHOLDER_ID</td>
<td></td>
<td>ADMINISTRATOR_C_CREDIT_CARD_CARDHOLDER_ID</td>
<td></td>
<td>Integer</td>
<td>10</td>
<td></td>
<td>No</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CATEGORY</td>
<td></td>
<td>ADMINISTRATOR_C_CREDIT_ACCOUNT CATEGORY</td>
<td></td>
<td>Char</td>
<td>10</td>
<td></td>
<td>No</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CREDIT_HIS_IND</td>
<td></td>
<td>ADMINISTRATOR_C_CREDIT_ACCOUNT CREDIT_HIS_IND</td>
<td></td>
<td>Char</td>
<td>1</td>
<td>No</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PKG_ID</td>
<td></td>
<td>ADMINISTRATOR_C_CREDIT_CARD_PKG_ID</td>
<td></td>
<td>Integer</td>
<td>10</td>
<td></td>
<td>No</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACTIVATION_DT</td>
<td></td>
<td>ADMINISTRATOR_C_CREDIT_CARD_ACTIVATION_DT</td>
<td></td>
<td>Date</td>
<td>10</td>
<td>Yes</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXPIRY_DT</td>
<td></td>
<td>ADMINISTRATOR_C_CREDIT_CARD_EXPIRY_DT</td>
<td></td>
<td>Date</td>
<td>10</td>
<td></td>
<td>Yes</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CREATED_DATE</td>
<td></td>
<td>ADMINISTRATOR_C_CREDIT_ACCOUNT CREATED_DATE</td>
<td></td>
<td>Date</td>
<td>10</td>
<td>No</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANNUAL_FEE</td>
<td></td>
<td>ADMINISTRATOR_C_PRODUCT ANNUAL_FEE</td>
<td></td>
<td>Decimal</td>
<td>3</td>
<td>1</td>
<td>Yes</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CREDIT_LIMIT</td>
<td></td>
<td>ADMINISTRATOR_C_CREDIT_ACCOUNT CREDIT_LIMIT</td>
<td></td>
<td>Decimal</td>
<td>10</td>
<td>2</td>
<td>Yes</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATUS</td>
<td></td>
<td>ADMINISTRATOR_C_CREDIT_ACCOUNT STATUS</td>
<td></td>
<td>Char</td>
<td>1</td>
<td>No</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PURCHASE_DATE</td>
<td></td>
<td>ADMINISTRATOR_C_PRODUCT PURCHASE_DATE</td>
<td></td>
<td>Decimal</td>
<td>3</td>
<td>1</td>
<td>Yes</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEXT_PAY_DUE</td>
<td></td>
<td>ADMINISTRATOR_C_CREDIT_ACCOUNT NEXT_PAY_DUE</td>
<td></td>
<td>Date</td>
<td>10</td>
<td>Yes</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAST_TRANS_TS</td>
<td></td>
<td>ADMINISTRATOR_C_CREDIT_ACCOUNT LAST_TRANS_TS</td>
<td></td>
<td>Timestamp</td>
<td>25</td>
<td>6</td>
<td>No</td>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CASH_ADV_RATE</td>
<td></td>
<td>ADMINISTRATOR_C_CREDIT_ACCOUNT CASH_ADV_RATE</td>
<td></td>
<td>Decimal</td>
<td>3</td>
<td>1</td>
<td>Yes</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BAL_Owed</td>
<td></td>
<td>ADMINISTRATOR_C_CREDIT_ACCOUNT BAL_Owed</td>
<td></td>
<td>Decimal</td>
<td>5</td>
<td>2</td>
<td>No</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE</td>
<td></td>
<td>ADMINISTRATOR_C_PRODUCT_TYPE</td>
<td></td>
<td>Char</td>
<td>20</td>
<td>No</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure C-12  DataStage Designer Transformer Stage

Figure C-12 describes the **Transformer Stage**, which includes the mapping between the source columns (Extract_Credit link) and the target columns (Load_Credit_Data link), and any transformations to be performed. Note the simple data type transform in the SQL type of the CC_NBR column from CHAR to BigInt.
Figure C-13  DataStage Designer Load_Credit_Data ODBC Stage

Figure C-13 shows the options for loading the target table from the input link Load_Credit_Data. Ascential DataStage provides a number of options on how the target is to be updated as shown under the Update action field. For the initial load, the Insert rows without clearing was appropriate in our case.
Figure C-14 Describes the loading of the credit summary data warehouse table from the credit transactions data warehouse table as its source. Note the additional stage of aggregating the data.
Figure C-15 displays the contents of the Columns tab for the EXTRACT link, and identifies the data source columns in the Derivation field. The Column name field lists the user-defined names of the extracted columns. The ODBC driver is used to perform this extraction and is called the ODBC Stage.
The Aggregator Stage in Figure C-16 identifies the input columns in the EXTRACT link under the **Columns** tab available to the aggregation function, while the **Outputs** tab in Figure C-17 on page 274 for the TRANSFORM link identifies the types of aggregations to be performed.
In Figure C-17, note the grouping that occurred on (CARDHOLDER_ID, CC_NBR) and the columns generated (TOTAL_NUMBER_OF_TRANS, TOTAL_MONEY_SPENT, and LAST_TRANSACTION_TS) using Count(CC_NBR), Sum(AMOUNT), and Max(TRANSACTION_TS) functions.
Appendix C. Data models, table/file definitions, et al used in the DFC Customer Insight business solution

Figure C-18  DataStage Designer Credit Transactions Transformer Stage

Figure C-18 describes the Transformer Stage, which includes the mapping between the source columns (TRANSFORM link) and the target columns (LOAD link), and any transformations to be performed.

Note: Similar jobs can be defined for the other transaction summaries.

Incremental update of the data warehouse

The Ascential DataStage jobs for incrementally updating the data warehouse on a recurring basis are quite similar to the processes described in “Initial load of the data warehouse” on page 266 except for the following differences:

1. The Update action field in Figure C-13 on page 270 should be Insert new or update existing rows.

2. The data sources are different because they need to only have access to changed data, that is, incremental changes that have occurred since the last update.
– Changed data was captured in the relational database operational systems using the appropriate Capture feature of DataPropagator™ (DPROPR on z/OS), DB2 UDB replication, and Oracle replication.
– The non-relational VSAM data changes were extracted from a journal maintained by the rewards operational system.

**Note:** Nicknames were created on the federated server for the tables containing changed data, which were then defined as the data source for Ascential DataStage.

**Data mart details**

As mentioned earlier, one or more data marts would be created on an on demand basis for analyses and mining to plan targeted marketing campaigns and/or alter the rating of a particular customer based on prior transaction history. One such example may be creating a data mart of all customers in the western United States having assets above $100,000 with a history of limited transaction activity in their brokerage accounts in order to explore offering possible incentives to them for becoming more active.

In such cases, the data mart data model would be a derivative of the data warehouse, with the data warehouse being the only data source for populating the data mart.

In our case we built a data mart with a data model identical to that of the data warehouse (see Figure C-9 on page 258); the data mart DDL is also similar to the data warehouse DDL described in Example C-6 on page 258.

Our scenario involved DFC exploring a new product offering (insurance services) with customers in a high revenue generating region such as the north eastern part of the US. We extracted customer data in this region (from the data warehouse) having home owner demographics for further analysis and mining in order to generate an appropriate targeted marketing campaign.

Figure C-19 on page 277 shows the simple model for loading the credit card related data mart tables. It shows a single transform stage that connects to the data source using the EXTRACT link, and the target table using the LOAD link.
Since this data mart performed very simple filtering of data from the data warehouse, we have not included the screens defining this activity.

