Programming
Microsoft
SQL Server 2008

Leonard Lobel
Andrew J. Brust
Stephen Forte

twentysix
NEW YORK
To my partner, Mark, and our children, Adam, Jacqueline, and Joshua, for standing by me through every one of life’s turns.

—Leonard Lobel

To my wife, Lauren, and my sons, Sean and Miles. Thank you for your love, your support, and your accommodation of the unreasonable.

—Andrew Brust

To Kathleen, thanks for your support and making me run marathons, which are more painful than book writing and building beta machines.

—Stephen Forte
# Contents at a Glance

## Part I  **Core Fundamentals**
1. Overview .......................................................... 3  
2. T-SQL Enhancements ............................................... 13  
3. Exploring SQL CLR .................................................. 111  
4. Server Management .................................................. 161  
5. Security in SQL Server 2008 ........................................ 189  

## Part II  **Beyond Relational**
6. XML and the Relational Database ............................... 231  
7. Hierarchical Data and the Relational Database ............... 281  
8. Using FILESTREAM for Unstructured Data Storage .......... 307  
9. Geospatial Data Types ............................................... 341  

## Part III  **Reach Technologies**
10. The Microsoft Data Access Machine .......................... 377  
11. The Many Facets of .NET Data Binding ....................... 419  
12. Transactions ........................................................ 449  
13. Developing Occasionally Connected Systems ................ 491  

## Part IV  **Business Intelligence**
14. Data Warehousing .................................................. 563  
15. Basic OLAP .......................................................... 611  
16. Advanced OLAP ..................................................... 639  
17. OLAP Queries, Tools, and Application Development ........ 717  
18. Expanding Your Business Intelligence with Data Mining .... 793  
19. Reporting Services .................................................. 879
# Table of Contents

Acknowledgments ................................................................. xxii
Introduction ........................................................................ xxv

## Part I  Core Fundamentals

### 1  Overview ................................................................. 3

- Just How Big Is It? ........................................................ 3
- A Book for Developers .................................................. 5
- A Book by Developers .................................................. 6
- A Book to Show You the Way.
  - Core Technologies ....................................................... 7
  - Beyond Relational ....................................................... 8
  - Reaching Out ............................................................ 9
  - Business Intelligence Strategies ................................. 10
- Summary ........................................................................ 12

### 2  T-SQL Enhancements ............................................... 13

- Common Table Expressions ......................................... 14
- Creating Recursive Queries with CTEs ......................... 18
- The PIVOT and UNPIVOT Operators ............................ 21
- Using UNPIVOT .......................................................... 22
- Dynamically Pivoting Columns .................................... 23
- The APPLY Operator .................................................. 25
- TOP Enhancements ....................................................... 26
- Ranking Functions ....................................................... 28
  - The ROW_NUMBER Function ..................................... 28
  - The RANK Function .................................................. 32
  - The DENSE_RANK and NTILE Functions .................. 34

---

**What do you think of this book? We want to hear from you!**

Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

[www.microsoft.com/learning/booksurvey](http://www.microsoft.com/learning/booksurvey)
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using All the Ranking Functions Together</td>
<td>36</td>
</tr>
<tr>
<td>Ranking over Groups Using <code>PARTITION BY</code></td>
<td>37</td>
</tr>
<tr>
<td>Exception Handling in Transactions</td>
<td>40</td>
</tr>
<tr>
<td>The <code>varchar(max)</code> Data Type</td>
<td>42</td>
</tr>
<tr>
<td>The <code>WAITFOR</code> Statement</td>
<td>43</td>
</tr>
<tr>
<td>DDL Triggers</td>
<td>43</td>
</tr>
<tr>
<td>SNAPSHOT Isolation</td>
<td>45</td>
</tr>
<tr>
<td>Table-Valued Parameters</td>
<td>45</td>
</tr>
<tr>
<td>More than Just Another Temporary Table Solution</td>
<td>46</td>
</tr>
<tr>
<td>Working with a Multiple-Row Set</td>
<td>48</td>
</tr>
<tr>
<td>Using TVPs for Bulk Inserts and Updates</td>
<td>49</td>
</tr>
<tr>
<td>Working with a Single Row of Data</td>
<td>51</td>
</tr>
<tr>
<td>Creating Dictionary-Style TVPs</td>
<td>54</td>
</tr>
<tr>
<td>Passing TVPs Using ADO.NET</td>
<td>56</td>
</tr>
<tr>
<td>TVP Limitations</td>
<td>59</td>
</tr>
<tr>
<td>New Date and Time Data Types</td>
<td>59</td>
</tr>
<tr>
<td>Separation of Dates and Times</td>
<td>59</td>
</tr>
<tr>
<td>More Portable Dates and Times</td>
<td>60</td>
</tr>
<tr>
<td>Time Zone Awareness</td>
<td>61</td>
</tr>
<tr>
<td>Date and Time Accuracy, Storage, and Format</td>
<td>62</td>
</tr>
<tr>
<td>New and Changed Functions</td>
<td>65</td>
</tr>
<tr>
<td>The <code>MERGE</code> Statement</td>
<td>68</td>
</tr>
<tr>
<td>Defining the Merge Source and Target</td>
<td>70</td>
</tr>
<tr>
<td>The <code>WHEN MATCHED</code> Clause</td>
<td>71</td>
</tr>
<tr>
<td>The <code>WHEN NOT MATCHED BY TARGET</code> Clause</td>
<td>72</td>
</tr>
<tr>
<td>Using <code>MERGE</code> for Table Replication</td>
<td>73</td>
</tr>
<tr>
<td>The <code>WHEN NOT MATCHED BY SOURCE</code> Clause</td>
<td>74</td>
</tr>
<tr>
<td><code>MERGE</code> Output</td>
<td>76</td>
</tr>
<tr>
<td>Choosing a Join Method</td>
<td>78</td>
</tr>
<tr>
<td><code>MERGE</code> DML Behavior</td>
<td>79</td>
</tr>
<tr>
<td>Doing the “Upsert”</td>
<td>81</td>
</tr>
<tr>
<td>The <code>INSERT OVER</code> DML Syntax</td>
<td>90</td>
</tr>
<tr>
<td>Extending <code>OUTPUT...INTO</code></td>
<td>90</td>
</tr>
<tr>
<td>Consuming <code>CHANGES</code></td>
<td>94</td>
</tr>
<tr>
<td>The <code>GROUPING SETS</code> Operator</td>
<td>97</td>
</tr>
<tr>
<td>Rolling Up by Level</td>
<td>99</td>
</tr>
<tr>
<td>Rolling Up All Level Combinations</td>
<td>101</td>
</tr>
<tr>
<td>Returning Just the Top Level</td>
<td>103</td>
</tr>
</tbody>
</table>
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixing and Matching</td>
<td>104</td>
</tr>
<tr>
<td>Handling NULL Values</td>
<td>105</td>
</tr>
<tr>
<td>New T-SQL Shorthand Syntax</td>
<td>109</td>
</tr>
<tr>
<td>Summary</td>
<td>110</td>
</tr>
<tr>
<td>3 Exploring SQL CLR</td>
<td>111</td>
</tr>
<tr>
<td>Getting Started: Enabling CLR Integration</td>
<td>112</td>
</tr>
<tr>
<td>Visual Studio/SQL Server Integration</td>
<td>113</td>
</tr>
<tr>
<td>SQL Server Projects in Visual Studio</td>
<td>114</td>
</tr>
<tr>
<td>Automated Deployment</td>
<td>117</td>
</tr>
<tr>
<td>SQL CLR Code Attributes</td>
<td>117</td>
</tr>
<tr>
<td>Your First SQL CLR Stored Procedure</td>
<td>118</td>
</tr>
<tr>
<td>CLR Stored Procedures and Server-Side Data Access</td>
<td>120</td>
</tr>
<tr>
<td>Piping Data with <code>SqlDataRecord</code> and <code>SqlMetaData</code></td>
<td>123</td>
</tr>
<tr>
<td>Deployment</td>
<td>125</td>
</tr>
<tr>
<td>Deploying Your Assembly</td>
<td>125</td>
</tr>
<tr>
<td>Deploying Your Stored Procedures</td>
<td>127</td>
</tr>
<tr>
<td>Testing Your Stored Procedures</td>
<td>129</td>
</tr>
<tr>
<td>CLR Functions</td>
<td>131</td>
</tr>
<tr>
<td>CLR Triggers</td>
<td>136</td>
</tr>
<tr>
<td>CLR Aggregates</td>
<td>140</td>
</tr>
<tr>
<td>SQL CLR Types</td>
<td>145</td>
</tr>
<tr>
<td>Security</td>
<td>150</td>
</tr>
<tr>
<td>Examining and Managing SQL CLR Types in a Database</td>
<td>152</td>
</tr>
<tr>
<td>Best Practices for SQL CLR Usage</td>
<td>159</td>
</tr>
<tr>
<td>Summary</td>
<td>160</td>
</tr>
<tr>
<td>4 Server Management</td>
<td>161</td>
</tr>
<tr>
<td>What Is SMO?</td>
<td>161</td>
</tr>
<tr>
<td>What About SQL-DMO?</td>
<td>162</td>
</tr>
<tr>
<td>Latest Features in SMO</td>
<td>166</td>
</tr>
<tr>
<td>Working with SMO in Microsoft Visual Studio</td>
<td>167</td>
</tr>
<tr>
<td>Iterating Through Available Servers</td>
<td>169</td>
</tr>
<tr>
<td>Retrieving Server Settings</td>
<td>171</td>
</tr>
<tr>
<td>Creating Backup-and-Restore Applications</td>
<td>175</td>
</tr>
<tr>
<td>Performing Programmatic DBCC Functions with SMO</td>
<td>181</td>
</tr>
<tr>
<td>Policy-Based Management</td>
<td>183</td>
</tr>
<tr>
<td>A Simple Policy</td>
<td>184</td>
</tr>
<tr>
<td>Summary</td>
<td>188</td>
</tr>
</tbody>
</table>
## 5 Security in SQL Server 2008 ........................................ 189

- Four Themes of the Security Framework ........................................ 189
- Secure by Design ................................................................. 189
- Secure by Default ............................................................... 190
- Secure by Deployment .......................................................... 190
- Secure Communications ......................................................... 190

### SQL Server 2008 Security Overview ........................................ 191

- SQL Server Logins ............................................................... 192
- Database Users ................................................................. 193
- The guest User Account ....................................................... 194

### Authentication and Authorization ........................................... 195

- How Clients Establish a Connection ........................................ 195
- Password Policies .............................................................. 197
- User-Schema Separation ..................................................... 198
- Execution Context ............................................................. 200

### Encryption Support in SQL Server ......................................... 203

- Encrypting Data on the Move ............................................... 204
- Encrypting Data at Rest ...................................................... 206

### Transparent Data Encryption in SQL Server 2008 .................. 211

- Creating Keys and Certificates ........................................... 211
- Enabling TDE ................................................................. 213
- Querying TDE Views .......................................................... 213
- Backing Up the Certificate .................................................. 214
- Restoring an Encrypted Database .......................................... 215

### SQL Server Audit ............................................................... 216

- Creating an Audit Object .................................................... 216
- Auditing Options ............................................................... 217
- Recording Audits to the File System .................................... 219
- Recording Audits to the Windows Event Log ......................... 220
- Auditing Server Events ...................................................... 220
- Auditing Database Events .................................................. 221
- Viewing Audited Events ..................................................... 222
- Querying Audit Catalog Views ........................................... 224

### How Hackers Attack SQL Server ........................................... 225

- Direct Connection to the Internet ........................................ 225
- Weak System Administrator Account Passwords .................... 226
- SQL Server Browser Service .............................................. 226
SQL Injection .................................................. 226
Intelligent Observation ......................................... 227
Summary ............................................................... 228

Part II Beyond Relational

6 XML and the Relational Database ............................. 231
   XML in SQL Server 2000 ........................................ 233
   XML in SQL Server 2008—the xml Data Type ............... 234
      Working with the xml Data Type as a Variable .......... 234
      Working with XML in Tables ................................. 235
      XML Schemas .................................................. 237
      XML Indexes ................................................. 244
   FOR XML Commands ............................................. 247
      FOR XML RAW ................................................ 248
      FOR XML AUTO ............................................... 248
      FOR XML EXPLICIT ......................................... 250
   FOR XML Enhancements ......................................... 253
   OPENXML Enhancements in SQL Server 2008 ............... 261
   XML Bulk Load ................................................ 262
   Querying XML Data Using XQuery ............................. 263
      Understanding XQuery Expressions and XPath ......... 263
   SQL Server 2008 XQuery in Action .......................... 266
   SQL Server XQuery Extensions ............................... 275
   XML DML ...................................................... 276
      Converting a Column to XML ................................ 278
   Summary ........................................................... 280

7 Hierarchical Data and the Relational Database .............. 281
   The hierarchyid Data Type .................................... 282
   Creating a Hierarchical Table ................................ 283
      The GetLevel Method ........................................ 284
   Populating the Hierarchy ..................................... 285
      The GetRoot Method ........................................ 286
      The GetDescendant Method ................................ 286
      The ToString Method ....................................... 288
      The GetAncestor Method ................................... 293
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchical Table Indexing Strategies</td>
<td>296</td>
</tr>
<tr>
<td>Depth-First Indexing</td>
<td>297</td>
</tr>
<tr>
<td>Breadth-First Indexing</td>
<td>298</td>
</tr>
<tr>
<td>Querying Hierarchical Tables</td>
<td>299</td>
</tr>
<tr>
<td>The IsDescendantOf Method</td>
<td>299</td>
</tr>
<tr>
<td>Reordering Nodes Within the Hierarchy</td>
<td>301</td>
</tr>
<tr>
<td>The GetReparentedValue Method</td>
<td>301</td>
</tr>
<tr>
<td>Transplanting Subtrees</td>
<td>303</td>
</tr>
<tr>
<td>More hierarchyid Methods</td>
<td>305</td>
</tr>
<tr>
<td>Summary</td>
<td>306</td>
</tr>
<tr>
<td>8 Using FILESTREAM for Unstructured Data Storage</td>
<td>307</td>
</tr>
<tr>
<td>BLOBs in the Database</td>
<td>307</td>
</tr>
<tr>
<td>BLOBs in the File System</td>
<td>309</td>
</tr>
<tr>
<td>What’s in an Attribute?</td>
<td>309</td>
</tr>
<tr>
<td>Enabling FILESTREAM</td>
<td>310</td>
</tr>
<tr>
<td>Enabling FILESTREAM for the Machine</td>
<td>311</td>
</tr>
<tr>
<td>Enabling FILESTREAM for the Server Instance</td>
<td>312</td>
</tr>
<tr>
<td>Creating a FILESTREAM-Enabled Database</td>
<td>313</td>
</tr>
<tr>
<td>Creating a Table with FILESTREAM Columns</td>
<td>315</td>
</tr>
<tr>
<td>The OpenSqlFilestream Native Client API</td>
<td>318</td>
</tr>
<tr>
<td>File-Streaming in .NET</td>
<td>319</td>
</tr>
<tr>
<td>Understanding FILESTREAM Data Access</td>
<td>321</td>
</tr>
<tr>
<td>The Payoff</td>
<td>331</td>
</tr>
<tr>
<td>Creating a Streaming HTTP Service</td>
<td>333</td>
</tr>
<tr>
<td>Building the WPF Client</td>
<td>338</td>
</tr>
<tr>
<td>Summary</td>
<td>340</td>
</tr>
<tr>
<td>9 Geospatial Data Types</td>
<td>341</td>
</tr>
<tr>
<td>SQL Server 2008 Spaces Out</td>
<td>341</td>
</tr>
<tr>
<td>Spatial Models</td>
<td>342</td>
</tr>
<tr>
<td>Planar (Flat-Earth) Model</td>
<td>342</td>
</tr>
<tr>
<td>Geodetic (Round-Earth) Model</td>
<td>343</td>
</tr>
<tr>
<td>Spatial Data Types</td>
<td>344</td>
</tr>
<tr>
<td>Defining Space with Well-Known Text</td>
<td>344</td>
</tr>
<tr>
<td>Working with geometry</td>
<td>345</td>
</tr>
<tr>
<td>The Parse Method</td>
<td>346</td>
</tr>
<tr>
<td>The STIntersects Method</td>
<td>347</td>
</tr>
</tbody>
</table>
Part III Reach Technologies

10 The Microsoft Data Access Machine ......................... 377

ADO.NET and Typed DataSets ................................ 378
Typed DataSet Basics ......................................... 378
TableAdapter Objects .......................................... 380
Connection String Management ............................... 381
Using the TableAdapter Configuration Wizard ............... 382
More on Queries and Parameters ............................. 385
DBDirect Methods and Connected Use of Typed DataSet Objects ... 387
“Pure” ADO.NET: Working in Code ......................... 387
Querying 101 .................................................. 388
LINQ: A New Syntactic Approach to Data Access ............ 392
LINQ to DataSet .............................................. 392
LINQ Syntax, Deconstructed .................................. 393
LINQ to SQL and the ADO.NET Entity Framework: ORM Comes to .NET ... 395
Why Not Stick with ADO.NET? ............................... 396
Building an L2S Model ....................................... 397
The Entity Framework: Doing ORM the ADO.NET Way .... 402
XML Behind the Scenes ....................................... 405
Querying the L2S and EF Models ............................. 406
Adding Custom Validation Code ................................ 410
Web Services for Data: Using ADO.NET Data Services Against EF Models ... 411
Creating the Service ........................................... 412
### Table of Contents

- **Testing the Service** ............................................. 414
- **Building the User Interface** ...................................... 414
- **Data as a Hosted Service: SQL Server Data Services** ................. 415
- **Summary: So Many Tools, So Little Time** ............................... 417

#### 11 The Many Facets of .NET Data Binding ................................. 419

- **Windows Forms Data Binding: The Gold Standard** .................... 420
  - **Getting Ready** .................................................... 420
  - **Generating the UI** ............................................. 421
  - **Examining the Output** ......................................... 423
  - **Converting to LINQ to SQL** .................................... 424
  - **Converting to Entity Framework** ................................ 425
  - **Converting to ADO.NET Data Services** ............................ 426
- **Data Binding on the Web with ASP.NET** ................................ 427
  - **L2S and EF Are Easy** ........................................... 428
  - **Beyond Mere Grids** ............................................ 429
  - **Data Binding Using Markup** ..................................... 430
  - **Using AJAX for Easy Data Access** ................................ 430
  - **ASP.NET Dynamic Data** .......................................... 435
- **Data Binding for Windows Presentation Foundation** .................... 438
  - **Design Time Quandary** .......................................... 439
  - **Examining the XAML** ............................................ 441
- **Grand Finale: Silverlight** ........................................... 445
- **Summary** .................................................................. 447

#### 12 Transactions ................................................................. 449

- **What Is a Transaction?** .................................................. 450
  - **Understanding the ACID Properties** ................................ 450
- **Local Transaction Support in SQL Server 2008** .......................... 453
  - **Autocommit Transaction Mode** ..................................... 453
  - **Explicit Transaction Mode** ........................................... 453
  - **Implicit Transaction Mode** .......................................... 456
  - **Batch-Scoped Transaction Mode** ................................... 457
- **Using Local Transactions in ADO.NET** .................................. 459
- **Transaction Terminology** .................................................. 461
- **Isolation Levels** ............................................................ 462
  - **Isolation Levels in SQL Server 2008** ...................... 462
  - **Isolation Levels in ADO.NET** .................................. 467
Distributed Transactions ............................................. 468
Distributed Transaction Terminology ................................ 469
Rules and Methods of Enlistment ...................................... 470
Distributed Transactions in SQL Server 2008 ..................... 472
Distributed Transactions in the .NET Framework .................. 473
Writing Your Own Resource Manager .............................. 477
Using a Resource Manager in a Successful Transaction ............ 481
Transactions in SQL CLR (CLR Integration) ....................... 485
Putting It All Together ............................................... 489
Summary ........................................................................ 490

13 Developing Occasionally Connected Systems ................. 491
Comparing Sync Services with Merge Replication .................. 492
Components of an Occasionally Connected System ............... 493
Merge Replication .......................................................... 494
  Getting Familiar with Merge Replication ......................... 494
  Creating an Occasionally Connected Application with Merge Replication ........................................ 496
  Configuring Merge Replication ...................................... 499
  Creating a Mobile Application Using Microsoft Visual Studio 2008 .................................................. 520
Sync Services for ADO.NET ............................................. 533
  Sync Services Object Model .......................................... 534
  Capturing Changes for Synchronization ............................ 538
  Creating an Application Using Sync Services ..................... 543
  Additional Considerations ............................................. 557
Summary ........................................................................ 560

Part IV Business Intelligence

14 Data Warehousing ...................................................... 563
Data Warehousing Defined .............................................. 563
The Importance of Data Warehousing ................................. 564
What Preceded Data Warehousing ..................................... 566
  Lack of Integration Across the Enterprise ......................... 567
  Little or No Standardized Reference Data ......................... 568
  Lack of History ............................................................ 568
Data Not Optimized for Analysis ....................................... 568
As a Result... ............................................................. 569
Data Warehouse Design .................................................. 570
## Table of Contents

The Top-Down Approach of Inmon ................................................. 572
The Bottom-Up Approach of Kimball ........................................... 574
What Data Warehousing Is Not .................................................. 580
OLAP .................................................................................. 580
Data Mining ........................................................................ 581
Business Intelligence .............................................................. 582
Dashboards and Scorecards ....................................................... 583
Performance Management ......................................................... 585
Practical Advice About Data Warehousing .................................... 585
Anticipating and Rewarding Operational Process Change ............ 586
Rewarding Giving Up Control .................................................... 586
A Prototype Might Not Work to Sell the Vision ............................ 586
Surrogate Key Issues ............................................................... 587
Currency Conversion Issues ....................................................... 587
Events vs. Snapshots .................................................................. 588
SQL Server 2008 and Data Warehousing ...................................... 589
T-SQL `MERGE` Statement .......................................................... 589
Change Data Capture ................................................................. 592
Partitioned Table Parallelism ....................................................... 600
Star-Join Query Optimization ..................................................... 603
`SPARSE` Columns .................................................................. 604
Data Compression and Backup Compression ................................. 605
Learning More .......................................................................... 610
Summary .................................................................................. 610

### 15 Basic OLAP ................................................................. 611

Wherefore BI? .......................................................................... 611
OLAP 101 ............................................................................. 613
OLAP Vocabulary ..................................................................... 614
Dimensions, Axes, Stars, and Snowflakes ................................. 615
Building Your First Cube ........................................................... 617
Preventing Star Schema Objects ............................................... 617
A Tool by Any Other Name ........................................................ 618
Creating the Project .................................................................. 619
Adding a Data Source View ....................................................... 621
Creating a Cube with the Cube Wizard ....................................... 625
Using the Cube Designer ............................................................ 626
Using the Dimension Wizard ..................................................... 629
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using the Dimension Designer</td>
<td>632</td>
</tr>
<tr>
<td>Working with the Properties Window and Solution Explorer</td>
<td>634</td>
</tr>
<tr>
<td>Processing the Cube</td>
<td>635</td>
</tr>
<tr>
<td>Running Queries</td>
<td>636</td>
</tr>
<tr>
<td>Summary</td>
<td>637</td>
</tr>
<tr>
<td><strong>16 Advanced OLAP</strong></td>
<td>639</td>
</tr>
<tr>
<td>What We’ll Cover in This Chapter</td>
<td>640</td>
</tr>
<tr>
<td>MDX in Context</td>
<td>640</td>
</tr>
<tr>
<td>And Now a Word from Our Sponsor</td>
<td>640</td>
</tr>
<tr>
<td>Advanced Dimensions and Measures</td>
<td>641</td>
</tr>
<tr>
<td>Keys and Names</td>
<td>641</td>
</tr>
<tr>
<td>Changing the All Member</td>
<td>644</td>
</tr>
<tr>
<td>Adding a Named Query to a Data Source View</td>
<td>645</td>
</tr>
<tr>
<td>Parent/Child Dimensions</td>
<td>647</td>
</tr>
<tr>
<td>Member Grouping</td>
<td>651</td>
</tr>
<tr>
<td>User Table Time Dimensions, Attribute Relationships, Best Practice Alerts, and Dimension/Attribute Typing</td>
<td>652</td>
</tr>
<tr>
<td>Server Time Dimensions</td>
<td>660</td>
</tr>
<tr>
<td>Fact Dimensions</td>
<td>661</td>
</tr>
<tr>
<td>Role-Playing Dimensions</td>
<td>664</td>
</tr>
<tr>
<td>Advanced Measures</td>
<td>665</td>
</tr>
<tr>
<td>Calculations</td>
<td>667</td>
</tr>
<tr>
<td>Calculated Members</td>
<td>668</td>
</tr>
<tr>
<td>Named Sets</td>
<td>673</td>
</tr>
<tr>
<td>More on Script View</td>
<td>674</td>
</tr>
<tr>
<td>Key Performance Indicators</td>
<td>677</td>
</tr>
<tr>
<td>KPI Visualization: Status and Trend</td>
<td>678</td>
</tr>
<tr>
<td>A Concrete KPI</td>
<td>679</td>
</tr>
<tr>
<td>Testing KPIs in Browser View</td>
<td>681</td>
</tr>
<tr>
<td>KPI Queries in Management Studio</td>
<td>683</td>
</tr>
<tr>
<td>Other BI Tricks in Management Studio</td>
<td>688</td>
</tr>
<tr>
<td>Actions</td>
<td>689</td>
</tr>
<tr>
<td>Actions Simply Defined</td>
<td>690</td>
</tr>
<tr>
<td>Designing Actions</td>
<td>690</td>
</tr>
<tr>
<td>Testing Actions</td>
<td>692</td>
</tr>
<tr>
<td>Partitions, Storage Settings, and Proactive Caching</td>
<td>693</td>
</tr>
<tr>
<td>Editing and Creating Partitions</td>
<td>694</td>
</tr>
</tbody>
</table>
### Table of Contents

Partition Storage Options ........................................... 696
Proactive Caching ..................................................... 697
Additional Features and Tips ........................................ 699
Aggregations ............................................................ 700
Algorithmic Aggregation Design ................................. 700
Usage-Based Aggregation Design ................................. 701
Manual Aggregation Design (and Modification) ............... 702
Aggregation Design Management ................................. 704
Aggregation Design and Management Studio ................... 705
Perspectives ............................................................. 705
Translations ............................................................ 707
Roles ........................................................................ 712
Summary .................................................................... 715

#### 17 OLAP Queries, Tools, and Application Development ........ 717

Using Excel ............................................................... 719
  Connecting to Analysis Services .................................. 719
  Building the PivotTable .............................................. 723
  Exploring PivotTable Data ......................................... 725
  Scorecards ................................................................ 727
  Creating and Configuring Charts ................................. 729
  In-Formula Querying of Cubes .................................... 732
  Visual Studio Tools for Office and Excel Add-Ins .......... 737
  Excel Services .......................................................... 738
Beyond Excel: Custom OLAP Development with .NET .......... 743
  MDX and Analysis Services APIs ................................. 744
  Moving to MDX ........................................................ 744
  Management Studio as an MDX Client ......................... 745
  OLAP Development with ADO MD.NET ......................... 758
  Using Analysis Management Objects ............................ 769
  XMLA at Your (Analysis) Service ............................... 771
  Analysis Services CLR Support: Server-Side ADO MD.NET .... 782
Summary .................................................................... 792

#### 18 Expanding Your Business Intelligence with Data Mining ...... 793

Why Mine Your Data? .................................................. 793
SQL Server 2008 Data Mining Enhancements .................. 797
Getting Started ......................................................... 798
Preparing Your Source Data ........................................ 798
Acknowledgments

Working on this book has truly been the most rewarding experience of my professional career thus far, and I need to thank a great many people who have made it possible.

I first met Andrew Brust about 10 years ago, and we’ve enjoyed a close working relationship and growing friendship for the past 7 of those. I can’t count the number of times Andrew has opened doors for me with project, writing, and speaking opportunities—and now, of course, this book. Andrew introduced me to Stephen Forte back in 2004, and after 30 years in the industry, I’ve learned to find new appreciation for the art of software development through Stephen’s enthusiastic (that is, wacky and wild) personality. Andrew and Stephen both made this project significantly easier by producing the original edition of this book—an excellent treatment of Microsoft SQL Server 2005 that set the starting point for this new 2008 edition. It’s been an absolute thrill and honor that they invited me to join them this time around and to assume the role of lead author for the new edition. Thanks to you both for entrusting me with that responsibility, as well as for your own hard work on this edition.

