Microsoft® SQL Server® 2012 T-SQL Fundamentals

Gain a solid understanding of T-SQL—and write better queries

Master the fundamentals of Transact-SQL—and develop your own code for querying and modifying data in Microsoft SQL Server 2012. Led by a SQL Server expert, you’ll learn the concepts behind T-SQL querying and programming, and then apply your knowledge with exercises in each chapter. Once you understand the logic behind T-SQL, you’ll quickly learn how to write effective code—whether you’re a programmer or database administrator.

Discover how to:

• Work with programming practices unique to T-SQL
• Create database tables and define data integrity
• Query multiple tables using joins and subqueries
• Simplify code and improve maintainability with table expressions
• Implement insert, update, delete, and merge data modification strategies
• Tackle advanced techniques such as window functions, pivoting, and grouping sets
• Control data consistency using isolation levels, and mitigate deadlocks and blocking
• Take T-SQL to the next level with programmable objects

Get code samples on the web
Ready to download at http://go.microsoft.com/fwlink/?Linkid=248717
For system requirements, see the Introduction.

About the Author
Itzik Ben-Gan, Microsoft MVP for SQL Server since 1999, is cofounder of SolidQ, where he teaches and consults internationally on T-SQL querying, programming, and query tuning. He’s a frequent contributor to SQL Server Pro and MSDN Magazine, and speaks at industry events such as Microsoft TechEd, DevTeach, PASS, and SQL Server Connections.

DEVELOPER ROADMAP

Start Here!
• Beginner-level instruction
• Easy to follow explanations and examples
• Exercises to build your first projects

Step by Step
• For experienced developers learning a new topic
• Focus on fundamental techniques and tools
• Hands-on tutorial with practice files plus eBook

Developer Reference
• Professional developers; intermediate to advanced
• Expertly covers essential topics and techniques
• Features extensive, adaptable code examples

Focused Topics
• For programmers who develop complex or advanced solutions
• Specialized topics; narrow focus; deep coverage
• Features extensive, adaptable code examples
To Dato

To live in hearts we leave behind,
Is not to die.

—THOMAS CAMPBELL
This page intentionally left blank
# Contents at a Glance

*Foreword*  
xix  

*Introduction*  
xxi  

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Background to T-SQL Querying and Programming</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Single-Table Queries</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>Joins</td>
<td>99</td>
</tr>
<tr>
<td>4</td>
<td>Subqueries</td>
<td>129</td>
</tr>
<tr>
<td>5</td>
<td>Table Expressions</td>
<td>157</td>
</tr>
<tr>
<td>6</td>
<td>Set Operators</td>
<td>191</td>
</tr>
<tr>
<td>7</td>
<td>Beyond the Fundamentals of Querying</td>
<td>211</td>
</tr>
<tr>
<td>8</td>
<td>Data Modification</td>
<td>247</td>
</tr>
<tr>
<td>9</td>
<td>Transactions and Concurrency</td>
<td>297</td>
</tr>
<tr>
<td>10</td>
<td>Programmable Objects</td>
<td>339</td>
</tr>
<tr>
<td>A</td>
<td>Getting Started</td>
<td>375</td>
</tr>
</tbody>
</table>

*Index*  
397  

*About the Author*  
413
This page intentionally left blank
Contents

Foreword ................................................................. xix
Introduction .......................................................... xxii

Chapter 1  Background to T-SQL Querying and Programming  1
Theoretical Background ............................................. 1
  SQL ........................................................................ 2
  Set Theory ............................................................ 3
  Predicate Logic ........................................................ 4
  The Relational Model .............................................. 4
  The Data Life Cycle ................................................ 9
SQL Server Architecture ........................................... 12
  The ABC Flavors of SQL Server ............................... 12
  SQL Server Instances .......................................... 14
  Databases ............................................................ 15
  Schemas and Objects ............................................. 18
Creating Tables and Defining Data Integrity .................. 19
  Creating Tables .................................................... 19
  Defining Data Integrity ......................................... 21
Conclusion ................................................................ 25

Chapter 2  Single-Table Queries  27
Elements of the SELECT Statement ............................... 27
  The FROM Clause ................................................ 29
  The WHERE Clause ............................................. 31
  The GROUP BY Clause ......................................... 32

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:
microsoft.com/learning/booksurvey
The HAVING Clause ................................................. 36
The SELECT Clause ................................................. 36
The ORDER BY Clause .............................................. 42
The TOP and OFFSET-FETCH Filters ......................... 44
A Quick Look at Window Functions ............................... 48
Predicates and Operators .......................................... 50
CASE Expressions .................................................... 53
NULL Marks ........................................................... 55
All-at-Once Operations ............................................. 59
Working with Character Data ....................................... 61
  Data Types .......................................................... 61
  Collation ............................................................. 62
  Operators and Functions ......................................... 64
  The LIKE Predicate ................................................ 71
Working with Date and Time Data ................................. 73
  Date and Time Data Types ....................................... 73
  Literals ............................................................... 74
  Working with Date and Time Separately ...................... 78
  Filtering Date Ranges ............................................ 79
  Date and Time Functions ........................................ 80
Querying Metadata ................................................... 88
  Catalog Views ...................................................... 88
  Information Schema Views ..................................... 89
  System Stored Procedures and Functions ..................... 89
Conclusion .................................................................. 91
Exercises .................................................................... 91
  1 ........................................................................ 91
  2 ........................................................................ 92
  3 ........................................................................ 92
  4 ........................................................................ 92
  5 ........................................................................ 93
  6 ........................................................................ 93
  7 ........................................................................ 94
  8 ........................................................................ 94
Chapter 3  Joins .................................................... 99

Cross Joins .......................................................... 99
  ANSI SQL-92 Syntax .................................................. 100
  ANSI SQL-89 Syntax .................................................. 101
  Self Cross Joins ..................................................... 101
  Producing Tables of Numbers ..................................... 102

Inner Joins .......................................................... 103
  ANSI SQL-92 Syntax .................................................. 103
  ANSI SQL-89 Syntax .................................................. 105
  Inner Join Safety ..................................................... 105

More Join Examples .................................................. 106
  Composite Joins ...................................................... 106
  Non-Equi Joins ......................................................... 107
  Multi-Join Queries ................................................... 109

Outer Joins .......................................................... 110
  Fundamentals of Outer Joins ....................................... 110
  Beyond the Fundamentals of Outer Joins ......................... 113

Conclusion .......................................................... 120

Exercises ............................................................ 120
  1-1 ................................................................. 120
  1-2 (Optional, Advanced) ......................................... 121
  2 ................................................................. 122
  3 ................................................................. 123
  4 ................................................................. 123
# Chapter 4  Subqueries

Self-Contained Subqueries .............................................................. 129
Self-Contained Scalar Subquery Examples .................................... 130
Self-Contained Multivalued Subquery Examples .......................... 132

Correlated Subqueries ................................................................. 136
The EXISTS Predicate ................................................................... 138

Beyond the Fundamentals of Subqueries ..................................... 140
Returning Previous or Next Values .............................................. 140
Using Running Aggregates ......................................................... 141
Dealing with Misbehaving Subqueries ........................................ 142

Conclusion ................................................................................... 147