CIIS details

The IBM CIIS offering has a customizable object data model that is implemented on DB2 on z/OS, UNIX and Windows platforms. CIIS can be populated in one of two ways, as follows:

1. A CIIS API to insert, update, and delete CIIS objects. This is the recommended approach since it ensures consistency and integrity of relationships between the various CIIS objects during updates.

2. Directly manipulate the underlying DB2 tables using SQL or load. This has to be done with great care to avoid consistency and integrity exposures. This
approach is only recommended for performance reasons during the initial population of the CIIS when extremely large volumes of data may be involved. A rigorous procedure has been developed to ensure an efficient and error-free initial load of the CIIS.

We customized the CIIS to only contain customer metadata of each of the operational systems, and the holistic view of a single customer across all the operational systems. Our CIIS did not contain any transaction information. Most of the information in the CIIS comes from the operational systems; the exception being with holistic information about the customer such as the customer rating, which would tend to be updated based on analysis and mining of information in the data warehouse.

We chose to use the CIIS API for the initial load of the CIIS, and direct update of the underlying tables for certain types of incremental updates such as those originating in the data warehouse.

Figure C-20 on page 279 shows the custom version of the CIIS data model for the DFC Customer Insight solution in the CIIS Workbench, while Figure C-21 on page 280 shows the typical representation.
Appendix C. Data models, table/file definitions, et al used in the DFC Customer Insight business solution
Once the custom version has been designed, the appropriate DDL is generated to create the underlying DB2 tables.

**Initial load of the CIIS**

CIIS generates surrogate keys for the CIIS objects that form the basis of establishing relationships between the objects. When multiple related objects are loaded independently, one needs to determine the generated surrogate keys in order to establish relationships between them. To avoid this complexity, the optimal way is to load a set of related objects together using artificial keys. Since multiple related objects exist in the different stovepipe operational systems, we adopted the following approach for the initial load of the CIIS.

1. Extract data from the individual operation systems using SQL scripts via federation and load the data into temporary tables corresponding the CIIS
objects. Also create tables that were templates for generating the CIIS API constructs. The objective of creating the temporary tables is to load multiple related objects together and avoiding the complexity of dealing with CIIS generated surrogate keys for the CIIS objects.

Example C-7 shows an example of the script we used to extract data from one of the operational systems.

**Example: C-7  Sample extract and load into temporary tables of a single customer**

```sql
--Insert Header
insert into gheader(header_id) values (1);
insert into a_person(header_id) values (1);

----Insert System
insert into a_system1(header_id, ext_party_ref)
    select 1, '00'||cast(cast(member_act_nbr as integer) as char(8))
    from s_account_member
    where first_name like 'LUCCHESSI%';
insert into a_system2(header_id, ext_party_ref)
    select 1, '00000'||cast(cardholder_id as char(5))
    from c_card_holder
    where first_name like 'LUCCHESSI%';
insert into a_system3(header_id, ext_party_ref)
    select 1, cast(owner_id as char(10))
    from b_owner
    where first_name like 'LUCCHESSI%';
insert into a_system4(header_id, ext_party_ref)
    select 1, cast(borrower_id as char(10))
    from l_borrower
    where fname like 'LUCCHESSI%';

--Insert Party
insert into a_party (header_id, last_name, first_name)
    values (1, 'VINCENZO', 'LUCCHESSI');
insert into a_postcontact(header_id, town, post_cde, country, state, addr1)
    select 1, city, zip, 'USA', state, street_adr
    from c_card_holder
    where first_name like 'LUCCHESSI%';
insert into a_postrpcontact(header_id) values (1);
insert into a_telcontact(header_id, area_cde, extn)
    select 1, '000'||substr(phone, 1,3), substr(phone, 5,8)
    from c_card_holder
    where first_name like 'LUCCHESSI%';
insert into a_telerpcontact(header_id) values (1);

--Insert SummaryAccount
insert into a_accsum1
    (header_id, account_no, account_type, atm_flg, credit_flg,
    ...
``
check_flg, product_id, product_desc)
select 1, cast(cast(b.act_nbr as integer) as char(4)),
    'Checking', '0', '0', '1',
    '0000000'||cast(cast(c.product_id as integer) as char(3)),
c.description
from s_account_member a, s_account b, s_product c
where a.first_name like 'LUCCHESSI%'
and a.master_act_nbr = b.master_act_nbr
and b.product_id = c.product_id ;

insert into a_rpaccsum1(header_id) values (1);

insert into a_accsum2
    (header_id, account_no, account_type, atm_flg,
     credit_flg, check_flg, product_id, product_desc,
     category, acc_limit)
select 1, c.cc_nbr, 'Credit', '0', '1', '0',
    '000000000'||cast(c.prod_id as char(1)), d.prod_desc,
    b.category, cast(cast(b.credit_limit as integer) as char(4))
from c_card_holder a, c_credit_account b, c_credit_card c, c_product d
where a.first_name like 'LUCCHESSI%'
and a.cardholder_id = c.cardholder_id
and c.cc_nbr = b.cc_nbr
and c.prod_id = d.prod_id;

insert into a_rpaccsum2(header_id) values (1);

insert into a_accsum3
    (header_id, account_no, account_type, atm_flg, credit_flg,
     check_flg, product_id, product_desc)
select 1, cast(a.account_nbr as char(13)), 'Brokerage', '0', '0', '0',
    '000000000'||cast(a.product_id as char(1)), c.product_desc
from b_account a, b_owner b, b_product c
where b.first_name like 'LUCCHESSI%'
and a.owner_id = b.owner_id
and a.product_id = c.product_id;

insert into a_rpaccsum3(header_id) values (1);

insert into a_accsum4(header_id, account_no, account_type, atm_flg,
credit_flg, check_flg, product_id, product_desc)
select 1, cast(b.id as char(9)), 'Loan', '0', '0', '0', b.type,
case b.type
    when 'HOUSE' then 'House Loan'
    when 'AUTO' then 'Auto Loan'
    when 'PERSONAL' then 'Personal Loan'
else 'NONE'
end
from l_borrower a, l_loan b
where a.fname like 'LUCCHESI%'
and (a.borrower_id = b.borrower_id1
or a.borrower_id = b.borrower_id2);

insert into a_rpacccsum4(header_id) values (1);

--Insert BalanceSummary
insert into a_balsum1(header_id, balance_type, debit_ind, amount)
select 1, 'Checking', '0107050002',
cast(cast(a.balance as integer) as char(5))
from s_account a, s_account_member b
where b.first_name like 'LUCCHESI%'
and a.master_act_nbr = b.master_act_nbr;

insert into a_balsum2(header_id, balance_type, debit_ind, amount)
select 1, 'Credit', '0107050001',
cast(cast(b.bal_owed as integer) as char(6))
from c_card_holder a, c_credit_account b
where a.first_name like 'LUCCHESI%'
and (a.cardholder_id = b.p_cardholder_id
or a.cardholder_id = b.s_cardholder_id);

insert into a_balsum3(header_id, balance_type, amount)
select 1, 'Brokerage', cast(cast(c.num_of_units as integer) as char(4))
from b_account a, b_owner b, b_portfolio c
where b.first_name like 'LUCCHESI%'
and a.owner_id = b.owner_id
and a.account_nbr = c.account_id;

insert into a_balsum4(header_id, balance_type, amount)
select 1, 'Loan', cast(cast(b.amount as integer) as char(7))
from l_borrower a, l_loan b
where a.fname like 'LUCCHESI%'
and (a.borrower_id = b.borrower_id1
or a.borrower_id = b.borrower_id2);

2. Create a java program that joins the related objects in the temporary tables
and generates the CIIS API. Example C-8 shows the java program used to
extract data from the temporary tables populated in the previous step and
generate the CIIS APIs to load all the related CIIS objects for a single
customer together.

Example: C-8  Java™ program for inserting into the CIIS