We could never have produced a book so broad and deep in scope without the additional aid of the other guest authors. Elsie Pan, Paul Delcogliano, Mark Frawley, and Jeff Bolton each did an outstanding job writing brand-new chapters, and Elsie also revised material from the last edition. Heartfelt thanks go out as well to Kenn Scribner, who performed an incredibly detail-oriented tech review of the entire book, and especially for helping out with updating important material at the bottom of the ninth with two men out and three men on. I’m very grateful for their contributions and professionalism, and I feel privileged to have worked with each of them. I’d also like to thank Jay Rabin, and all the wonderful folks at twentiesix New York, for their continuous stream of support and encouragement throughout this whole project.

I was very lucky to have worked closely with Kathleen Atkins, Sally Stickney, and Ken Jones of Microsoft Press; Steve Sagman from Waypoint Press; and copy editor Jennifer Harris. Their superb editorial contributions, project planning, and overall guidance were vital to the production of this book. Double thanks go to Sally, who was always available to me (weekends too) for much-needed guidance as I entered the world of book writing. And the assistance provided by a number of people from various Microsoft product teams helped tackle the challenge of writing about new software as it evolved through several beta releases. So thank you to Steve Lasker, for support with Compact and Sync Services, and to Roger Doherty and Isaac Kunen for support with the “beyond relational” features. In particular, Roger inspired several of the FILESTREAM and geospatial demos found in those chapters. George Sobhy was also a great help with geospatial—he even arranged for a shared desktop demo between New York and Cairo (and thanks to Stephen Forte too, for the introduction).
Acknowledgments

This would all be moot, of course, without the love and support of my family. Almost all of my spare time over the past year was spent working on this project in one form or another—researching and writing new material, editing the whole book, and coordinating its production—which at times transformed me into an absentee partner and father. I owe an enormous debt of gratitude to my wonderful partner, Mark, and my awesome kids, Adam, Jacqueline, and Josh, for putting up with it all. So thanks, gang, I’m back home now! And thanks most of all to dear Mom, bless her soul, for always encouraging me to write with “expression” since the first grade.

—Leonard Lobel

Writing a book is hard, especially for the people in the authors’ lives who lend heroic support. Revising a book is hard too, especially for the people in the authors’ lives who lend that support again.

With that in mind, I’d like to thank my wife, Lauren (who endured this project while at the same time earning her master’s degree and being an amazing mom to our two boys). And I thank our boys as well: Miles (who, though only four years old, is nonetheless a veteran of both editions of this book) and Sean (who, at age 18 months, has endured yet another thing that his older brother experienced first). All three have tolerated my intolerable absences from their events and their lives. Each one has my gratitude for the patience and understanding shown to me.

I’d also like to thank everyone at twentysix New York, especially Jay Rabin, for granting me a period of calm, with unprecedented duration, to get the work on this edition done.

Finally, but certainly not least of all, I’d like to thank Leonard Lobel for “taking the wheel” on this edition of the book. Had he not done so, we simply would not have this edition of the book. Lenni is prone to thanking me for exposing him to opportunity. What he fails to understand is that by repeatedly succeeding, he makes me look good simply for having the good taste to recommend him.

—Andrew Brust
It's nice to have your name on the cover of a book, but without help from various people, this book never would have happened. I'll start with Andrew and Lenni, my wonderful co-authors, both easy to work with, dedicated, and also very patient with me. The folks at Microsoft Press were all great to work with and had considerable energy and flexibility.

I would not have been able to take on this project if I did not have the support of the folks at Telerik, Triton Works, and Dash Soft, three companies that I work very closely with. I would like to give special thanks to the leaders of those firms, Vassil Terziev, Mark Storen, and Remon “FGD” Zakaria, for their understanding and support when deadlines for the book loomed.

We have also had tons of great reviewers. I was blessed to have folks like Kimberly Tripp, Peter DeBetta, and Roman Rehak help out with reading my chapters, as well as Kevin Collins, Remi Caron, Joel Semeniuk, Eileen Rumwell, Steve Lasker, Kirk Haselden, Ted Lee, Sergei Ivanov, Richard Campbell, Goksin Bakir, Malek Kemmou, Jason Gideon, Julie Lerman, Bill Ramos, Tom Halligan, and finally Jack Prilook—who looked at my manuscript 13 times.

I started this book on the first day of classes of my second year of my MBA education. Some days I had to choose whether to write on Policy-Based Management or macroeconomic trends in China and India. I’d like to thank all my group members at EMBA 26, most especially Dr. Ian Miller, Rosa Alvarado, Jason Nocco, Dmitriy Malinovskiy, and Cyrus Kazi. As fate would have it, I type these words on my last day of school. How fitting to finish a book and an MBA in the same weekend.

—Stephen Forte
Introduction

Welcome, developer!

The book you are holding, much like Microsoft SQL Server 2008 itself, builds on a great “previous release.” SQL Server 2005 was—architecturally speaking—a groundbreaking upgrade from earlier versions of the product, and the 2005 edition of this book was a new printed resource that provided comprehensive coverage of the revamped platform. This new edition includes thoroughly updated coverage of the most important topics from the past edition, plus brand-new coverage of all the new exciting and powerful features for developers in SQL Server 2008. As with the 2005 edition, we set out to produce the best book for developers who need to program SQL Server 2008 in the many ways that it can be programmed.

To best understand our approach, we ask that you consider likening SQL Server 2008 to, of all things, a Sunday newspaper. A Sunday newspaper is made up of multiple sections, each of which is written separately and appeals to a distinct audience. The sections do have overlapping content and share some readership, of course, but most people don’t read the whole paper, and they don’t need to. Meanwhile, the entire paper is considered a single publication, and those who read it think of themselves as readers of the paper rather than of one or more of its sections. Likewise, SQL Server has many pieces to it: few people will use them all, and people will need to learn about them gradually, over time, as their business needs dictate.

Our book reflects this reality and in many ways replicates the structure of a Sunday newspaper. For one thing, a great number of authors have been involved in producing the book, drawing on their expertise in their chapters’ specific subject matter. For another, the context of certain chapters differs markedly from those of other chapters. Some chapters cover specific subject matter deeply. Others cover a broader range of material, and do so at a higher level. That’s an approach we didn’t anticipate when we authored the 2005 edition of this book. But it’s the approach we found most effective by the time we finished it, and one which we continue to follow in this new edition for SQL Server 2008. We have found that it makes an otherwise overwhelming set of technologies much more approachable and makes the learning process much more modular.

Make no mistake, though—the overall vision for the book is a cohesive one: to explore the numerous programmability points of SQL Server 2008 and, in so doing, provide widespread coverage of the great majority of the product’s features, in a voice that caters to developers’ interests. Whether you read every chapter in the book or just some of them and whether you read the book in or out of order, our goal has been to provide you with practical information, numerous useful samples, and a combination of high-level coverage and detailed discussion, depending on how deep we thought developers would want to go.
Just as the Sunday newspaper doesn’t cover everything that's going on in the world, this book won’t teach you everything about SQL Server. For example, we don’t cover high-availability/fault tolerance features such as replication, clustering, or mirroring. We don’t discuss query plans and optimization, nor do we investigate SQL Server Profiler, SQL Trace, or the Database Engine Tuning Advisor. Some features covered in the 2005 edition have not changed significantly in SQL Server 2008, such as native XML Web Services, Service Broker, Integration Services, and cross-tier debugging. These topics are also not covered, in order to make room for new SQL Server 2008 features. (The 2005 edition chapters that cover those topics are available for you to download from this book’s companion Web site, which we explain toward the end of this introduction.)

We discovered as we wrote the book that covering everything in the product would result in a book unwieldy in size and unapproachable in style. We hope we struck the right balance, providing a digestible amount of information with enough developer detail and enough pointers to other material to help you become a seasoned SQL Server professional.

Who This Book Is For

Now that we have established what the book does and does not cover, we’d like to clarify just who we believe will be most interested in it and best served by it. In a nutshell, this book is for .NET and SQL Server developers who work with databases and data access, at the business logic/middle-tier layer as well as the application level.

In our perhaps self-centered view of the development world, we think this actually describes most .NET developers, but clearly some developers are more interested in database programming in general, and SQL Server specifically, than others, and it is this more interested group we want to reach.

We assume that you have basic, working knowledge of .NET programming on the client and Transact-SQL (T-SQL) on the server, although SQL experience on any platform can easily substitute. We also assume that you are comfortable with the basics of creating tables, views, and stored procedures on the server. On the client tools side, we assume that you are familiar with the prior generation of SQL Server and .NET development tools. If you’ve already been working with SQL Server Management Studio in SQL Server 2005, you’ll feel right at home with the 2008 version, which has been extended to support new server features (and now even includes IntelliSense for T-SQL!). If you’re still running SQL Server 2000 or earlier, you’ll definitely appreciate SQL Server Management Studio as a vast improvement over the two primary tools that preceded it—Enterprise Manager and Query Analyzer. SQL Server Management Studio essentially represents the fusion of those two tools, packaged in a modern user interface (UI) shell very similar to that provided by Microsoft Visual Studio—complete with customizable menus and toolbars, floatable and dockable panes, solutions, and projects. The primary tool for .NET devel-
opment is, of course, Visual Studio 2008, and experience with any version will also be beneficial for you to have.

Having said all that, we have a fairly liberal policy regarding these prerequisites. For example, if you've only dabbled with SQL and .NET, that's OK, as long as you're willing to try and pick up on things as you read along. Most of our code samples are not that complex. However, our explanations do assume some basic knowledge on your part, and you might need to do a little research if you lack the experience.

**Note** For the sake of consistency, all the .NET code in this book is written in C#. (The only exceptions to this rule will be found in Chapter 19 for Reporting Services, since only Visual Basic .NET is supported for scripting report expressions and deployments.) However, this book is in no way C#-oriented, and there is certainly nothing C#-specific in the .NET code provided. As we just stated, the code samples are not very complex, and if you are more experienced with Visual Basic .NET than you are with C#, you should have no trouble translating the C# code to Visual Basic .NET on the fly as you read it.

In addition to covering the SQL Server core relational engine, its latest breed of “beyond relational” capabilities, and its ancillary services, this book also provides in-depth coverage of SQL Server’s business intelligence (BI) features, including Reporting Services, and the online analytical processing (OLAP) and data mining components of Analysis Services. Although ours is not a BI book per se, it is a database developer’s book, and we feel strongly that all these features should be understood by mainstream database developers. BI is really one of the cornerstone features of SQL Server 2008, so the time is right for traditional database developers to “cross over” to the world of BI.

Realizing that these technologies, especially OLAP and data mining, will be new territory for many readers, we assume no knowledge of them on your part. Any reader who meets the prerequisites already discussed should feel comfortable reading about these BI features and, more than likely, feel ready and excited to start working with BI after reading the BI-focused chapters.

**How This Book Is Organized**

This book is broken down into four parts. Each part follows a specific SQL Server “theme,” if you will.

Part I begins with an overview that gives you a succinct breakdown of the chapters in all four parts of the book. Then it dives right in to core SQL Server technologies. We explore the many powerful enhancements made to Transact-SQL (T-SQL), both in SQL Server 2005 and 2008 (in that order). We also introduce you to SQL Server’s .NET Common Language Runtime (CLR) integration features, which cut across our discussions of data types and server-side
programming. You’ll learn how to programmatically administer the server using Server Management Objects (SMO), which were introduced in SQL Server 2005, and how to use the new administrative framework called Policy-Based Management (PBM) in SQL Server 2008. Then we tackle security. After quickly covering basic SQL Server security concepts, we show how to encrypt your data both while in transit (traveling across the network) and at rest (on disk). We’ll also teach the latest security features in SQL Server 2008, including Transparent Data Encryption (TDE) and SQL Server Audit, which you will find extremely useful in today’s world of regulatory compliance.

Part II is dedicated to the SQL Server 2008 “beyond relational” release theme, which is all about working with semistructured and unstructured data. This is a concept that broadens our traditional view of relational databases by getting us to think more “outside the box” in terms of all the different types of data that SQL Server can be used to manage, query, and manipulate. We begin with a chapter on XML support (which was spearheaded in SQL Server 2005), and provide detailed coverage that includes the recent XML enhancements made in SQL Server 2008. All the remaining chapters in Part II cover nonrelational features that are brand new in SQL Server 2008. These include hierarchical tables, native file streaming, and geospatial capabilities. These features are designed to enrich the native database engine by bringing unprecedented intelligence and programming convenience down to the database level.

In Part III, we move away from the server and discuss concepts relating to actual database software development, be it in the middle tier or at the application level. This includes data access using “traditional” ADO.NET, language-integrated query (LINQ), the ADO.NET Entity Framework, and the latest innovations, ADO.NET Data Services and SQL Server Data Services. After you succeed in accessing your data, you’ll need to deliver that data to your users, and that means data binding. We’ll dig in to data binding for Microsoft Windows and ASP.NET Web applications, as well as the newest UI platforms, Windows Presentation Foundation (WPF) and Silverlight. We also cover transactions and various other topics relevant to extending your databases’ reach with technologies such as merge replication, Sync Services for ADO.NET, and mobile database application development with SQL Server Compact 3.5.

Part IV is our BI section. In it, we provide efficient, practical coverage of SQL Server Analysis Services and Reporting Services. We are particularly proud of this section because we assume virtually no BI or OLAP knowledge on your part and yet provide truly deep coverage of SQL Server BI concepts, features, and programming. We have a chapter dedicated to the topic of data warehousing. In it, you’ll see how to use a new SQL Server 2008 feature called Change Data Capture (CDC) to facilitate incremental updating of large data warehouses. Furthermore, we cover all the new important BI features in SQL Server 2008, expanded to include the latest data mining add-ins for Microsoft Office Excel 2007. The Reporting Services chapter has been written from scratch for the completely reworked and enhanced Report Designer, and also teaches you the many ways that Reporting Services can be programmed and managed.
Together, the four parts of the book provide you with a broad inventory of a slew of SQL Server 2008 developer-relevant features and the conceptual material necessary to understand them. We don’t cover everything in SQL Server 2008, but we will arm you with a significant amount of core knowledge and give you the frame of reference necessary to research the product further and learn even more. Where appropriate, we refer you to SQL Server Books Online, which is the complete documentation library for SQL Server 2008 (available from the Start Menu under Programs, Microsoft SQL Server 2008, Documentation And Tutorials).

**Code Samples and the Book’s Companion Web Site**

All the code samples discussed in this book can be downloaded from the book’s companion Web site at the following address:

http://www.microsoft.com/mspress/companion/9780735625990/

**Important**  This book and its sample code were written for, and tested against, the Release Candidate (RC0) version of SQL Server 2008 Developer edition, released in June 2008. If and when we discover any incompatibilities with the Release To Manufacturer (RTM) version, or any further service packs that are later released, our intent is to update the sample code and post errata notes on the book’s companion Web site, available at http://www.microsoft.com/mspress/companion/9780735625990/. Please monitor that site for new code updates and errata postings.

In addition to all the code samples, the book’s companion Web site also contains several chapters from the 2005 edition of this book that were not updated for the 2008 edition. These include the chapters on native XML Web Services and Service Broker, which are features that have not been widely adopted since they were introduced in SQL Server 2005 but that continue to be supported in SQL Server 2008. The 2005 edition chapters covering SQL Server Management Studio (the primary graphical tool you’ll use for most of your database development work), SQL Server 2005 Express edition, Integration Services, and debugging are posted on the companion Web site as well. With the inclusion of all the new SQL Server 2008 coverage, space constraints simply did not permit us to include these topics (which have not changed significantly in SQL Server 2008) in this new edition. And while we provide completely new coverage on the latest data binding techniques, the 2005 edition covers ADO.NET programming techniques against then-new SQL Server features, and so it is posted on the companion Web site as well. This book’s chapter on OLAP Application development has also been revised to include Excel 2007 coverage, and the 2005 edition is available on the companion Web site for those developers who are still working with Excel 2003 against Analysis Service OLAP cubes.
Introduction

Because this is a developer book, we often include one or more Visual Studio projects as part of the sample code, in addition to SQL Server Management Studio projects containing T-SQL or Analysis Services script files. Within the companion materials parent folder is a child folder for each chapter. Each chapter's folder, in turn, contains either or both of the following two folders: SSMS and VS. The former contains a SQL Server Management Studio solution (.ssmssln file), the latter contains a Visual Studio solution (.sln file). Some chapters might have multiple Visual Studio solutions. After you've installed the companion files, double-click a solution file to open the sample scripts or code in the appropriate integrated development environment (IDE).

Because most of the code is explained in the text, you might prefer to create it from scratch rather than open the finished version supplied in the companion sample code. However, the finished version will still prove useful if you make a small error along the way or if you want to run the code quickly before reading through the narrative that describes it.

Some of the SQL Server Management Studio projects contain embedded connections that are configured to point to a default instance of SQL Server running on your local machine. Similarly, some of the Visual Studio source code contains connections or default connection strings (sometimes in code, sometimes in settings or configuration files, and other times in the Text property of controls on the forms in the sample projects) that are configured likewise. If you have SQL Server 2008 installed as the default instance on your local machine with Windows-integrated security and the AdventureWorks2008 sample database, the majority of the sample code should run without modification. If not, you’ll need to modify the server name, instance name, or user credentials accordingly to suit your environment. You’ll also need to install AdventureWorks2008 if you have not already done so. (Download instructions for all the sample databases are given in the sections ahead.)

A number of chapters rely on various popular sample databases available for SQL Server. These include the Northwind and just-mentioned AdventureWorks2008 sample transactional databases, the AdventureWorksDW2008 sample data warehouse database, and the Adventure Works DW 2008 Analysis Services sample database. None of our examples use the pubs database, which has been around since long before SQL Server 2000.

Using the Sample Northwind Database

You can use the version of Northwind that came with SQL Server 2000, if you have it, and attach it to your SQL Server 2008 server. Microsoft has also published a Windows Installer file (.msi) that will install the Northwind sample database on your server (even the older pubs sample database is included). The installer provides both the primary database file and the log file that can be directly attached, as well as T-SQL scripts, which can be executed.
to create the databases from scratch. At press time, the download page for the *Northwind*

Using the Sample *AdventureWorks2008* Databases

As of SQL Server 2005, and updated for SQL Server 2008, Microsoft provides the *AdventureWorks* family of databases. You can download these sample databases from CodePlex, which is Microsoft’s open source Web site (in fact, all of Microsoft’s official product code samples are hosted on CodePlex). This book uses the *AdventureWorks2008* relational online transaction processing (OLTP) database, the *AdventureWorksDW2008* relational data warehouse database, and the *AdventureWorksAS2008* Analysis Services database. The latest version of these sample databases are designed for use only with SQL Server 2008 and will not work with SQL Server 2005. (The older *AdventureWorks* databases for SQL Server 2005 are still available on CodePlex at the time of this writing, however.)

At press time, the download location for all sample *AdventureWorks2008* databases is [http://www.codeplex.com/MSFTDBProdSamples](http://www.codeplex.com/MSFTDBProdSamples). Click the Releases tab on this page to select any of the sample databases for downloading to your machine. An Internet shortcut to this URL is included on the book’s companion Web site. If the link does not work for you, try running a Web search on “SQL Server 2008 product sample databases.”

The *AdventureWorks2008* OLTP database uses the new FILESTREAM feature in SQL Server 2008, and therefore requires that FILESTREAM be enabled for the instance on which *AdventureWorks2008* is installed. Chapter 8 is devoted to FILESTREAM, and you should refer to the “Enabling FILESTREAM” section in that chapter, which shows how to enable FILESTREAM in order to support *AdventureWorks2008*.

**Important** The samples for this book are based on the 32-bit version of the sample *AdventureWorks2008* databases, which is almost—but not exactly—identical to the 64-bit version. If you are using the 64-bit version of these sample databases, some of your query results might vary slightly from those shown in the book’s examples.
**System Requirements**

To follow along with the book’s text and run its code samples successfully, we recommend that you install the Developer edition of SQL Server 2008, which is available to a great number of developers through Microsoft’s MSDN Premium subscription, on your PC. You will also need Visual Studio 2008; we recommend that you use the Professional edition or one of the Team edition releases, each of which is also available with the corresponding edition of the MSDN Premium subscription product.

**Important** To cover the widest range of features, this book is based on the Developer edition of SQL Server 2008. The Developer edition possesses the same feature set as the Enterprise edition of the product, although Developer edition licensing terms preclude production use. Both editions are high-end platforms that offer a superset of the features available in other editions (Standard, Workgroup, Web, and Express). We believe that it is in the best interest of developers for us to cover the full range of developer features in SQL Server 2008, including those available only in the Enterprise and Developer editions.


To run these editions of SQL Server and Visual Studio, and thus the samples in this book, you’ll need the following 32-bit hardware and software. (The 64-bit hardware and software requirements are not listed here but are very similar.)

- 600-MHz Pentium III–compatible or faster processor (1-GHz minimum, but 2GHz or faster processor recommended).
- Microsoft Windows 2000 Server with Service Pack (SP) 4 or later; Windows 2000 Professional Edition with SP4 or later; Windows XP with SP2 or later; Windows Server 2003 (any edition) with SP1 or later; Windows Small Business Server 2003 with SP1 or later; or Windows Server 2008 (any edition).
- For SQL Server 2008, at least 512 MB of RAM (1 GB or more recommended).
- For Visual Studio 2008, 192 MB (256 MB recommended).
- For SQL Server 2008, approximately 1460 MB of available hard disk space for the recommended installation. Approximately 200 MB of additional available hard disk space for SQL Server Books Online.
- For Visual Studio 2008, maximum of 20 GB of available space required on installation drive. This includes space for the installation of the full set of MSDN documentation.
Internet connection required to download the code samples for each chapter from the companion Web site. A few of the code samples require an Internet connection to run as well.

CD-ROM or DVD-ROM drive recommended.

Super VGA (1024 × 768) or higher resolution video adapter and monitor recommended.

Microsoft Mouse or compatible pointing device recommended.

Microsoft Internet Explorer 6.0 SP1 or later. Microsoft Internet Explorer 7.0 recommended.

For SQL Server Reporting Services, Microsoft Internet Information Services (IIS) 6.0 or later and ASP.NET 2.0 or later.

**Support for This Book**

Every effort has been made to ensure the accuracy of this book and the companion content. As corrections or changes are collected, they will be added to a Microsoft Knowledge Base article.

Microsoft Press provides support for books and companion content at the following Web site:

http://www.microsoft.com/learning/support/books/

**Questions and Comments**

If you have comments, questions, or ideas regarding the book or the companion content, or questions that are not answered by visiting the preceding sites, please send them to Microsoft Press via e-mail to:

mspinput@microsoft.com

Or send them via postal mail to

*Microsoft Press*
*Attn: Programming Microsoft SQL Server 2008 Editor*
*One Microsoft Way*
*Redmond, WA 98052-6399*

Please note that Microsoft software product support is not offered through the preceding addresses.
Chapter 14
Data Warehousing

—Mark Frawley

This chapter is all about data warehousing. If you’ve been avoiding this topic—dismissing it perhaps as being too advanced, esoteric, or abstract to be applicable—this chapter will help you cast those excuses aside and embrace data warehousing. The practical advice and guidance we give will empower you and your end users to glean more useful information and intelligence from your data. We will begin with an explanation of exactly what data warehousing is and why you should care about it, and then we’ll show how to take advantage of specific Microsoft SQL Server 2008 data warehousing features.

Data Warehousing Defined

You’re in good company if you wonder exactly what is meant by data warehousing—and indeed you might even wonder whether it has any precise meaning at all. The term has existed for almost two decades, and you might have seen a variety of definitions. Here is ours:

Data warehousing is both a vision of and a methodological approach toward organizing and managing enterprise data for the purpose of providing a trustworthy, consistent, integrated, and comprehensive data foundation for an enterprise’s data-driven requirements and applications, both tactical and strategic.

Why does our definition not include any technical references? Well, that’s just the point! While technology is essential to actually realizing the vision, data warehousing is not—or should not be—fundamentally about technology. It is about laying the data foundation needed to run an enterprise. Run as in making informed decisions. And enterprise rather than business because data warehousing is equally relevant whether the work is for-profit, not-for-profit, or in the public sector (a subtle distinction resulting from the unfortunate fact that the word business is embedded in the term business intelligence, or BI)—and, increasingly, whether the entity is small, medium, or large. Compared with what was true in the past, Microsoft’s data warehousing–related offerings under the SQL Server product umbrella have made it particularly feasible for data warehousing goals to be attainable by small and medium-size enterprises. Of course, Microsoft continues to deliver industrial-strength data warehousing performance for the largest enterprises—especially with the 2008 release of SQL Server.
The Importance of Data Warehousing

Today, data warehousing in some form has become a given, a must, for running an enterprise of any significant size. At its best, it enables actual competitive advantage, but even when focused more tactically or departmentally, it is now considered essential to being competitive—as basic and essential as the general ledger or payroll system. While it is often difficult to quantify the benefits of data warehousing in terms of return on investment (ROI), no one these days seriously questions its value and necessity. As a database developer, you are likely to be involved with data warehousing in one way or another—if not directly, at least in interfacing to a data warehouse. So it’s important for you to understand what data warehousing is all about.

Developing a data warehouse is in some ways a very different undertaking from traditional online transactional processing (OLTP) database development, with which you are probably more familiar. Two of the most notable differences are that data warehousing essentially emphasizes data and its relationships—as opposed to the emphasis on process found in the typical OLTP application—and that hard experience by practitioners has evolved specialized ways of modeling data that are particularly useful in achieving the goals of data warehousing.

Even if your role is primarily technical, you will be able to do a much better job of building or interfacing to a data warehouse if you know something about these differences from OLTP and the reasons for them. This will also help you appreciate the perspective of decision makers who rely on accurate data storage and analysis (see the next chapter), which will be very likely different from that of typical OLTP application stakeholders.

Data warehousing is an essential foundation for what has come to be known as business intelligence (BI). We’ll learn more about the close relationship between data warehousing and BI later in this chapter, but for now, appreciate that they are not synonymous. At the same time, in keeping with our earlier observation, mentally substitute enterprise when you hear business.

The remainder of this chapter consists of five sections that build upon one another as we progress through our treatment of data warehousing. Instead of immediately focusing on technical details and step-by-step procedures in SQL Server 2008, we review the history leading up to why data warehousing is today a distinct practice and how SQL Server 2008 represents an excellent data warehousing platform.

The first section, “What Preceded Data Warehousing,” focuses on the origins of data warehousing to help you appreciate why data warehousing emerged as a distinct practice responding to industry issues. The second section, “Data Warehouse Design,” describes the two principal approaches to data warehouse design. The third section, “What Data Warehousing
Is Not,” considers various terms often confused with data warehousing and gives them distinct definitions. The fourth section, “Practical Advice About Data Warehousing,” alerts you to various common but nonobvious issues that you might encounter when building a data warehouse. Last, the fifth section, “SQL Server 2008 and Data Warehousing,” discusses SQL Server 2008–specific details as they relate to data warehousing.

With this ambitious agenda to cover in just a single chapter, we will not actually tell you much about “how” to build the perfect data warehouse—dozens of entire books are available for that. Rather, what we aim to provide is a unique combination of background, clarification of terms, identification of tricky spots, and finally some technical details about the specific data warehousing platform offered by SQL Server 2008.

Data vs. Information

At the risk of sounding pedantic, fully appreciating why data warehousing is valuable requires drawing the distinction between data and information. Data consists of recorded, characterized “facts”—for example, sale amounts initiated by customer A at store B on date C, paid for with credit card D. These facts are the amounts of the sale (numbers), while the characteristics give these numbers meaning or context. This is the sort of transactional data typically captured by an operational application.

Such characterized facts are essential, but information involves interpreting facts, identifying the relationships between them, and finding the more abstract “meaning” (if it exists) implied by them. Each characteristic, such as customer, store, date, and so on, could serve as a predicate in a query. For example, what is the pattern of sales vs. store for this customer? Or what stores have the highest sales by date? Of course, there are countless others. These sorts of questions are higher order, or value adding, because their answers enable informed decision making for the future, as opposed to mere question answering of the sort that a customer service representative might do from the facts themselves (for example, when answering the question, “what is this charge on my statement that I don’t recognize?”).

This might not seem an important distinction, but historically, it often simply wasn’t technically feasible to assemble the available data in a form suitable for informed decision making. Often, what passed for that instead was instinct and educated guesswork. In contrast, data warehousing emphasizes organizing, standardizing, and formatting facts in such a way as to enable deriving such “information” from them. Building on that, BI is then concerned with defining, extracting, delivering, and acting on that information.
What Preceded Data Warehousing

Depending on your experience, you might remember the term *electronic data processing*, also known as EDP or DP, which was used to describe the use of computers in enterprise applications for much of the 55+ years of computing history. Over the last 15 to 20 years, the term has morphed into today’s *information technology*, commonly referred to simply as IT. Although unintentional, the timing of the change and the implication of the two terms could also stand for “pre–data warehousing” and “post–data warehousing.”