Exercises ...................................................................................... 147
1 .................................................................................................. 147
2 (Optional, Advanced) .............................................................. 148
3 .................................................................................................. 149
4 .................................................................................................. 149
5 .................................................................................................. 150
6 .................................................................................................. 150
7 (Optional, Advanced) .............................................................. 151
8 (Optional, Advanced) .............................................................. 151
Solutions .......................................................... 152
1 ................................................................. 152
2 ................................................................. 152
3 ................................................................. 153
4 ................................................................. 153
5 ................................................................. 153
6 ................................................................. 154
7 ................................................................. 154
8 ................................................................. 155

Chapter 5  Table Expressions  157

Derived Tables .................................................. 157
Assigning Column Aliases .................................. 159
Using Arguments ............................................. 161
Nesting .......................................................... 161
Multiple References ........................................ 162

Common Table Expressions ................................. 163
Assigning Column Aliases in CTEs ....................... 164
Using Arguments in CTEs ................................. 165
Defining Multiple CTEs ..................................... 165
Multiple References in CTEs .............................. 166
Recursive CTEs .............................................. 166

Views .............................................................. 169
Views and the ORDER BY Clause ......................... 170
View Options .................................................. 172

Inline Table-Valued Functions ............................ 176
The APPLY Operator ......................................... 178

Conclusion ....................................................... 181

Exercises ........................................................ 182
1-1 ................................................................. 182
1-2 ................................................................. 182
2-1 ................................................................. 183
2-2 ................................................................. 183
3 (Optional, Advanced) ..................................... 184
## Chapter 6  Set Operators

The **UNION** Operator ................................................................. 192
The **UNION ALL** Multiset Operator ................................. 192
The **UNION** Distinct Set Operator ............................... 193

The **INTERSECT** Operator ...................................................... 194
The **INTERSECT** Distinct Set Operator ........................ 195
The **INTERSECT ALL** Multiset Operator ....................... 195

The **EXCEPT** Operator ............................................................... 198
The **EXCEPT** Distinct Set Operator ............................... 198
The **EXCEPT ALL** Multiset Operator ............................. 199

Precedence .............................................................................. 200

Circumventing Unsupported Logical Phases ...................... 202

Conclusion .............................................................................. 204

Exercises .............................................................................. 204
1 ..................................................................................... 204
2 ..................................................................................... 204
3 ..................................................................................... 206
4 ..................................................................................... 206
5 (Optional, Advanced) ...................................................... 206
Chapter 7  Beyond the Fundamentals of Querying  211

Window Functions ................................. 211
Ranking Window Functions ................. 214
Offset Window Functions ..................... 217
Aggregate Window Functions .................. 220

Pivoting Data ....................................... 222
Pivoting with Standard SQL ................. 224
Pivoting with the Native T-SQL PIVOT Operator .... 225

Unpivoting Data .................................... 228
Unpivoting with Standard SQL .............. 229
Unpivoting with the Native T-SQL UNPIVOT Operator .... 231

Grouping Sets .................................... 232
The GROUPING SETS Subclause ............. 234
The CUBE Subclause ............................. 234
The ROLLUP Subclause ........................... 235
The GROUPING and GROUPING_ID Functions ................ 236

Conclusion ........................................ 239

Exercises ........................................... 239
1 ........................................... 239
2 ........................................... 240
3 ........................................... 240
4 ........................................... 241
5 ........................................... 242
Chapter 8  Data Modification

Inserting Data .................................................. 247
  The INSERT VALUES Statement ................................. 247
  The INSERT SELECT Statement ................................ 249
  The INSERT EXEC Statement .................................... 250
  The SELECT INTO Statement ................................... 251
  The BULK INSERT Statement .................................... 252
  The Identity Property and the Sequence Object .......... 252

Deleting Data ....................................................... 261
  The DELETE Statement ......................................... 262
  The TRUNCATE Statement ...................................... 263
  DELETE Based on a Join ........................................ 263

Updating Data ....................................................... 264
  The UPDATE Statement ......................................... 265
  UPDATE Based on a Join ....................................... 267
  Assignment UPDATE ............................................ 269

Merging Data ........................................................ 270

Modifying Data Through Table Expressions .................... 274

Modifications with TOP and OFFSET-FETCH .................... 277

The OUTPUT Clause ............................................... 280
  INSERT with OUTPUT ........................................... 280
  DELETE with OUTPUT .......................................... 282
  UPDATE with OUTPUT .......................................... 283
  MERGE with OUTPUT .......................................... 284
  Composable DML .................................................. 285

Conclusion ............................................................ 287
Chapter 9  Transactions and Concurrency  297

Transactions .......................................................... 297
Locks and Blocking ..................................................... 300
  Locks ............................................................... 300
  Troubleshooting Blocking ......................................... 303
Isolation Levels .......................................................... 309
  The READ UNCOMMITTED Isolation Level ............... 310
  The READ COMMITTED Isolation Level ................. 311
  The REPEATABLE READ Isolation Level ................ 313
  The Serialization Isolation Level ......................... 314
Isolation Levels Based on Row Versioning .................. 316
Summary of Isolation Levels ...................................... 323
Deadlocks .............................................................. 323
Conclusion .............................................................. 326
Chapter 10 Programmable Objects

Variables ................................................. 339
Batches ...................................................... 341
  A Batch As a Unit of Parsing ....................... 342
  Batches and Variables .............................. 343
  Statements That Cannot Be Combined in the Same Batch .................. 343
  A Batch As a Unit of Resolution .................. 344
  The GO n Option .................................... 344
Flow Elements .......................................... 345
  The IF . . . ELSE Flow Element .................. 345
  The WHILE Flow Element ......................... 346
  An Example of Using IF and WHILE ............... 348
Cursors .................................................. 348
### Contents

Temporary Tables ............................................. 353  
Local Temporary Tables ..................................... 353  
Global Temporary Tables .................................... 355  
Table Variables ............................................. 356  
Table Types .................................................. 357  
Dynamic SQL .................................................. 359  
The EXEC Command .......................................... 359  
The sp_executesql Stored Procedure ....................... 360  
Using PIVOT with Dynamic SQL ............................ 361  
Routines ....................................................... 362  
  User-Defined Functions .................................. 362  
  Stored Procedures ........................................ 364  
  Triggers .................................................. 366  
Error Handling ................................................ 370  
Conclusion .................................................... 374  

### Appendix A  Getting Started  375

Getting Started with SQL Database ....................... 375  
Installing an On-Premises Implementation of SQL Server 376  
  1. Obtain SQL Server .................................... 376  
  2. Create a User Account .................................. 376  
  3. Install Prerequisites .................................... 377  
  4. Install the Database Engine, Documentation, and Tools 377  
Downloading Source Code and Installing the Sample Database ........ 385  
Working with SQL Server Management Studio ............ 387  
Working with SQL Server Books Online .................. 393  

Index 397  

About the Author 413
This page intentionally left blank
Foreword

I'm very happy that Itzik has managed to find the time and energy to produce a book about T-SQL fundamentals. For many years, Itzik has been using his great Microsoft SQL Server teaching, mentoring, and consulting experience to write books on advanced programming subjects, leaving a significant gap not only for the novice and less experienced users but also for the many experts working with SQL Server in roles where T-SQL programming is not a high priority.