```java
import java.net.URL;
import java.sql.*;

public class CIISAccess {
```

Appendix C. Data models, table/file definitions, et al used in the DFC Customer Insight business solution
String dbname, id, password;
final String CIISStoreProcedure = "{call RELSPROC(?)}";

public CIISAccess(String dbname, String id, String password)
{
    this.dbname = dbname;
    this.id = id;
    this.password = password;
}

public static void main (String args[])
{
    if (args.length != 3)
    {
        System.out.println("Usage: java CIISAccess <db name> <user id> <password>");
        return;
    }

    String dbname = args[0];
    String id = args[1];
    String password = args[2];

    CIISAccess obj = new CIISAccess(dbname, id, password);
    obj.begin();
}

public void begin()
{
    String url = "jdbc:db2:" + dbname;
    Connection con = null;

    try {
        // Load the jdbc-odbc bridge driver
        Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");

        // Attempt to connect to a driver. Each one
        // of the registered drivers will be loaded until
        // one is found that can process this URL

        // Connection con =
        DriverManager.getConnection(url,"db2admin","admindb2");
        con = DriverManager.getConnection(url, id, password);
// If we were unable to connect, an exception
// would have been thrown. So, if we get here,
// we are successfully connected to the URL

// Check for, and display and warnings generated
// by the connect.

checkForWarning(con.getWarnings());

// Get the DatabaseMetaData object and display
// some information about the connection

DatabaseMetaData dma = con.getMetaData();

System.out.println("\nConnected to " + dma.getURL());
System.out.println("Driver " + dma.getDriverName());
System.out.println("Version " + dma.getDriverVersion() + "\n");

// AddPerson

long totalStartTime =
    java.util.Calendar.getInstance().getTime().getTime();
AddPerson(con);
long totalEndTime =
    java.util.Calendar.getInstance().getTime().getTime();

System.out.println("Total Time: " + (totalEndTime - totalStartTime));

try
{
    con.close();
}
catch (Exception e)
{
}

} catch (SQLException ex) {

    // A SQLException was generated. Catch it and
    // display the error information. Note that there
    // could be multiple error objects chained

Appendix C. Data models, table/file definitions, et al used in the DFC Customer Insight business solution
// together
System.out.println("\n*** SQLException caught ***\n");

while (ex != null)
{
    System.out.println("SQLState: " + ex.getSQLState ());
    System.out.println("Message: " + ex.getMessage ());
    System.out.println("Vendor:   " + ex.getErrorCode ());
    ex = ex.getNextException ();
    System.out.println ("");
}

}

} catch (java.lang.Exception ex)
{
    // Got some other type of exception. Dump it.
    ex.printStackTrace ();
}

}

private void AddPerson(Connection con2)
{

    try
    {
        final String CIISStoreProcedure = "{call RELSPROC(?)}";
        // final String strAddPerson =""
        // final String strAddPerson =""
        String select1="SELECT GHEADER.BLK_LENGTH|| GHEADER.SID|| GHEADER.BLK_NO||
        GHEADER.NEXT_BLK||
        String select2="GHEADER.SYSTEM_ID|| GHEADER.TOTAL_LEN|| GHEADER.RUN_ENV_FLG||
        String select3="GHEADER.CALL_ENV_FLG|| GHEADER.RESERVED1||
        GHEADER.TRACE_FLG||
        String select4="GHEADER.REF_INT_FLG|| GHEADER.NUM_BLK||
        GHEADER.USERID||
        String select5="GHEADER.TERMID|| GHEADER.REQID|| GHEADER.BLK_FILENAME||
        GHEADER.RESERVED3||
        String select6="GHEADER.RESERVED4|| GHEADER.NUM_ELEMENTS||
        GHEADER.RESERVED5||
        String select7="GHEADER.ERR_BOUNDARY|| GHEADER.RESERVED6||
        GHEADER.VALIDATE_FLG||
        String select8="GHEADER.TOKEN|| GHEADER.DELIMT|| GHEADER.RESERVED7||
        GHEADER.NUM_ERR||
        String select9="GHEADER.MAX_ERRCODE|| GHEADER.MAX_ERRTYPE||
        GHEADER.ID_GEN_PREF||GHEADER.FILLER||
        String select10="A_PERSON.CRUD_LEN|| A_PERSON.BO_OBJECT|| A_PERSON.ACTION||
        A_PERSON.GROUP_NO||";
String select11="A_PERSON.NO_ROWS|| A_PERSON.NO_ROW_FOUND|| A_PERSON.PARTY_ID||
A_PERSON.SEC_NO||";
String select12="A_PERSON.NAT_ID_NO|| A_PERSON.BIRTH|| A_PERSON.EMPLOYER||
A_PERSON.EMP_CONTACT||";
String select13="A_PERSON.NUM_YRS_EMP|| A_PERSON.SALARY|| A_PERSON.CITIZEN||
A_PERSON.START_DATE||A_PERSON.END_DATE|| A_PERSON.DELIMT||";

String select14="A_SYSTEM1.CRUD_LEN|| A_SYSTEM1.BO_OBJECT|| A_SYSTEM1.ACTION||
A_SYSTEM1.GROUP_NO||";
String select15="A_SYSTEM1.NO_ROWS|| A_SYSTEM1.NO_ROW_FOUND||
A_SYSTEM1.SYS_DFLT_ID||";
String select16="A_SYSTEM1.SEQ_NO|| A_SYSTEM1.PARTY_ID||
A_SYSTEM1.PARTY_TYPE||A_SYSTEM1.EXT_PARTY_REF|| A_SYSTEM1.EXT_SYS_ID||
A_SYSTEM1.START_DATE||A_SYSTEM1.END_DATE|| A_SYSTEM1.DELIMT||";
String select17="A_SYSTEM2.CRUD_LEN|| A_SYSTEM2.BO_OBJECT|| A_SYSTEM2.ACTION||
A_SYSTEM2.GROUP_NO||";
String select18="A_SYSTEM2.NO_ROWS|| A_SYSTEM2.NO_ROW_FOUND||
A_SYSTEM2.SYS_DFLT_ID||A_SYSTEM2.SEQ_NO|| A_SYSTEM2.PARTY_ID||
A_SYSTEM2.PARTY_TYPE||";
String select19="A_SYSTEM2.EXT_PARTY_REF|| A_SYSTEM2.EXT_SYS_ID||
A_SYSTEM2.START_DATE||A_SYSTEM2.END_DATE|| A_SYSTEM2.DELIMT||";
String select20="A_SYSTEM3.CRUD_LEN|| A_SYSTEM3.BO_OBJECT|| A_SYSTEM3.ACTION||
A_SYSTEM3.GROUP_NO||";
String select21="A_SYSTEM3.NO_ROWS|| A_SYSTEM3.NO_ROW_FOUND||
A_SYSTEM3.SYS_DFLT_ID||A_SYSTEM3.SEQ_NO|| A_SYSTEM3.PARTY_ID||
A_SYSTEM3.PARTY_TYPE||";
String select22="A_SYSTEM3.EXT_PARTY_REF|| A_SYSTEM3.EXT_SYS_ID||
A_SYSTEM3.START_DATE||A_SYSTEM3.END_DATE|| A_SYSTEM3.DELIMT||";
String select23="A_SYSTEM4.CRUD_LEN|| A_SYSTEM4.BO_OBJECT|| A_SYSTEM4.ACTION||
A_SYSTEM4.GROUP_NO||A_SYSTEM4.NO_ROWS|| A_SYSTEM4.NO_ROW_FOUND||
A_SYSTEM4.SYS_DFLT_ID||";
String select24="A_SYSTEM4.SYS_DFLT_ID||A_SYSTEM4.PARTY_ID||