Until the early to mid-1990s (when the client/server architectural paradigm reached its peak), the application of computers to enterprise needs had a strong emphasis on streamlining or automating manual clerical processes and relatively simple, repetitive high-volume tasks such as billing, payroll, inventory, and maintaining the general ledger (GL). Such applications were obvious initial targets for the application of computers in the business environment for at least three reasons:

- Their repetitive, highly constrained nature (making them relatively easy to model and suitable for automation)
- The presumed cost savings associated with that automation
- The technical feasibility given the state of the art at the time

Early input and output formats were very crude. For a long time, batch-mode processing—based on input via punched cards and output on green-bar lined printer paper—was the norm. Eventually, the state of the art advanced to allow interactive activities (giving us the now quaint and superfluous but persistent adjective *online*). Still, the application of computers to the enterprise remained largely driven by the aforementioned factors. A natural consequence was that each DP-targeted application was closely aligned with the operational process it supported, and marginally if at all with other processes. DP was about recording the basic facts of enterprise transactions while ensuring data integrity and then summarizing the results in fixed reports. The well-known term *online transaction processing* (OLTP) developed as a label for all of this.

*Electronic data processing* was an apt description of what computers and their users were doing during the pre–data warehousing period—processing data as transactions electronically (as opposed to manually)—and also what they were frequently *not* doing—turning data into information (as previously defined).

While this focus in many cases addressed operational needs adequately, it also led to a host of issues that impeded extracting a higher level of value from the data being collected. Data warehousing evolved, among other things, as a way of addressing these impediments. Let’s explore how.
Lack of Integration Across the Enterprise

The emphasis on operational processes inevitably created nonintegrated, stand-alone applications. From both enterprise and technical perspectives, each application defined essential entities as it saw fit—not just the entities unique to itself but also those “master data” entities such as customers and products that exist across the enterprise. There was typically no common understanding of what was meant by these key entities, so each application kept its own version, leading to lots of data duplication.

With this state of affairs, it was difficult or impossible to create a meaningful enterprise-wide view of just about anything. When attempted, such views were necessarily at a high level of summarization, time-consuming, and expensive to create and therefore were created only infrequently. Enterprise decision making, especially at the operational and tactical level, still depended greatly on intuition, experience, and instinct. It often simply wasn’t possible to base decisions on hard, accurate, up-to-date information. Late in the pre–data warehousing age, there were attempts to address this in the form of applications known as executive information systems (EIS) and decision support systems (DSS). These were generally ineffective because relative to their cost, they didn’t deliver enough value to their small, high-level audience.

Management Reporting and the GL

The one application that typically was enterprise-wide was the general ledger (GL). Every other major application concerned with financial information (which was many, if not most applications) had to feed accounting entries to the GL. As a result, the GL often was the single point of integration between applications because it existed and had those connections already. Also, it was accepted as an enterprise-wide single version of “the truth” by its very nature. For these reasons, most early attempts at enterprise-wide reporting were driven from the GL.

There was value in this, but there were grave limitations as well. A GL is not well suited to “management reporting,” except possibly at the highest aggregated levels, such as annual report line items. Management reporting is mostly focused on measurements of enterprise performance at much lower levels, levels which are irrelevant to the concerns of a GL—such as the profitability of specific customers. Yet once the GL became the single point of integration and thereby the source of management reporting, it started getting abused. All sorts of accounts and subledgers to support detailed management reporting proliferated in the GL, and modifications to the GL interface of source systems were made to feed them. Over time, this situation had a tendency to collapse under its own maintenance weight, especially when the GL chart of accounts needed to be restructured in the event of a merger. One of the impetuses of data warehousing was to address all this by providing a separate, appropriate environment for management reporting.
Little or No Standardized Reference Data

Closely related to lack of integration, there typically existed no single, agreed-upon “system of record” for key or master referential data across the enterprise, such as customer and product. Problems that stemmed from this included incomplete and inaccurate data, duplicated data entry (and resultant errors), and wasted effort synchronizing multiple versions from different applications. Most important of all was the inability to derive, except possibly at great effort, a consistent, comprehensive, and up-to-date view of the enterprise. In addition to these obvious consequences were some less obvious ones—for example, the embarrassment of severing a relationship with a customer who is unprofitable in one region but is overall very profitable, because you could not see the “big picture” of all your relationships with the customer across all regions, products, and organizational units.

To be sure, these problems and the reasons behind them were well recognized by the DP department and by the operational level of the enterprise almost from the beginning, and this led to attempts to create “master file” versions of the most important referentials—typically, customers, rates, products, and the organizational hierarchy. But technical limitations, political turf battles, and a lack of recognition at senior management levels of the costs of this fragmentation generally kept such efforts suboptimal.

Lack of History

Operational applications (let’s call them “OpApps”) by their very nature tend to neither require nor maintain historical data going back very far—often not more than a year or two. There are exceptions of course, such as an application that manages mortgage loans at a bank or life insurance at an insurer. These are certainly operational in nature and must also retain historical activity going back even decades perhaps. But in most cases, OpApps maintain a minimum of history in order to optimize their OLTP performance and minimize storage cost, and because there is simply no requirement to do more.

In any case, within the same enterprise, OpApps differ in the length of history maintained, its periodicity (that is, hourly, daily, weekly, monthly, and so on), and the way changes in referential data over time are handled (that is, whether a history of changes is maintained, and if so, on which attributes, and how many versions; for example, is the history of marital status or address of a customer maintained). These differences make integrating the historical data of multiple OpApps difficult, to say the least.

Data Not Optimized for Analysis

There are more significant differences between OpApps and analytical applications (“AApps,” for short). As described so far, OpApps—especially in the pre–data warehousing era—were and still are concerned mainly with reliably recording the facts of current transactions. They
have limited concern with past history or with other OpApps, which is why they came to be referred to as “islands of automation.”

In contrast, AApps are concerned with “digesting” OpApp data to provide actionable insights, predictions, and an apples-to-apples view of the entire enterprise. Sometimes such applications even combine internal and external data, such as benchmarks regarding competitors, providing a view of how the enterprise looks in a larger context. Achieving these goals requires solving all kinds of problems that OpApps do not need to be concerned with. In addition to these general differences, here are some more specific ones:

- Given their uses, OpApps are physically optimized for insert, update, and delete operations, while AApps require read or query optimization.
- The amount of data required to answer a typical OpApps query is quite small, while the amount required to answer a typical AApp query can be huge. Imagine the amount of atomic data that must be digested to answer a query such as “Who were the top 5 customers by purchases for 2007, and what were the top 5 products purchased by each of them?”
- Among the various OpApps that must be integrated for an enterprise-wide view, there are many impediments to integration, in addition to those mentioned earlier. Here are a few:
  - Entities that mean the same thing but that are named differently
  - Entities that mean different things but that are named the same
  - Different encodings of the same thing (for example, country codes)
  - Different scale and precision of measures
  - Different lengths of descriptive text for the same thing
  - Different conventions for the primary key of the same entity
  - “Smart keys”—where information is encoded in primary keys

As a Result...

- Creating any particular view of enterprise data, especially one integrated across multiple applications, was a very technical undertaking that only the DP staff could perform. Usually, there was a large backlog of requests for such views or reports.
- Many such requests (the fulfillment of which might have helped run the enterprise better) never materialized in the first place. That was because users knew that by the time the DP department could fulfill them, it would be too late to meet the business opportunity.
Each request that was fulfilled was usually implemented through a new report or extract, even if its requirements varied only slightly from an existing one. Given the technology of the time, even something as simple (as we would consider it today) as aggregating the data at a different level—say, quarterly rather than monthly—resulted in a new report. Further, even when a report already existed that could fulfill a request, there was typically no way to know that because no effective metadata was maintained about existing reports—and so a new one would be created.

Every report or extract would become permanently enshrined in the system infrastructure, forever. There was often no way to track who was using what report for what purpose (if it was being used at all), so once a report was running, it was easier and safer to just keep supporting it.

Eventually, there were extracts of extracts—one “report” would become the source for another. Keeping track of the dependencies became difficult if not impossible.

It should be obvious how all this represented a huge maintenance nightmare. But up through the early 1990s, this situation was all too common in the average “DP shop,” and it just kept getting worse. It became increasingly evident that this was a crisis in the making, and what we today call data warehousing was born in response.

In fairness, it should be noted that there were efforts to build what effectively were data warehouses long before the term was coined. But in those days, such efforts essentially re-invented the wheel each time. They could not benefit from what is available today now that techniques have matured and become codified and, thanks to the advent of the Internet, shared. It is also true that hardware advances in the form of drastically lower storage costs and fantastically improved CPU capacities have had a profound impact on the practice of data warehousing and are essential to its viability today.

**Data Warehouse Design**

The preceding discussion gives you an idea of the issues that data warehousing evolved to address. In this section, we only scratch the surface of design considerations in bringing a data warehouse into existence and hope that will whet your appetite to learn more. Fortunately, it has never been easier to learn more about data warehousing than it is today.

*Note* The value of data warehousing was not always widely accepted. In its early days, it was viewed suspiciously and considered to be just a fad or an expensive waste of time by many IT practitioners. At best it was thought of as “nice to have” and something that only the largest, best funded, and mostly for-profit enterprises could consider. Fortunately, none of this is true any longer.
Building a data warehouse requires addressing a myriad of technical *and* nontechnical issues, including the following:

- Determination of enterprise goals and objectives to be served by the data warehouse and gaining organizational buy-in for them.
- Identification of the various audiences for the data and their varying requirements.
- Addressing of latency requirements with the appropriate data architecture.
- Extract, transform, and load (ETL)—the process and tools by which data is extracted from source OpApps, cleaned and otherwise transformed as needed, and then loaded into the data warehouse. SQL Server Integration Services (SSIS) is Microsoft’s primary ETL tool for data warehousing.
- Design of entitlement, backup, mobility, scalability, delivery, and training schemes.
- Methods of end-user access to the information, including the distinction often made between reporting and analysis. The tools and products for this usually receive a disproportionate amount of attention in a data warehousing project because they are so visible.
- The embedding of an organizational ethos that the data warehouse will constantly evolve with the ever-changing needs it supports. The effort is never “done.”

The primary goal of any data warehouse is to integrate data from disparate sources into a centralized store (at least logically speaking), in a form that can be used across the enterprise for decision support by all who need it. Merely dumping all the data from various stand-alone applications into a common database is not the sort of integration we mean. Rather, a data warehouse requires a schema of some sort to which all the data brought in is made to conform. The data also needs to be “clean”—meaning that all the different ways of representing the “same” thing in the various source systems have been converted to a single consistent form. Both of these tasks are ETL responsibilities, as previously mentioned.

Based on what we’ve said so far, the 35,000-foot view of a data warehouse is shown in Figure 14-1.
With this background in place, we can now consider the two predominant data warehousing architectures guiding practice today.

The Top-Down Approach of Inmon

William Inmon is recognized as “the father of data warehousing,” having invented the term in 1990. The data warehousing features he characterized can seem self-evident today, but no one had codified them previously as he did. According to his definition, the essential characteristics of data in a data warehouse are as follows:

- **Subject-oriented**  Major entities are common across multiple OpApps. Customer, Product, Shipment, and Account are typical subject areas.
- **Integrated**  Data sources are consistent with one another along common themes.
- **Nonvolatile**  Data, once loaded, is usually never changed (updated or deleted).
- **Time-variant**  Time is part of the key to everything—"as it was at this point in time," also known as “history,” is preserved.

These features enable the previously stated goals of any data warehouse.

While an oversimplification, the Inmon style of data warehousing presumes that an enterprise data model has been or will be created—one that identifies all the “subject-oriented” entities common across multiple OpApps, the required numeric measures, the required detail level of each, and the relationships between them. It is posited that the logical data model representing this within the data warehouse is a normalized relational model of the sort associated with OLTP applications. Inmon refers to this as the “enterprise data warehouse” and to the data as being “architected.” The emphasis is on a centralized, normalized data store.

Since the typical complexity of a normalized model does not lend itself to direct query from ease of use and performance perspectives, this architecture also posits various datamarts,
which are additional derived databases whose structure is optimized for query, and which generally contain only aggregated data derived from the data warehouse. The key point is that their architecture is secondary and separate from the data warehouse proper. A refinement of Figure 14-1 that represents Inmon’s datamart concept is shown Figure 14-2.

![Figure 14-2](image)

Because this approach generally insists that a large-scale model already exists or will be created before construction of the data warehouse begins, it is usually characterized as **top-down**.

Inmon has written several books elaborating the principles and refinements of this architecture, and along with Claudia Imhoff (a long-term associate), he has elucidated an even larger architecture, the Corporate Information Factory (CIF), of which data warehousing is only a part. Space constraints preclude us from delving into further detail about the Inmon and CIF approaches. We do want to make two points before moving on, however.

The first you are probably already thinking—that requiring the existence or creation of an enterprise data model is impractical in many organizations. It **has** been successfully done, typically in larger enterprises, but many would find it impossible to justify the time and expense required to develop the model (with nothing to show at the end but documentation). No doubt when it **can** be done, it lays a very powerful foundation for informational applications, but in many cases, it is not feasible.

The second point is that many find this approach relatively abstract—useful in articulating high-level architecture but less helpful with practical details during actual development. The next approach to data warehousing that we’ll discuss, at the other end of the design spectrum, evolved to address both these realities.
The Bottom-Up Approach of Kimball

From the mid 1990s to the present, Ralph Kimball has publicized an alternative to the Inmon approach to data warehousing, the heart of which he called the Dimensional Model. If the Inmon approach can be called top-down, Kimball’s is definitely bottom-up, although both advocate a step-by-step approach. Just as Inmon articulated and formalized concepts that were already in use by practitioners, Kimball codified several practices already in use but lacking an integrative vision.

The first is the Dimensional Model, held to represent the most elegant tradeoffs between end-user intelligibility, ease of use, good performance for both predefined and ad hoc queries, and easy extensibility. The second is the idea of building the data warehouse incrementally, something most enterprises find much more palatable than the all-at-once, “big bang” approach implied by Inmon’s architecture. A key part of this is the concept of “conformed dimensions” (which we’ll define in a moment) to ensure that each new incremental data warehouse development could be integrated with what was already built, as opposed to each effort becoming the next-generation “island of automation,” or as it is usually called today, “stovepipe,” application. Third, Kimball emphasizes implementation practicality, with very specific advice on a host of data design issues advanced through his books, Web site, regular seminars, and training offerings.

Many indeed seem to find this approach desirable, as evidenced by the fact that most data analysis tools on the market today, including Microsoft SQL Server Analysis Services (which we cover in Chapters 15 through 18), have a definite affinity for the Dimensional Model. For this reason, as well as because it is less abstract, we will devote the rest of this section to an overview of this approach.

Important  Inmon and Kimball are by far the best-known data warehousing pundits. For better or worse, because their approaches are often seen as so different, each has developed a “camp” of supporters who criticize each others’ views of data warehousing best practices with sometimes religious zeal. Nonetheless, both share an emphasis on adhering to an architecture for the data warehousing design and on a step-by-step approach to design and construction. Most data warehousing projects in fact combine elements of the two approaches, which is as it should be, because each has excellent ideas to contribute. This is why it is prudent for you to be aware of them both.

This section does not purport to teach the Kimball approach. Space permits us merely to expose you to a few key concepts associated with it. This should make your further investigations easier and more effective.
Terminology

You should be aware of several useful data warehousing terms that—while closely associated with (if not always originated by) Kimball and the Dimensional Model—have come to be more broadly understood due to their representation in many tools (especially OLAP tools). You’ll see most of these terms again in the chapters that cover SQL Server Analysis Services (Chapters 15 through 18).

- **Measure** A typically numeric value of interest in reporting and analysis, such as price, balance, or inventory. As stored in a data warehouse, the relevant measures are defined by the industry of the enterprise and come from the OpApps that are its data sources. A measure is also characterized by grain, defined later in this list.

- **Dimension** The heart of the Dimensional Model, a dimension is variously described as an “axis of analysis” or a “what” qualifier. A dimension helps qualify a measure and give it context (discussed in the next section). In a query, a dimension can be part of the query result and/or part of the query constraints. The most fundamental dimension is **Time**, essential in almost any context. Others are industry-specific but typically include at a minimum **Customer, Product, and Geography**. Dimensions are typically recognized as referential or master data entities. A dimension is a collection of related values called **members**—for example, 2008 might be a member of the **Time** dimension and **John Smith** a member of the **Customer** dimension. In a Dimensional Model, the dimensions are considered to be independent of one another, even if they really are not. For example, **Customer** and **Product** are not independent, since not every customer buys every product, but by modeling each as a dimension, we treat them as if they are independent because doing so simplifies the conceptual model on which queries are based. Few if any dimensions have zero correlation with any other dimensions.

- **Hierarchy** A particular parent-child organization of members within a dimension. Each distinct set of parents is called a **level** of the hierarchy. For example, a **Time** dimension might have levels named **Year** and **Month**. The **Year** level might have members like **2007** and **2008**, while the **Month** level might have members like **Jan 2007** and **Jan 2008**, with parent members at the **Year** level of **2007** and **2008**. Hierarchies occur naturally in a wide range of applications and are nothing more than a way of grouping members for summarization. A hierarchy reflects the fact that different members of the same dimension represent different levels of detail.

- **Dimension table** A relational table containing (typically) one row per member of the dimension (depending on what form of history, if any, is maintained in the dimension). A dimension table usually has a minimum of two columns, one representing the key or identifier that uniquely defines members of the dimension and another giving a descriptive name for the member.

- **Fact table** A relational table that functions, from a data modeling perspective, as an associative entity between various dimensions. It contains one or more measure columns, and key columns of all related dimensions. It is populated (by ETL) in such a
way that the measure values are completely described by the related dimensional keys. A fact table is also characterized by its **grain** (defined later in this list), and all measures in the same fact table (should) have the same grain.

- **Star schema**  Based on what an Entity Relationship (E/R) diagram of a fact table and its related dimension tables look like, this has become a generic term for that pattern (discussed later in this section).

- **Grain**  A characteristic of a measure that is defined in terms of its related dimensions. Grain has two properties: first, precisely those dimensions that define the context of the measure; second, for each such dimension, the *level* within a hierarchy from the dimension that defines the level of detail of the measure. These two properties together define the measure’s grain. For example, if all measures in a fact table pertain to values of the *Month* level of the *Year-Month* hierarchy of the *Time* dimension, the *Time* grain of that fact table is *Month*. The overall grain of the fact table, referred to as its **granularity**, is defined by such characteristics for all its dimensions.

- **Conformed dimension**  A dimension, as previously defined, that has been designed and built in such a way that each star schema that includes the dimension can be meaningfully joined (logically) on such dimension. From a practical perspective, this means that all occurrences of such dimension in various fact tables mean the same thing—each includes exactly the same members, and each member has exactly the same meaning in relation to the facts whose context it helps define. Kimball refers to this state of affairs as the “Bus Architecture.”

It is not the case that each fact table using the dimension must use it at the same level (if it has a hierarchy). For example, if one fact table is at the *Year* level of the *Time* dimension and another is at the *Month* level, data from the two can still be meaningfully combined—it is simply necessary to aggregate the *Month* data to the level of *Year* first. Without conformed dimensions, various star schemas cannot be meaningfully combined along their common dimensions—in which case, the incremental approach to building up the data warehouse is not possible. Creating conformed dimensions is probably the most difficult part of the Dimensional Model approach, and where it most intersects with the Inmon approach—it is here that organizational agreement about which dimensions can be conformed, and what they will mean, must be secured. This is also where a lack of needed data (that is, at the required grain) in source OpApps will become apparent.

**Note**  While the term *conformed dimension* concentrates on dimensions, the grain of the measures to be given context by such dimensions is equally important. To define conformed dimensions, there must exist measure definitions whose grain in the proposed conformed dimensions is the same in all existing or contemplated fact tables.
Context and the Star Schema

As mentioned earlier, dimensions provide the context of a measure. Figure 14-3 depicts an imaginary conversation that demonstrates how context is needed to make sense of data.

<table>
<thead>
<tr>
<th>What does 492 mean?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not much. Could be a count, or an amount...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>How about 492.00?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Looks like it’s a financial amount...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>It occurred on February 1, 2004.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Well...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The Central Division organization ...?</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK, 492.00 on Feb 1, 2004 associated with Central Division—what am I supposed to do with that?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Corporate Department ...?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes, but...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Travel Lodging ...?</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK, now we’re getting somewhere. That sounds like an expense. So, on Feb 1, 2004, the Corporate department of the Central Division incurred 492.00 in Travel Lodging expense. But wait a minute, expenses can be actual or budgeted...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>It’s an Actual...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Now I know what 492.00 means!</td>
</tr>
</tbody>
</table>

**FIGURE 14-3** Determining the context of a measure

**Note** Actually, do we really now know everything necessary to give 492.00 complete context? Not unless we make a further assumption. Can you guess what? Of course—what currency is this in?

Now let’s diagram this conversation, as shown in Figure 14-4.
We can examine an actual implementation of the preceding example. Run the code shown in Listing 14-1 against the AdventureWorksDW2008 sample database to retrieve our exact case.

**LISTING 14-1** Querying AdventureWorksDW2008 for the value of a particular measure

```sql
USE AdventureWorksDW2008
GO

SELECT
dd.FullDateAlternateKey,
do.OrganizationName,
ddg.DepartmentGroupName,
da.AccountDescription,
ds.ScenarioName,
ff.Amount
FROM
FactFinance ff
INNER JOIN DimDate AS dd
ON ff.DateKey = dd.DateKey
INNER JOIN DimOrganization AS do
ON ff.OrganizationKey = do.OrganizationKey
INNER JOIN DimDepartmentGroup AS ddg
ON ff.DepartmentGroupKey = ddg.DepartmentGroupKey
INNER JOIN DimScenario AS ds
ON ff.ScenarioKey = ds.ScenarioKey
INNER JOIN DimAccount AS da
ON ff.AccountKey = da.AccountKey
WHERE
dd.FullDateAlternateKey = '2/1/2004' AND
do.OrganizationName = 'Central Division' AND
ddg.DepartmentGroupName = 'Corporate' AND
da.AccountDescription = 'Travel Lodging' AND
ds.ScenarioName = 'Actual'
```

**Note** The sample AdventureWorksDW2008 database implements a schema that illustrates a Kimball-inspired data warehouse. Refer to this book's Introduction for instructions on locating and downloading this sample database.
From this query and the E/R diagram that represents the tables involved, we can see in Figure 14-5 what is meant by a star schema.

![Diagram of a star schema from AdventureWorksDW2008](image)

**FIGURE 14-5** A star schema from *AdventureWorksDW2008*

### Surrogate Keys

The *surrogate key* concept is not original to Kimball or the Dimensional Model, but it is something they strongly advocate. A surrogate key is a system-assigned, typically integer, primary key to a table. In SQL Server, the surrogate key would typically be an identity column, although sometimes a particular architecture might find it preferable to have a central *key generator* that gives out surrogate keys as needed. Surrogate keys have two important characteristics, as follows:

- They have no embedded encodings—that is, they are not “smart” keys. This makes them immune to changes in the source data that would plague nonsurrogate primary keys. One reasonable exception to this is the surrogate key of the *Time* dimension, where making the surrogate integer key smart by representing *YYYYMMDD* (when applicable to the grain of the fact tables) can make partitioning the fact tables much easier.

- As integers, they are the most efficient possible primary keys, both from performance and storage perspectives.

This concludes our brief review of the Kimball approach to data warehousing. You are strongly encouraged to consult the references at the end of this section, as well as appropriate Web searches, for a great deal more information. We'll close here with Figure 14-6, which illustrates what a data warehouse built to Kimball principles looks like. An important aspect to observe in this figure is that the data warehouse *is* the collection of star schemas—there are no separate datamarts, as in the Inmon approach. (And by the way, in an Inmon data warehouse, there is no objection to the datamarts following the Kimball architecture.) Although not shown in this figure, it is assumed that the various star schemas are not *disjoint*, meaning that wherever they share a functional dimension such as *Customer* or *Product*, they have been constructed in such a way as to actually share a single version of the dimension. When this is done, the data in the various star schemas can be validly combined along the common
dimensions—a property derived from them having been constructed to be “conformable,” in the parlance of the Dimensional Model.

**FIGURE 14-6** A Kimball-oriented data warehouse

## What Data Warehousing Is Not

Much confusion exists in the literature and among practitioners because many terms are regularly conflated with data warehousing, even now when the maturity of the field should preclude this. A charitable view is that this was at least understandable in the past when the field was evolving rapidly in theory, practice, and product. But today, there ought to be more clarity, precision, and common understanding. In furtherance of this, we feel it is worth asserting that there are worthwhile distinctions still represented by certain overused and misused terms. This section provides a brief summary of some of these terms.

### OLAP

The term *online analytical processing*, or OLAP, was coined by Dr. E. F. Codd (the originator of the relational model) in 1994 to distinguish a set of properties that analytical applications should satisfy (in contrast with his famous 1985 publication of “12 Rules” that a relational database management system should satisfy; see [http://en.wikipedia.org/wiki/Codd’s_12_rules](http://en.wikipedia.org/wiki/Codd’s_12_rules)). The term was intended to draw distinctions between the at-the-time well-known properties of OLTP applications and the less-well-defined properties of analytical applications. It is probably most valuable simply for emphasizing that such a distinction should be made. Today the term can be understood also as referring to a response to the limitations of spreadsheet-based approaches. While not strictly part of the definition, as a practical matter, cube-based technology is now usually associated with OLAP.
Note  As with data warehousing, there were OLAP-like efforts long before the term OLAP was coined that were recognizable precursors, going back to the 1960s.

An OLAP application often, although not of necessity, draws its data from some form of star schema. The various OLAP tools on the market today form a spectrum in the degree to which they require a recognizable star schema as their data source. At one end, some tools can deliver OLAP functionality, with relatively simple calculations, from just about any data source with any organization, while at the other end are tools that can use only cubes (a data structure designed to facilitate fast analysis, further described in Chapter 15) as their data source (and hopefully can fully exploit their power). A data warehouse is very helpful as the source anywhere on this spectrum and is a virtual necessity on the cube-oriented end of it.

More Info  The Fast Analysis of Shared Multidimensional Information (FASMI) test is a more precise, alternative definition of the properties that the term OLAP aspired to distinguish, developed by the authors of The OLAP Report. For a detailed definition of FASMI, as well as links to a wealth of other excellent OLAP information (much of it free), see http://www.olapreport.com/fasmi.htm.

In the context of SQL Server, Analysis Services is Microsoft's full-featured OLAP engine; it is covered in detail in Chapters 15 through 18.

Data Mining

The traditional way of extracting information from data requires a skilled analyst with a deep understanding of the enterprise who formulates ad hoc queries, the answers to which he or she think would be interesting—for example, “What was the impact of last month’s sales promotion on sales?” or “Which stores in the top 10 by sales this year were also in the top 10 by sales last year?” In effect, the analyst forms hypotheses of cause and effect and then tests them against the data. To be effective, this rather hit-or-miss style of information discovery requires tools that permit easily formulating the queries and fast response so that the analyst can maintain his or her train of thought. OLAP technology is ideally suited for this.

In contrast, data mining is an approach in which correlations that might exist in a data set are automatically “discovered” using specialized data models and statistical algorithms. Because it is automated, it is more thorough in finding correlations, and it is unaffected by the prejudices and blind spots that an analyst would have using an ad hoc approach. The analyst still needs to evaluate each correlation found to determine whether it is meaningful or merely correlative, however.

In principle, data mining does not require a data warehouse for its source data. However, a well-crafted data warehouse with clean data could be an ideal source. The intended analysis
and the allowable latency also affect whether a data warehouse as an analysis source is feasible. For example, in detecting credit card fraud, is the data warehouse updated often enough to be useful?

Starting with SQL Server 2000, Microsoft has invested much effort in giving SQL Server Analysis Services data mining capabilities that are much easier for relative nonspecialists to use than what has previously been available on the market. These capabilities are covered in detail in Chapter 18.