When it comes to T-SQL, Itzik is one of the most knowledgeable people in the world. In fact, we (members of the SQL Server development team), turn to Itzik for expert advice on most of the new language extensions we plan to implement. His feedback and consultations have become an important part of our SQL Server development process.

It is never an easy task for a person who is a subject matter expert to write an introductory book; however, Itzik has the advantage of having taught both introductory and advanced programming classes for many years. Such experience is a great asset when differentiating the fundamental T-SQL information from the more advanced topics. But in this book, Itzik is not simply avoiding anything considered advanced; he is not afraid to take on inherently complex subjects such as set theory, predicate logic, and the relational model, introducing them in simple terms, and providing just enough information for readers to understand their importance to the SQL language. The result is a book that rewards readers with an understanding of not only what and how T-SQL works, but also why.

In programming manuals and books, there is no better way to convey the subject under discussion than with a good example. This book includes many examples—and you can download them all from Itzik's website, http://tsql.solidq.com. T-SQL is a dialect of the official ISO and ANSI standards for the SQL language, but it has numerous extensions that can improve the expressiveness and brevity of your T-SQL code. Many of Itzik's examples show the T-SQL dialect solution and the equivalent ANSI SQL solution to the same exercise side by side. This is a great advantage for readers who are familiar with the ANSI version of SQL but who are new to T-SQL, as well as for programmers who need to write SQL code that can be deployed easily across several different database platforms.
Itzik’s deep connection to the SQL Server team shows in his explanation of the Appliance, Box, Cloud (ABC) flavors of SQL Server in Chapter 1, “Background to T-SQL Querying and Programming.” So far, I have seen the term “ABC” used only internally within the Microsoft SQL Server team, but I’m sure it is only a matter of time until the term spreads around. Itzik developed and tested the examples in the book against both the “B” (box) and “C” (cloud) flavors of SQL Server. And the Appendix points out where you can get started with the cloud version of SQL Server, known as Windows Azure SQL Database. Therefore, you can use this book as a starting point for your own cloud experiences. The Azure website shows how to start your free subscription to the Azure services, so you can then execute the examples in the book.

The cloud extension of SQL Server is an extremely important point that you should not miss. I consider it to be so important that I’m doing something here that never should be done in a Foreword—advertising another book (sorry, Itzik, I have to do this!). My own interest and belief in cloud computing skyrocketed after reading Nicholas G. Carr’s *The Big Switch* (W.W. Norton and Company, 2009), and I want to share that experience. It is a great book that compares the advancement of cloud computing to electrification in the early 1900s. My certainty in the future of cloud computing was further cemented by watching James Hamilton’s “Cloud Computing Economies of Scale” presentation at the MIX10 conference (the recording is available at http://channel9.msdn.com/events/MIX/MIX10/EX01).

Itzik mentions one more cloud-related change that you should be aware of. We were used to multi-year gaps between SQL Server releases, but that pattern is changing significantly with the cloud; you should instead be prepared for several smaller cloud releases (called Service Updates) deployed in the Microsoft Data Centers around the world every year. Therefore, Itzik wisely documents the discrepancies between SQL Server and Windows Azure SQL Database T-SQL on his http://tsql.solidq.com website rather than in the book, so he can easily keep the information up to date.

Enjoy the book—and even more—enjoy the new insights into T-SQL that this book will bring to you.

*Lubor Kollar, SQL Server development team, Microsoft*
Introduction

This book walks you through your first steps in T-SQL (also known as Transact-SQL), which is the Microsoft SQL Server dialect of the ISO and ANSI standards for SQL. You’ll learn the theory behind T-SQL querying and programming and how to develop T-SQL code to query and modify data, and you’ll get an overview of programmable objects.

Although this book is intended for beginners, it is not merely a set of procedures for readers to follow. It goes beyond the syntactical elements of T-SQL and explains the logic behind the language and its elements.

Occasionally, the book covers subjects that may be considered advanced for readers who are new to T-SQL; therefore, those sections are optional reading. If you already feel comfortable with the material discussed in the book up to that point, you might want to tackle the more advanced subjects; otherwise, feel free to skip those sections and return to them after you’ve gained more experience. The text will indicate when a section may be considered more advanced and is provided as optional reading.

Many aspects of SQL are unique to the language and are very different from other programming languages. This book helps you adopt the right state of mind and gain a true understanding of the language elements. You learn how to think in terms of sets and follow good SQL programming practices.

The book is not version-specific; it does, however, cover language elements that were introduced in recent versions of SQL Server, including SQL Server 2012. When I discuss language elements that were introduced recently, I specify the version in which they were added.

Besides being available in an on-premises flavor, SQL Server is also available as a cloud-based service called Windows Azure SQL Database (formerly called SQL Azure). The code samples in this book were tested against both on-premises SQL Server and SQL Database. The book’s companion website (http://tsql.solidq.com) provides information about compatibility issues between the flavors—for example, features that are available in SQL Server 2012 but not yet in SQL Database.

To complement the learning experience, the book provides exercises that enable you to practice what you’ve learned. The book occasionally provides optional exercises that are more advanced. Those exercises are intended for readers who feel very comfortable with the material and want to challenge themselves with more difficult problems. The optional exercises for advanced readers are labeled as such.
Who Should Read This Book

This book is intended for T-SQL developers, DBAs, BI practitioners, report writers, analysts, architects, and SQL Server power users who just started working with SQL Server and need to write queries and develop code using Transact-SQL.

Assumptions

To get the most out of this book, you should have working experience with Windows and with applications based on Windows. You should also be familiar with basic concepts concerning relational database management systems.

Who Should Not Read This Book

Not every book is aimed at every possible audience. This book covers fundamentals. It is mainly aimed at T-SQL practitioners with little or no experience. With that said, several readers of the previous edition of this book have mentioned that—even though they already had years of experience—they still found the book useful for filling gaps in their knowledge.

Organization of This Book

This book starts with both a theoretical background to T-SQL querying and programming in Chapter 1, laying the foundations for the rest of the book, and also coverage of creating tables and defining data integrity. The book moves on to various aspects of querying and modifying data in Chapters 2 through 8, then to a discussion of concurrency and transactions in Chapter 9, and finally provides an overview of programmable objects in Chapter 10. The following section lists the chapter titles along with a short description:

- Chapter 1, “Background to T-SQL Querying and Programming,” provides a theoretical background of SQL, set theory, and predicate logic; examines the relational model and more; describes SQL Server’s architecture; and explains how to create tables and define data integrity.
- Chapter 2, “Single-Table Queries,” covers various aspects of querying a single table by using the SELECT statement.
Chapter 3, “Joins,” covers querying multiple tables by using joins, including cross joins, inner joins, and outer joins.

Chapter 4, “Subqueries,” covers queries within queries, otherwise known as subqueries.

Chapter 5, “Table Expressions,” covers derived tables, common table expressions (CTEs), views, inline table-valued functions, and the APPLY operator.

Chapter 6, “Set Operators,” covers the set operators UNION, INTERSECT, and EXCEPT.

Chapter 7, “Beyond the Fundamentals of Querying,” covers window functions, pivoting, unpivoting, and working with grouping sets.

Chapter 8, “Data Modification,” covers inserting, updating, deleting, and merging data.