A_SYSTEM4.PARTY_TYPE||A_SYSTEM4.EXT_PARTY_REF|| A_SYSTEM4.EXT_SYS_ID||
A_SYSTEM4.START_DATE||";
String select25="A_SYSTEM4.END_DATE|| A_SYSTEM4.DELIMT||A_PARTY.CRUD_LEN||
A_PARTY.BO_OBJECT|| A_PARTY.ACTION|| A_PARTY.GROUP_NO||";
String select26="A_PARTY.NO_ROWS|| A_PARTY.NO_ROW_FOUND||
A_PARTY.PARTY_NAME_ID|| A_PARTY.SEC_NO||";
String select27="A_PARTY.ROLE_PLAYER_ID|| A_PARTY.PARTY_TYPE||
A_PARTY.USAGE_SEQ_NO||";
String select28="A_PARTY.SUFFIX_TITLE|| A_PARTY.DFLT_FLG|| A_PARTY.LAST_NAME||
A_PARTY.ADNL_NAME||";
String select29="A_PARTY.PREF_TITLE|| A_PARTY.FIRST_NAME|| A_PARTY.START_DATE||
A_PARTY.END_DATE||A_PARTY.DELIMT||";
String select30="A_POSTCONTACT.CRUD_LEN|| A_POSTCONTACT.BO_OBJECT||
A_POSTCONTACT.ACTION||";
String select31="A_POSTCONTACT.GROUP_NO|| A_POSTCONTACT.NO_ROWS||
A_POSTCONTACT.NO_ROW_FOUND||";
String select32 = "A_POSTCONTACT.CONTACT_ID || A_POSTCONTACT.SEQ_NO ||
A_POSTCONTACT.CONTACT_TYPE ||";
String select33 = "A_POSTCONTACT.TOWN || A_POSTCONTACT.POST_CDE ||
A_POSTCONTACT.COUNTRY ||";
String select34 = "A_POSTCONTACT.STATE || A_POSTCONTACT.CONTACT_STR ||
A_POSTCONTACT.CNTRY_CDE ||";
String select35 = "A_POSTCONTACT.AREA_CDE || A_POSTCONTACT.NO_BODY ||
A_POSTCONTACT.EXTN ||";
String select36 = "A_POSTCONTACT.ADDR1 || A_POSTCONTACT.ADDR2 ||
A_POSTCONTACT.START_DATE ||";
String select37 = "A_POSTCONTACT.END_DATE || A_POSTCONTACT.DELIMT ||";
String select38 = "A_POSTRPCONTACT.CRUD_LEN || A_POSTRPCONTACT.BO_OBJECT ||
A_POSTRPCONTACT.ACTION ||";
String select39 = "A_POSTRPCONTACT.GROUP_NO || A_POSTRPCONTACT.NO_ROWS ||
A_POSTRPCONTACT.NO_ROW_FOUND ||";
String select40 = "A_POSTRPCONTACT.ROLE_PLYER_ID ||
A_POSTRPCONTACT.NATURE_1100 ||";
String select41 = "A_POSTRPCONTACT.CONTACT_ID || A_POSTRPCONTACT.PARTY_TYPE ||
A_POSTRPCONTACT.PARTY_NAME_ID || A_POSTRPCONTACT.DFLT_FLG ||";
String select42 = "A_POSTRPCONTACT.USAGE_SEQ_NO ||
A_POSTRPCONTACT.CONTACT_TYPE ||";
String select43 = "A_POSTRPCONTACT.START_DATE || A_POSTRPCONTACT.END_DATE ||
A_POSTRPCONTACT.DELIMT ||";
String select44 = "A_TELCONTACT.CRUD_LEN || A_TELCONTACT.BO_OBJECT ||
A_TELCONTACT.ACTION ||";
String select45 = "A_TELCONTACT.GROUP_NO || A_TELCONTACT.NO_ROWS ||
A_TELCONTACT.NO_ROW_FOUND ||";
String select46 = "A_TELCONTACT.CONTACT_ID || A_TELCONTACT.SEQ_NO ||
A_TELCONTACT.CONTACT_TYPE ||";
String select47 = "A_TELCONTACT.TOWN || A_TELCONTACT.POST_CDE ||
A_TELCONTACT.COUNTRY ||";
String select48 = "A_TELCONTACT.STATE || A_TELCONTACT.CONTACT_STR ||
A_TELCONTACT.CNTRY_CDE ||";
String select49 = "A_TELCONTACT.AREA_CDE || A_TELCONTACT.NO_BODY ||
A_TELCONTACT.EXTN ||";
String select50 = "A_TELCONTACT.ADDR1 || A_TELCONTACT.ADDR2 ||
A_TELCONTACT.START_DATE ||";
String select51 = "A_TELCONTACT.END_DATE || A_TELCONTACT.DELIMT ||";
String select52 = "A_TELRPCONTACT.CRUD_LEN || A_TELRPCONTACT.BO_OBJECT ||
A_TELRPCONTACT.ACTION ||";
String select53 = "A_TELRPCONTACT.GROUP_NO || A_TELRPCONTACT.NO_ROWS ||
A_TELRPCONTACT.NO_ROW_FOUND ||";
String select54 = "A_TELRPCONTACT.ROLE_PLYER_ID ||
A_TELRPCONTACT.NATURE_1100 ||";
String select55 = "A_TELRPCONTACT.CONTACT_ID || A_TELRPCONTACT.PARTY_TYPE ||
A_TELRPCONTACT.PARTY_NAME_ID || A_TELRPCONTACT.DFLT_FLG ||";
String select56 = "A_TELRPCONTACT.USAGE_SEQ_NO ||
A_TELRPCONTACT.CONTACT_TYPE ||";
String select59="A_TELERPCONTACT.START_DATE|| A_TELERPCONTACT.END_DATE||
A_TELERPCONTACT.DELIM||";
String select60="A_ACCSUM1.CRUD_LEN|| A_ACCSUM1.BO_OBJECT|| A_ACCSUM1.ACTION||
A_ACCSUM1.GROUP_NO||";
String select61="A_ACCSUM1.NO_ROWS|| A_ACCSUM1.NO_ROW_FOUND||
A_ACCSUM1.ACCOUNT_ID|| A_ACCSUM1.SEQ_NO||";
String select62="A_ACCSUM1.ACCOUNT_NO|| A_ACCSUM1.ACCOUNT_TYPE||
A_ACCSUM1.ACCOUNT_NAME||";
String select63="A_ACCSUM1.ATM_FLG|| A_ACCSUM1.CREDIT_FLG||
A_ACCSUM1.CHECK_FLG||";
String select64="A_ACCSUM1.STAT_ACCT|| A_ACCSUM1.PRODUCT_ID||
A_ACCSUM1.PRODUCT_DESC||";
String select65="A_ACCSUM1.CATEGORY|| A_ACCSUM1.ACC_LIMIT||
A_ACCSUM1.APP_STATUS||";
String select66="A_ACCSUM1.START_DATE|| A_ACCSUM1.END_DATE||
A_ACCSUM1.DELIM||";
String select67="A_ACCSUM2.CRUD_LEN|| A_ACCSUM2.BO_OBJECT|| A_ACCSUM2.ACTION||
A_ACCSUM2.GROUP_NO||";
String select68="A_ACCSUM2.NO_ROWS|| A_ACCSUM2.NO_ROW_FOUND||
A_ACCSUM2.ACCOUNT_ID||";
String select69="A_ACCSUM2.SEQ_NO|| A_ACCSUM2.ACCOUNT_NO||
A_ACCSUM2.ACCOUNT_TYPE||";
String select70="A_ACCSUM2.ACCOUNT_NAME|| A_ACCSUM2.ATM_FLG||
A_ACCSUM2.CREDIT_FLG||";
String select71="A_ACCSUM2.CHECK_FLG|| A_ACCSUM2.STAT_ACCT||
A_ACCSUM2.PRODUCT_ID||";
String select72="A_ACCSUM2.PRODUCT_DESC|| A_ACCSUM2.CATEGORY||
A_ACCSUM2.ACC_LIMIT||";
String select73="A_ACCSUM2.APP_STATUS|| A_ACCSUM2.START_DATE||
A_ACCSUM2.END_DATE|| A_ACCSUM2.DELIM||";
String select74="A_ACCSUM3.CRUD_LEN|| A_ACCSUM3.BO_OBJECT|| A_ACCSUM3.ACTION||
A_ACCSUM3.GROUP_NO||";
String select75="A_ACCSUM3.NO_ROWS|| A_ACCSUM3.NO_ROW_FOUND||
A_ACCSUM3.ACCOUNT_ID||";
String select76="A_ACCSUM3.SEQ_NO|| A_ACCSUM3.ACCOUNT_NO||
A_ACCSUM3.ACCOUNT_TYPE||";
String select77="A_ACCSUM3.ACCOUNT_NAME|| A_ACCSUM3.ATM_FLG||
A_ACCSUM3.CREDIT_FLG||";
String select78="A_ACCSUM3.CHECK_FLG|| A_ACCSUM3.STAT_ACCT||
A_ACCSUM3.PRODUCT_ID||";
String select79="A_ACCSUM3.PRODUCT_DESC|| A_ACCSUM3.CATEGORY||
A_ACCSUM3.ACC_LIMIT||";
String select80="A_ACCSUM3.APP_STATUS|| A_ACCSUM3.START_DATE||
A_ACCSUM3.END_DATE|| A_ACCSUM3.DELIM||";
String select81="A_ACCSUM4.CRUD_LEN|| A_ACCSUM4.BO_OBJECT|| A_ACCSUM4.ACTION||
A_ACCSUM4.GROUP_NO||";
String select82="A_ACCSUM4.NO_ROWS|| A_ACCSUM4.NO_ROW_FOUND||
A_ACCSUM4.ACCOUNT_ID||";

Appendix C. Data models, table/file definitions, et al used in the DFC Customer Insight business solution
String select83="A_ACCSUM4.SEQ_NO|| A_ACCSUM4.ACCOUNT_NO||
A_ACCSUM4.ACCOUNT_TYPE||";
String select84="A_ACCSUM4.ACCOUNT_NAME|| A_ACCSUM4.ATM_FLG||
A_ACCSUM4.CREDIT_FLG||";
String select85="A_ACCSUM4.CHECK_FLG|| A_ACCSUM4.STAT_ACCT||