Business Intelligence

The term business intelligence (BI), coined by analyst Howard Dressner in 1989, has turned out to be quite popular. Today it is applied in so many contexts that you would be right to wonder whether it distinguishes anything anymore. Some argue that it doesn’t, but we think that it still does. It is unfortunate that the business in BI obscures the fact that BI can be valuable in any enterprise, not just the for-profit ones implied by the B. So as suggested earlier, think enterprise intelligence when you hear business intelligence.

The most important thing to be clear about is that BI, properly understood, is not about any particular technology—although its implementation certainly depends on technology. BI is fundamentally a management approach and philosophy. Like most good ideas, its basic premise sounds so obvious when stated that it hardly seems worth noting: management decisions should be based on facts, not on educated guesswork, politics, or other subjective bases. Of course, management of an enterprise has always been based at some level on objective information—accounting being the most elemental form. But in the past, such objective measures, especially at the enterprise level, were at a summary level, produced infrequently (if periodically), rigidly structured, and incapable of easily revealing the detail from which they were derived.

BI aims to change all this by ensuring that information is accurate, reliable, updated as frequently as necessary, and readily accessible to whoever needs it, regardless of their level in the organization. One focus of BI is on the technologies required to achieve these goals, which generally include some form of data warehouse—hence the association. But the technology focus, especially on user interfaces (UIs), tends to receive disproportionate attention. An equally important focus should be on the vision of fact-based decision making that is supported by senior management and influences the way the enterprise will be run.

Initially, BI often faced significant resistance in the enterprise. If knowledge is power, losing control of knowledge feels like (and often is) losing power. BI threatened this with its emphasis on making information available to a much broader audience. Fortunately by now, the value of BI is recognized in most enterprises.

Last, we must mention that historically, many BI projects and their supporting data warehouse implementations have overpromised and underdelivered, giving BI a bad reputation
for being expensive and risky. As a result, some are beginning to rethink the necessity of creating a data warehouse to support BI and instead are using existing reports and other existing data sources directly as BI sources. While this approach has its appeal, only time will tell whether it becomes an important theme in BI implementation.

Dashboards and Scorecards

The terms *dashboard* and *scorecard* are often used synonymously. They both represent information graphically, summarizing it with various elements showing relative magnitudes, trends, and other meaningful relationships. But they are not synonymous.

Dashboards

A dashboard, like its automobile namesake, displays measures without the context of related goals. It has a “just the facts” tactical orientation and is updated as often as necessary for the (typically) operational process that it supports. It is more generic than a proper scorecard in that it can display anything (including a scorecard). Figure 14-7 shows a typical dashboard.

*FIGURE 14-7* A typical dashboard
Scorecards

A scorecard displays base measures in the context of related goals, objectives, or target measures and provides at-a-glance visual cues as to whether each such base measure is lagging, achieving, or surpassing its goal measure. Obviously, therefore, a scorecard is not possible unless such goal measures exist in addition to the base measures. A strategy must be devised for such goal measures to exist. It follows that a scorecard is strategic, whereas a dashboard is tactical and operational.

The term *key performance indicator* (KPI) is closely associated with scorecards. The traffic light and trend indicators in Figure 14-8 are KPIs. A KPI encapsulates a measure, a related goal measure, a calculation about the relationship of the two, and a graphic that expresses a “good or bad” indication based on the calculation.

![A typical scorecard](image)

**FIGURE 14-8** A typical scorecard

Goal measures are usually not defined at lower levels of detail. Consider the difference in grain between *Actual* and *Plan* measures—the former derive from individual transactions, while the latter are created at a much more summarized level, at least in the *Time* dimension. For this reason, scorecards tend to report at a more summarized level than dashboards, which is consistent with their strategic vs. tactical orientation. This in turn also means that changes occur more slowly, so scorecards are usually refreshed less often than dashboards. In a financial scorecard like the one shown in Figure 14-8, an *Actual vs. Plan* KPI exhibits all
these principles and is seen as a traffic light in the *Plan* columns. Notice the *Trend* indicator, which is also a KPI that uses some calculation between prior-period *Actual* and *Plan* values.

Since SQL Server 2005, Analysis Services provides KPI objects that can be stored in cubes. They can be consumed and displayed by Microsoft Office Excel 2007, Microsoft Office SharePoint Server, and Microsoft Performance Point, each of which also allows creating and storing KPIs within its respective environment.

More Info  See Chapter 16 for advanced OLAP coverage that includes KPIs.

**Performance Management**

*Performance management* is a relatively recent term that is a particular flavor of BI but rates its own discussion because of its currency in the literature and market today as a distinct entity. Performance management implies BI—but the converse is not true, because BI is the more general term. As noted earlier, BI’s techniques can be focused in many different directions. Performance management is a specific application of BI. It is first about establishing organizational goals and objectives and ways of measuring progress toward meeting them—often using BI techniques to help determine what those goals and measures should be. Once these goals are established, it is then about gathering past, current, and projected performance, explicitly measuring these against the established goals, and widely disseminating how well goals are being met. This is usually achieved in the form of scorecards, which are again facilitated by BI tools and techniques.

The Balanced Scorecard (BSC) is a well-known example of performance management that predates the term. It is worth becoming familiar with the BSC approach, not least because it can help you better understand the factors driving enterprise strategy, and how to ensure that the strategy is enacted.


**Practical Advice About Data Warehousing**

A data warehousing effort requires both theory and discovery. Although the theory associated with building a data warehouse could be considered a rather well understood topic today, practical experience still has much to offer. In this section, we’ll look at a few of the data warehousing best practices that we have found most valuable.
Anticipating and Rewarding Operational Process Change

It is almost certain that a data warehousing effort will identify data elements and relationships essential to realizing the enterprise goals that are not currently captured in the operational processes. It is also likely that those who would be most directly affected in their day-to-day work by addressing this will feel that they have nothing to gain by doing so, and often something to lose. For example, an enterprise goal might be to capture which sales groups should get credit, and in what proportion, for working together to make a sale happen—the better to apportion the bonus pool of the sales force. Enabling this requires capturing information about which sales groups were involved at the time the sales transaction is recorded. This is information that is likely not currently available in the workflow of the back-office staff who record the transaction, and moreover, even if it is (or is made to be), the extra time it would take them to record it will reduce the number of transactions they can process per hour. They will most likely resist, given the impact on their productivity, unless this effort is officially recognized and proper incentives are put in place to motivate their cooperation.

Rewarding Giving Up Control

As suggested earlier in this chapter in the section “Business Intelligence,” a successful data warehousing/BI effort often requires those who have traditionally been in control of key data to relinquish that control in the interest of the greater good. Any organizational change effort will threaten those who perceive themselves the losers in some way (often correctly), and it is only natural for them to resist the change. If the enterprise recognizes this and provides positive motivators to take this risk, the chances of success are increased. How feasible this is, of course, depends greatly on the organizational culture. The BSC approach can be particularly valuable in this regard.

A Prototype Might Not Work to Sell the Vision

Building a prototype or proof of concept (POC) for a data warehousing/BI approach is often recommended as a way to achieve buy-in from important stakeholders. It is easy to assume that a representative POC will do the trick. By representative, we mean that the important technical capabilities are demonstrated as feasible (such as whether particular relationships can be modeled successfully), even if this is illustrated with fictitious data such as the AdventureWorksDW2008 database.

What you might not realize until it is too late is that stakeholders can find it difficult to appreciate such an approach, particularly when the POC is not based on measures they recognize or the values used are not realistic. If you hear people in your audience calling out “Hey, that number isn’t right!” while you are demonstrating the POC, that’s exactly what’s happening. Logically, in a POC, it might not matter whether the data is accurate, but once your
stakeholders lose interest or faith, it can be very difficult to regain. Focusing on such issues is also a favored tactic of those who oppose the data warehouse for whatever reason.

For a POC to have the best chance of success, it should be as realistic and as attuned to the work of the stakeholders who will be judging it as possible. This often runs counter to the idea that a POC requires a minimal investment, which is exactly why we are making this point. The data warehousing project can get shot down before it even gets off the ground with an ill-conceived POC.

**Surrogate Key Issues**

The value of using integer surrogate keys in a data warehouse was discussed earlier in this chapter in the section “Data Warehouse Design.” But their use is not without issues, as described here:

- In general, surrogate keys should not be “smart”—that is, they should not have any significant meaning encoded in their values. However, an exception might be worth considering for the Time dimension. At the physical level, there can be value in the Time surrogate key taking the form YYYYMMDD, YYYYMM, or YYYYWW (where Y, M, D, and W are year, month, day, and week values), all of which are easily represented as an integer. Two reasons justify this violation of the normal best practice. First, if the Time surrogate key column is the first in the composite primary key of the fact table (as it usually should be) and the primary key has a clustered index, the fact data will be optimally organized for the Time constraint of the typical query—which is usually either a point in time or a range. Second, such a smart Time key will make it much easier to implement and maintain physical partitioning of the Time dimension at the relational database level.

- Surrogate keys can be generated in several ways, two principal ones being `IDENTITY` columns or a row-by-row assignment facility—for example, `SELECT MAX(Id) + 1`—using appropriate locking mechanisms. Regardless of the method, complications can arise in the typical multi-environment setting—that is, development, quality assurance (QA), and production. Assume that at the start of a development cycle, your development environment is refreshed from production. Then you also copy over ETL input files from production and run the ETL process in development (perhaps as part of a parallel test). Depending on how surrogate keys are assigned, there can be a good chance that the same data (from a business key perspective) is assigned different surrogate keys in development and production. This can greatly complicate reconciliation between the two.

**Currency Conversion Issues**

Particularly in larger, multinational enterprises, financial applications usually require currency conversion in order to compare similar items (apples to apples). Be aware that this is a subject
fraught with business rule and design conundrums. Since SQL Server 2005, Analysis Services has provided features that can make implementation of currency conversion calculations in the cube easier.

But this does not address the issues we want to highlight here, which relate to the tension between designing for ad hoc, not-known-in-advance queries and needing to know something, possibly a lot, about likely queries, if a suitable design is to be derived. Issues around currency conversion illustrate this particularly well. There are no “right” answers to the following questions, but you would do well to consider all of them if currency conversion is in any way a part of your business perspective:

- What flexibility is required? Will there be one master currency in which all comparisons are expressed, several standard currencies, or in any existing currency?
- Closely related to the preceding questions, does it make sense to precalculate and store converted amounts, or must this be done on the fly?
  
  As with all rates and ratios, care must be taken where aggregation is involved to force the currency conversion to be at the appropriate leaf level of detail, followed by aggregation to the required summary level. The capabilities of your OLAP tool influence this greatly.
- Are converted amounts to be at the rate in effect at their original point in time only, or should amounts also be convertible based on the rates at any point in time?
- At what rates should future values (for example, Budget) be converted: the rates in effect when the budget is finalized, never after to be adjusted? Or should current rates be used, adjusting the projections every period? Must you be able to distinguish how much of a variance between Actual and Budget is due to currency conversion vs. changes in the Budget measure itself?

The design driven by answers to these business questions has profound effects on both the questions that can be answered later and the technical complexity required.

**Events vs. Snapshots**

There are two complementary approaches to data warehouse logical design: the event-driven approach and the snapshot approach. Both involve tradeoffs in complexity and in the sort of inquiries they can support.

On the one hand, it can be argued that everything of analytical interest in an enterprise can be represented as an event. Events are items like a payment or an order being received or a shipment getting delivered. Events by definition occur asynchronously at points in time. In principle at least, if all relevant events can be identified and captured, it is possible to deduce the state of affairs at any point in time, as well as how that state came to be. For some informational applications, this is critical. Constructing the point in time from events can, however, be exceedingly complex.
On the other hand, a snapshot-based approach does not record events at all. Instead, it simply periodically records the aggregate effect of events. Answering queries about the points in time where snapshots were taken is obviously much easier than it would be with a purely event-based approach, where the state at the point in time would need to be reconstructed.

These approaches sometimes need to be combined. For example, with an Account entity, often the only thing of interest is the account balance at periodic points in time, such as month-end. On the other hand, it is also imperative to be able to query each and every event (debit or credit) that affected the balance since the previous snapshot.

Events and snapshots have considerations in addition to which functional questions they support. There is the question of what the source system can provide in terms of either events or snapshots, which has an impact on how much work must be done in the data warehouse ETL to create one or the other. Also, a snapshot approach that takes a snapshot of everything, regardless of how much or little has changed since the last snapshot can lead to data proliferation and can be inefficient compared with an event-based approach when changes are relatively few—although this can be addressed with techniques such as Change Data Capture, detailed later in this chapter.

It is well worth spending considerable time during the design phase thinking through the implications of both approaches before determining the best choices for your requirements.

**SQL Server 2008 and Data Warehousing**

Earlier versions of SQL Server had new features related to data warehousing, most notably Analysis Services, Reporting Services, and in SQL Server 2005, certain features of SQL Server Integration Services such as the Slowly Changing Dimensions task. But these earlier versions had very little at the level of the relational engine specifically targeting the particular needs of data warehousing. SQL Server 2008 delivers new features that squarely target data warehousing, particularly in relation to making very large databases more manageable and cost effective. This section will review the most important of the data warehousing–oriented enhancements in SQL Server 2008, starting with the Transact-SQL (T-SQL) enhancements aimed at working with data warehousing.

**T-SQL MERGE Statement**

The `MERGE` statement is covered in more depth in Chapter 2 and is applicable to many more scenarios than data warehousing. We cover it here too because it is also very relevant to data warehousing, specifically in the ETL context.

The `MERGE` statement provides what’s commonly referred to as upsert—meaning update the row if it already exists; otherwise, insert it. But there is more as well. `MERGE` requires a target table, which is joined in some relationship to a source table. The source table contains the data to be merged or synchronized with the target table. The `MERGE` statement supports
up to three types of clauses defining the row-by-row action to be taken on the target table based on how it compares with the source table:

- **WHEN MATCHED**  The row exists in both merge and target tables (performs an inner join and allows *UPDATE* or *DELETE*).

- **WHEN NOT MATCHED BY TARGET**  The row exists in the source table but not the target table (performs a left outer join and allows *INSERT*).

- **WHEN NOT MATCHED BY SOURCE**  The row exists in the target table but not the source table (performs a right outer join and allows *UPDATE* or *DELETE*).

More Info  Each merge clause can also state constraints in addition to the implied join, such as another condition comparing column values between source and target. However, there are some very particular rules governing the use of multiple merge clauses and their various combinations. We cover those in the full treatment given to the new *MERGE* statement in Chapter 2.

In the data warehousing context, the *MERGE* statement is particularly suited to the maintenance of the dimension tables of star schemas. It is also very helpful in maintaining Type 1 slowly changing dimensions (SCDs), where changes simply overlay existing values, and Type 2 SCDs, where *MERGE* can do part of the job (a separate *INSERT* operation is still needed when an existing row is updated, to create the new version of it.) See the section entitled “Data Warehouse Design” earlier in this chapter for more details. (A full treatment of SCDs is beyond the scope of this chapter.)

In SQL Server 2008 Integration Services, *MERGE* can streamline and simplify the insert/update pattern that would be required under SQL Server 2005 Integration Services. Previously, the decision to insert or update in SQL Server 2005 Integration Services had to be based on a lookup of the source row using a Lookup task that was loaded with the target rows and two output data flows based on the failure or success of the lookup: one doing inserts and one doing updates against the target. With *MERGE*, the Lookup task is no longer needed, which simplifies the Integration Services package and avoids the performance, memory, and deadlock issues that can arise with the Lookup task if the target table is large.

Syntactically, *MERGE* requires two joinable tables or table-equivalents. (The target must be either a table or an updatable view; the source can be any table-equivalent.) For Integration Services, this means that the source table must exist or must be created in the package (as a temporary table, common table expression [CTE], or other equivalent).

The code in Listing 14-2 shows a series of representative T-SQL expressions using *MERGE* against the *AdventureWorksDW2008* database. Run each statement by hand as directed by the comments, followed by running the *MERGE* statement at the end. Note that *GeographyKey* is an identity column in *DimGeography*, so the column list must be explicit in the *INSERT* statement in the *MERGE* statement’s *WHEN NOT MATCHED BY TARGET* clause. Also note that the ending semicolon is required to terminate the *MERGE* statement.
More Info  All the data manipulation language (DML) statements in T-SQL (INSERT, UPDATE, DELETE, and MERGE) support an OUTPUT clause, which can be quite useful for archiving changed data. In addition, the new INSERT OVER DML feature in SQL Server 2008 enhances the OUTPUT clause with filtering capabilities. See Chapter 2 for details of the OUTPUT clause and INSERT OVER DML.

LISTING 14-2 Using MERGE for a data warehousing update

```sql
USE AdventureWorksDW2008
GO

-- Make a copy of the table.
SELECT * INTO DimGeographyTest FROM DimGeography

-- Create "Changes" table as another copy of same data.
SELECT * INTO Changes FROM DimGeography

-- If you now run the MERGE statement below, no changes will be reported. Note
-- the condition on the UPDATE clause, which prevents unnecessary updates.

-- Now force some UPDATES (53):
UPDATE Changes
SET SalesTerritoryKey = 11
WHERE SalesTerritoryKey = 10

-- Now running MERGE reports 53 updates.

-- Now force DELETES (empty table will effectively delete every row in
-- DimGeographyTest):
DELETE Changes

-- Now running MERGE will delete all 653 rows in DimGeographyTest.

-- Testing INSERT is left as an exercise for the reader.

-- MERGE statement:

MERGE DimGeographyTest AS dg
USING (SELECT * FROM Changes) AS c
ON dg.GeographyKey = c.GeographyKey
WHEN MATCHED and dg.SalesTerritoryKey <> c.SalesTerritoryKey THEN
  UPDATE SET dg.SalesTerritoryKey = c.SalesTerritoryKey
WHEN NOT MATCHED BY TARGET THEN
  INSERT (City, StateProvinceCode, StateProvinceName,
         CountryRegionCode, EnglishCountryRegionName,
         SpanishCountryRegionName, FrenchCountryRegionName,
         PostalCode, SalesTerritoryKey)
```
The deletion possibilities of \textit{MERGE} would be rare in a data warehousing scenario except in single-instance fixes of erroneous data, but it is worth knowing about for that purpose alone. In general, beware of using \textit{DELETE} with \textit{MERGE}. If your source table is inadvertently empty (as it is eventually in our example), \textit{MERGE} with a \textit{WHEN NOT MATCHED BY SOURCE} clause specifying \textit{DELETE} could unintentionally delete every row in the target (depending on what other conditions were in the \textit{WHEN NOT MATCHED BY SOURCE} clause).

\section*{Change Data Capture}

Like one use of \textit{MERGE}, the new Change Data Capture (CDC) feature in SQL Server 2008 targets the ETL component of data warehousing. CDC is available only in the Enterprise edition of SQL Server 2008 (and of course, the functionally equivalent Developer and Evaluation editions).

\begin{quote}
\textbf{Note} SQL Server 2008 provides a number of change tracking features—each one tailored for a specific purpose. In particular, CDC addresses data warehousing, SQL Server Audit addresses security (see Chapter 5), and SQL Server Change Tracking targets synchronization of occasionally connected systems and mobile devices using ADO.NET Sync Services (see Chapter 13).
\end{quote}

CDC is designed to efficiently capture and record relevant changes in the context of a data warehouse. Traditionally, detecting changes in an OpApp table that need to be applied to a data warehouse has required relatively brute force methods such as the following:

\begin{itemize}
  \item For updates, using the \textit{CHECKSUM} function as a shortcut to detecting inequality of columns between source and target rows (SQL Server only), or comparing time stamps.
  \item For inserts, outer-joining source and target rows and testing for \textit{NULL} on the target.
  \item For inserts and updates, implementing triggers on the source table to detect changes and take appropriate action against the target, or performing a lookup (perhaps using an Integration Services Lookup task) to compare source against target and then driving the update or insert by the success or failure of the lookup.
  \item For inserts and updates, using the \textit{OUTPUT} clause (SQL Server 2005 and 2008) or \textit{INSERT OVER DML} (SQL Server 2008 only), which we cover in Chapter 2.
\end{itemize}
The CDC feature introduced in SQL Server 2008 provides a valuable new way of laying the groundwork for maintaining changing data in a data warehouse. Without resorting to triggers or other custom code, it allows capturing changes that occur to a table into a separate SQL Server Change Tracking table (the change table). This table can then be queried by an ETL process to incrementally update the data warehouse as appropriate. Querying the change table rather than the tracked table itself means that the ETL process does not affect the performance of applications that work with the transactional tables of your database in any way. CDC is driven by a SQL Server Agent job that recognizes changes by monitoring the SQL Server transaction log. This provides much better performance than using triggers, especially in bulk load scenarios typical in a data warehouse—and there’s no code to write or maintain with CDC. The tradeoff is somewhat more latency, which in a data warehouse is often perfectly acceptable. Figure 14-9 depicts a high-level view of CDC architecture using an illustration taken from SQL Server Books Online.

FIGURE 14-9 High-level architecture of CDC
Several new system stored procedures and table-valued functions (TVFs) are provided to enable, monitor, and consume SQL Server Change Tracking output. To begin, you execute the `sp_cdc_enable_db` procedure to enable CDC on the current database. (You must be in the `sysadmin` role to do this.) When you run this procedure, a new `cdc` user, `cdc` schema, and `CDC_admin` role are created. These names are hard-coded, so in the event that you already have a user or schema named `cdc`, you will need to rename it before using CDC.

Once the database is CDC-enabled, you enable CDC on a given table by executing `sp_cdc_enable_table`. (You must be in the `db_owner` role to do this.) When you do that, several objects are created in the `cdc` schema: a change table and at least one (but possibly two) TVFs. Let’s look at each of these objects in turn.

When CDC is enabled on a table, SQL Server creates a change table in the `cdc` schema corresponding to the table on which CDC is being enabled. The change table will be populated with change data automatically by CDC and is assigned a name based on both the schema and the table being tracked. For example, when you enable CDC on the `Employee` table in the `dbo` schema (as we’ll do shortly), SQL Server creates a corresponding change table named `cdc.dbo_Employee_CT` that will record all changes to the `dbo.Employee` table. The schema of the tracked table (`dbo` in this case) is part of the change table name so that same-named tables from different schemas can all be unambiguously tracked in the `cdc` schema. It is also possible to explicitly name the change table, as long as it’s unique in the database.

The ETL process will query this change table for change data in order to populate your data warehouse, but it will not normally do so by selecting directly from it. Instead, the ETL process will call a special TVF to query the change table for you. This TVF is also created for you by SQL Server automatically when the change table is created, and—like the change table—the TVF is also created in the `cdc` schema with a name based on the schema and table name of the tracked table. So again, if we’re tracking the `dbo.Employee` table, SQL Server creates a TVF named `cdc.fn_cdc_get_all_changes_dbo_Employee` that accepts parameters to select all changes that occur to `dbo.Employee` between any two desired points in time.

If you specify `@supports_net_changes=1` when calling `sp_cdc_enable_table`, a second TVF is created for the change table as well. Like the first TVF, this one allows you to select changes between any two points in time, except that this TVF returns just the `net` (final) changes that occurred during that time frame. This means, for example, that if a row was added and then deleted within the time frame being queried using this second TVF, data for that row would not be returned—whereas the first TVF would return data that reflects both the insert and the delete. This second TVF is named in a similar fashion as the first, except using the word `net` instead of `all`. For `dbo.Employee`, this TVF is named `cdc.fn_cdc_get_net_changes_dbo_Employee`. Note that querying for net changes requires the tracked table to have a primary key or unique index.

Neither of these TVFs accept start and end times directly but instead require the range to be expressed as log sequence numbers (LSNs) by first calling `sys.fn_cdc_map_time_to_lsn`. So to
query between two points in time, you call `sys.fn_cdc_map_time_to_lsn` twice—one for the start time and once for the end time—and then use the LSN values returned by this function as input values to the TVFs for querying change data. This might seem unnecessarily cumbersome, but in fact has good reason related to supporting two change tables on the same table, one feeding the production systems and another supporting ongoing development.

**Tip** The start and end times this function is called with are not required to fall within the range of time actually represented in the log. If either time falls outside the boundaries in the log, the function "does the right thing": it returns the earliest existing LSN if the specified start time is prior to the earliest LSN, and it returns the latest existing LSN if the specified end time is after the latest LSN. This will be implicitly illustrated shortly in Listing 14-3 for both start and end times.

The `sp_cdc_enable_table` stored procedure has several optional parameters that give you a lot of flexibility. You can, among other options, specify your own name for the change table, a role that a user must belong to in order to query changes (if not in `sysadmin` or `db_owner`), which columns of the table should be tracked (you don’t need to track all of them), the filegroup on which to create the change table, and whether the `SWITCH_PARTITION` option of `ALTER TABLE` can be executed against the tracked table (which has very important implications). Consult SQL Server Books Online for more details of `sp_cdc_enable_table` parameters.

When you no longer require CDC on a particular table, you can call the `sp_cdc_disable_table` stored procedure on the table. This procedure drops the change table and the TVFs and updates the system metadata to reflect that the table is no longer tracked. When you no longer require CDC on the database, call the `sp_cdc_disable_db` stored procedure to completely disable CDC for the entire database.

**Important** You should be aware of several considerations before dropping a database on which CDC has been enabled. To drop a CDC-enabled database, you must either stop SQL Server Agent or first disable CDC by running `sp_cdc_disable_db` on the database to be dropped. If you take the former approach, the SQL Server Agent jobs will be deleted automatically when SQL Server Agent is next started upon detecting that the database the jobs were associated with is no longer present. Of course, SQL Server Change Tracking for other databases running on the server instance will also be suspended while SQL Server Agent is stopped. The latter approach is the preferred method, since it does not interfere with other CDC-enabled databases and will remove all CDC artifacts related to the database being dropped.

The change table records all changes to the requested columns, including intermediate states (per DML statement) between two points in time. Note that CDC supports sparse columns (covered later in this section) but not sparse column sets. Each change table row also includes five metadata columns of great value for change-consuming processes to determine what type of change (insert, update, or delete) each row represents and to group and order all changes belonging to the same transaction. One item it cannot capture is who made the
change, which is why it is not ideal for maintaining audit trails. For that, you can use SQL Server Audit, which will track and record which users are performing data modifications as well as any other activity of interest. (We cover SQL Server Audit in Chapter 5.)

As we mentioned earlier, CDC relies on SQL Server Agent for automating the capture process. The first time `sp_cdc_enable_table` is executed on any table in a database, SQL Server also creates two SQL Server Agent jobs for that database. The first is a change-capture job, which performs the actual transaction log monitoring to apply changes on the tracked table to the corresponding change table. The second is a cleanup job, which deletes rows from change tables after a configurable interval (three days, by default) and removes all CDC artifacts if the tracked table is dropped. Therefore, SQL Server Agent must be running the first time this procedure is run to CDC-enable a table on any database in the server instance. Subsequently, if SQL Server Agent stops running, changes to tracked tables will accumulate in the transaction log but not be applied to the change tables until SQL Server Agent is restarted.

CDC can at first appear rather cumbersome to use, but it is well thought out in terms of its configuration flexibility and support for various scenarios. Some of these might not be immediately obvious—for example, what happens if a tracked table is dropped, or its structure changed, after CDC is enabled on it? We lack the space to delve into these essential aspects, but you’ll find comprehensive details in SQL Server Books Online. The code in Listing 14-3 shows a complete example of using CDC.