Chapter 9, “Transactions and Concurrency,” covers concurrency of user connections that work with the same data simultaneously; it covers concepts including transactions, locks, blocking, isolation levels, and deadlocks.

Chapter 10, “Programmable Objects,” provides an overview of the T-SQL programming capabilities in SQL Server.

The book also provides an appendix, “Getting Started,” to help you set up your environment, download the book’s source code, install the TSQL2012 sample database, start writing code against SQL Server, and learn how to get help by working with SQL Server Books Online.

**System Requirements**

The Appendix, “Getting Started,” explains which editions of SQL Server 2012 you can use to work with the code samples included with this book. Each edition of SQL Server might have different hardware and software requirements, and those requirements are well documented in SQL Server Books Online under “Hardware and Software Requirements for Installing SQL Server 2012.” The Appendix also explains how to work with SQL Server Books Online.

If you’re connecting to SQL Database, hardware and server software are handled by Microsoft, so those requirements are irrelevant in this case.
Code Samples

This book features a companion website that makes available to you all the code used in the book, the errata, and additional resources.

http://tsql.solidq.com

Refer to the Appendix, “Getting Started,” for details about the source code.

Acknowledgments

Many people contributed to making this book a reality, whether directly or indirectly, and deserve thanks and recognition.

To Lilach, for giving reason to everything I do, and for not complaining about the endless hours I spend on SQL.

To my parents Mila and Gabi and to my siblings Mickey and Ina, thanks for the constant support. Thanks for accepting the fact that I’m away, which is now harder than ever. Mom, we’re all counting on you to be well and are encouraged by your strength and determination. Dad, thanks for being so supportive.

To members of the Microsoft SQL Server development team; Lubor Kollar, Tobias Ternstrom, Umachandar Jayachandran (UC), and I’m sure many others. Thanks for the great effort, and thanks for all the time you spent meeting me and responding to my email messages, addressing my questions and requests for clarification. I think that SQL Server 2012 and SQL Database show great investment in T-SQL, and I hope this will continue.

To the editorial team at O’Reilly Media and Microsoft Press; to Ken Jones, thanks for all the Itzik hours you spent, and thanks for initiating the project. To Russell Jones, thanks for your efforts in taking over the project and running it from the O’Reilly side. Also thanks to Kristen Borg, Kathy Krause, and all others who worked on the book.

To Herbert Albert and Gianluca Hotz, thanks for your work as the technical editors of the book. Your edits were excellent and I’m sure they improved the book’s quality and accuracy.

To SolidQ, my company for the last decade: it’s gratifying to be part of such a great company that evolved to what it is today. The members of this company are much more than colleagues to me; they are partners, friends, and family. Thanks to Fernando G. Guerrero, Douglas McDowell, Herbert Albert, Dejan Sarka, Gianluca Hotz, Jeanne Reeves,
Glenn McCoin, Fritz Lechnitz, Eric Van Soldt, Joelle Budd, Jan Taylor, Marilyn Templeton, Berry Walker, Alberto Martin, Lorena Jimenez, Ron Talmage, Andy Kelly, Rushabh Mehta, Eladio Rincón, Erik Veerman, Jay Hackney, Richard Waymire, Carl Rabeler, Chris Randall, Johan Åhlén, Raoul Illyés, Peter Larsson, Peter Myers, Paul Turley, and so many others.

To members of the SQL Server Pro editorial team, Megan Keller, Lavon Peters, Michele Crockett, Mike Otey, and I’m sure many others; I’ve been writing for the magazine for more than a decade and am grateful for the opportunity to share my knowledge with the magazine’s readers.

To SQL Server MVPs Alejandro Mesa, Erland Sommarskog, Aaron Bertrand, Tibor Karaszi, Paul White, and many others, and to the MVP lead, Simon Tien; this is a great program that I’m grateful and proud to be part of. The level of expertise of this group is amazing and I’m always excited when we all get to meet, both to share ideas and just to catch up at a personal level over beer. I believe that, in great part, Microsoft’s inspiration to add new T-SQL capabilities in SQL Server is thanks to the efforts of SQL Server MVPs, and more generally the SQL Server community. It is great to see this synergy yielding such a meaningful and important outcome.

To Q2, Q3, and Q4, thanQ.

Finally, to my students: teaching SQL is what drives me. It’s my passion. Thanks for allowing me to fulfill my calling, and for all the great questions that make me seek more knowledge.

Errata & Book Support

We’ve made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site:

http://www.microsoftpressstore.com/title/9780735658141

If you find an error that is not already listed, you can report it to us through the same page.

If you need additional support, email Microsoft Press Book Support at mspinput@microsoft.com.

Please note that product support for Microsoft software is not offered through the addresses above.
We Want to Hear from You

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://www.microsoft.com/learning/booksurvey

The survey is short, and we read every one of your comments and ideas. Thanks in advance for your input!

Stay in Touch

Let's keep the conversation going! We're on Twitter: http://twitter.com/MicrosoftPress.
CH A P T E R 4

Subqueries

SQL supports writing queries within queries, or nesting queries. The outermost query is a query whose result set is returned to the caller and is known as the outer query. The inner query is a query whose result is used by the outer query and is known as a subquery. The inner query acts in place of an expression that is based on constants or variables and is evaluated at run time. Unlike the results of expressions that use constants, the result of a subquery can change, because of changes in the queried tables. When you use subqueries, you avoid the need for separate steps in your solutions that store intermediate query results in variables.

A subquery can be either self-contained or correlated. A self-contained subquery has no dependency on the outer query that it belongs to, whereas a correlated subquery does. A subquery can be single-valued, multivalued, or table-valued. That is, a subquery can return a single value (a scalar value), multiple values, or a whole table result.

This chapter focuses on subqueries that return a single value (scalar subqueries) and subqueries that return multiple values (multivalued subqueries). I’ll cover subqueries that return whole tables (table subqueries) later in the book in Chapter 5, “Table Expressions.”

Both self-contained and correlated subqueries can return a scalar or multiple values. I’ll first describe self-contained subqueries and demonstrate both scalar and multivalued examples, and explicitly identify those as scalar or multivalued subqueries. Then I’ll describe correlated subqueries, but I won’t explicitly identify them as scalar or multivalued, assuming that you will already understand the difference.

Again, exercises at the end of the chapter can help you practice what you’ve learned.

Self-Contained Subqueries

Every subquery has an outer query that it belongs to. Self-contained subqueries are subqueries that are independent of the outer query that they belong to. Self-contained subqueries are very convenient to debug, because you can always highlight the subquery code, run it, and ensure that it does what it’s supposed to do. Logically, it’s as if the subquery code is evaluated only once before the outer query is evaluated, and then the outer query uses the result of the subquery. The following sections take a look at some concrete examples of self-contained subqueries.
Self-Contained Scalar Subquery Examples

A scalar subquery is a subquery that returns a single value—regardless of whether it is self-contained. Such a subquery can appear anywhere in the outer query where a single-valued expression can appear (such as WHERE or SELECT).

For example, suppose that you need to query the Orders table in the TSQL2012 database and return information about the order that has the maximum order ID in the table. You could accomplish the task by using a variable. The code could retrieve the maximum order ID from the Orders table and store the result in a variable. Then the code could query the Orders table and filter the order where the order ID is equal to the value stored in the variable. The following code demonstrates this technique.