A_ACCSUM4.PRODUCT_ID||";
String select86="A_ACCSUM4.PRODUCT_DESC|| A_ACCSUM4.CATEGORY||
A_ACCSUM4.ACC_LIMIT||";
String select87="A_ACCSUM4.APP_STATUS|| A_ACCSUM4.START_DATE||
A_ACCSUM4.END_DATE|| A_ACCSUM4.DELIMT||";
String select88="A_BALSUM1.CRUD_LEN|| A_BALSUM1.BO_OBJECT|| A_BALSUM1.ACTION||
A_BALSUM1.GROUP_NO||";
String select89="A_BALSUM1.NO_ROWS|| A_BALSUM1.NO_ROW_FOUND||
A_BALSUM1.BALANCE_ID||";
String select90="A_BALSUM1.SEQ_NO|| A_BALSUM1.AGREEMENT_ID||
A_BALSUM1.BALANCE_TYPE||";
String select91="A_BALSUM1.DEBIT_IND|| A_BALSUM1.AMOUNT||
A_BALSUM1.ACCTING_DATE||";
String select92="A_BALSUM1.INTRATE_DEBIT|| A_BALSUM1.INTRATE_CREDIT||
A_BALSUM1.EXPIRY_DATE||";
String select93="A_BALSUM1.START_DATE|| A_BALSUM1.END_DATE||
A_BALSUM1.DELIMT||";
String select94="A_BALSUM2.CRUD_LEN|| A_BALSUM2.BO_OBJECT|| A_BALSUM2.ACTION||
A_BALSUM2.GROUP_NO||";
String select95="A_BALSUM2.NO_ROWS|| A_BALSUM2.NO_ROW_FOUND||
A_BALSUM2.BALANCE_ID||";
String select96="A_BALSUM2.SEQ_NO|| A_BALSUM2.AGREEMENT_ID||
A_BALSUM2.BALANCE_TYPE||";
String select97="A_BALSUM2.DEBIT_IND|| A_BALSUM2.AMOUNT||
A_BALSUM2.ACCTING_DATE||";
String select98="A_BALSUM2.INTRATE_DEBIT|| A_BALSUM2.INTRATE_CREDIT||
A_BALSUM2.EXPIRY_DATE||";
String select99="A_BALSUM2.START_DATE|| A_BALSUM2.END_DATE||
A_BALSUM2.DELIMT||";
String select100="A_BALSUM3.CRUD_LEN|| A_BALSUM3.BO_OBJECT|| A_BALSUM3.ACTION||
A_BALSUM3.GROUP_NO||";
String select101="A_BALSUM3.NO_ROWS|| A_BALSUM3.NO_ROW_FOUND||
A_BALSUM3.BALANCE_ID||";
String select102="A_BALSUM3.SEQ_NO|| A_BALSUM3.AGREEMENT_ID||
A_BALSUM3.BALANCE_TYPE||";
String select103="A_BALSUM3.DEBIT_IND|| A_BALSUM3.AMOUNT||
A_BALSUM3.ACCTING_DATE||";
String select104="A_BALSUM3.INTRATE_DEBIT|| A_BALSUM3.INTRATE_CREDIT||
A_BALSUM3.EXPIRY_DATE||";
String select105="A_BALSUM3.START_DATE|| A_BALSUM3.END_DATE||
A_BALSUM3.DELIMT||";
String select106="A_BALSUM4.CRUD_LEN|| A_BALSUM4.BO_OBJECT|| A_BALSUM4.ACTION||
A_BALSUM4.GROUP_NO||";
String select107="A_BALSUM4.NO_ROWS|| A_BALSUM4.NO_ROW_FOUND||
A_BALSUM4.BALANCE_ID||";
String select108="A_BALSUM4.SEQ_NO|| A_BALSUM4.AGREEMENT_ID||
A_BALSUM4.BALANCE_TYPE||";
String select109="A_BALSUM4.DEBIT_IND|| A_BALSUM4.AMOUNT||
A_BALSUM4.ACCTING_DATE||";
String select110="A_BALSUM4.INTRATE_DEBIT|| A_BALSUM4.INTRATE_CREDIT||
A_BALSUM4.EXPIRY_DATE||";
String select111="A_BALSUM4.START_DATE|| A_BALSUM4.END_DATE||
A_BALSUM4.DELIMT"

String select112=" FROM DB2ADMIN.GHEADER AS GHEADER, DB2ADMIN.A_PERSON AS 
A_PERSON,";

String select113="DB2ADMIN.A_SYSTEM1 AS A_SYSTEM1, DB2ADMIN.A_SYSTEM2 AS 
A_SYSTEM2,";
String select114="DB2ADMIN.A_SYSTEM3 AS A_SYSTEM3, DB2ADMIN.A_SYSTEM4 AS 
A_SYSTEM4,";
String select115="DB2ADMIN.A_PARTY AS A_PARTY, DB2ADMIN.A_POSTCONTACT AS 
A_POSTCONTACT,"
String select116="DB2ADMIN.A_POSTRPCONTACT AS A_POSTRPCONTACT, 
DB2ADMIN.A_TELCONTACT AS A_TELCONTACT,";
String select117="DB2ADMIN.A_TELERPCONTACT AS A_TELERPCONTACT, 
DB2ADMIN.A_ACCSUM1 AS A_ACCSUM1,";
String select118="DB2ADMIN.A_ACCSUM2 AS A_ACCSUM2, DB2ADMIN.A_ACCSUM3 AS 
A_ACCSUM3,";
String select119="DB2ADMIN.A_ACCSUM4 AS A_ACCSUM4, DB2ADMIN.A_BALSUM1 AS 
A_BALSUM1,";
String select120="DB2ADMIN.A_BALSUM2 AS A_BALSUM2, DB2ADMIN.A_BALSUM3 AS 
A_BALSUM3,";
String select121="DB2ADMIN.A_BALSUM4 AS A_BALSUM4"

String select122=" WHERE GHEADER.HEADER_ID = A_PERSON.HEADER_ID"

String select123=" AND GHEADER.HEADER_ID = A_SYSTEM1.HEADER_ID"
String select124=" AND GHEADER.HEADER_ID = A_SYSTEM2.HEADER_ID"
String select125=" AND GHEADER.HEADER_ID = A_SYSTEM3.HEADER_ID"
String select126=" AND GHEADER.HEADER_ID = A_SYSTEM4.HEADER_ID"
String select127=" AND GHEADER.HEADER_ID = A_PARTY.HEADER_ID"
String select128=" AND GHEADER.HEADER_ID = A_POSTCONTACT.HEADER_ID"
String select129=" AND GHEADER.HEADER_ID = A_POSTRPCONTACT.HEADER_ID"
String select130=" AND GHEADER.HEADER_ID = A_TELCONTACT.HEADER_ID"
String select131=" AND GHEADER.HEADER_ID = A_TELERPCONTACT.HEADER_ID"
String select132=" AND GHEADER.HEADER_ID = A_ACCSUM1.HEADER_ID"
String select133=" AND GHEADER.HEADER_ID = A_ACCSUM2.HEADER_ID"
String select134=" AND GHEADER.HEADER_ID = A_ACCSUM3.HEADER_ID"
String select135=" AND GHEADER.HEADER_ID = A_ACCSUM4.HEADER_ID"
String select136=" AND GHEADER.HEADER_ID = A_BALSUM1.HEADER_ID"
String select137=" AND GHEADER.HEADER_ID = A_BALSUM2.HEADER_ID";
String select138=" AND GHEADER.HEADER_ID = A_BALSUM3.HEADER_ID";
String select139=" AND GHEADER.HEADER_ID = A_BALSUM4.HEADER_ID";
String 
select=select1+select2+select3+select4+select5+select6+select7+select8+select9+
select10+select11+select12+select13+select14+select15+select16+select17+select1
8+select19+select20+select21+select22+select23+select24+select25+select26+selec
t27+select28+select29+select30+select31+select32+select33+select34+select35+sel
ect36+select37+select38+select39+select40+select41+select42+select43+select44+s
elect45+select46+select47+select48+select49+select50+select51+select52+select53
+select54+select55+select56+select57+select58+select59+select60+select61+select
62+select63+select64+select65+select66+select67+select68+select69+select70+sele
ct71+select72+select73+select74+select75+select76+select77+select78+select79+se
lect80+select81+select82+select83+select84+select85+select86+select87+select88+
select89+select90+select91+select92+select93+select94+select95+select96+select9
7+select98+select99+select100+select101+select102+select103+select104+select105
+select106+select107+select108+select109+select110+select111+select112+select11
3+select114+select115+select116+select117+select118+select119+select120+select1
21+select122+select123+select124+select125+select126+select127+select128+select
129+select130+select131+select132+select133+select134+select135+select136+selec
t137+select138+select139;
System.out.println("SQL:"+select);