**LISTING 14-3 Using Change Data Capture**

```sql
-- Create test database
CREATE DATABASE CDCDemo
GO

USE CDCDemo
GO

-- Enable CDC on the database
EXEC sp_cdc_enable_db

-- Show CDC-enabled databases
SELECT name, is_cdc_enabled FROM sys.databases

-- View the new "cdc" user and schema
SELECT * FROM sys.schemas WHERE name = 'cdc'
SELECT * FROM sys.database_principals WHERE name = 'cdc'

-- Create Employee table
CREATE TABLE Employee(
    EmployeeId    int NOT NULL PRIMARY KEY,
    EmployeeName  varchar(100) NOT NULL,
    EmailAddress  varchar(200) NOT NULL
)

-- Enable CDC on the table (SQL Server Agent *should* be running when you run this)
```
EXEC sp_cdc_enable_table
   @source_schema = N'dbo',
   @source_name = N'Employee',
   @role_name = N'CDC_admin',
   @capture_instance = N'dbo_Employee',
   @supports_net_changes = 1

-- Show CDC-enabled tables
SELECT name, is_tracked_by_cdc FROM sys.tables

-- Insert some employees...
INSERT INTO Employee VALUES(1, 'John Smith', 'john.smith@ourcorp.com')
INSERT INTO Employee VALUES(2, 'Dan Park', 'dan.park@ourcorp.com')
INSERT INTO Employee VALUES(3, 'Jay Hamlin', 'jay.hamlin@ourcorp.com')
INSERT INTO Employee VALUES(4, 'Jeff Hay', 'jeff.hay@ourcorp.com')

-- Select them from the table and the change capture table
SELECT * FROM Employee
SELECT * FROM cdc.dbo_employee_ct -- See note above

-- Delete Jeff
DELETE Employee WHERE EmployeeId = 4

-- Results from Delete
SELECT * FROM Employee
SELECT * FROM cdc.dbo_employee_ct
-- (Note: result of DELETE may take several seconds to show up in CT table)

-- Update Dan and Jay
UPDATE Employee SET EmployeeName = 'Dan P. Park' WHERE EmployeeId = 2
UPDATE Employee SET EmployeeName = 'Jay K. Hamlin' WHERE EmployeeId = 3

-- Results from update
SELECT * FROM Employee
SELECT * FROM cdc.dbo_employee_ct -- See note above

-- Give the CDC job a chance to initialize before accessing the TVFs
WAITFOR DELAY '00:00:20'

-- To access change data, use the CDC TVFs, not the change tables directly
DECLARE @begin_time datetime
DECLARE @end_time datetime
DECLARE @from_lsn binary(10)
DECLARE @to_lsn binary(10)
SET @begin_time = GETDATE() - 1
SET @end_time = GETDATE()

-- Map the time interval to a CDC LSN range
SELECT @from_lsn =
   sys.fn_cdc_map_time_to_lsn('smallest greater than or equal', @begin_time)
SELECT @to_lsn =
   sys.fn_cdc_map_time_to_lsn('largest less than or equal', @end_time)

SELECT @begin_time AS BeginTime, @end_time AS EndTime
SELECT @from_lsn AS FromLSN, @to_lsn AS ToLSN

-- Return the changes occurring within the query window.

-- First, all changes that occurred...
SELECT *
FROM cdc.fn_cdc_get_all_changes_dbo_employee(@from_lsn, @to_lsn, N'all')

-- Then, net changes, that is, final state...
SELECT *
FROM cdc.fn_cdc_get_net_changes_dbo_employee(@from_lsn, @to_lsn, N'all')

Let’s examine this code closely. After creating our sample database `CDCDemo`, we enable 
CDC on that database by calling `EXEC sp_cdc_enable_db`. The next several `SELECT` queries 
demonstrate how to retrieve various kinds of CDC-related information. The first `SELECT` 
query shows how the `is_cdc_enabled` column in `sys.databases` returns true (1) or false (0), 
making it easy to find out which databases are CDC-enabled and which aren’t. The next two 
`SELECT` queries show how the new `cdc` schema and user can be found in `sys.schemas` and `sys. 
database_principals`.

The code then proceeds to create the `Employee` table, which has only three columns to keep 
our example simple. CDC is then enabled on the `Employee` table by calling `EXEC sp_cdc_en- 
able_table` and passing parameters that identify the `Employee` table in the `dbo` schema for 
change capture. (Remember that SQL Server Agent must be running at this point.) The next 
`SELECT` statement shows how to query the `is_tracked_by_cdc` column in `sys.tables` to find out 
which tables are CDC-enabled and which aren’t.

Recall that enabling CDC on the `Employee` table creates a TVF for retrieving all changes 
made to the table between any two points in time. Recall too that by specifying `@supports_ 
net_changes = 1`, this also creates a second TVF for retrieving only the `net` changes made 
between any two points in time. The difference between `all` changes and `net` changes will be 
very clear in a moment, when we call both of these TVFs and compare their results. But first 
the code performs a mix of `INSERT`, `UPDATE`, and `DELETE` operations against the `Employee` 
table to simulate database activity and engage the capture process. In Listing 14-3, these 
operations are accompanied by `SELECT` statements that query the change table `cdc.dbo_em- 
ployee_ct`. This is done purely to demonstrate that change data for the `Employee` table is be- 
ing captured to the change table. However, you should normally not query the change tables 
directly in this manner and should instead use the generated TVFs to extract change infor- 
mation about the `Employee` table, as demonstrated by the rest of the code.

Our code then executes a `WAITFOR` statement to pause for 20 seconds before calling the 
TVFs, in order to give the SQL Server Agent change capture job a chance to initialize. This is 
a one-time latency only; it does not represent the normal latency for CDC-tracked changes 
to be recorded, which is on the order of 2 to 3 seconds. Without this delay, or if SQL Server
Agent is not running when you call the TVFs, you will receive a rather misleading error message that unfortunately does not describe the actual problem.

To call either of the generated TVFs, you need to provide a value range that defines the window of time during which you want change data returned. As already explained, this range is expressed using LSN values, which you can obtain by calling `sys.fn_cdc_map_time_to_lsn` and passing in the desired start and end points in time. So first we establish a time range for the past 24 hours, which we obtain by assigning `GETDATE() – 1` and `GETDATE()` to the start and end time variables. Then we call `sys.fn_cdc_map_time_to_lsn` on the start and end time variables to obtain the LSN values corresponding to the last 24 hours. (Note that the starting LSN gets adjusted automatically to compensate for the fact that there are no LSNs from 24 hours ago, as does the ending LSN, since there might not be any from a moment ago either.) We then issue two `SELECT` statements so that we can view the time and LSN range values, an example of which is shown here:

```
BeginTime               EndTime
----------------------- -----------------------
(1 row(s) affected)
```

```
FromLSN                ToLSN
---------------------- ----------------------
0x0000001A0000001E0039 0x00000020000000A50001
(1 row(s) affected)
```

Equipped with the LSN range values, we issue two more `SELECT` statements. (These are the last two statements in Listing 14-3.) The first statement queries the range against the all changes TVF, and the second statement queries the range against the net changes TVF. Comparing the results of these two queries clearly illustrates the difference between the TVFs, as shown here:

```
__$start_lsn           __$seqval              __$operation __$update_mask EmployeeId
EmployeeName      EmailAddress
---------------------- ---------------------- ------------ -------------- ---------- -------
---------- --------------------------
0x00000001E0000007C0013 0x00000001E0000007C0012 2            0x07           1          John
Smith       john.smith@ourcorp.com
0x00000001E000000800003 0x00000001E000000800002 2            0x07           2          Dan
Park        dan.park@ourcorp.com
0x00000001E000000810003 0x00000001E000000810002 2            0x07           3          Jay
Hamlin      jay.hamlin@ourcorp.com
0x00000001E000000820003 0x00000001E000000820002 2            0x07           4          Jeff
Hay         jeff.hay@ourcorp.com
0x00000001E000000850004 0x00000001E000000850002 1            0x07           4          Jeff
Hay         jeff.hay@ourcorp.com
0x00000001E0000001AC0004 0x00000001E0000001AC0002 4            0x02           2          Dan P.
Park        dan.park@ourcorp.com
0x00000001E0000001AE0004 0x00000001E0000001AE0002 4            0x02           3          Jay K.
```
(7 row(s) affected)

<table>
<thead>
<tr>
<th>__start_lsn</th>
<th>__operation</th>
<th>__update_mask</th>
<th>EmployeeId</th>
<th>EmployeeName</th>
<th>EmailAddress</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000001E0000007C0013</td>
<td>2</td>
<td>NULL</td>
<td>1</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@ourcorp.com">john.smith@ourcorp.com</a></td>
</tr>
<tr>
<td>0x00000001E0000001AC0004</td>
<td>2</td>
<td>NULL</td>
<td>2</td>
<td>Dan P. Park</td>
<td><a href="mailto:steven.jones@ourcorp.com">steven.jones@ourcorp.com</a></td>
</tr>
<tr>
<td>0x00000001E0000001AE0004</td>
<td>2</td>
<td>NULL</td>
<td>3</td>
<td>Jay K. Hamlin</td>
<td><a href="mailto:jay.hamlin@ourcorp.com">jay.hamlin@ourcorp.com</a></td>
</tr>
</tbody>
</table>

(3 row(s) affected)

The first result set includes all the information about all changes made during the specified LSN range, including all interim changes. Thus, the information returned from the first all changes TVF shows every stage of change, or seven changes in total. In our scenario, John was inserted once and then never changed. So only his insert (__operation value 2) is shown. Dan and Jay were inserted (__operation value 2) and updated (__operation value 4), so both changes (insert and update) are returned for each of them. Jeff, on the other hand, was deleted (__operation value 1) after being inserted, so both changes (insert and delete) are returned for Jeff.

The second result set includes only the final changes made during the specified LSN range. So for the same LSN range, we receive only three change records from the second net changes TVF, each of which provides the final column values in the specified LSN range. John appears only once as in the previous query, since he was inserted only once and never modified or deleted within the LSN range. However, although Dan and Jay were inserted and updated, they each appear only once (with their final values for the LSN range), and not twice as in the previous query. And since Jeff was inserted and deleted within the window of time specified by the LSN range, no change data for Jeff is returned at all by the net changes TVF.

**Partitioned Table Parallelism**

In SQL Server, a partitioned table is a table whose physical storage is divided horizontally (that is, as subsets of rows) into multiple filegroups (invisibly to queries and DML) for the purpose of improved manageability and isolation of various kinds of otherwise potentially conflicting access. For example, different partitions of the same table can have different backup and compression strategies and indexes, each optimized to the use of the partition. Given the large size of many data warehouses, this flexibility can be invaluable.
The typical (although by no means required) partition key is *Time*, since that is so often the natural horizontal dividing line. Partitioning by *Time* allows, for example, "old" data to be indexed more lightly than current, more frequently accessed data. Old data can also be backed up and deleted without affecting simultaneous queries against more recent data. Partitioning is an important tool of physical implementation, particularly when building a very large data warehouse.

Another potential benefit of well-designed partitioning is more efficient query plans. Queries specifying the partitioning key that involve only a single partition benefit from having less data to traverse (and potentially more optimized indexes if the partition is for newer data). In addition, when SQL Server is running on multiple-core or multiple-CPU hardware and configured appropriately, multiple worker threads are available and can achieve parallelism in processing a query by assigning multiple threads to it.

**Note** For maximum partitioning benefit, it is crucial to physically isolate each partition of a table from each of the others. In practice, this means that each filegroup of each partition should be on a different physical disk and, in extreme cases, even on a different disk controller. In general, however, this book does not explain the mechanics of partitioned tables, which are well covered in SQL Server Books Online.

**Thread Management**

SQL Server 2005 optimized parallelism for queries involving only a single partition, by allocating all available threads to the one partition. However, on a multipartition query, performance could suffer badly because then only one thread is allocated per partition—leading to some parallelism for the query as a whole but none per partition. The result was that queries varying only slightly in their partitioning key constraint could exhibit vastly different degrees of performance.

The new Partitioned Table Parallelism feature in SQL Server 2008 directly addresses this shortcoming by allocating all available threads to a multipartition query in round-robin fashion. The result is that each partition, as well as the query as a whole, achieves some degree of parallelism. This is automatic when applicable. The best gains will be achieved when the number of threads (that is, cores or CPUs) is significantly larger than the number of partitions on the table. The difference between SQL Server 2005 and 2008 in thread allocation for multipartition queries is illustrated in Figure 14-10. Under the latter in this example, three times as many threads per partition operate on the Feb YTD query, and with all else being equal, this should translate to a 200 percent performance improvement.
SQL Server 2005

Jan YTD Query

Feb YTD Query

SQL Server 2008

Jan YTD Query

Feb YTD Query

**FIGURE 14-10** The difference between SQL Server 2005 and 2008 in how threads are allocated to multipartition queries

**Note** Partitioned Table Parallelism is available only in the Enterprise edition of SQL Server 2008.

**Lock Escalation**

Another important feature of Partitioned Table Parallelism relates to table locking behavior. Previously, when deciding whether to elevate to a table-level lock on a partitioned table, the database engine did not take into account whether concurrent statements against the same table were each accessing a different partition. When they were, each was logically independent and there would be no reason for one to block another. But by not recognizing this and escalating one of the statements to a table lock, the database engine could unnecessarily block the remaining statements, in the process also enhancing the possibility of deadlocks among them. In SQL Server 2008, the default behavior on a partitioned table behaves as before, but Partitioned Table Parallelism enables a new `ALTER TABLE` option, which directs the database engine to use partition-level lock escalation, instead of table-level, on a partitioned table. The syntax is shown here:

```
ALTER TABLE MyTable SET (LOCK_ESCALATION = <option>)
```
The `LOCK_ESCALATION` option can be specified as `TABLE`, `AUTO`, or `DISABLE`. The default is `TABLE`, which means that only table-level lock escalation will occur. If you specify `AUTO`, you get partition-level locking on partitioned tables, table-level otherwise. With `DISABLE`, no lock escalation will occur (in most cases).

**Star-Join Query Optimization**

Star-Join Query Optimization is an important new feature in SQL Server 2008 (again, available in Enterprise edition only) in the context of data warehouse-oriented performance enhancements, but it does not lend itself to deep explanation in a book like this because it does not offer any user-adjustable properties and its operation is largely buried within the database engine. The good news is that you need not do anything to get the benefit of it when applicable.

As noted earlier, the star schema is a common physical data model in Kimball-style data warehousing architectures. Queries against such a physical model are typically characterized by a central fact table joined to multiple dimension tables, each on single-column equijoins (joins based on equality), where the fact table has much higher cardinality than the dimension tables (more rows in the fact table as compared with the dimension table), and the constraints of the query are all on the dimension tables—a pattern now known as a star-join. Since this pattern is common across a large range of data warehousing scenarios, it became apparent that a query optimizer that could recognize such a pattern could potentially produce more efficient query plans than otherwise.

Here’s the basic idea. Eliminate as many candidate rows from the fact table as early as possible in the query-resolution pipeline, since the fact table typically has by far the highest cardinality of the tables involved. In practice, this means determining the candidate join keys from the dimension tables first (taking advantage of the winnowing effect of the constraints typically on them) and then using this information to eliminate candidate rows from the fact table ahead of, and more efficiently than, the actual join process further down the pipeline would. The heuristics—or in other words the rules by which the optimizer recognizes a star-join query—are important to the effectiveness of this strategy.

Such mechanisms are complex and, for our purposes, largely opaque. SQL Server 2005 introduced some star-join optimization based on these principles, but SQL Server 2008 extends the degree to which it can recognize and optimize this pattern. Microsoft benchmarks assert that the degree of performance improvement on typical data warehouse queries at which this feature is targeted can range from 10% to 30%. The SQL Server 2008 enhancements in this area also include more detailed information in query plans, which help the designer to understand when or if this feature is being applied to particular queries.
Note  This enhancement will be of most value when a significant part of the SQL Server work-
load involves ad hoc SQL queries against a star schema. If your architecture directs most ad hoc
queries to an OLAP cube, it will be of lesser, if any, benefit, unless your cube is hosted by SQL
Server Analysis Services and uses the Relational OLAP (ROLAP) or Hybrid OLAP (HOLAP) storage
mode (since in these cases a significant number of cube queries might become SQL star schema
queries).

Space considerations preclude us from discussing this feature in more detail here. To learn
more, we recommend that you visit the links provided at the end of this section.

**SPARSE Columns**

Not all practitioners are happy with *NULL* values in a relational database schema, but for
better or worse, they are widely used in practice. Without engaging that debate, some will
rationalize allowing nullable columns when physically modeling a type (for example, *Product*)
that has many subtypes that have few attributes in common and many attributes unique
to each subtype. It can be convenient, despite going against the grain of normalization, to
physically model this situation as a single table with a column for every attribute across all
subtypes. In such a case, each attribute column must be nullable and will be sparse—that is,
containing *NULL* in a high percentage of cases. It would be beneficial if the storage for such
sparsely populated nullable columns were optimized, particularly in the data warehousing
context, given the often large database sizes involved.

In versions earlier than SQL Server 2008, storing *NULL* values was not optimized—it re-
quired storage for every *NULL* occurrence. SQL Server 2008 introduces the notion of the
**SPARSE** column, a nullable column whose storage is optimized for *NULL* values—at the cost
of increased storage overhead for non-*NULL* values. With this option enabled, occurrences
of *NULL* use no storage. (Note that this is also true when SQL Server Data Compression,
detailed in the next section, is used—although the two are not equivalent.) The density of
a column’s *NULL* values required to achieve a 40 percent space saving using the **SPARSE** at-
tribute, the nominal space saving value as reported by SQL Server Books Online, depends on
the column’s data type and ranges from 42 percent for 38-digit high-precision numeric types
to 98 percent for *bit*. The **SPARSE** attribute in particular benefits Microsoft Office SharePoint
Server, which by its generic and end-user-driven nature is a particular case of the preceding
scenario—needing to store many user-defined attributes that are sparse by nature.

A few data types cannot be **SPARSE**, and there are other, potentially significant, restric-
tions on using **SPARSE** columns—for example, they cannot have default values or rules or
be part of a clustered index or unique primary key index. SQL Server Books Online provides
full details.

This feature is enabled by decorating column definitions in your *CREATE TABLE* and *ALTER
TABLE* statements with the new **SPARSE** attribute. Obviously, the column must also be de-
clared *NULL*. Listing 14-4 shows an example of usage.
SQL Server 2008 introduces two other new features that have a relationship to the **SPARSE** feature but do not depend on it. The first is the **column set**, an optionally declared set of specified columns on a table that, once declared, associates an *xml* column with the table as metadata (that is, no additional storage is used). This column represents the specified columns as an XML document and allows querying and updating of the columns as a group using XQuery and XML DML (which we cover in depth in Chapter 6). The individual columns can still be referenced in the usual way, but the column set representation can be a more convenient method when a table has a large number of columns and might provide performance improvements in some cases. **SPARSE** columns relate to column sets in that a column set cannot be added to an existing table already containing any **SPARSE** columns, and if **SPARSE** columns are later added to a table with a column set, they automatically become part of the column set.

The second new feature is the **filtered index**. A filtered index is an optimized nonclustered index whose declaration includes a *WHERE* clause that restricts the values included in the index to those specified. This can have wide-ranging implications for index maintenance, index storage, and query plan optimization. This feature is most useful when the query patterns against the table are well understood and they naturally relate to distinct subsets of rows. **SPARSE** columns are good candidates to participate in a filtered index because they represent distinct, well-defined subsets (rows with *NULLs* in the columns and rows with non-*NULLs*). For more details of both these features, which involve considerable complexity in their own right, see SQL Server Books Online.

A final benefit of **SPARSE** columns is that, by their nature, they can reduce the size of large backups, potentially more so than any of the new compression features we cover in the next section.

### Data Compression and Backup Compression

Data compression and backup compression are long-awaited enhancements to SQL Server—not surprisingly, also available only in the Enterprise edition (with one exception, as we’ll see when we discuss backup compression). They are of benefit in all scenarios, but especially for large data warehouses. Many factors cause a data warehouse to grow at least linearly with
time: the desire to facilitate trend analyses, personalization, and data mining; the fact that most data warehouses increase the number of data sources included over time; and last that multiple copies of the data warehouse often exist for redundancy and development and QA purposes. SQL Server 2008 provides both data compression, targeting the database itself, and backup compression, targeting the backup/restore process.

As the size of the data warehouse increases, it affects the cost and complexity of maintaining the online version and of taking backups of it. SQL Server 2008 Data Compression provides many benefits. It aids online query performance by increasing the number of rows stored per page, lessening disk I/O and saving costs in disk space. It improves performance for a given amount of memory, as more rows can be held in memory at the same time. It can benefit the backup/restore process by minimizing the I/O and therefore time and media required, since less physical data needs to be transferred. Last, replication and mirroring scenarios can also benefit for all the same reasons.

**Data Compression**

SQL Server 2005 made a start at targeting data compression concerns with both its table-level *vardecimal* storage format (in Service Pack 2 for the Enterprise edition) and its ability to use NTFS file system file compression on SQL Server read-only secondary data files (or all files, including log files, if the database is read-only).

These enhancements remain supported in SQL Server 2008, although use of the *vardecimal* option is deprecated and use of NTFS compression for SQL Server data files is mostly not recommended. Instead, SQL Server 2008 goes considerably beyond these earlier enhancements in the features it provides for data compression.

The most basic form of data compression uses a storage format that eliminates unneeded precision in fixed-length data types—that is, representing each value in a column with the minimal number of bits necessary. For example, any value of 255 or less stored in an integer data type could be stored in one byte instead of four (neglecting some slight overhead). SQL Server 2005 provided such compression or variable-length storage only for the *decimal* and *numeric* data types, but SQL Server 2008 provides it for all formerly fixed-length data types (including *decimal* and *numeric*). Note that what is changing is storage format, not data type, so the semantics of each data type remain the same to T-SQL queries as well as applications.

Data compression comes in two forms: *row compression* (RC) and *page compression* (PC). RC is another name for the variable-length storage approach just detailed. With RC, all occurrences of 0 (zero) and NULL consume no space. RC is not effective for variable-length data types (they are already effectively compressed), for some shorter data types (where the overhead of compression outweighs the benefit), and for some other data types for technical reasons.
Note To summarize, RC does not apply to tinyint, smalldatetime, date, time, varchar, text, nvarchar, ntext, varbinary, image, cursor, sql_variant, uniqueidentifier, table, xml, and user-defined types (UDTs).

PC is a superset of RC and provides potentially greater overall compression than RC alone, at the cost of greater CPU overhead. Where RC is concerned with compressing scale and precision on each individual row-column value, PC is concerned with compressing redundancy across all the rows and their columns on a particular page. PC can be used with all the same database objects as RC. It applies three steps to the enabled object, in the order indicated:

1. RC to the leaf level of a table and to all levels of an index.
2. PC—on each page, for each column of each row on that the page, any common prefixes among all values stored in that column (if any) are identified and tokenized. Each such prefix value is stored once in the new Compression Information (CI) section of the page (by column), and values in each column are replaced with short encoded values that identify the prefix and how much of it applies (as a prefix to the remainder of the value).
3. Dictionary compression—on each page, repeating values from any column in any row on the page are identified and stored in the CI area, and the values are replaced with a pointer to the repeated value. This can further compress the results of the first two steps.

As data is added to a PC-enabled object, these operations are initiated only when a page becomes full. If PC is enabled on an existing object containing data, that object must be rebuilt, a potentially expensive operation.

The code in Listing 14-5 shows an example of creating a table enabled for PC.

**Listing 14-5** Enabling PC on a table

```sql
CREATE TABLE RowCompressionDemo
(FirstName char(10),
 LastName  char(30),
 Salary    decimal(8,2))
WITH (DATA_COMPRESSION = PAGE)
```

SQL Server 2008 provides a system stored procedure associated with both forms of compression aptly named `sp_estimate_data_compression_savings`, which can be used to evaluate whether compression is worth applying to a given object. It can be run for a given uncompressed table, index, or partition to estimate the size it would be, using both RC and PC. It can also do the reverse; reporting the size a compressed object would be if uncompressed. This procedure works by sampling the data of the indicated object into a temporary store and running the indicated compression or decompression on it. It is possible for it to report a larger size for compressed than uncompressed data, which indicates clearly that the nature
of the data is such that the storage overhead associated with compression outweighs any benefit.

Of course, these forms of compression require more CPU cycles to use than would otherwise be required, both when writing (compressing) and reading (decompressing) data. Each represents a tradeoff between saving space (disk and memory) and increasing CPU use. In addition, the effectiveness of any compression scheme is sensitive to the data type and statistical distribution of the values being compressed. For example, compression of an `int` column (4 bytes) in which most values do not exceed 255 (which fit in 1 byte) would exhibit much more benefit from RC than if the values were evenly distributed or if the column were already declared as a `tinyint` (1 byte). For these reasons, as well as the fine grain of data types that this feature allows to be individually tuned for compression, it is advisable to experiment with the various compression options to determine the optimal combination of settings.

Data compression must be enabled—it is disabled by default. It can be enabled on an entire table (which applies to all of its partitions), on individual partitions of a table, on individual indexes of a table, on individual index partitions, and on the clustered index of an indexed view. These features, together with the separately selectable options of row or page compression, give the database administrator great flexibility in tuning the use of compression to achieve the best tradeoffs.

Data compression is enabled by `CREATE TABLE` (as shown earlier) and `CREATE INDEX` statements, and also by `ALTER TABLE` and `ALTER INDEX`. Note that SQL Server Data Compression is not automatically enabled on existing or subsequently created nonclustered indexes of a table on which data compression is enabled—each such index must be separately and explicitly enabled. The one exception to this is that a clustered index does inherit the compression setting of its table.

Last but not least, an uncompressed table can be rebuilt with either form of compression via the new `ALTER TABLE...REBUILD WITH (DATA_COMPRESSION=xxx)` statement, where `xxx` is either `ROW` or `PAGE`. As the compression process is CPU intensive, it lends itself to parallelism, and SQL Server 2008 can take advantage of the availability of multiple CPUs. The `REBUILD` clause therefore supports a `MAXDOP` option to control how many CPUs are allocated to the process.

Backup Compression

SQL Server Backup Compression is a new option with the `BACKUP` statement. Although only the Enterprise edition can create a compressed backup, any edition can restore one.

Compared with data compression, backup compression is extremely coarse grained. It is either enabled or it isn’t for the entire backup—there are no options to tune the compression, and the compression methods are opaque. Nevertheless, it is a welcome enhancement since no earlier version of SQL Server provided any form of backup compression, forcing practitioners to compress backups in a separate step with other, non–SQL Server, utilities.
Chapter 14 Data Warehousing

The option is disabled by default, but the default can be changed via server-level configuration or overridden in the BACKUP statement. It should be noted that an uncompressed 2008 backup operation (both create and restore) can benefit when SQL Server Data Compression has been used on a significant scale in the database being backed up, as a direct result of reduced I/O. If data compression has been used, backup compression will likely provide a smaller (possibly much smaller) space-saving benefit, and because of the additional CPU overhead, backup/restore time might perform worse than without backup compression. This feature is therefore most valuable when the database being backed up has not had significant data compression applied—your own experimentation is warranted.

**Note** Compressed and uncompressed backups cannot be mixed in a backup media set.

As a simple example of the potential efficiency of backup compression, compare the size and time required to back up and restore the AdventureWorksDW2008 database, as shown in Listing 14-6. The CHECKPOINT and DBCC DROPCLEANBUFFERS statements are used to ensure that all cache buffers are empty so that one test does not misleadingly improve the performance of the next. Create the directory C:\Backups prior to running the following code.

**LISTING 14-6** Comparing the time and size between compressed and uncompressed backups

```sql
CHECKPOINT
DBCC DROPCLEANBUFFERS
BACKUP DATABASE AdventureWorksDW2008 TO DISK='C:\Backups\AWDWUncompressed.bak'
-- 10.661 sec, 71 Mb

CHECKPOINT
DBCC DROPCLEANBUFFERS
BACKUP DATABASE AdventureWorksDW2008 TO DISK='C:\Backups\AWDWCompressed.bak'
WITH COMPRESSION
-- 6.408 sec, 13 Mb

CHECKPOINT
DBCC DROPCLEANBUFFERS
RESTORE DATABASE AWDWUncompressed FROM DISK = 'C:\Backups\AWDWUncompressed.bak'
WITH MOVE 'AdventureWorksDW2008_Data' TO 'C:\Backups\AWDWUncompressed.mdf',
    MOVE 'AdventureWorksDW2008_Log' TO 'C:\Backups\AWDWUncompressed.ldf'
-- 9.363 sec

CHECKPOINT
DBCC DROPCLEANBUFFERS
RESTORE DATABASE AWDWCompressed FROM DISK = 'C:\Backups\AWDWCompressed.bak'
WITH MOVE 'AdventureWorksDW2008_Data' TO 'C:\Backups\AWDWCompressed.mdf',
    MOVE 'AdventureWorksDW2008_Log' TO 'C:\Backups\AWDWCompressed.ldf';
-- 6.101 sec
```

In this example, you can see that there is much more improvement in the backup (compression) stage than the restore stage, but in both cases, performance for the compressed
backup is superior to the uncompressed backup. This is due to the reduction in I/O required for processing the smaller (compressed) backup file. Of course, experiments are warranted in your particular scenario to determine exactly what improvements you will see for yourself.

Learning More

We’ve made several references to SQL Server Books Online for more detailed information about many of the new data warehousing features in SQL Server 2008. In addition, you can learn more about all of these SQL Server 2008 data warehousing–oriented features by visiting the following links:


These links were valid as of press time, but if they don’t work, you can perform a Web search on “SQL Server 2008 data warehouse enhancements.”