USE TSQL2012;

DECLARE @maxid AS INT = (SELECT MAX(orderid) FROM Sales.Orders);

SELECT orderid, orderdate, empid, custid FROM Sales.Orders WHERE orderid = @maxid;

This query returns the following output.

<table>
<thead>
<tr>
<th>orderid</th>
<th>orderdate</th>
<th>empid</th>
<th>custid</th>
</tr>
</thead>
<tbody>
<tr>
<td>11077</td>
<td>2008-05-06 00:00:00.000</td>
<td>1</td>
<td>65</td>
</tr>
</tbody>
</table>

You can substitute the technique that uses a variable with an embedded subquery. You achieve this by substituting the reference to the variable with a scalar self-contained subquery that returns the maximum order ID. This way, your solution has a single query instead of the two-step process.

SELECT orderid, orderdate, empid, custid FROM Sales.Orders WHERE orderid = (SELECT MAX(O.orderid) FROM Sales.Orders AS O);

For a scalar subquery to be valid, it must return no more than one value. If a scalar subquery can return more than one value, it might fail at run time. The following query happens to run without failure.

SELECT orderid FROM Sales.Orders WHERE empid = (SELECT E.empid FROM HR.Employees AS E WHERE E.lastname LIKE N'B%');

The purpose of this query is to return the order IDs of orders placed by any employee whose last name starts with the letter B. The subquery returns employee IDs of all employees whose last names start with the letter B, and the outer query returns order IDs of orders where the employee ID is equal to the result of the subquery. Because an equality operator expects single-valued expressions
from both sides, the subquery is considered scalar. Because the subquery can potentially return more than one value, the choices of using an equality operator and a scalar subquery here are wrong. If the subquery returns more than one value, the query fails.

This query happens to run without failure because currently the Employees table contains only one employee whose last name starts with B (Sven Buck with employee ID 5). This query returns the following output, shown here in abbreviated form.

<table>
<thead>
<tr>
<th>orderid</th>
</tr>
</thead>
<tbody>
<tr>
<td>10248</td>
</tr>
<tr>
<td>10254</td>
</tr>
<tr>
<td>10269</td>
</tr>
<tr>
<td>10297</td>
</tr>
<tr>
<td>10320</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>10874</td>
</tr>
<tr>
<td>10899</td>
</tr>
<tr>
<td>10922</td>
</tr>
<tr>
<td>10954</td>
</tr>
<tr>
<td>11043</td>
</tr>
</tbody>
</table>

(42 row(s) affected)

Of course, if the subquery returns more than one value, the query fails. For example, try running the query with employees whose last names start with D.

```sql
SELECT orderid
FROM Sales.Orders
WHERE empid =
  (SELECT E.empid
   FROM HR.Employees AS E
   WHERE E.lastname LIKE N'D%');
```

Apparently, two employees have a last name starting with D (Sara Davis and Zoya Dolgopyatova). Therefore, the query fails at runtime with the following error.

Msg 512, Level 16, State 1, Line 1
Subquery returned more than 1 value. This is not permitted when the subquery follows =, !=, <, <=, >, >= or when the subquery is used as an expression.

If a scalar subquery returns no value, it returns a NULL. Recall that a comparison with a NULL yields UNKNOWN and that query filters do not return a row for which the filter expression evaluates to UNKNOWN. For example, the Employees table currently has no employees whose last names start with A; therefore, the following query returns an empty set.

```sql
SELECT orderid
FROM Sales.Orders
WHERE empid =
  (SELECT E.empid
   FROM HR.Employees AS E
   WHERE E.lastname LIKE N'A%');
```
Self-Contained Multivalued Subquery Examples

A multivalued subquery is a subquery that returns multiple values as a single column, regardless of whether the subquery is self-contained. Some predicates, such as the IN predicate, operate on a multivalued subquery.

Note There are other predicates that operate on a multivalued subquery; those are SOME, ANY, and ALL. They are very rarely used and therefore not covered in this book.

The form of the IN predicate is:

<scalar_expression> IN (<multivalued subquery>)

The predicate evaluates to TRUE if scalar_expression is equal to any of the values returned by the subquery. Recall the last request discussed in the previous section—returning order IDs of orders that were handled by employees with a last name starting with a certain letter. Because more than one employee can have a last name starting with the same letter, this request should be handled with the IN predicate and a multivalued subquery, and not with an equality operator and a scalar subquery. For example, the following query returns order IDs of orders placed by employees with a last name starting with D.

```
SELECT orderid
FROM Sales.Orders
WHERE empid IN
  (SELECT E.empid
   FROM HR.Employees AS E
   WHERE E.lastname LIKE N'D%');
```

Because it uses the IN predicate, this query is valid with any number of values returned—none, one, or more. This query returns the following output, shown here in abbreviated form.

```
orderid
---------
10258
10270
10275
10285
10292
...
10978
11016
11017
11022
11058

(166 row(s) affected)
```
You might wonder why you wouldn’t implement this task by using a join instead of subqueries, like this.

```sql
SELECT O.orderid
FROM HR.Employees AS E
JOIN Sales.Orders AS O
  ON E.empid = O.empid
WHERE E.lastname LIKE N'D%';
```

Similarly, you are likely to stumble into many other querying problems that you can solve with either subqueries or joins. In my experience, there’s no reliable rule of thumb that says that a subquery is better than a join. In some cases, the database engine interprets both types of queries the same way. Sometimes joins perform better than subqueries, and sometimes the opposite is true. My approach is to first write the solution query for the specified task in an intuitive form, and if performance is not satisfactory, one of my tuning approaches is to try query revisions. Such query revisions might include using joins instead of subqueries or using subqueries instead of joins.

As another example of using multivalued subqueries, suppose that you need to write a query that returns orders placed by customers from the United States. You can write a query against the `Orders` table that returns orders where the customer ID is in the set of customer IDs of customers from the United States. You can implement the last part in a self-contained, multivalued subquery. Here’s the complete solution query.

```sql
SELECT custid, orderid, orderdate, empid
FROM Sales.Orders
WHERE custid IN
  (SELECT C.custid
   FROM Sales.Customers AS C
   WHERE C.country = N'USA');
```

This query returns the following output, shown here in abbreviated form.

<table>
<thead>
<tr>
<th>custid</th>
<th>orderid</th>
<th>orderdate</th>
<th>empid</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>10262</td>
<td>2006-07-22 00:00:00.000</td>
<td>8</td>
</tr>
<tr>
<td>89</td>
<td>10269</td>
<td>2006-07-31 00:00:00.000</td>
<td>5</td>
</tr>
<tr>
<td>75</td>
<td>10271</td>
<td>2006-08-01 00:00:00.000</td>
<td>6</td>
</tr>
<tr>
<td>65</td>
<td>10272</td>
<td>2006-08-02 00:00:00.000</td>
<td>6</td>
</tr>
<tr>
<td>65</td>
<td>10294</td>
<td>2006-08-30 00:00:00.000</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>11040</td>
<td>2008-04-22 00:00:00.000</td>
<td>4</td>
</tr>
<tr>
<td>32</td>
<td>11061</td>
<td>2008-04-30 00:00:00.000</td>
<td>4</td>
</tr>
<tr>
<td>71</td>
<td>11064</td>
<td>2008-05-01 00:00:00.000</td>
<td>1</td>
</tr>
<tr>
<td>89</td>
<td>11066</td>
<td>2008-05-01 00:00:00.000</td>
<td>7</td>
</tr>
<tr>
<td>65</td>
<td>11077</td>
<td>2008-05-06 00:00:00.000</td>
<td>1</td>
</tr>
</tbody>
</table>

(122 row(s) affected)
As with any other predicate, you can negate the \textit{IN} predicate with the \textit{NOT} logical operator. For example, the following query returns customers who did not place any orders.

\begin{verbatim}
SELECT custid, companyname
FROM Sales.Customers
WHERE custid NOT IN
  (SELECT 0.custid
   FROM Sales.Orders AS O);
\end{verbatim}

Note that best practice is to qualify the subquery to exclude \textit{NULL} marks. Here, to keep the example simple, I didn’t exclude \textit{NULL} marks, but later in the chapter, in the “\textit{NULL Trouble}” section, I explain this recommendation.