PreparedStatement pstmt = con2.prepareStatement(select);
ResultSet rset = pstmt.executeQuery();
while(rset.next()) {
    System.out.println(rset.getString(1));
    // Create a Store procedure Statement object so we can execute
    CallableStatement cs2 = con2.prepareCall(CIISStoreProcedure);
    //cs2.setString(1, strAddPerson);
    cs2.setString(1, rset.getString(1));
    cs2.registerOutParameter(1,Types.VARCHAR);

    // Submit a query, creating a ResultSet object
    cs2.execute();
    String answer = cs2.getString(1);
    System.out.println("Add Person = " + answer);
System.out.println(answer.substring(0,10000));
System.out.println("="); 

cs2.close();
}
rset.close();
pstmt.close();
    // con2.close();
    
}
catch (Exception e)
private boolean checkForWarning (SQLException warn) throws SQLException {
    boolean rc = false;

    // If a SQLWarning object was given, display the
    // warning messages.  Note that there could be
    // multiple warnings chained together
    if (warn != null) {
        System.out.println ("\n *** Warning ***\n");
        rc = true;
        while (warn != null) {
            System.out.println ("SQLState: " +
                   warn.getSQLState ());
            System.out.println ("Message:  " +
                   warn.getMessage ());
            System.out.println ("Vendor:   " +
                   warn.getErrorCode ());
            System.out.println ("\n");
            warn = warn.getNextWarning ();
        }
    }
    return rc;
}

Attention: As mentioned earlier, for performance reasons, direct loading of the underlying DB2 tables is recommended when very large volumes are involved. However, considerable care should be taken to ensure that appropriate steps are taken to ensure the integrity and consistency of the loaded data. For further details, please refer to the CIIS documentation.
Incremental update of CIIS objects

Update of the CIIS objects have to occur when:

- Certain changes occur to the operational systems that need to be reflected in CIIS.
- Results of analysis and mining of the data warehouse requires the status of a customer to be updated in the CIIS.

In our DFC Customer Insight solution, we chose to directly update the customer status field in the underlying DB2 table of the CIIS based on analysis of the data warehouse. The assumption is that the changes have been captured and loaded onto a separate table. We used Ascential DataStage to perform this update as shown in Figure C-22 through Figure C-26 on page 298. Since this update only affects a single table in the CIIS, there are no consistency concerns of directly updating the underlying DB2 table.

![DataStage Designer](image)

*Figure C-22  Incremental update of CIIS*
Figure C-23  Update CIIS Extract Stage
Figure C-24  Update CIIS Transform Stage
Figure C-25  Update CIIS Load Stage - Update existing rows only

Note the **Update action** for the load as being Update existing rows only in Figure C-25.
Figure C-26  Update CIIS Load Stage
Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 301. Note that some of the documents referenced here may be available in softcopy only.

For more information on applying pattern approaches:

- *Applying Pattern Approaches*, SG24-6805 (external version)

For more information on the Self-Service business pattern:

- *Patterns: Service Oriented Architecture and Web Services*, SG24-6303
- *Patterns: Self-Service Application Solutions Using WebSphere V5.0*, SG24-6591
- *Patterns: iSeries: Connecting Self-Service Applications to the Enterprise - WebSphere V5*, REDP3670
- Patterns: Self-Service Application Solutions Using WebSphere for z/OS V5, SG24-7092
- *Self-Service Applications using IBM WebSphere V5.0 and WebSphere MQ Integrator V2.1 Patterns for e-business Series*, SG24-6875
- *Patterns: Connecting Self-Service applications to the Enterprise*, SG24-6572
- *Patterns: Connecting Self-Service applications to the Enterprise, iSeries*, REDP0213
- *Patterns: Connecting Self-Service applications to the Enterprise, zSeries®,* SG24-6827
- *Self-Service Patterns using WebSphere Application V4.0*, SG24-6175
- *Self-Service Applications Using IBM WebSphere V4.0 and MQSeries Integrator*, SG24-6160-01

For more information on the Access Integration pattern:

- *Patterns: Pervasive Portals patterns for e-business series*, SG24-6876
- **Access Integration pattern Using WebSphere Portal Server, SG24-6267**
- **Mobile Application with IBM WebSphere Everyplace® Access Design and Development, SG24-6259**

For more information on Extended Enterprise and Application Integration business pattern:
- **Patterns: Direct Connections for Intra-and-Inter-enterprise Applying the Patterns approach, REDP3754**
- **Patterns: Direct Connections for Intra-and Inter-enterprise, SG24-6933**
- **Patterns: Broker Interactions for Intra - Inter Enterprise, SG24-6075**
- **Patterns: Serial and Parallel Processes for WebSphere Process Choreographer and WebSphere MQ Workflow, SG24-6306**
- **Serial Managed Process Flows for Intra- and Inter-enterprise, SG24-6305**

For more information on the Composite pattern:
- **Patterns: A Portal composite pattern using WebSphere Portal V5, SG24-6077**
- **Patterns: A Portal composite pattern using WebSphere Portal V4.1, SG24-6869**
- **B2B e-commerce With WebSphere Business Edition V5.4, SG24-6194**
- **WebSphere Commerce Portal V5.4 Solutions, SG24-6890**
- **WebSphere Commerce Edition V5.4 Catalog and Content Management**
- **e-Commerce Patterns for Building B2C Web Sites Using IBM WebSphere Commerce Suite V5.1, SG24-6180**
- **e-commerce Patterns for z/Linux using WebSphere Commerce Suite V5.1 Patterns for e-business Series, REDP0411**
- **WebSphere Commerce Suite V5.1 for i/Series Implementation and Deployment Guide, REDP0159**