We can also recommend these additional resources to learn more about the recommended practices of data warehousing:


- The Data Warehousing Institute, http://www.tdwi.org/TDWI

Summary

Data warehousing has become a key component of any enterprise-wide data architecture and is no longer only practical for the largest enterprises. Data warehousing developed as a response to the many impediments to creating actionable information from the data collected by operational applications, impediments that only gradually became recognized as significantly undermining the potential of computers to help turn data into information. The issues existed not only because of historical technical limitations but also because of fundamental differences in optimum design between operational and informational applications.

A data warehouse provides the foundation for many data-driven applications. SQL Server 2008 provides a full-featured, powerful, and cost-effective platform on which to build a data warehouse. You’ve seen how SQL Server 2008 is particularly targeted to data warehousing issues and provides a number of long-awaited features in that sphere. In addition, Microsoft also offers a wide range of integrated and complementary technology, including Microsoft Office SharePoint, Microsoft Performance Point, and the 2007 Microsoft Office system, which enable you to build informational applications on top of your SQL Server data warehouse foundation.
Index

A
access, cube. See roles (OLAP cubes)
access level for FILESTREAM feature, 311, 313
Accordion controls, 432
Accumulate method (aggregate classes), 141
accuracy of date-time values, 62
ACID properties, 449, 450–452
actions (OLAP cubes), 689–693
designing, 690
testing, 692
actions, report, 918
ADD MINING MODEL clause, ALTER MINING STRUCTURE, 839
adjacent groups, tablix data regions, 912
administering Reporting Services, 937–951
Report Manager and Management Studio, 940–949
caching reports, 943
data-driven report subscriptions, 942
securing reports, 944–949
standard report subscriptions, 940–942
Reporting Services Configuration Manager, 937–939
SharePoint integration, 949–951
ADO MD.NET data provider, 758–769
cell references, 763
executing MDX queries, 760
exploring metadata, 763–769
fetching metadata in cubes, 764
prediction queries with, 860
server-side, 782–792
calling managed code from MDX, 786
debugging managed code, 786
result set–returning functions, 789–792
void functions, 788
working with multiple axes, 761–763
XMLA response content from, manipulating, 780
AdomdCommand objects, 760
ExecuteXmlReader method, 780
AdomdConnection objects, 760
Cubes collection of, 764
GetSchemaDataSet method, 765
AdomdDataAdapter objects, 760
ADO.NET, 377
coding, 387–392
connected DataReader objects, 390
typed DataSet objects, 389
untyped DataSet objects, 388–389
validation logic in typed DataSets, 391
Entity Framework (EF) model. See Entity Framework (EF) model, ADO.NET
LINQ, reasons to use, 396–397. See also LINQ
local transactions in, 459–461
native, why not to use, 396–397
passing TVPs using, 56
Sync Services. See Sync Services for ADO.NET
typed DataSet objects, 378–387
adding validation logic to, 391
connected use of, 387
connection string management, 381
data binding example, 420–424
TableAdapter Configuration Wizard, 382–385
TableAdapter objects, 380
using DataSet designer, 379
ADO.NET Data Services, 377, 411–415
AJAX client library for, 434
binding to, 426
ADO.NET Entity Data Model. See Entity Framework (EF) model, ADO.NET
AFTER trigger, MERGE statement and, 79
aggregates (SQL CLR), 140–145
managing with Object Explorer, 152
aggregations (OLAP), 700
algorithmic design, 700
design management, 704
Management Studio for, 705
manual design, 702–704
special aggregates, 665
usage-based design, 701
AJAX Control Toolkit, 432
AJAX-enabled Web pages, 367
for data access, 430–435
ADO.NET Data Services client, 434
algorithmic aggregation design, 700
All Tables diagram (Analysis Services), 623
alphabetical format for date-time values, 64
ALTER DATABASE statement
AUDIT SPECIFICATION clause, 221
SET ENCRYPTION ON clause, 213
ALTER INDEX statement, enabling data compression, 608
ALTER MINING STRUCTURE statement, 839
ALTER SERVER AUDIT SPECIFICATION statement

ALTER SERVER AUDIT SPECIFICATION statement, 220
ALTER SERVER AUDIT statement, 217
  WITH AUDIT_GUID option, 218
  WITH ON_FAILURE option, 218
  WITH QUEUE_DELAY option, 217
  WITH STATE option, 218
ALTER SERVICE MASTER KEY REGENERATE statement, 207
ALTER TABLE statement
  CHANGE_TRACKING option, 541
  enabling data compression, 608
  LOCK_ESCALATION option, ALTER TABLE, 603
  AS TYPE, TVPs and, 59
ambient transactions, 476
AMO (Analysis Management Objects), 769–770
Analysis Manager, 618
analysis optimization, lack of, 568
Analysis Services, 611, 618–619. See also OLAP
  adding data source views, 621–625
  APIs for custom OLAP development, 744
  BPAs (best practice alerts), 654–656
Calculation Tools, 671
CLR support with server-side ADO MD.NET, 782–792
  calling managed code from MDX, 786
  debugging managed code, 786
  result set–returning functions, 789–792
  void functions, 788
connecting to (Excel), 719–723
creating projects, 619, 800
data mining with. See data mining
key performance indicators (KPIs), 677–689
  queries in Management Studio, 683–688
  status and trend expressions, 678
  testing in Browser View, 681–683
MDX queries, building, 748
MDX queries, opening, 747
member grouping, 651
partitions, 693–700
  editing and creating, 694
  proactive caching, 697–699
  perspectives, 705–707
  proactive caching, 697–699
processing cubes, 635
roles, 712–715
server time dimensions, 660–685
  fact dimensions, 661–664
  role-playing dimensions, 664
Time Series algorithm, 797
translations, 707–711
XMLA queries. See XMLA
XMLA scripts, 688
analytical applications, 568
anchor member, CTE, 18
anchor value, Sync Services, 543
API programming, data mining and, 857
application mobilization. See mobile application (example), creating
APPLY operator, T-SQL, 25
area calculations, geospatial data, 355
articles (Merge Replication), 495
ASP.NET
  data binding with, 427–438
  alternatives to GridView controls, 429
  ASP.NET Dynamic Data, 435–438
  using AJAX, 430–435
data mining and, 861
  ReportViewer controls, 930
ASP.NET Dynamic Data, 435–438
assemblies (SQL CLR)
  deploying, 119, 125–127
  loading (server-side ADO MD.NET), 783
  security, 150
association properties (L2S models), 399
Association Rules algorithm, 846
associations (L2S models), 399
Astoria. See ADO.NET Data Services
asymmetric key encryption, 204
atomicity, 451, 452
attack surface, 191
attribute profiles, 826
attribute-based XML, creating, 248
  OPENXML function with, 261
AttributeHierarchies collection (Dimension objects), 764
auditing, 216–224
  creating audit objects, 216
  of database events, 221–222
  options for, 217
  querying audit catalog views, 224
  recording audits
    to file system, 219
    to Windows event log, 220
  of server events, 217
  viewing audited events, 222
authentication and authorization, 195–203
  creating client connections, 195
  execution context, 200–203
  guest account, 195
  mobile application synchronization, 513, 518
  password policies, 197
  user-schema separation, 198–200
authorities (SQL Server Data Services), 415
authorization. See authentication and authorization
AUTHORIZATION clause, CREATE ASSEMBLY, 125, 127
AUTO mode (FOR XML statement), 248
ELEMENTS keyword, 260
AUTO_CLEANUP option, 539
autocommit transaction mode, 453
automated deployment, 117
axes in MDX queries, 754

B
BACKUP CERTIFICATE statement, 215
backup compression options, 608–610
DECRYPTION BY PASSWORD clause, 215
backup compression with data warehousing, 605, 608–610
BACKUP SERVICE MASTER KEY statement, 207
backup-and-restore applications, creating with SMO, 175–181
backups of certificates, 214
balanced scorecard (BSC), 585
Basic Authentication. See authentication and authorization
batch-scoped transaction mode, 457–459
BEGIN DISTRIBUTED TRANSACTION statement, 472
BEGIN TRANSACTION statement, 462
beginning transactions, defined, 461
BeginTransaction method (SqlConnection), 487
best practice alerts (BPAs), 654–656
BI (business intelligence), 611–612
data mining and. See data mining
key performance indicators (KPIs), 583, 677–689
status and trend expressions, 678
testing in Browser View, 681–683
Management Studio for, 683, 688–689
with PivotTables. See PivotTables in Excel prototype for, building, 586
Business Intelligence Projects, creating new, 619

C
caching, OLAP cubes, 697–699
caching reports, 943
calculated members in MDX queries, 757
Calculation Tools (Analysis Services), 671
calculations. See also aggregations (OLAP)
calculated measures, 668–673
named sets, 673
Script view, 674–676
CALL command (MDX), 789
canonical (element-based) XML, producing, 260
OPENXML function with, 261
cascading parameters, reports, 896
CascadingDropdown controls, 433
CASE construct, GROUPING function, 107
case tables in mining models, 805
CAST function
converting columns to XML, 278–279
extracting from datetime2 data, 64
CATCH block. See TRY and CATCH blocks, T-SQL
CDC (Change Data Capture), 592–600
CellSet objects, 760
cell references, 763
certificates, 204
backing up, 214
creating, 206
Change Data Capture (CDC), 592–600
change tracking for Sync Services, 538–543
changing tracking functions, 542
configuring database for, 538–543
CHANGE_RETENTION option, 539
CHANGE_TRACKING_CURRENT_VERSION function, 542
surrogate keys, 579
BPAs (best practice alerts), 654–656
breadth-first index, 296, 298
browsing mining models (Data Mining Client add-in), 876
BSC (balanced scorecard), 585
built-in fields, reports, 888
bulk inserts and updates with TVPs, 48, 49
bulkadmin server role, 193
business intelligence (BI), 582, 611–612
data mining and. See data mining
key performance indicators (KPIs), 583, 677–689
status and trend expressions, 678
testing in Browser View, 681–683
Management Studio for, 683, 688–689
with PivotTables. See PivotTables in Excel prototype for, building, 586
Business Intelligence Projects, creating new, 619
CHANGE_TRACKING_MIN_VALID_VERSION function, 542
CHANGES keyword, INSERT OVER DML syntax, 94–97
CHANGETABLE function, 542
Chaos isolation level (ADO.NET), 467
CHAPTERS axis (MDX queries), 754
chart data regions in reports, 915–917
charts (Excel), creating from PivotTables, 729
CHECK constraint, XML schemas vs., 239
CheckAllocations method, 182
CheckAllocationsDataOnly method, 182
CheckCatalog method, 182, 183
CHECKPOINT statement, 609
CheckTables method, 182
CheckTablesDataOnly method, 182
child groups, tablix data regions, 911
CIF (Corporate Information Factory), 573
Class Library projects, 118
CLR functions, 134
CREATE PROCEDURE commands, 128
deployment, 125
removing SQL CLR objects, 157
testing stored procedures, 129
client connections. See also occasionally connected systems
creating, 195
direct Internet connections, 225
encrypting, 205
client file I/O streaming, FILESTREAM for, 312
ClientDeleteServerUpdate conflict (synchronization), 550
ClientID property (SyncSession), 535
ClientInsertServerInsert conflict (synchronization), 550
ClientUpdateServerDelete conflict (synchronization), 550
ClientUpdateServerUpdate conflict (synchronization), 550
CLR. See SQL CLR feature
cracteristics, 822
cluster diagrams, 819
cluster discrimination, 822
cluster profiles, 820
clustering models, 807, 811
viewing with Cluster Viewer, 818–823
ViewMode property, 862
Clustering Sequence algorithm, 848
code attributes, SQL CLR, 117–118
COLLATE statement, xml data type and, 236
collections of XML schemas, 237
column function (sql), 275
column sets, 605
columns
defined as XML, 235–237
converting columns to XML, 278–279
default and constraints, 236
primary and secondary XML indexes, 244–247
querying. See XQuery language
encrypting data within, 206
FILESTREAM columns, 315–318
pivoting dynamically, 23
statements without explicit column lists, TVPs and, 53, 54
COLUMNS axis (MDX queries), 754
<Command> element, XMLA, 774
command-line report deployment, 923–927
cmment test function (XML PATH), 257
Commit method (IEnlistmentNotification), 478
cmmitt phase (transactions), 470
committing transactions, defined, 461
Common Language Runtime. See SQL CLR feature
common table expressions (CTEs), 14–20
creating recursive queries with, 18–20
ranking functions with GROUP BY function, 30–33
communications security, 190. See also security
Compact version, SQL Server, 496–497
binding data to Pocket PC forms, 523
comparing data mining models, 827–829
comparing instances of xml data type, 234, 235
comparison operations in indexed hierarchical tables, 297
compensated transactions, 450
compile errors, statement-level, 41
Complete method (TransactionScope), 476
complex MDX expressions, 670
compression (backup) with data warehousing, 605, 608–610
compression (data) with data warehousing, 605–608
condition (PBM), defined, 184
configuring Report Services, 937–951
Report Manager and Management Studio, 940–949
caching reports, 943
data-driven report subscriptions, 942
securing reports, 944–949
standard report subscriptions, 940–942
Reporting Services Configuration Manager, 937–939
SharePoint integration, 949–951
ConflictResolver property (SqlCeClientSyncProvider), 552
conflicts, synchronization, 543–552
conformed dimensions (in data warehousing), 576
CONNECTION permissions, 196
Connection property (TableAdapter), 381
connection string management (DataSets), 381
connections (client). See also occasionally connected systems
creating, 195
direct Internet connections, 225
crypting, 205
ConnectionString property (SqlConnection), 389
consistency (transactions), 451, 452
containers (SQL Server Data Services), 415
context of measures, 575, 576, 577. See also star schemas
continuous data, defined, 796
Control Toolkit (AJAX), 432
CONVERT function
  converting columns to XML, 278–279
  extracting from datetime2 data, 64
Convert To Formulas option (Excel), 733
cordinate-based cel references, 763
Corporate Information Factory (CIF), 573
currency conversion issues in data warehousing, 587
CREATE ASSEMBLY statement, 119, 125
  WITH PERMISSION_SET clause, 126, 127, 152
CREATE CERTIFICATE statement, 215
  ENCRYPTION BY PASSWORD clause, 215
CREATE DATABASE statement
  AUDIT SPECIFICATION clause, 221
  FILEGROUP...CONTAINS FILESTREAM clause, 313
  FOR SERVER AUDIT clause, 222
CREATE FUNCTION statement, 132
  ORDER clause, 135
CREATE INDEX statement, enabling data compression, 608
CREATE LOGIN statement, 193, 197
CREATE MASTER KEY statement, 207
CREATE MINING MODEL statement, 839
CREATE MINING STRUCTURE statement, 838
CREATE PROCEDURE statement, EXTERNAL NAME clause, 127
CREATE SCHEMA statement, 199
CREATE SERVER AUDIT SPECIFICATION statement, 220
CREATE SERVER AUDIT statement, 216
  TO APPLICATION_LOG and TO SERVER_LOG options, 220
  WITH AUDIT_GUID option, 218
  WITH ON_FAILURE option, 218
  WITH QUEUE_DELAY option, 217
  WITH STATE option, 218
  STATE option, 220
CREATE TABLE statement
  enabling data compression, 608
  ROWGUIDCOL attribute, 315
CREATE TRIGGER statements, 138
CREATE USER statement, 194
CROSS APPLY operator, 25
CROSS JOIN queries, with geospatial data, 347
CROSSJOIN function (MDX), 756
CTEs (common table expressions), 14–20
  creating recursive queries with, 18–20
  ranking functions with GROUP BY function, 30–33
Cube Designer, 626–629. See also cubes, OLAP
CUBE keyword, GROUP BY clause, 104. See also WITH CUBE operator
Cube Wizard, 625
CUBEMEMBER function (Excel), 733
cubes, OLAP. See also OLAP
  actions, 689–693
  designing, 690
  testing, 692
building, 617–635
  adding data source view, 621–625
  creating project, 619
  with Cube Designer, 626–629
  with Cube Wizard, 625
  preparing star schema objects, 617
Properties Windows and Solution Explorer, 634
  using Dimension Designer, 632
  using Dimension Wizard, 629
defined, 614
dimensions. See dimensions
Excel with
  connecting PivotTables to cubes, 719
  querying in Excel formulas, 732–737
measures. See measures
partitions, 693–700
  editing and creating, 694
  proactive caching, 697–699
perspectives, 705–707
  proactive caching, 697–699
processing, 635
roles, 712–715
server time dimensions, 660–685
  fact dimensions, 661–664
  role-playing dimensions, 664
translations, 707–711
CUBEVALUE function (Excel), 733, 735
Current property (Transaction), 486
CurrentDatabaseName property (Context), 791
CURRENTMEMBER property (MDX), 674
CurrentServerId property (Context), 791

D
dashboards, 583
data, information vs., 565
data access, 377–418
ADO.NET. See ADO.NET
LINQ (Language Integrated Query). See LINQ
typed DataSet objects, 378–387
adding validation logic to, 391
connected use of, 387
connection string management, 381
data binding example, 420–424
TableAdapter Configuration Wizard, 382–385
TableAdapter objects, 380
using DataSet designer, 379
data access infrastructure. See also data binding
ADO.NET Data Services. See ADO.NET Data Services
LINQ (Language Integrated Query). See LINQ
typed DataSet objects, 377, 415
data binding, 419–447
to ADO.NET Data Services, 426
ASP.NET, 427–438
alternatives to GridView controls, 429
ASP.NET Dynamic Data, 435–438
using AJAX, 430–435
Silverlight 2.0, 438, 445–446
Windows Forms, 420–427
WPF (Windows Presentation Foundation), 438–445
lack of design support, 439–514
XAML for, 441–445
data compression with data warehousing, 605–608
data conflicts in synchronization, 543–552
data definition language. See DDL triggers
data encryption. See encryption
data hierarchies. See hierarchical data
data mining, 581, 793–878
applications of, 856
API programming, 857
ASP.NET and, 861
Excel 2007 add-ins, 866–877
managed stored procedures, 863
model content queries, 860
prediction queries with ADO MD.NET, 860
Web controls, 862
Windows Forms Model content browser controls, 858
XMLA scripts, 865
getting started, 798–802
mining models, 805–816
adding, 811–813
changing column usage, 813
creating new, 805–809
data types and, 813
data mining models, 813
design
bottom-up approach (Kimball), 574–580
technical and nontechnical issues, 571
top-down approach (Inmon), 572–573
design of, 570–580
getting started, 798–802
history of shortcomings before, 566–570
importance of, 564–565
practical advice on, 585–589
validating and comparing, 827–829
viewing, 818–826
mining structures, creating, 804
nested tables, 830–836
DMX statements for, 843, 851
reasons for, 793–796
SQL Server 2008 enhancements, 797
using DMX, 836–856
for modeling, 837–848
for predictions, 848–855
templates, 856
Data Mining Client add-in (Excel 2007), 867,
873–877
Data Mining Extensions. See DMX
Data Mining Structure Designer, 802. See also
data mining
Data Mining Wizard, 802. See also data mining
data regions (reports), 886
chart data regions in reports, 915–917
formatting, 889
matrix data regions in reports, 906–910
subtotals and totals, 908
tablix data region, 881, 886, 910–915
extending with groups, 910–912
fixed columns and rows, adding, 913
report actions, 918
Data Source View designer, 623–624
Data Source View Wizard, 621–625
Data Source Views
adding named queries to, 645
adding tables to, 628
data sources
adding to Analysis Services projects, 621–625
for mobile applications, 521–523
for reports, 883
Data Sources window, 421
data storage, unstructured. See FILESTREAM feature
data synchronization, 493
for mobile applications. See Web Synchronization
data test function (XML PATH), 257
data types
data mining models, 813
with Sync Services, 559
data warehousing, 563–610
defined, 563
design
bottom-up approach (Kimball), 574–580
technical and nontechnical issues, 571
top-down approach (Inmon), 572–573
design of, 570–580
getting started, 798–802
history of shortcomings before, 566–570
importance of, 564–565
practical advice on, 585–589
data warehousing (continued)
with SQL Server 2008, 589–610
  Change Data Capture (CDC), 592–600
data and backup compression, 605–610
Partitioned Table Parallelism, 600–603
SPARSE columns, 604–605
Star-Join Query Optimization, 603–604
data warehousing (continued)
  T-SQL MERGE statement, 589–592
what it isn’t, 580–585
database encryption key (DEK), 212
database integrity check with SMO, 181
database master keys (DMKs), 206
database roles, logins with, 194
database schemas. See schemas, XML
Database task (Reporting Services Configuration Manager), 938
database users, 193
  authenticating. See authentication and authorization
guest account, 194
user-schema separation, 198–200
databases
  BLOBs in, 307. See also BLOB data; FILESTREAM feature
encrypted, restoring, 215
encrypting data within, 206
event auditing, 221–222. See also auditing
FILESTREAM-enabled, creating, 313–315. See also FILESTREAM feature
data-driven report subscriptions, 942
DataGridView controls (examples), 422, 424
  alternative approaches, 429
  viewing schema DataSet objects, 765
datamarts, 572, 617
DataReader objects, coding with, 390
DataSet, LINQ to. See L2S (LINQ to SQL) models
DataSet designer, 379
DataSet objects, typed, 378–387, 424
  adding validation logic to, 391
coding against, 389
connected use of, 387
connection string management, 381
data binding example, 420–424
schema DataSet objects, 765–769
TableAdapter Configuration Wizard, 382–385
TableAdapter objects, 380
  using DataSet designer, 379
DataSet objects, untyped, 388–389
DataTable objects in typed DataSets, 379
datetime data type, replacing, 59
date and time data types, T-SQL, 59–67
  accuracy, storage, and format, 62–64
new and changed since SQL Server 2000, 65
  separation of dates and times, 59
time zone awareness, 61–62
date data type, 59
  storage and precision, 60, 62
DATEADD function, 65
DATEDIFF function, 65
DATENAME function, 65, 67
DATEPART function, 65, 67
dateTime data type, 240
datetime2 data type, 59, 60
  storage and precision, 60, 62
datetimeoffset data type, 59, 61–62, 65
  storage and precision, 60, 62
DateTimePicker control (example), 423
db_accessadmin role, 194
db_backupoperator role, 194
db_datareader role, 194
db_datawriter role, 194
db_ddladmin role, 194
db_denydatareader role, 194
db_denydatawriter role, 194
db_owner role, 194
db_securityadmin role, 194
DBCC functions
  DROPCLEANBUFFERS statement, 609
  performing with SMO, 181
dbcreator server role, 193
DBDirect methods, 387
dbo schema, 198
DbServerSyncProvider class, 534
DbSyncServerProvider class, 536
DDL statements, 836. See also DMX (Data Mining Extensions)
DDL triggers, 43, 136
Debug toolbar (Calculations tab), 675
debugging server-side managed code, 786
Decision Trees model, 806, 812
dm_x function, 840
  viewing with Tree Viewer, 823–825
  ViewType property, 862
DECLARE statement, 235
decryption. See encryption
DECRYPTION BY PASSWORD clause, BACKUP CERTIFICATE statement, 215
Default property (Properties.Settings), 389
dek (database encryption key), 212
DELETE statements
  bulk deletes with TVPs, 51
  INSERT INTO...SELECT statement around, 90–94
  merges, 79. See also MERGE statement
  OUTPUT clause, 90
  TOP parameter in, 26–27
  WAITFOR statement with, 43
DELETED pseudo-table
  merges and, 76, 90
  querying with DML triggers, 137
delivering reports, 919–937
  accessing reports programmatically, 928–937
    ReportViewer controls, 929
    URL access, 928
  deploying to report server, 919–927
    using command line, 923–927
    using Report Manager, 921–923
    using Visual Studio, 920
  scripting report generation with Integration Services, 930–937
denial of service, 226
DENSE_RANK function, 34–36
  using with other ranking functions, 36
deny permission, setting using schemas, 199
deployment
  automated, 117
  of data mining objects, 816
  of mobile applications (example), 530–532
to report server, 919–927
  using command line, 923–927
  using Report Manager, 921–923
  using Visual Studio, 920
security, 190. See also security
  SQL CLR assembly security, 151
  user-defined aggregates, 142
  user-defined types (UDTs), 149
depth-first indexing, 284, 296, 297
descriptive data mining models, 806
deserialization, with hierarchyid data type, 306
  DetailsView control (example), 429
Diagram Organizer pane (Analysis Services), 623
dictionary compression, 607
dictionary-style TVPs, 54
Dimension Designer, 632. See also dimensions
  attribute relationships, 656–659
dimension tables
  defined, 575
  in OLAP cubes, 615
  Star-Join Query Optimization, 603–604
Dimension Wizard, 629
  server time dimensions, 660
Dimensional Model (data warehousing), 574–580
  context and star schema, 577
  Star-Join Query Optimization, 603–604
  surrogate keys, 579
dimensions, 614, 626, 641–659. See also cubes, OLAP
  adding named queries to Data Source Views, 645
  attribute relationships, 656–659
  attribute typing, 652
  creating with Dimension Wizard, 629
  defined (in data warehousing), 575
  geospatial, 350
key performance indicators (KPIs), 677–689
  queries in Management Studio, 683–688
  status and trend expressions, 678
testing in Browser View, 681–683
  keys and names, 641
  member grouping, 651
  multidimensional OLAP databases (MOLAP), 615, 640
  parent/child dimensions, 647–651
  role-playing, 712–715
  server time dimensions, 660–685
  fact dimensions, 661–664
  role-playing dimensions, 664
  user table time dimensions, 652
  writeback dimensions, 699
direct file I/O streaming, FILESTREAM for, 312
direct Internet connections, 225
dirty reads, 451, 462
  defined, 461
Disabled value (TransactionOption attribute), 473
  <Discover> element, XMLA, 773
discrete data, defined, 796
discretization, 651
disk space for audit files, 220
diskadmin server role, 193
Dispose method (TransactionScope), 476
distance calculations, geospatial data, 355, 363
distributed transaction coordinators (DTCs). See transaction managers (TMs)
distributed transactions, 453, 468–484
  in .NET Framework, 473
terminology of, 461, 469
distribution, defined, 796
distributors (Merge Replication), 495
configuring, 499–503
DMKs (database master keys), 206
DML statements, 836. See also DMX (Data Mining Extensions)
DML triggers, 136–140
DMX (Data Mining Extensions), 836–856
  for modeling, 837–848
  for predictions, 848–855
  query window, 837
  templates, 856
documenting mining structures and models (Excel 2007), 877
DP (electronic data processing), 566
DRILLDOWNMEMBER function (MDX), 756
drillthrough feature, 842
  with Excel PivotTables, 726
  with schema DataSet objects, 766–769
DTCs (distributed transaction coordinators). See transaction managers (TMs)
Dundas charting controls, 354
Dundas Data Visualization, 881
duplicates, finding with self-joins vs. CTEs, 16, 282
durability, 451, 452
durable enlistment, 471
Dynamic Data Web Applications, 435
dynamic properties (SQL Server Data Services), 415

E
Edit Relationship dialog box (Analysis Services), 623
ing editing data mining models, 811
EDP (electronic data processing), 566
EF (Entity Framework) model. See Entity Framework (EF) model, ADO.NET
electronic data processing (EDP), 566
element-based (canonical) XML, producing, 260
OPENXML function with, 261
ELEMENTS keyword, FOR XML clause, 260
E-Mail Settings task (Reporting Services Configuration Manager), 938
encryption, 203–210. See also Transparent Data Encryption (TDE)
ENCRIPTION BY PASSWORD clause, CREATE CERTIFICATE statement, 215
Encryption Keys task (Reporting Services Configuration Manager), 938
endpoints, 195
enlisting within a transaction, 470, 471–472
enterprise integration, lack of, 567
EnterpriseServices namespace, 473
entities (SQL Server Data Services), 415
Entity Data Model Wizard, 402
Entity Framework (EF) model, ADO.NET, 395–397, 402–405
adding validation logic to, 410
data binding, 425
with ASP.NET, 428
querying, 406–409
XML in, 405
EnumAvaiableSqlServers method
(SmoApplication), 169
enumerating SQL Server instances, 169
error handling, in T-SQL transactions, 40–42
ErrorsOccurred conflict (synchronization), 550
ETL step of data mining, 794
event log, recording audits to, 220
Event Viewer, 222
EVENTDATA function, 43
EventData property (SqlTriggerContext), 139
event-driven data warehouse design, 588
Excel 2007
data mining add-ins, 866–877
Data Mining Client add-in (Excel 2007), 867, 873–877
Table Analysis Tools add-in (Excel 2007), 867–873
OLAP with, 719–743
building PivotTable interface, 723–724
connecting to Analysis Services, 719–723
creating and configuring charts, 729
Excel Services, 738–743
exploring PivotTable data, 725–727
in-formula querying of cubes, 732–737
scorecards, 727–729
Visual Studio Tools for, 737
Excel Services, 719, 738–743
exception handling, in T-SQL transactions, 40–42
EXECUTE AS <login> context, 200
EXECUTE AS CALLER context, 200
EXECUTE AS OWNER context, 190
EXECUTE AS SELF context, 200
<Execute> element, XMLA, 774
calling Execute programmatically from .NET, 778
ExecuteAndSend method (SqlPipe), 121
ExecuteCellSet method (AdomdCommand), 760
ExecuteNonQuery method (SqlCommand), 113
ExecuteReader method (SqlCommand), 121
ExecuteXmlReader method (AdomdCommand), 780
Execution Account task (Reporting Services Configuration Manager), 938
execution context, 200–203
execution modes, SMO, 186
exist method (xml data type), 266, 269
expectations. See key performance indicators (KPIs)
EXPLICIT mode (FOR XML statement), 250
EXPLICIT option, 233
explicit transaction mode, 453–456
Express edition, SQL Server 2008, 496–497
Expression Blend, 439
expressions, report, 889
External Access permission set (SQL CLR assemblies), 150, 785
EXTERNAL NAME clause, CREATE PROCEDURE, 127