The self-contained, multivalued subquery returns all customer IDs that appear in the \textit{Orders} table. Naturally, only IDs of customers who did place orders appear in the \textit{Orders} table. The outer query returns customers from the \textit{Customers} table where the customer ID is not in the set of values returned by the subquery—in other words, customers who did not place orders. This query returns the following output.

\begin{verbatim}
custid   companyname
--------- ----------------
 22       Customer DTDMN
 57       Customer WVAXS
\end{verbatim}

You might wonder whether specifying a \textit{DISTINCT} clause in the subquery can help performance, because the same customer ID can occur more than once in the \textit{Orders} table. The database engine is smart enough to consider removing duplicates without you asking it to do so explicitly, so this isn’t something you need to worry about.

The last example in this section demonstrates the use of multiple self-contained subqueries in the same query—both single-valued and multivalued. Before I describe the task at hand, run the following code to create a table called \textit{dbo.Orders} in the \textit{TSQL2012} database (for test purposes), and populate it with order IDs from the \textit{Sales.Orders} table that have even-numbered order IDs.

\begin{verbatim}
USE TSQL2012;
IF OBJECT_ID('dbo.Orders', 'U') IS NOT NULL DROP TABLE dbo.Orders;
CREATE TABLE dbo.Orders(orderid INT NOT NULL CONSTRAINT PK_Orders PRIMARY KEY);
INSERT INTO dbo.Orders(orderid)
  SELECT orderid
  FROM Sales.Orders
  WHERE orderid % 2 = 0;
\end{verbatim}
I describe the *INSERT* statement in more detail in Chapter 8, “Data Modification,” so don’t worry if you’re not familiar with it yet.

The task at hand is to return all individual order IDs that are missing between the minimum and maximum in the table. It can be quite complicated to solve this problem with a query without any helper tables. You might find the *Nums* table introduced in Chapter 3, “Joins,” very useful here. Remember that the *Nums* table contains a sequence of integers, starting with 1, with no gaps. To return all missing order IDs from the *Orders* table, query the *Nums* table and filter only numbers that are between the minimum and maximum in the *dbo.Orders* table and that do not appear in the set of order IDs in the *Orders* table. You can use scalar self-contained subqueries to return the minimum and maximum order IDs and a multivalued self-contained subquery to return the set of all existing order IDs. Here’s the complete solution query.

```sql
SELECT n
FROM dbo.Nums
WHERE n BETWEEN (SELECT MIN(O.orderid) FROM dbo.Orders AS O)
    AND (SELECT MAX(O.orderid) FROM dbo.Orders AS O)
    AND n NOT IN (SELECT O.orderid FROM dbo.Orders AS O);
```

Because the code that populated the *dbo.Orders* table filtered only even-numbered order IDs, this query returns all odd-numbered values between the minimum and maximum order IDs in the *Orders* table. The output of this query is shown here in abbreviated form.

```
n
----------
10249
10251
10253
10255
10257
...
11067
11069
11071
11073
11075
```

(414 row(s) affected)

When you’re done, run the following code for cleanup.

```sql
DROP TABLE dbo.Orders;
```
Correlated subqueries are subqueries that refer to attributes from the table that appears in the outer query. This means that the subquery is dependent on the outer query and cannot be invoked independently. Logically, it’s as if the subquery is evaluated separately for each outer row. For example, the query in Listing 4-1 returns orders with the maximum order ID for each customer.

**LISTING 4-1  Correlated Subquery**

```sql
USE TSQL2012;

SELECT custid, orderid, orderdate, empid
FROM Sales.Orders AS O1
WHERE orderid =
  (SELECT MAX(O2.orderid)
   FROM Sales.Orders AS O2
   WHERE O2.custid = O1.custid);
```

The outer query is against an instance of the `Orders` table called `O1`; it filters orders where the order ID is equal to the value returned by the subquery. The subquery filters orders from a second instance of the `Orders` table called `O2`, where the inner customer ID is equal to the outer customer ID, and returns the maximum order ID from the filtered orders. In simpler terms, for each row in `O1`, the subquery is in charge of returning the maximum order ID for the current customer. If the order ID in `O1` and the order ID returned by the subquery match, the order ID in `O1` is the maximum for the current customer, in which case the row from `O1` is returned by the query. This query returns the following output, shown here in abbreviated form.

<table>
<thead>
<tr>
<th>custid</th>
<th>orderid</th>
<th>orderdate</th>
<th>empid</th>
</tr>
</thead>
<tbody>
<tr>
<td>91</td>
<td>11044</td>
<td>2008-04-23 00:00:00.000</td>
<td>4</td>
</tr>
<tr>
<td>90</td>
<td>11005</td>
<td>2008-04-07 00:00:00.000</td>
<td>2</td>
</tr>
<tr>
<td>89</td>
<td>11066</td>
<td>2008-05-01 00:00:00.000</td>
<td>7</td>
</tr>
<tr>
<td>88</td>
<td>10935</td>
<td>2008-03-09 00:00:00.000</td>
<td>4</td>
</tr>
<tr>
<td>87</td>
<td>11025</td>
<td>2008-04-15 00:00:00.000</td>
<td>6</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10924</td>
<td>2008-03-04 00:00:00.000</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>11016</td>
<td>2008-04-10 00:00:00.000</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>10856</td>
<td>2008-01-28 00:00:00.000</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>10926</td>
<td>2008-03-04 00:00:00.000</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>11011</td>
<td>2008-04-09 00:00:00.000</td>
<td>3</td>
</tr>
</tbody>
</table>

(89 row(s) affected)

Correlated subqueries are usually much harder to figure out than self-contained subqueries. To better understand the concept of correlated subqueries, I find it useful to focus attention on a single row in the outer table and understand the logical processing that takes place for that row. For example, focus your attention on the order in the `Orders` table with order ID 10248.
With respect to this outer row, when the subquery is evaluated, the correlation or reference to \textit{O1.custid} means 85. After substituting the correlation with 85, you get the following.

\begin{verbatim}
SELECT MAX(O2.orderid)
FROM Sales.Orders AS O2
WHERE O2.custid = 85;
\end{verbatim}

This query returns the order ID 10739. The outer row’s order ID—10248—is compared with the inner one—10739—and because there’s no match in this case, the outer row is filtered out. The subquery returns the same value for all rows in \textit{O1} with the same customer ID, and only in one case is there a match—when the outer row’s order ID is the maximum for the current customer. Thinking in such terms will make it easier for you to grasp the concept of correlated subqueries.