For more information on Custom designs:
- **IBM WebSphere V5 Edge of Network Patterns, SG24-6896**
- **Patterns: Portal Search Custom Design, SG24-6881**
- **Patterns for the Edge of Network, SG24-6822**
- **Custom designs for Domino and WebSphere Integration, SG24-6903**
Older patterns:
- Applying the Patterns for e-business to Domino and WebSphere Scenarios, SG24-6255
- User-to-Business Pattern Using WebSphere Personalization, SG24-6213
- User-to-Business Patterns Using WebSphere Enterprise Edition, SG24-6151
- User-to-Business Patterns Using WebSphere Advanced and MQSI, SG24-6160
- User-to-Business Patterns Using WebSphere Commerce Suite, SG24-6156

Other publications
These publications are also relevant as further information sources:

Online resources
These Web sites and URLs are also relevant as further information sources:
- The Patterns for e-business Web site at:

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
Index

Numerics
80/20 situation 2

A
Access Integration patterns 124
ACCOUNTING_STRING 214
actors
Brokerage application 121
Browser 121
Checkings and Savings application 121
Credit Card application 121
Customer Service Representative 121
Internet 121
Knowledge users 121
Loans application 121
Rewards application 121
Agent
Federation variation application pattern 112,
127, 129, 135–136
Federation variation pattern 111
Federation variation runtime pattern 113
Agent IT drivers 127
Agent tier 127
analysis and mining 128, 131
Application Integration
Federation 124
Application Integration patterns 124–125
Application patterns 3, 9, 116, 126, 147
Single Sign-on and Role-Based Access 129
Application tier 129
Apply 77
Apply step 78
ASCII encoding 41
audit trail 145

B
Best practices 3, 14
blank padding 49
Business and Integration patterns overview 59
Business drivers 143
business functions
Analyze and mine 122
Customer Information File (CIF) 121
Data mart 121
Data warehouse 121
Record customer interaction 122
Search on lastname, city and zipcode 121
Stock purchase/sale 122
Transfer assets between accounts 122
Update CIF 122
Update customer metadata and relationships
122
View customer metadata and transaction history
122
Business patterns 2, 4–5, 116, 122
Extended Enterprise 7
Self-Service 4
business requirements 116
Business, Integration, and related Application pat-
terns 61

C
cache 27
change capture 145
CIF 122, 126–127, 130, 136
CIIS 144–145
Client Information Integration Solution 144
Client tier 129
collating sequence 47
COLLATING_SEQUENCE 41, 48, 50
collocate 145
COMM_RATE 48
compensation 29
Composite patterns 3–4, 8, 116, 126
configuring data sources
DB2 UDB for z/OS data source 206, 219
Oracle 9i 166
XML data source 182
Connectivity 173, 210
connectors 122
CPU_RATIO 48
CSR 118, 122, 126, 136, 143
CSR scenario 147
Custom design 6
customer information file 118
Customer Insight solution 115, 117, 119, 122, 124, 126, 134, 141
customer service representative 118

D
Data consolidation or placement 22–23
data federation 20, 26, 35
Data Integration 62
  Federation 65
  Population 72
  Two-way Synchronization 93
Data Integration application patterns 63
Data Integration node 68, 91
Data Integration patterns 147
data marts 118, 122, 128, 145
data movement 135
Data placement 47
Data Server/Services node 69, 75
Data type mappings 43
data warehouse 118, 122, 126, 128, 130, 145
data-focused Application Integration 62
DataStage 144, 146
DB2 Enterprise Server Edition 34
DB2 for Multiplatforms user mappings 222
DB2 for Multiplatforms wrapper 220
DB2 for z/OS nicknames 215, 224
DB2 for z/OS server 209, 220
DB2 for z/OS user mappings 212
DB2 for z/OS wrapper 208
DB2 II V8.1 26, 29
  components 34
data sources supported 30
data types supported 31
  overview 26
DB2 Information Integration overview 21
DB2 Information Integrator 145–146
  creating Oracle nicknames 177
  creating the ODBC nickname 202
  creating the ODBC server 198
  creating the ODBC wrapper 196
  creating the XML nicknames 186
  creating the XML server 185
DB2 Information Integrator (DB2 II) 25
DB2 Information Integrator Classic Federation for z/OS (DB2 IICF) 26
DB2 Information Integrator for Content (DB2 IIC) 25
DB2 Information Integrator portfolio 24
DB2 Net Search Extender 35
DB2 on AIX
  creating DB2 for Multiplatforms nicknames 224
  creating DB2 for z/OS nicknames 215
  creating DB2 for z/OS user mappings 212, 222
  creating the DB2 for z/OS server 209
  creating the DB2 for z/OS wrapper 208
  integrating DB2 for z/OS 206, 219
DB2 optimizer 50, 52
DB2 UDB for z/OS data source 206, 219
DB2 UDB for z/OS server definition 209
DB2 UDB for z/OS server options 210
DB2_MAXIMAL_PUSHDOWN 41–42, 174, 212
DFC 115, 117, 119
Distributed access 22, 24
Druid Financial Corporation 115

E
EBCDIC encoding 41
EII 22
enterprise information integration (EII) 22

F
FEDERATED 38
federated database 27, 50, 184, 196, 208
federated global catalog 215, 225
federated server 48, 51
federated system 27, 38, 50, 176
  configuring a data source 40
Federation 24, 63, 130
  Cache variation pattern 69
Federation application pattern 66, 132, 135, 137
Federation pattern 66
Federation runtime pattern 68
flat files 183
Function mappings 44, 48

G
Gather 77
Gather 1 81, 83, 131
Gather 2 85
Gather step 78
get_stats 53
global catalog 35–36, 51–52
global catalog views 36
GRANT 51
Grid computing 19
Guidelines 3, 14

H
holistic view 117–118

I
incremental update 141
index specification 51–52
indexes 50–51, 53, 56
Information Aggregation

User Information Access 98
Information Aggregation application patterns 64
Information Aggregation patterns 122–123, 125,
128, 147
information integration 20–22
Integrated Runtime environment 141
integration
Application connectivity 19
Build to integrate 19
Information integration 19
Process integration 19