F
fact dimensions, 661–664
fact tables
defined, 575
in OLAP cubes, 615
Star-Join Query Optimization, 603–604
fanouts, 282
file I/O streaming, FILESTREAM for

file I/O streaming, FILESTREAM for, 312
file streams, BLOBs in, 309. See also BLOB data;
FILESTREAM feature
file system, recording audits to, 219
FILEGROUP...CONTAINS FILESTREAM clause,
CREATE DATABASE statement, 313
FILEPATH option, TO FILE clause (CREATE SERVER
AUDIT), 219
FILESTREAM feature, 307–340
about, 307, 309
enabling, 310–318
creating databases, 313–315
creating tables with FILESTREAM columns,
315–318
for machine, 311–312
for server instance, 312
OpenSqlFilestream function, 316, 318–340
building WPF client, 338
creating streaming HTTP service, 333–338
data access, about, 321–331
file streaming in .NET, 319
spatial features with (example), 352–373
Fill method (SqlDataAdapter), 389
Fill method (TableAdapter), 423
FillComboWithDatabases method (SMOUtilities),
173
FillRowMethodName parameter, SqlFunction
attribute, 134
filtered data mining models, 815
filtered indexes, 605
filtered views of cubes, 705–707
fixed columns and rows, tablix regions, 913
fixed database roles, logins with, 194
fixed server roles, logins with, 193
flat XML output, 248
flat-earth (planar) model, 342. See also geometry
data type
FLATTENED keyword (DMX), 854, 860
FLWOR expressions, 264
for keyword (XQuery), 264
FOR PATH clause (USING XML INDEX), 246
FOR PROPERTY clause (USING XML INDEX), 246
FOR SERVER AUDIT clause
  CREATE DATABASE AUDIT SPECIFICATION
  statement, 222
  CREATE SERVER AUDIT SPECIFICATION
  statement, 220
FOR VALUE clause (USING XML INDEX), 246
FOR XML clause, SELECT statement, 233, 247–263,
  248, 250
FOR XML AUTO, 248
FOR XML EXPLICIT, 250
FOR XML PATH, 255–258
FOR XML RAW, 248
element-based (canonical) XML, 260
ELEMENTS keyword, 260
~ FOR XML PATH, 253
inline XSD schemas, 259
ROOT option, 258
TYPE option, 253, 254
XMLSCHEMA option, 259
Force Protocol Encryption option, 206
ForceRollBack method (PreparingEnlistment), 481
forcing encryption, 205
formats for date-time values, 64
formatting measures, 665
formatting reports, 887–890
subtotals and totals, 909
FormView control, 429
PIVOT and UNPIVOT operators, 21–24

G
general ledger (GL), 567
GenerateDBDirectMethods property, 387
geodetic (round-earth) model, 343. See also
geography data type
geography data type, 150, 351
  integrating with Microsoft Virtual Earth,
  364–373
loading data with Parse method, 346
Parse method, 346
spatial reference IDs, 355
STArea and STLength methods, 355
STDimension method, 350
STDistance method, 363
STIntersection method, 350
STIntersects method, 347
ToString method, 349
gemeans data type, 150, 345–351
Parse method, 346
STDimension method, 350
STIntersection method, 350
STIntersects method, 347
ToString method, 349
geospatial data types, 341–374
generality data type, 351
  integrating with Microsoft Virtual Earth,
  364–373
spatial reference IDs, 355
STArea and STLength methods, 355
STDistance method, 363
geography data type, 345–351
Parse method, 346
STDimension method, 350
STIntersection method, 350
STIntersects method, 347
ToString method, 349
geospatial data types (continued)
  STIntersection method, 350
  STIntersects method, 347
  ToString method, 349
spatial data types, 344
spatial models, 342–343
Well-Known Text (WKT) markup language, 344
GET_FILESTREAM_TRANSACTION_CONTEXT function, 318, 327
GetAncestor method (hierarchical tables), 293–296, 300
GetDescendant method (hierarchical tables), 286
GetLevel method (hierarchical tables), 284
GetMembers method (Level), 764
GetReparentedValue method (hierarchical tables), 301–303
GetRoot method (hierarchical tables), 286, 301
GetSchemeDataSet method (AdomdConnection), 765–769
giving up control (data warehousing), 586
GLArgumentException exception, 351
globals, report, 888
goals. See key performance indicators (KPIs)
grains (in data warehousing), 576
GRANT permission, setting using schemas, 199
GridView controls, 422, 424
  alternative approaches, 429
  viewing schema DataSet objects, 765
GROUP BY clause, SELECT statement, 97, 99
  ROLLUP and CUBE keywords, 104
GROUP BY function, ranking functions with, 30
GROUPING function, 106
GROUPING SETS operator, T-SQL, 97–109
  NULL values, handling, 105–109
  returning top level only, 103
  rolling up all level combinations, 101
  rolling up by level, 99
  rollup and cube operations together, 103
groups, adding to tablix data regions, 910–912
  fixed columns and rows in, 913
  guest user account, 194

H
hacker attacks, types of, 225–227
header on report pages, 888
hierarchical data, 281
  creating hierarchical tables, 283–285
  hierarchyid data type, about, 282
  indexing strategies, 296
Parse, Read, and Write methods, 305
populating hierarchies, 285–296
  GetAncestor method, 293–296, 300
  GetDescendant method, 286
GetLevel method, 284
GetRoot method, 286, 301
ToString method, 288–292
querying hierarchical tables, 299
  indexing strategies and, 296
  reordering nodes, 301
XML output as, 248
hierarchical queries with CTEs, 18–20
hierarchies
  creating with Dimension Designer, 632
  defined (in data warehousing), 575
dimension attribute relationships, 656–659
in OLAP cubes, 614. See also cubes, OLAP
Hierarchies collection (Dimension objects), 764
hierarchyid data type, 282. See also hierarchical data
  GetAncestor method, 293–296, 300
  GetDescendant method, 286
  GetLevel method, 284
  GetReparentedValue method, 301–303
  GetRoot method, 286, 301
  IsDescendantOf method, 299–301
  Parse method, 305
  Read method, 306
  ToString method, 288–292
  Write method, 306
historical data, lack of, 568
HOLAP (Hybrid OLAP), 688
hosted data services. See SQL Server Data Services
HTTP service, FILESTREAM storage as, 333–338
Hybrid OLAP (HOLAP), 688

I
IDENTITY values, 284
IEnlistmentNotification interface, 478
IEnumerable interface, 133
IIS, configuring for Web Synchronization, 514–516
implicit transaction mode, 456
inconsistent state, 451
indexes, XML, 244–247
indexing hierarchical tables, 296
  breadth-first index, 296, 298
depth-first indexing, 284, 296, 297
  filtered indexes, 605
InDoubt method (IEnlistmentNotification), 478
information vs. data, 565
Init method (aggregate classes), 141
initiating transactions, 470
  enlistment, 470, 471–472
injection attacks, 162, 226
inline XSD schemas, 259
Inmon approach to data warehousing, 572–573
INSERT INTO statements (DMX), 840
INSERT INTO...SELECT statement, 90–94
SHAPE clause with, 847, 854
INSERT OVER DML syntax (T-SQL), 90–97
  CHANGES keyword, 94–97
  OUTPUT...INTO, extending, 90–94
INSERT statements
  bulk inserts with TVPs, 49
  editing BLOB data, 316
  INSERT INTO...SELECT statement around, 90–94
  merges, 79. See also MERGE statement
  OUTPUT clause, 90
  TOP parameter in, 26–27
  WAITFOR statement with, 43
  without explicit column lists, TVPs and, 53
XML Bulk Load with, 262
INSERTED pseudo-table
  merges and, 76, 90
  querying with DML triggers, 137
INSTEAD OF trigger, MERGE statement and, 79
Integrated Authentication. See authentication and
authorization
integration across enterprise, lack of, 567
Integration Services to script report generation,
930–937
integrity check with SMO, 181
intelligent observation, 227
IntelliSense technology
  for connection string management, 381
  with Excel, 733
interactive reports, 917–919
Internet connections, direct, 225
intersections, geospatial, 347, 350
IsDescendantOf method (hierarchical tables),
299–301
ISO8601 format for date-time values, 64
isolation (transactions), 451, 452
isolation levels, 462–468
  in ADO.NET
    Chaos isolation level (ADO.NET), 467
    ReadCommitted isolation level (ADO.NET), 468
    ReadUncommitted isolation level (ADO.NET), 467
    RepeatableRead isolation level (ADO.NET), 468
    Serializable isolation level (ADO.NET), 468
    Snapshot isolation level (ADO.NET), 468
    Unspecified isolation level (ADO.NET), 468
  in SQL Server 2008
    read committed isolation level, 464
    read uncommitted isolation level, 462
    repeatable read isolation level, 464
    serializable isolation level, 465
    snapshot isolation level, 45, 465

J
join method, choosing with MERGE, 78

K
key generators, 579
key performance indicators (KPIs), 583, 677–689
  status and trend expressions, 678
  testing in Browser View, 681–683
keys, dimension, 641
keys, encryption, 204
  creating master key, 211
  Service Master Key (SMK), 206
Kimball approach to data warehousing, 574–580
  context and star schema, 577
  Star-Join Query Optimization, 603–604
surrogate keys, 579
KPIs (key performance indicators), 583, 677–689
  with MDX (multidimensional expression
  language), 757
queries in Management Studio, 683–688
  testing in Browser View, 681–683

L
L2S (LINQ to SQL) model, 395–401
  adding validation logic to, 410
  building, 397–401
  data binding, 424
  querying, 406–409
L2S (LINQ to SQL) models
  data binding, 424
  with ASP.NET, 428
Lat property (SqlGeography), 370
lattitudes, obtaining, 351. See also geography
data type
lax validation, XSD support for, 241
length calculations, geospatial data, 355, 363
length for passwords, setting minimum, 197
let keyword (XQuery), 264
Level Naming Template dialog box, 651
LEVEL property (MDX), 672
Levels collection (Hierarchy objects), 764
lift charts, 829
LINQ (Language Integrated Query), 377, 392–395
  LINQ to SQL (L2S) models, 395–401
    adding validation logic to, 410
    building, 397–401
    data binding, 424
    querying, 406–409
    syntax of, 393–395
list data type (XSD), 242
local data caches with Sync Services, 557
Local Security Settings applet (example), 197
local transactions, 452, 453–461
in ADO.NET, 459–461
  autocommit transaction mode, 453
  batch-scoped transaction mode, 457–459
  explicit transaction mode, 453–456
  implicit transaction mode, 456
terminology of, 461
lock escalation, 602
LOCK_ESCALATION option, ALTER TABLE, 603
LOG ON clause, CREATE DATABASE statement, 314
logging. See also auditing
LOGINPROPERTY function, 198
logins, 192
  encrypting, 204
  as execution context, 200
  password policies, 197
  permissions. See authentication and authorization
Long property (SqlGeography), 370
longitudes, obtaining, 351. See also geography data type

M
management facet (PBM), defined, 184
management reporting, 567
Management Studio
  as MDX client, 744, 745–758
  Reporting Services administration, 940–949
    caching reports, 943
    data-driven report subscriptions, 942
    securing reports, 944–949
    standard report subscriptions, 940–942
    with spatial data, 354
  using XMLA from, 772–778
  viewing audited events with, 223
manual aggregation design, 702–704
mash-up, Virtual Earth (example), 364–373
master key, creating, 211
matrix data regions in reports, 906–910
  subtotals and totals, 908
MAX_ROLLOVER_FILES option, TO FILE clause (CREATE SERVER AUDIT), 219
maximum audit file size, 219
MAXRECURSION option, 19
MAXSIZE option, TO FILE clause (CREATE SERVER AUDIT), 219
MDX (multidimensional expression language), 615, 640, 897–900
  actions (OLAP cubes), 689–693
designing, 690
testing, 692
building queries, 748
calculated queries and named sets, 757
custom OLAP development, 744
expressions, storing for queries, 667. See also calculations
important concepts of, 751–756
in key performance indicators. See KPIs
KPIs with, 757
Management Studio for, 684
OLAP development with ADO MD.NET. See ADO MD.NET data provider
opening queries, 747
queries, writing
  with MDX Query Designer, 901–904
  in text mode, 904–906
  via XMLA. See XMLA
server-side ADO MD.NET, 782–792
calling managed code, 786
debugging managed code, 786
result set–returning functions, 789–792
void functions, 788
SQL Server Management Studio (SSMS) as client, 744, 745–758
MDX Query Designer, 901–904
text mode, 904–906
MDX templates, 748
mean, defined, 796
measures, 614, 625, 627. See also cubes, OLAP calculated, 668–673
defined (in data warehousing), 575
display folders, 666
formatting, 665
key performance indicators (KPIs), 677–689
  queries in Management Studio, 683–688
  status and trend expressions, 678
  testing in Browser View, 681–683
in OLAP cubes, 641–659
special aggregates, 665
writeback measure groups, 699
median, defined, 796
member grouping (Analysis Services), 651
MEMBERS function (MDX), 751
Merge Agents, 513
Merge method (aggregate classes), 141, 144
Merge Replication, 492, 494–532
  configuring, 499–520
  configuring distribution, 499–503
  creating publications, 504–512
  verifying subscription, 520
  Web Synchronization. See Web Synchronization
mobile application (example), creating, 520–532
binding data to Pocket PC forms, 523
deployment, 530–532
Merge Replication (continued)
programmatic synchronization, 526–530
setting control properties, 524–526
setting up data source, 521–523
occasionally connected application (example), 496–498
replication term glossary, 494
vs. Sync Services for ADO.NET, 492
MERGE statement, 68–90
in data warehousing, 589–592
defining merge source and target, 70
DML behavior, 79–81
INSERT INTO...SELECT statement around, 90–94
join method, choosing, 78
OUTPUT clause, 68, 76, 90
for table replication, 73
upsert operations, 81–90
WHEN MATCHED clause, 71
WHEN NOT MATCHED BY SOURCE clause, 74
WHEN NOT MATCHED BY TARGET clause, 72
metadata, 204
Microsoft Clustering algorithm, 811
Microsoft IIS, configuring for Web Synchronization, 514–516
Microsoft SQL Server 2008 Reporting Services. See Reporting Services
Microsoft Streets & Trips 2008, 351
Microsoft Virtual Earth, 364–373
Microsoft Visual Studio, SMO with, 167–183
backup and restore applications, 175–181
interacting through available servers, 169
programmatic DBCC functions, 181
retrieving server settings, 171–174
Microsoft.AnalysisServices.AdomdClient assembly, 759
Microsoft.SqlServer.Server namespace, 121
Microsoft.SqlServer.Types namespace, 283
millisecond accuracy, 60, 62
minimum password length, 197
Mining Accuracy Chart view, 827
mining models, 805–816. See also data mining adding, 811–813
changing column usage, 813
creating new, 805–809
data types and, 813
documenting (Data Mining Client add-in), 877
editing, 811
filtering, 815
validating and comparing, 827, 827–829
viewing, 818–826
mining structures, 804, 808. See also data mining mobile application (example), creating, 520–532.
See also occasionally connected systems
binding data to Pocket PC forms, 523
deployment, 530–532
programmatic synchronization, 526–530
setting control properties, 524–526
setting up data source, 521–523
modify method (xml data type), 266, 276–277
MODIFY NAME clause, ALTER SERVER AUDIT statement, 217
MOLAP (Multidimensional OLAP), 615, 640, 696
moving hierarchical subtrees, 303
multidimensional expression language. See MDX multidimensional OLAP databases (MOLAP), 615, 640
multipartition query, 601
multiple-row sets, TVPs with, 48

N
Naïve Bayes algorithm, 813
ViewerMode property, 862
viewing with Naïve Bayes Viewer, 826
named queries, adding to Data Source Views, 645
named sets (Analysis Services), 673
with MDX (multidimensional expression language), 757
names, dimension, 641
NativeSqlClient class, 321
NATURAL PREDICTION JOIN (DMX), 850
navigation properties (EF models), 403
nested tables in data mining, 805, 830–836
DMX statements for, 843, 851
.NET Framework
calling XMLA Execute programmatically, 778
code attributes, 117–118
data access. See data access infrastructure
data binding. See data binding
distributed transactions in, 473
file streaming in, 319
network connections. See client connections
NEWSEQUENTIALID function, 315
node test function (XML PATH), 257
nodes method (xml data type), 266, 275
NONEMPTY function (MDX), 756
NONEMPTYCROSSJOIN function (MDX), 756
nonrepeatable reads, 464
defined, 461
preventing, 464
nonvolatile, data warehouses as, 572
NotSupportedException (TransactionOption attribute), 473
N-tier applications with Sync Services, 554
NTILE functions, 34–36
using with other ranking functions, 36
NULL values
in rollup rows, handling, 105
SPARSE columns, 604–605
numeric format for date-time values, 64
tvarchar(max) data type, 42

Object Explorer, SQL CLR types with, 152
occasionally connected systems, 491–560
components of, 493
configuring Merge Replication, 499–520
configuring distribution, 499–503
creating publications, 504–512
verifying subscription, 520
Web Synchronization. See Web Synchronization
creating (example), 496–498
mobile application (example), creating, 520–532
binding data to Pocket PC forms, 523
deployment, 530–532
programmatic synchronization, 526–530
setting control properties, 524–526
setting up data source, 521–523
Sync Services for ADO.NET, 492, 533–560
capturing changes for synchronization, 538–543
creating applications that use, 543–557
data type considerations, 559
local data caches, 557
vs. Merge Replication, 492
object model for, 534–537
security considerations, 560
SqlSyncAdapterBuilder class, 558
ODBC format for date-time values, 64
.odc files, 721
Office Data Connection files, 721
OLAP (online analytical processing), 580, 611–792
actions, 689–693
aggregations, 700
algorithmic design, 700
design management, 704
manual design, 702–704
usage-based design, 701
building a cube, 617–635
adding data source view, 621–625
creating project, 619
with Cube Designer, 626–629
with Cube Wizard, 625
preparing star schema objects, 617
Properties Windows and Solution Explorer, 634
using Dimension Designer, 632
using Dimension Wizard, 629
calculations, 667–676

custom development with .NET, 743–792
ADO MD.NET. See ADO MD.NET data provider
AMO (Analysis Management Objects), 769–770
MDX and Analysis Services APIs, 744
server-side ADO MD.NET, 782–792
XXMLA for. See XMLA
data mining vs., 856
dimensions in. See dimensions with Excel, 719–743
building PivotTable interface, 723–724
connecting to Analysis Services, 719–723
creating and configuring charts, 729
Excel Services, 738–743
exploring PivotTable data, 725–727
in-formula querying of cubes, 732–737
scorecards, 727–729
Visual Studio Tools for, 737
introduction to, 613–617
key performance indicators (KPIs), 677–689
queries in Management Studio, 683–688
status and trend expressions, 678
testing in Browser View, 681–683
measures in. See measures, in OLAP cubes
partitions, 693–700
editing and creating, 694
proactive caching, 697–699
perspectives, 705–707
proactive caching, 697–699
processing cubes, 635
reports, 900–906
using MDX Query Designer, 901
roles, 712–715
running queries, 636
server time dimensions, 660–685
fact dimensions, 661–664
role-playing dimensions, 664
translations, 707–711
On Change execution modes (SMO), 186
ON COLUMNS clause, 751
On Demand execution mode (SMO), 186
ON keyword, MERGE statement, 70
ON PRIMARY clause, CREATE DATABASE statement, 314
ON ROWS clause, 751
On Schedule execution mode (SMO), 186
ON_FAILURE option, ALTER SERVER AUDIT statement, 218
one-way data synchronization, 493
Open Geospatial Consortium (OGC), 342
OpenGIS Simple Features for SQL standard, 342
OPENROWSET function, 263
OpenSqlFilestream function

OpenSqlFilestream function, 316, 318–340
  building WPF client, 338
  creating streaming HTTP service, 333–338
  file streaming in .NET, 319
  FILESTREAM data access, about, 321–331

OPENXML function, 233
  SQL Server 2008 enhancements, 261–262

operational applications, 568–569
  operational process change, anticipating, 586
  optimistic concurrency checks, 84

ORDER BY clause, DMX statements, 849
  order by keyword (XQuery), 264
  ORDER BY statement, ranking functions, 28–30
  ordinal cell references, 763

ORDERBY clause, CREATE FUNCTION, 135
  ORDINAL property (MDX), 672

ORM (Object Relational Mapping), 395–411
  ADO.NET Entity Framework. See Entity Framework (EF) model, ADO.NET
  L2S (LINQ to SQL) model, 395–401
  adding validation logic to, 410
  building, 397–401
  data binding, 424
  querying, 406–409

OUTER APPLY operator, 25

OUTPUT clause
  INSERT, UPDATE, and DELETE statements, 90
  MERGE statement, 68, 76, 90
  CHANGES keyword with, 94–97
  TVPs and, 59
  OUTPUT...INTO clause, extending with INSERT OVER DML syntax, 90–94

OVER clause, ORDER BY statement, 29, 36
  owner execution context, 190

P

page compression (PC), 606
  page header, reports, 888

PAGES axis (MDX queries), 754

parallelism of partitioned tables, 600
  parameters in reports, 892–897
  adding to OLAP reports, 902
  cascading parameters, 896

parent groups, tablix data regions, 911

parent/child dimensions, 647–651

parents for subtrees, changing, 303

Parse method
  geometry and geography data types, 346
  hierarchyid data type, 305

partial page updates, 431

PARTITION BY statement, ranking over groups with, 37–40

Partitioned Table Parallelism feature, 600–603
  lock escalation, 602
  thread management, 601

partitions in OLAP cubes, 693–700
  editing and creating, 694
  proactive caching, 697–699

passwords. See also authentication and authorization
  policies for, 197
  weak, for system administrators, 226

path expressions. See XPath expression
  path index (XML), 246

PATH mode (FOR XML statement), 253, 255–258

PBM (policy-based management), 161, 183–188, 191
  defined, 184

PC (page compression), 606

PERCENT option, TOP parameter, 26

PercentComplete event, 178

performance
  audit file disk space, 220
  BLOB data storage, 308, 309. See also FILESTREAM feature
  caching reports, 943
  Compact vs. Express edition, 496–497
  data compression methods, 608
  indexing hierarchical tables, 296
  transactions
    reliability, 449
    resource management, 470
  performance management, 585

permissions. See also authentication and authorization
  execution context, 200–203
  guest account, 195
  setting using schemas, 199

perspectives (OLAP cubes), 705–707

phantom reads
  defined, 461
  preventing, 465
  with read committed isolation level, 464
  with repeatable read isolation level, 465

PIVOT operator (T-SQL), 21–24
  dynamically pivoting columns, 23

PivotCharts, 729

PivotTables in Excel
  building, 723–724
  connecting to cubes, 719
  creating charts from, 729
  exploring data in, 725–727
  in-formula querying of cubes, 732–737
  scorecards, building, 727–729

planar (flat-earth) model, 342. See also geometry data type
Pocket PC. See mobile application (example), creating policies for passwords, 197 policy-based [server] management. See PBM populating hierarchies, 285–296 GetAncestor method, 293–296, 300 GetDescendant method, 286 GetLevel method, 284 GetRoot method, 286, 301 ToString method, 288–292 preallocating disk space for audit files, 220 precision of date-time values, 60, 62 predictive data mining models, 806 DMX statements for, 848–855 prediction queries with ADO MD.NET, 860 Prepare method (IEnlistmentNotification), 478, 481 prepare phase (transactions), 470 Prepared method (PreparingEnlistment), 481 PreparingEnlistment class, 481 PREVMEMBER property (MDX), 674 primary keys, hierarchical tables, 284 primary XML indexes, 244 proactive caching, OLAP cubes, 697–699 processadmin server role, 193 processing data mining objects, 816 processing OLAP cubes, 635 processing-instructiontest function (XML PATH), 257 progress meter, implementing with SMO, 178 promotable single-phase enlistment, 471–472 proof of concept for data warehousing, 586 Properties window (Analysis Services), 634, 665 <Properties> element, XMLA, 773 Properties.Settings objects, 389 property index (XML), 246 Protocols For MSSQLSERVER Properties dialog box, 205 prototypes for data warehousing, 586 PSPE (promotable single-phase enlistment), 471–472 public database role, 194 public server role, 193 publications (Merge Replication), 495 creating, 504–512 publishers (Merge Replication), 495 synchronizing database for. See Web Synchronization publishing Excel content at SharePoint. See Excel Services queries added to TableAdapter objects, 385 in ADO.NET, code for, 387–392 connected DataReader objects, 390 typed DataSet objects, 389 untyped DataSet objects, 388–389 validation logic in typed DataSets, 391 of audit catalog views, 224 DMX queries. See DMX of hierarchical tables, 299 indexing strategies and, 296 LINQ, syntax of, 393–395. See also LINQ MDX queries. See MDX named, adding to Data Source views, 645 on OLAP cubes, 636. See also OLAP (online analytical processing) from Excel formulas, 732–737 Partitioned Table Parallelism feature, 601 performance of. See performance Report Designer, importing into, 885 Star-Join Query Optimization, 603–604 of TDE views, 213 of XML data with XQuery. See XQuery language query method (xml data type), 266, 272–274 Query objects, 385 query parameters (reports), 892 adding to OLAP reports, 902 cascading parameters, 896 QUEUE_DELAY option, ALTER SERVER AUDIT statement, 217 range, defined, 796 RANK function, 32 using with other ranking functions, 36 ranking functions, T-SQL, 28–40 DENSE_RANK and NTILE functions, 34–36 RANK function, 32 ROW_NUMBER function, 28–31 using all together, 36 windowing functions with, 37–40 RAW mode (FOR XML statement), 248 ELEMENTS keyword, 260 RDL (report definition language), 880 uploading files, 919–927 using command line, 923–927 using Report Manager, 921–923 using Visual Studio, 920 read committed isolation level (SQL Server 2008), 464
read committed snapshot isolation level (SQL Server 2008), 466
Read method, hierarchyid data type, 306
read uncommitted isolation level (SQL Server 2008), 462
ReadCommitted isolation level (ADO.NET), 468
ReadFile function, with BLOB data, 316, 319
READONLY keyword, TVPs and, 59
ReadUncommitted isolation level (ADO.NET), 467
REBUILD WITH clause, ALTER TABLE statement, 608
recording audits. See auditing
RecoveryInformation method (PreparingEnlistment), 481
recursive queries with CTEs, 18–20
reference data standardization, lack of, 568
relational databases
    hierarchical tables in. See hierarchical data
    SPARSE columns, 604–605
    Relational OLAP (ROLAP), 688
reliability, 449. See also transactions
remote client file I/O streaming, FILESTREAM for, 312
reordering nodes in hierarchical tables, 301
reorganizing hierarchical data, 301
reparenting subtrees, 284, 303
repeatable read isolation level (SQL Server 2008), 464
RepeatableRead isolation level (ADO.NET), 468
replicating tables, MERGE statement for, 73
replication. See Merge Replication; Sync Services
    for ADO.NET
report definition language (RDL), 880
uploading files, 919–927
    using command line, 923–927
    using Report Manager, 921–923
    using Visual Studio, 920
Report Designer, 880–919
    chart data regions, 915–917
    creating reports
        basic reports, 883–887
        interactive reports, 917–919
        OLAP reports, 900–906
        formatting reports, 887–890
        subtotals and totals, 909
    report groups, 890–892
    tablix data region, 881, 886, 910–915
        extending with groups, 910–912
        fixed columns and rows, adding, 913
        report actions, 918
        working with parameters, 892–897
        adding parameters to OLAP reports, 902
        cascading parameters, 896
        writing custom code, 897–900
    report groups, 890–892
    Report Manager, 940–949
        caching reports, 943
        creating report subscriptions, 940
        data-driven report subscriptions, 942
        deploying reports, 921–923
        securing reports, 944–949
        security of reports, 944
        standard report subscriptions, 940–942
    Report Manager URL task (Reporting Services Configuration Manager), 938
    report parameters, 892, 894
    cascading parameters, 896
    Report Server Web Services, 938
    Report Wizard, templates in, 897–900
    Reporting Services, 879–951
        administering, 937–951
        Report Manager and Management Studio, 940–949
        Reporting Services Configuration Manager, 937–939
        SharePoint integration, 949–951
        chart data regions, 915–917
        creating reports
            basic reports, 883–887
            interactive reports, 917–919
            with matrix data regions, 906–910
            OLAP reports, 900–906
            deploying reports, 919–937
            accessing reports programmatically, 928–937
            deploying to report server, 919–927
            formatting reports, 887–890
            subtotals and totals, 909
            report groups, 890–892
            tablix data region, 881, 886, 910–915
            extending with groups, 910–912
            fixed columns and rows, adding, 913
            report actions, 918
            working with parameters, 892–897
            adding parameters to OLAP reports, 902
            cascading parameters, 896
            writing custom code, 897–900
    Reporting Services Configuration Manager, 937–939
    ReportViewer controls, 929
    <RequestType> element, XAML, 773
    Required value (TransactionOption attribute), 473
    RequiresNew value (TransactionOption attribute), 473
    RESERVE_DISK_SPACE option, TO FILE clause (CREATE SERVER AUDIT), 220
    ResolveAction enumeration, 552
    resource managers (RMs), 469
        how to use, 481–484
        performance of, 470
        writing, 477–481
restore applications, creating with SMO, 180
RESTORE SERVER MASTER KEY statement, 207
restoring encrypted databases, 215
<Restrictions> element, XMLA, 773
return keyword (XQuery), 264
REVOKE permission, setting using schemas, 199
RMs. See resource managers (RMs)
ROLAP (Relational OLAP), 688
role-playing dimensions, 664
roles (OLAP cubes), 712–715
roles (server), logins with, 193
roles (user). See user roles
ROLLBACK command (T-SQL), 486
Rollback method (IEnlistmentNotification), 478
rolling back transactions
defined, 461
issued by resource managers, 482
ROLLUP keyword, GROUP BY clause, 104. See also
WITH ROLLUP operator
rollback rows, 99–103
 NULL values in, 105
ROOT option, FOR XML clause, 258
round-earth (geodetic) model, 343. See also
g eo d e t ic data type
row compression (RC), 606
row groups in reports, 890–892
row visibility in reports, 917
ROW_NUMBER function, 28–31
 ORDER BY options, 29–30
 with PARTITION BY statement, 37
using with other ranking functions, 36
ROWGUIDCOL attribute, CREATE TABLE
statement, 315
ROWS axis (MDX queries), 754
rs utility, 923–927