The fact that correlated subqueries are dependent on the outer query makes them harder to debug than self-contained subqueries. You can’t just highlight the subquery portion and run it. For example, if you try to highlight and run the subquery portion in Listing 4-1, you get the following error.

\begin{verbatim}
Msg 4104, Level 16, State 1, Line 1
The multi-part identifier "O1.custid" could not be bound.
\end{verbatim}

This error indicates that the identifier \textit{O1.custid} cannot be bound to an object in the query, because \textit{O1} is not defined in the query. It is only defined in the context of the outer query. To debug correlated subqueries you need to substitute the correlation with a constant, and after ensuring that the code is correct, substitute the constant with the correlation.

As another example of a correlated subquery, suppose that you need to query the \textit{Sales.OrderValues} view and return for each order the percentage that the current order value is of the total values of all of the customer’s orders. In Chapter 7, “Beyond the Fundamentals of Querying,” I provide a solution to this problem that uses window functions; here I’ll explain how to solve the problem by using subqueries. It’s always a good idea to try to come up with several solutions to each problem, because the different solutions will usually vary in complexity and performance.

You can write an outer query against an instance of the \textit{OrderValues} view called \textit{O1}; in the \textit{SELECT} list, divide the current value by the result of a correlated subquery that returns the total value from a second instance of \textit{OrderValues} called \textit{O2} for the current customer. Here’s the complete solution query.

\begin{verbatim}
SELECT orderid, custid, val,
    CAST(100. * val / (SELECT SUM(O2.val)
                        FROM Sales.OrderValues AS O2
                        WHERE O2.custid = O1.custid)
    AS NUMERIC(5,2)) AS pct
FROM Sales.OrderValues AS O1
ORDER BY custid, orderid;
\end{verbatim}
The `CAST` function is used to convert the datatype of the expression to `NUMERIC` with a precision of 5 (the total number of digits) and a scale of 2 (the number of digits after the decimal point).

This query returns the following output.

<table>
<thead>
<tr>
<th>orderid</th>
<th>custid</th>
<th>val</th>
<th>pct</th>
</tr>
</thead>
<tbody>
<tr>
<td>10643</td>
<td>1</td>
<td>814.50</td>
<td>19.06</td>
</tr>
<tr>
<td>10692</td>
<td>1</td>
<td>878.00</td>
<td>20.55</td>
</tr>
<tr>
<td>10702</td>
<td>1</td>
<td>330.00</td>
<td>7.72</td>
</tr>
<tr>
<td>10835</td>
<td>1</td>
<td>845.80</td>
<td>19.79</td>
</tr>
<tr>
<td>10952</td>
<td>1</td>
<td>471.20</td>
<td>11.03</td>
</tr>
<tr>
<td>11011</td>
<td>1</td>
<td>933.50</td>
<td>21.85</td>
</tr>
<tr>
<td>10308</td>
<td>2</td>
<td>88.80</td>
<td>6.33</td>
</tr>
<tr>
<td>10625</td>
<td>2</td>
<td>479.75</td>
<td>34.20</td>
</tr>
<tr>
<td>10759</td>
<td>2</td>
<td>320.00</td>
<td>22.81</td>
</tr>
<tr>
<td>10926</td>
<td>2</td>
<td>514.40</td>
<td>36.67</td>
</tr>
</tbody>
</table>

...(830 row(s) affected)

The **EXISTS** Predicate

T-SQL supports a predicate called `EXISTS` that accepts a subquery as input and returns `TRUE` if the subquery returns any rows and `FALSE` otherwise. For example, the following query returns customers from Spain who placed orders.

```
SELECT custid, companyname
FROM Sales.Customers AS C
WHERE country = N'Spain'
AND EXISTS
  (SELECT * FROM Sales.Orders AS O
   WHERE O.custid = C.custid);  
```

The outer query against the `Customers` table filters only customers from Spain for whom the `EXISTS` predicate returns `TRUE`. The `EXISTS` predicate returns `TRUE` if the current customer has related orders in the `Orders` table.

One of the benefits of using the `EXISTS` predicate is that it allows you to intuitively phrase English-like queries. For example, this query can be read just as you would say it in ordinary English: select the customer ID and company name attributes from the `Customers` table, where the country is equal to Spain, and at least one order exists in the `Orders` table with the same customer ID as the customer’s customer ID.
This query returns the following output.

custid  companyname
-------- ----------------
 8       Customer QUHWH
 29      Customer MDLWA
 30      Customer KSLQF
 69      Customer SIUIH

As with other predicates, you can negate the EXISTS predicate with the NOT logical operator. For example, the following query returns customers from Spain who did not place orders.

```sql
SELECT custid, companyname
FROM Sales.Customers AS C
WHERE country = N'Spain'
   AND NOT EXISTS
      (SELECT * FROM Sales.Orders AS O
       WHERE O.custid = C.custid);
```

This query returns the following output.

custid  companyname
-------- ----------------
 22      Customer DTDMN

Even though this book’s focus is on logical query processing and not performance, I thought you might be interested to know that the EXISTS predicate lends itself to good optimization. That is, the Microsoft SQL Server engine knows that it is enough to determine whether the subquery returns at least one row or none, and it doesn’t need to process all qualifying rows. You can think of this capability as a kind of short-circuit evaluation.

Unlike most other cases, in this case it’s logically not a bad practice to use an asterisk (*) in the SELECT list of the subquery in the context of the EXISTS predicate. The EXISTS predicate only cares about the existence of matching rows regardless of the attributes specified in the SELECT list, as if the whole SELECT clause were superfluous. The SQL Server database engine knows this, and in terms of optimization, ignores the subquery’s SELECT list. So in terms of optimization, specifying the column wildcard * in this case has no negative impact when compared to alternatives such as specifying a constant. However, some minor extra cost might be involved in the resolution process of expanding the wildcard against metadata info. But this extra resolution cost is so minor that you will probably barely notice it. My opinion on this matter is that queries should be natural and intuitive, unless there’s a very compelling reason to sacrifice this aspect of the code. I find the form EXISTS(SELECT * FROM ...) much more intuitive than EXISTS(SELECT 1 FROM ...). Saving the minor extra cost associated with the resolution of * is something that is not worth the cost of sacrificing the readability of the code.

Finally, another aspect of the EXISTS predicate that is interesting to note is that unlike most predicates in T-SQL, EXISTS uses two-valued logic and not three-valued logic. If you think about it, there’s no situation where it is unknown whether a query returns rows.
Beyond the Fundamentals of Subqueries

This section covers aspects of subqueries that you might consider to be beyond the fundamentals. I provide it as optional reading in case you feel very comfortable with the material covered so far in this chapter.

Returning Previous or Next Values

Suppose that you need to query the Orders table in the TSQL2012 database and return, for each order, information about the current order and also the previous order ID. The concept of “previous” implies logical ordering, but because you know that the rows in a table have no order, you need to come up with a logical equivalent to the concept of “previous” that can be phrased with a T-SQL expression. One example of such a logical equivalent is “the maximum value that is smaller than the current value.” This phrase can be expressed in T-SQL with a correlated subquery like this.

```
SELECT orderid, orderdate, empid, custid,
       (SELECT MAX(O2.orderid)
        FROM Sales.Orders AS O2
        WHERE O2.orderid < O1.orderid) AS prevorderid
FROM Sales.Orders AS O1;
```

This query produces the following output, shown here in abbreviated form.