User interaction 19
Integration patterns 3, 5, 116
IO_RATIO 48
IT architect 2
IT drivers 124

J
join 38, 45–47

L
LIKE 41
Lotus Extended Search 45

M
metadata 128
Microsoft Excel files 183
middleware 135
MQT 46, 54
Multi Step Federated Gather variation 92
Multi Step Gather and Process variations 92
Multi Step Gather variation 91
Multi Step Process variation 92
Multi Step variation 91

N
native API 145
nicknames 36, 51–53, 55, 145
non relational wrappers 35
NUMERIC_STRING 50

O
ODBC additional server options 200
ODBC nickname 202
ODBC server definition 200
ODBC server options 200
ODBC servers 198
ODBC wrapper 196, 200
ODS 128
on demand 18
Automation 18
definition 18
Integration 18
Virtualization 18
operational systems 122, 126, 130
Optimization 173, 211
Oracle
COLLATE USING IDENTITY 41
Oracle 9i data source 166
Oracle additional server options 174
Oracle Client 167
Oracle server definition 169
Oracle server options 173
Oracle user mappings 174
Oracle wrapper 168

P
passthru 43
Passthru privileges 37
Patterns for e-business 1, 116
Application patterns 3, 9
Best practices 3, 14
Business patterns 2, 5
Composite patterns 3, 8
Guidelines 3, 14
Integration patterns 3, 5
layered asset model 2
Product Mappings 3, 13
Runtime patterns 3, 11
Web site 4
performance
federated server options 48
performance considerations 45
performance factors 46
PLAN_HINTS 174
Population 63
  Multi Step Federated Gather variation application pattern 89
  Multi Step Federated Gather variation pattern 88
  Multi Step Federated Gather variation runtime pattern 90
  Multi Step Gather variation application pattern 80, 130, 136, 140
  Multi Step Gather variation pattern 80
  Multi Step Gather variation runtime pattern 84
  Multi Step Process variation application pattern 85
  Multi Step Process variation pattern 85
  Multi Step Process variation runtime pattern 87
  Multi Step variation application pattern 76, 129, 135, 139
  Multi Step variation pattern 76
  Multi Step variation runtime pattern 79
Population application pattern 73, 91, 131, 135, 138
Population pattern 73
Population runtime pattern 75
portal 126, 145
portlets 145
Presentation tier 127
Process 77
Process Integration 62
Process node 91
Process step 78
process-focused Application Integration 62
Product Mappings 3, 13, 116, 142, 147
PUSHDOWN 41, 47, 199
pushdown 46–47, 173, 211
pushdown analysis 44, 46–47
pushdown factors 47

Q
query rewrite 46
Query, Analyze and Search tier 128

R
Redbooks Web site 301
  Contact us xxv
relational wrappers 34
reuse of assets 2

runstats 53
Runtime patterns 3, 11, 116, 135, 147
  Federation 137

S
Self Service
  Agent pattern 110
  Self-Service pattern 122–123, 125
  semantics 131
SET SERVER OPTION 42
SOD 116, 120, 123, 125
Solution Overview Diagram 116, 120
statistics 50–53, 56
stovepipe operational systems 119
SYSCAT.COLOPTIONS 37
SYSCAT.COLUMNS 37
SYSCAT.FUNCMAPOPTIONS 37
SYSCAT.FUNCMAPPARMOPTIONS 37
SYSCAT.FUNCMAPPINGS 37, 44
SYSCAT.FUNCTIONS 37
SYSCAT.INDEXES 37
SYSCAT.INDEXOPTIONS 37
SYSCAT.KEYCOLUSE 37
SYSCAT.PASSTHRUAUTH 37
SYSCAT.ROUTINES 37
SYSCAT.SERVEROPTIONS 36
SYSCAT.SERVERS 36
SYSCAT.TABLES 37
SYSCAT.TABOPTIONS 37
SYSCAT.TYPEMAPPINGS 37
SYSCAT.WRAPOPTIONS 36
SYSCAT.WRAPPERS 36
SYSSTAT 52
SYSSTAT.COLUMNS 37, 53
SYSSTAT.INDEXES 37, 53
SYSSTAT.TABLES 37, 52

T
table structured files 183
TCO 117, 127
Temporary/Persistent Store 82
tnsnames.ora 167
Two-way Synchronization 63
  Multi Step variation application pattern 96
  Multi Step variation pattern 96
  Multi Step variation runtime pattern 98
Two-way Synchronization application pattern 94
Two-way Synchronization pattern  94
Two-way Synchronization runtime pattern  96

U
UIA  128
  Federation variation application pattern  104
  Federation variation runtime pattern  106
  Write-back variation application pattern  106
  Write-back variation runtime pattern  108
UIA application pattern  132
User Information Access
  Federation variation pattern  104
  Write-back variation pattern  106
User Information Access (UIA) application pattern  135–136
User Information Access application pattern  100
User Information Access pattern  100
User Information Access runtime pattern  102
user mapping  42

V
VARCHAR_NO_TRAILING_BLANKS  49–50
VARCHAR2  49

W
wrappers  28
  Communication with the data source  28
  data modelling  29
  development kit  34
  Federated object registration  28
  Services and operations  29
  wrapper module  34
write-back  87

X
XML  145
XML data source  182
XML files  183
XML nickname  192
XML nicknames  186
XML server  185
XML server definition  185
XML wrapper  183, 186
Patterns: Information Aggregation and Data Integration with DB2 Information Integrator
This IBM Redbook documents and prototypes the role of DB2 Information Integrator technologies and architecture in IBM’s Patterns for e-business using a typical customer insight e-business scenario. It is aimed at an audience of IT architects and data integration professionals responsible for developing e-business solutions that integrate processes and data from multiple distributed heterogeneous systems.

This publication provides an overview of IBM’s Patterns for e-business, and the DB2 Information Integration architecture. It also describes the various application and runtime patterns identified to date for Information Aggregation and Data Integration, as defined in "Patterns for e-business".

Using a typical customer insight e-business scenario, this publication documents the step-by-step patterns approach to implementing the e-business solution. At each level in the hierarchy, each available pattern is evaluated and chosen before drilling down to the next lower layer where the pattern evaluation and selection process is repeated. The final drill down leads to product selection and implementation information.