S
Safe permission set (SQL CLR assemblies), 150, 785
SafeFileHandle class, 324
Scale-Out Deployment task (Reporting Services
Configuration Manager), 938
schema DataSet objects, 765–769
executing actions with, 766–769
Schema Generation Wizard, 631
schemas, separated from users, 198–200
schemas, XML, 237–244
 inline XSD schemas, 259
 lax validation, 241
SQL Server collections of, 237
union and list types, 242
xsl:dateTime enhancements, 240
scorecards, 583–585, 677. See also key
performance indicators (KPIs)
BSC (balanced scorecard), 585
OLAP with Excel, 727–729
Script Tasks (Integration Services), 930–937
Script view, Analysis Services, 674–676
ScriptManager controls, 367, 431
secondary XML indexes, 244, 246
SECTIONS axis (MDX queries), 754
Secure by Default approach, 190
Secure by Deployment approach, 190
Secure by Design approach, 189
secure communications, 190
security, 189–228
 auditing, 216–224
 authentication and authorization, 195–203
 encryption support, 203–210
 Transparent Data Encryption (TDE), 211–216
 how hackers attack, 225–227
overview of, 191–195
reports, 944–949
SMO and, 162
SQL CLR assemblies, 150
Sync Services for ADO.NET, 560
themes of security framework, 189–191
securityadmin server role, 193
SELECT clause (T-SQL), XQuery and, 273
SELECT INTO statements (DMX), 849
SELECT statement
 FOR XML clause, 233, 247–263, 248, 250
 FOR XML AUTO, 248
 FOR XML EXPLICIT, 250
 FOR XML PATH, 253, 255–258
 FOR XML RAW, 248
 element-based (canonical) XML, 260
 ELEMENTS keyword, 260
 inline XSD schemas, 259
 ROOT option, 258
 TYPE option, 253, 254
 XMLSCHEMA option, 259
GROUP BY function, 97, 99
GROUPING SETS operator, 97–109
merges, 79. See also MERGE statement
TOP parameter, 26–27
viewing BLOB data (example), 331
WAITFOR statement with, 43
without explicit column lists, TVPs and, 54
self-joining, 282
 CTEs and, 16, 282
self-signed certificates, 204
Send method (SqlPipe), 122
Serializable isolation level (ADO.NET), 468
serializable isolation level (SQL Server 2008), 465
serialization, with hierarchyid data type, 306
server event auditing, 217
server instance, FILESTREAM access level for, 312
server logins, 192
  encrypting, 204
  as execution context, 200
  password policies, 197
  permissions. See authentication and authorization
server management, 161–188
  policy-based management, 183–188
  SMO in Microsoft Visual Studio, 167–183
    creating backup and restore applications, 175–181
    interacting through available servers, 169
    programmatic DBCC functions, 181
    retrieving server settings, 171–174
Server Management Objects. See SMO
server roles, logins with, 193
server time dimensions (OLAP), 660–685
  fact dimensions, 661–664
  role-playing dimensions, 664
serveradmin server role, 193
ServerConnection object, 168
server-side ADO MD.NET, 782–792
  calling managed code from MDX, 786
  debugging managed code, 786
  result set–returning functions, 789–792
  void functions, 788
server-side data access (SQL CLR), 120–125
ServerSyncProviderProxy class, 534, 537
Service Account task (Reporting Services
  Configuration Manager), 938
Service Master Key (SMK), 206
service-based synchronization, 556
services without components, 474
session variables (Sync Services), 535
SessionId property (SyncSession), 535
SET ENCRYPTION ON clause, ALTER DATABASE
  statement, 213
SET TRANSACTION ISOLATION LEVEL statement, 462
setupadmin server role, 193
SHAPE clause, with DMX statements, 846, 854
SharePoint
  publishing Excel content to. See Excel Services Reporting Services integration with, 949–951
shred and compose (XML solution), 233
Silverlight 2.0, data binding with, 438, 445–446
single-row operations, TVPs with, 51–54
size, audit file, 219
SKIP keyword, 847
smalldatetime data type, replacing, 59
SMK. See Service Master Key
SMO (Server Management Objects), 161–167
  latest features in, 166
  in Microsoft Visual Studio, 167–183
  creating backup and restore applications, 175–181
  interacting through available servers, 169
  programmatic DBCC functions, 181
  retrieving server settings, 171–174
  namespace map (partial), 162
SQL-DMO vs., 162–166
Snapshot isolation level (ADO.NET), 468
snapshot isolation level (SQL Server 2008), 45, 465–467
SQL Server Change Tracking, 539
snapshot-based data warehouse design, 588
snapshots (Merge Replication), 495
  configuring for mobile application delivery. See Web Synchronization
snowflake schemas, 616
Solution Explorer (Analysis Services), 634
sort order, report groups, 891
sorting hierarchical data, 301
source data, preparing for mining, 798
source of MERGE statement, 70. See also MERGE statement
  WHEN NOT MATCHED BY SOURCE clause, 74–76
sp_addlogin procedure, 197
sp_configure procedure, 191
sp_estimate_data_compression_savings
  procedure, 607
sp_xml_preparedocument procedure, 261
SPARSE columns, 604–605
spatial data types, 344. See also geospatial data types
spatial models, 342–343
spatial reference IDs (SRIDs), 355
Specify Relationship dialog box (Analysis Services), 623
splits, 813
SQL, LINQ and. See LINQ (Language Integrated Query)
SQL CLR feature, 111–160, 283
aggregates, 140–145
assemblies, deploying, 119, 125–127
best practices, 159
deployment, 125–131
  enabling CLR integration, 112
functions, 131–135
managing types in database, 152–158
security, 150
server-side ADO MD.NET, 782–792
  calling managed code from MDX, 786
  debugging managed code, 786
  result set–returning functions, 789–792
  void functions, 788
stored procedures
  creating (example), 118
SQL CLR feature (continued)
deploying, 127–128
server-side data access and, 120–125
testing, 129–131
usage guidelines, 124
transactions, 485–488
example of, 489–490
triggers, 136–140
types, 145–150
Visual Studio/SQL Server integration, 113–118
SQL injection attacks, 162, 226
SQL Server 2000, XML in, 233
SQL Server 2008
about, 3–5
Express edition, 496–497
integration with Visual Studio, 113–118
XML in. See XML
SQL Server 2008 Reporting Services. See Reporting Services
SQL Server Audit feature. See auditing
SQL Server Browser service, 196, 226
SQL Server Change Tracking
changing tracking functions, 542
configuring database for, 538–543
SQL Server Compact 3.5, 496–497
SQL Server Configuration Manager tool, 205
access level for FILESTREAM feature, 311, 313
SQL Server Data Services, 377, 415
SQL Server instances. See also entries at server
connections to, 168
e numating available, 169
settings of, retrieving, 171
SQL Server Integration Services, 930–937
SQL Server logins, 192
encrypting, 204
as execution context, 200
password policies, 197
permissions. See authentication and authorization
SQL Server Management Studio (SSMS)
as MDX client, 744, 745–758
Reporting Services administration, 940–949
 caching reports, 943
data-driven report subscriptions, 942
securing reports, 944–949
standard report subscriptions, 940–942
with spatial data, 354
using XMLA from, 772–778
viewing audited events with, 223
SQL Server projects, 114–116
SQL Slammer virus, 226
sql:column function, 275
sql:variable function, 276
SqlCeClientSyncProvider class, 534, 536, 552
SqlCommand objects
avoiding creating with SqlDataAdapter, 388, 389
preventing SQL injection attacks, 227
SqlConnection objects, 487
avoiding creating with SqlDataAdapter, 388, 389
SqlContext class, 121
SqlDataRecord class, 123–124
SqlDbType.Structured enumeration, 56
SQL-DMO (Distributed Management Objects), 161
SMO (Server Management Objects) vs., 162–166
SqlFunction attribute, 131
SqlGeography class, 370
SqlPipe class, 121
SqlProcedure attribute (StoredProcedures), 127
SqlSyncAdapterBuilder class, 558
SqlTrigger attribute, 137
SqlUserDefinedAggregate attribute, 141
SqlUserDefinedType attribute, 147
SRIDs (spatial reference IDs), 355
SSL, for mobile application synchronization, 513
SSMS. See SQL Server Management Studio
standard deviation, defined, 796
standardized reference data, lack of, 568
star schemas, 576, 577
preparing to build OLAP cubes, 617
Star-Join Query Optimization, 603–604
STArea method, geography data type, 355
Star-Join Query Optimization, 603–604
STATE option
ALTER SERVER AUDIT statement, 218
CREATE SERVER AUDIT SPECIFICATION
statement, 220
statement-level recompile errors, 41
static properties (SQL Server Data Services), 415
status expressions, KPI, 678
StatusGraphic property (Kpis collection), 764
STDimension method, geometry and geography
data types, 350
STDistance method, geography data type, 363
STIntersection method, geometry and geography
data types, 350
STIntersects method, geometry and geography
data types, 347
STLength method, geography data type, 355
storage of date-time values, 60, 62
stored procedures
for data mining, 863
in EF models, 404–405
in L2S models, 399–401
managed (server-side ADO MD.NET)
calling code from MDX, 782
debugging, 786
stored procedures (continued)
result set–returning functions, 789
void functions, 788
writing code for, 782
parameters, TableAdapter objects and, 385
SQL CLR stored procedures
creating (example), 118
deploying, 127–128
managing with Object Explorer, 152
server-side data access, 120–125
testing, 129–131
usage guidelines, 124
TVPs with. See TVPs (table-valued parameters)
storing BLOBs. See BLOB data; FILESTREAM feature
streamed file access, FILESTREAM for, 312. See also FILESTREAM feature
as HTTP service, 333–338
streams, BLOBs in, 309. See also BLOB data; FILESTREAM feature
Streets & Trips 2008, 351
string formats for date-time values, 64
strings, converting to XML, 278–279
strongly typed DataSet objects. See typed DataSet objects
structured and unstructured data storage. See BLOB data; FILESTREAM feature
Structured enumeration (SqlDbType), 56
structures, mining, 804, 808. See also data mining
documenting (Data Mining Client add-in), 877
validating models in, 812
subject-orientation of data warehouses, 572
subscribers (Merge Replication), 495
synchronizing database for, 517–519. See also Web Synchronization
subscriptions, Merge Replication, 495, 520
subscriptions, reports
data-driven subscriptions, 942
standard subscriptions, 940–942
subtotals in matrix reports, 908
subtrees, transplanting. See also hierarchical data subtrees, transplanting (reparenting), 303
Supported value (TransactionOption attribute), 473
surface area for attack, 191
surrogate keys, 579, 587
switching execution context, 203
SWITCHOFFSET function, 66, 67
symmetric key encryption, 204
Service Master Key (SMK), 206
Sync Services for ADO.NET, 492, 533–560
capturing changes for synchronization, 538–543
changing tracking functions, 542
configuring database for, 538–543
creating applications that use, 543–557
data conflicts, handling, 543–552
N-tier applications, 554–556
service-based synchronization, 556
data type considerations, 559
local data caches, 557
vs. Merge Replication, 492
object model for, 534–537
security considerations, 560
SqlSyncAdapterBuilder class, 558
SyncAdapter class, 534, 536
SyncAgent class, 534
SyncGroup class, 534, 537
synchronization, 493
for mobile applications. See Web Synchronization
SyncInitialized constant, 536
SyncLastReceivedAnchor constant, 536
SyncNewReceivedAnchor constant, 536
SyncParameters property (SyncSession), 535
SyncRowCount constant, 536
SyncSession class, 534, 535
SyncSessionId constant, 536
SyncStatistics class, 534
SyncTable class, 534, 537
syntax errors, TRY/CATCH blocks and, 41
sysadmin server role, 193
sys.certificates view, 212
sys.database_audit_specification_details view, 225
table Analysis Tools add-in (Excel 2007), 867–873
table columns
defined as XML, 235–237
converting columns to XML, 278–279
SYSDATETIME function, 65
SYSDATETIMEOFFSET function, 65
sys.dm_audit_actions view, 225
sys.dm_audit_class_type_map view, 225
tspsk_data_exchange_plugin property, 543
sys.dm_database_encryption_keys view, 214
sys.dm_database_encryption_keys view, 214
sys.server_audit_specification_details view, 225
sys.server_audit_specifications view, 225
sys.server_file_audits view, 225
sys.sql_logins view, 198
system administrator passwords, 226
System.Data.SqlClient namespace, 122, 388
System.EnterpriseServices namespace, 473
System.Transactions namespace, 474, 476
SYSTIMESTAMP function, 65
table columns (continued)
default and constraints, 236
primary and secondary XML indexes, 244–247
querying. See XQuery language
pivoting dynamically, 23
statements without explicit column lists, TVPs and, 53, 54
table join types, MERGE with, 78
table locking, with Partitioned Table Parallelism, 602
table replication, MERGE statement for (T-SQL), 73
TableAdapterManager class (example), 424
TableCreationOption enumeration, 537
tables
adding to Data Source View pane, 628
for data mining models, 805
with FILESTREAM columns, 315–318
hierarchical. See hierarchical data
table-valued functions (TVFs), 132
table-valued parameters (TVPs), 45–59
bulk inserts and updates, 49
dictionary-style TVPs, 54
limitations of, 59
multiple-row sets, 48
passing TVPs using ADO.NET, 56
single rows of data, 51–54
tablix data region, 881, 886, 910–915
extending with groups, 910–912
fixed columns and rows, adding, 913
report actions, 918
target of MERGE statement, 70. See also MERGE statement
WHEN NOT MATCHED BY TARGET, 72
target type (PBM), defined, 184
TCs (transaction coordinators). See transaction managers (TMs)
TDE. See Transparent Data Encryption
templates
for DMX queries, 856
for reports, 897–900
Terminate method (aggregate classes), 141
testing
actions, 692
data mining, test sets for, 808
data web services (ADO.NET), 414
KPIs (key performance indicators), 681–683
SQL CLR stored procedures, 120, 129–131
text test function (XML PATH), 257
thread management (data warehousing), 601
time data type, 59
time data types, T-SQL. See date and time data types, T-SQL
time dimensions (OLAP). See dimensions; server time dimensions
Time Series algorithm (Analysis Services), 797
time zone awareness
T-SQL data types, 61–62, 65
xsd:dateTime data type, 240
time-variant, data warehouses as, 572
TMs. See transaction managers (TMs)
TO APPLICATION_LOG option, CREATE SERVER AUDIT statement, 220
TO FILE clause, CREATE SERVER AUDIT, 216, 219
TO SECURITY_LOG option, CREATE SERVER AUDIT statement, 220
TODATETIMEOFFSET function, 66
TOP clause
DMX statements, 849
MERGE statement and, 79
TOP parameter, T-SQL, 26–27
top-down approach to data warehousing, 572–573
ToString method
geometry and geography data types, 349
corehyid data type, 288–292
totals in matrix reports, 908
TRACK_COLUMNS_UPDATED option, 540
tracking changes for Sync Services, 538–543
changing tracking functions, 542
configuring database for, 538–543
training sets for data mining, 808
transaction abort error, 41
transaction coordinators. See transaction managers (TMs)
transaction managers (TMs), 469
TransactionOption attribute, 473
transactions, 449–490. See also T-SQL
ACID properties, 449, 450–452
defined, 450
distributed transactions, 453, 468–484
in .NET Framework, 473
terminology of, 461, 469
isolation levels. See isolation levels
local transactions, 452, 453–461
autocommit transaction mode, 453
batch-scoped transaction mode, 457–459
explicit transaction mode, 453–456
implicit transaction mode, 456
reliability of, 449
in SQL CLR (CLR integration), 485–488
example of, 489–490
terminology of, 461, 469–470
Transactions namespace, 477
TransactionScope class, 476, 487
example of, 489–490
translations (OLAP cubes), 707–711
Transparent Data Encryption (TDE), 211–216
backing up certificates, 214
enabling, 213
querying views, 213
restoring encrypted databases, 215
transplanting subtrees (hierarchical data), 303
tree structures. See hierarchical data
Tree Viewer, 823–825
ViewType property, 862
trend expressions, KPI, 678
TrendGraphic property (Kpis collection), 764
TriggerAction property (TriggerContext), 139
triggers
  DDL triggers, 43, 136
  SQL CLR triggers, 136–140
TRUSTWORTH property, 151
TRY and CATCH blocks, T-SQL, 40–42
T-SQL, 13–111, 111. See also transactions
aggregations, 140
APPLY operator, 25
CAST function
  converting columns to XML, 278–279
  extracting from datetime2 data, 64
common table expressions (CTEs), 14–20
creating recursive queries with, 18–20
CONVERT function
  converting columns to XML, 278–279
  extracting from datetime2 data, 64
date and time data types, 59–67
  accuracy, storage, and format, 62–64
  new and changed since SQL Server 2000, 65
  separation of dates and times, 59
  time zone awareness, 61–62
DDL triggers, 43
enhancements in SQL Server 2008, 4
FILESTREAM access, enabling, 311
GROUPING SETS operator, 97–109
  NULL values, handling, 105–109
  returning top level only, 103
  rolling up all level combinations, 101
  rolling up by level, 99
  rollup and cube operations together, 103
INSERT OVER DML syntax, 90–97
CHANGES keyword, 94–97
OUTPUT...INTO, extending, 90–94
MERGE statement, 68–90
  defining merge source and target, 70
  DML behavior, 79–81
  join method, choosing, 78
  OUTPUT clause, 76
for table replication, 73
upsert operations, 81–90
WHEN MATCHED clause, 71
WHEN NOT MATCHED BY SOURCE clause, 74
WHEN NOT MATCHED BY TARGET clause, 72
PIVOT and UNPIVOT operators, 21–24
dynamically pivoting columns, 23
ranking functions, 28–40
DENSE_RANK and NTILE functions, 34–36
RANK function, 32
ROW_NUMBER function, 28–31
  using all together, 36
  windowing functions with, 37–40
shorthand syntax, 109
SNAPSHOT isolation level, 45, 465–467
SQL Server Change Tracking, 539
table-valued parameters (TVPs), 45–59
  bulk inserts and updates, 49
dictionary-style TVPs, 54
  limitations of, 59
  multiple-row sets, 48
  passing TVPs using ADO.NET, 56
  single rows of data, 51–54
  TOP parameter, 26–27
  triggers, 43, 136
  TRY and CATCH blocks, 40–42
  varchar(max) data type, 42
  WAITFOR statement, 43
xml data type as variable, 234
tuples, 752, 753
TVFs (table-valued functions), 132
TVPs (table-valued parameters), 45–59
  bulk inserts and updates, 49
dictionary-style TVPs, 54
  limitations of, 59
  multiple-row sets, 48
  passing TVPs using ADO.NET, 56
  single rows of data, 51–54
two-phase commit transactions, 470. See also transactions
two-way data synchronization, 494
TYPE option, FOR XML clause, 253, 254
typed DataSet objects, 378–387
  adding validation logic to, 391
  connected use of, 387
  connection string management, 381
data binding example, 420–424
TableAdapterManager Configuration Wizard, 382–385
TableAdapterManager objects, 380
  using DataSet designer, 379
types, user-defined, 145–150
  examining and managing in database, 152–158
  managing with Object Explorer, 152
U
UDM (unified dimensional model) paradigm, 705
UDTs (user-defined types), 145–150
  examining and managing in database, 152–158
  managing with Object Explorer, 152
UNION ALL operator, recursive queries with CTEs, 18
union data type (XSD), 242
UNION statement, FOR XML EXPLICIT with, 250
Unknown conflict (synchronization), 550
UNLIMITED option (audit file size), 219
UNPIVOT operator (T-SQL), 21–24
  dynamically pivoting columns, 23
UNSAFE ASSEMBLY permission, 151
Unsafe permission set (SQL CLR assemblies), 151, 785
Unspecified isolation level (ADO.NET), 468
unstructured storage, FILESTREAM for. See FILESTREAM feature
UPDATE statements
  bulk updates with TVPs, 51
  INSERT INTO...SELECT statement around, 90–94
  merges, 79. See also MERGE statement
  OUTPUT clause, 90
  TOP parameter in, 26–27
  WAITFOR statement with, 43
UpdatePanel controls, 431
upsert operations, 79
  MERGE statement (T-SQL) for, 81–90
URL access for viewing reports, 928
usage-based aggregation design, 701
user interface
  for data web services (ADO.NET), 414
  Windows Forms data binding, 421
user roles
  logins with, 194
  for report security, 944–949
user table time dimensions, 652
user-defined aggregates, 140–145
  managing with Object Explorer, 152
user-defined types (UDTs), 145–150
  examining and managing in database, 152–158
  managing with Object Explorer, 152
users, 193
  authenticating. See authentication and authorization
  guest account, 194
user-schema separation, 198–200
USING keyword, MERGE statement, 70
USING XML INDEX statement, 247
V
validating data mining models, 827–829
validating mining models, 812
validation logic, adding
  to ORM models, 410
  to typed DataSets, 391
validation of XML. See schemas, XML
value index (XML), 246
value method (xml data type), 266, 270
varbinary(max) data type, 42
varchar(max) data type, T-SQL, 42
variable function (sql), 276
vector objects, 344
verifying backups (example application), 178
ViewerMode property (Naïve Bayes and Cluster Viewer controls), 862
viewing data mining models, 818–826
  Cluster Viewer, 818–823
  ViewerMode property, 862
  Naïve Bayes Viewer, 826
  Tree Viewer, 823–825
viewing reports. See delivering reports
ViewType property (Tree Viewer), 862
Virtual Earth, 364–373
Visual Studio Analysis Services. See Analysis Services
Visual Studio, deploying reports with, 920
Visual Studio Tools for Office, 737
Visual Studio/SQL Server integration, 113–118
visualization of KPIs, 678
void functions in server-side managed code, 788
volatile enlistment, 471
W
W3C XML format for date-time values, 64
WAITFOR statement, T-SQL, 43
Web controls for data mining, 862
Web Service URL task (Reporting Services Configuration Manager), 938
Web services for data. See ADO.NET Data Services
Web services operations for deploying reports, 924
Web Services, Report Server, 938
Web Synchronization, 513–519
  configuring IIS for, 514–516
  configuring subscriber database for, 517–519
  programmatic, 526–530
  Sync Services for ADO.NET and, 538–543
  changing tracking functions, 542
  configuring database for, 538–543
Well-Known Text (WKT) markup language

Well-Known Text (WKT) markup language, 344
WHEN MATCHED clause, MERGE statement, 71, 78
WHEN NOT MATCHED clause, MERGE statement, 78
WHEN NOT MATCHED BY SOURCE clause, MERGE statement, 74–76, 78
WHEN NOT MATCHED BY TARGET clause, MERGE statement, 72, 78
WHERE clause (MDX queries), 752
WHERE clause (T-SQL), XQuery and, 273
where keyword (XQuery), 264
windowing functions with ranking functions (T-SQL), 37–40
Windows Authentication. See authentication and authorization
Windows event log, recording audits to, 220
Windows Forms data binding, 420–427
Windows Forms Model content browser controls, 858
Windows login, 192
Windows Presentation Foundation (WPF)
building client, 338
data binding for, 438–445
lack of design support, 439–514
XAML for, 441–445
WITH ALGORITHM clause, CREATE DATABASE ENCRYPTION statement, 212
WITH CHANGE_TRACKING_CONTEXT clause, 542, 548
WITH CUBE operator, 97, 99
WITH DRILLTHROUGH clause, 842
WITH EXECUTE AS CALLER clause, CREATE FUNCTION, 132
WITH PERMISSION_SET clause, CREATE ASSEMBLY, 126, 127, 152
WITH ROLLUP operator, 97, 99
WKT (Well-Known Text) markup language, 344
WPF (Windows Presentation Foundation)
building client, 338
data binding for, 438–445
lack of design support, 439–514
XAML for, 441–445
Xml.exist method, 266, 269
XAML (Extensible Application Markup Language), 438
for WPF data binding, 441–445
XML (Extensible Markup Language), 231
FOR XML clause, SELECT statement, 233, 247–259
AUTO keyword, 248
ELEMENTS keyword, 260
EXPLICIT keyword, 250
PATH keyword, 253, 255–258
RAW keyword, 248
ROOT option, 258
TYPE option, 253, 254
XMLSCHEMA option, 259
element-based (canonical), producing, 260
indexes, 244–247
OPENXML function, 249, 261–262
querying data using XQuery. See XQuery language
schemas, 237–244
inline XSD schemas, 259
lax validation, 241
SQL Server collections of, 237
union and list types, 242
xsl:.dateTime enhancements, 240
in SQL Server 2000, 233
XML BULK LOAD, 262
xml data type. See xml data type
XML DML language, 276
XMLA (XML for Analysis), 688, 755, 771–781, 863
as API, 771
calling Execute from .NET, 778
data mining and, 865
manipulating ADO MD.NET response content, 780
using from SQL Server Management Studio, 772–778
xml.exist method, 266, 269
xml.modify method, 266, 276–277
xml.nodes method, 266, 275
xml.query method, 266, 272–274
XMLSCHEMA option, FOR XML clause, 259
xml.value method, 266, 270
XPath expression, 263
   FOR XML PATH statement, 253, 255–258
XQuery language, 263–279
   converting columns to XML, 278–279
   SQL Server extensions, 275–276
   XML DML language, 276
xml.exist method, 266, 269
xml.modify method, 266, 276–277
xml.nodes method, 266, 275
xml.query method, 266, 272–274
xml.value method, 266, 270
XSD (XML Schema Definition), 237. See also
   schemas, XML
   inline schemas, 259
xsd:list data type, 242
xsd:union data type, 242