<table>
<thead>
<tr>
<th>orderid</th>
<th>orderdate</th>
<th>empid</th>
<th>custid</th>
<th>prevorderid</th>
</tr>
</thead>
<tbody>
<tr>
<td>10248</td>
<td>2006-07-04 00:00:00.000</td>
<td>5</td>
<td>85</td>
<td>NULL</td>
</tr>
<tr>
<td>10249</td>
<td>2006-07-05 00:00:00.000</td>
<td>6</td>
<td>79</td>
<td>10248</td>
</tr>
<tr>
<td>10250</td>
<td>2006-07-08 00:00:00.000</td>
<td>4</td>
<td>34</td>
<td>10249</td>
</tr>
<tr>
<td>10251</td>
<td>2006-07-08 00:00:00.000</td>
<td>3</td>
<td>84</td>
<td>10250</td>
</tr>
<tr>
<td>10252</td>
<td>2006-07-09 00:00:00.000</td>
<td>4</td>
<td>76</td>
<td>10251</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11073</td>
<td>2008-05-05 00:00:00.000</td>
<td>2</td>
<td>58</td>
<td>11072</td>
</tr>
<tr>
<td>11074</td>
<td>2008-05-06 00:00:00.000</td>
<td>7</td>
<td>73</td>
<td>11073</td>
</tr>
<tr>
<td>11075</td>
<td>2008-05-06 00:00:00.000</td>
<td>8</td>
<td>68</td>
<td>11074</td>
</tr>
<tr>
<td>11076</td>
<td>2008-05-06 00:00:00.000</td>
<td>4</td>
<td>9</td>
<td>11075</td>
</tr>
<tr>
<td>11077</td>
<td>2008-05-06 00:00:00.000</td>
<td>1</td>
<td>65</td>
<td>11076</td>
</tr>
</tbody>
</table>

(830 row(s) affected)

Notice that because there’s no order before the first, the subquery returned a NULL for the first order.

Similarly, you can phrase the concept of “next” as “the minimum value that is greater than the current value.” Here’s the T-SQL query that returns for each order the next order ID.

```
SELECT orderid, orderdate, empid, custid,
       (SELECT MIN(O2.orderid)
        FROM Sales.Orders AS O2
        WHERE O2.orderid > O1.orderid) AS nextorderid
FROM Sales.Orders AS O1;
```
This query produces the following output, shown here in abbreviated form.

<table>
<thead>
<tr>
<th>orderid</th>
<th>orderdate</th>
<th>empid</th>
<th>custid</th>
<th>nextorderid</th>
</tr>
</thead>
<tbody>
<tr>
<td>10248</td>
<td>2006-07-04 00:00:00.000</td>
<td>5</td>
<td>85</td>
<td>10249</td>
</tr>
<tr>
<td>10249</td>
<td>2006-07-05 00:00:00.000</td>
<td>6</td>
<td>79</td>
<td>10250</td>
</tr>
<tr>
<td>10250</td>
<td>2006-07-08 00:00:00.000</td>
<td>4</td>
<td>34</td>
<td>10251</td>
</tr>
<tr>
<td>10251</td>
<td>2006-07-08 00:00:00.000</td>
<td>3</td>
<td>84</td>
<td>10252</td>
</tr>
<tr>
<td>10252</td>
<td>2006-07-09 00:00:00.000</td>
<td>4</td>
<td>76</td>
<td>10253</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11073</td>
<td>2008-05-05 00:00:00.000</td>
<td>2</td>
<td>58</td>
<td>11074</td>
</tr>
<tr>
<td>11074</td>
<td>2008-05-06 00:00:00.000</td>
<td>7</td>
<td>73</td>
<td>11075</td>
</tr>
<tr>
<td>11075</td>
<td>2008-05-06 00:00:00.000</td>
<td>8</td>
<td>68</td>
<td>11076</td>
</tr>
<tr>
<td>11076</td>
<td>2008-05-06 00:00:00.000</td>
<td>4</td>
<td>9</td>
<td>11077</td>
</tr>
<tr>
<td>11077</td>
<td>2008-05-06 00:00:00.000</td>
<td>1</td>
<td>65</td>
<td>NULL</td>
</tr>
</tbody>
</table>

(830 row(s) affected)

Notice that because there’s no order after the last, the subquery returned a NULL for the last order.

Note that SQL Server 2012 introduces new window functions called LAG and LEAD that allow the return of an element from a “previous” or “next” row based on specified ordering. I will cover these and other window functions in Chapter 7.

Using Running Aggregates

Running aggregates are aggregates that accumulate values over time. In this section, I use the Sales.OrderTotalsByYear view to demonstrate the technique for calculating running aggregates. The view shows total order quantities by year. Query the view to examine its contents.

```sql
SELECT orderyear, qty
FROM Sales.OrderTotalsByYear;
```

You get the following output.

<table>
<thead>
<tr>
<th>orderyear</th>
<th>qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007</td>
<td>25489</td>
</tr>
<tr>
<td>2008</td>
<td>16247</td>
</tr>
<tr>
<td>2006</td>
<td>9581</td>
</tr>
</tbody>
</table>

Suppose you need to return for each year the order year, quantity, and running total quantity over the years. That is, for each year, return the sum of the quantity up to that year. So for the earliest year recorded in the view (2006), the running total is equal to that year’s quantity. For the second year (2007), the running total is the sum of the first year plus the second year, and so on.

You can complete this task by querying one instance of the view (call it O1) to return for each year the order year and quantity, and then by using a correlated subquery against a second instance of the view (call it O2) to calculate the running-total quantity. The subquery should filter all years in O2
that are smaller than or equal to the current year in $O_1$, and sum the quantities from $O_2$. Here’s the solution query.

```sql
SELECT orderyear, qty, 
      (SELECT SUM(O2.qty) 
       FROM Sales.OrderTotalsByYear AS O2 
       WHERE O2.orderyear <= O1.orderyear) AS runqty 
FROM Sales.OrderTotalsByYear AS O1 
ORDER BY orderyear;
```

This query returns the following output.

<table>
<thead>
<tr>
<th>orderyear</th>
<th>qty</th>
<th>runqty</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006</td>
<td>9581</td>
<td>9581</td>
</tr>
<tr>
<td>2007</td>
<td>25489</td>
<td>35070</td>
</tr>
<tr>
<td>2008</td>
<td>16247</td>
<td>51317</td>
</tr>
</tbody>
</table>

Note that SQL Server 2012 enhances the capabilities of window aggregate functions, allowing new, highly efficient solutions for running totals needs. As mentioned, I will discuss window functions in Chapter 7.

### Dealing with Misbehaving Subqueries

This section introduces cases in which subqueries might behave counter to your expectations, and provides best practices that you can follow to avoid logical bugs in your code that are associated with those cases.

#### NULL Trouble

Remember that T-SQL uses three-valued logic. In this section, I will demonstrate problems that can evolve with subqueries when NULL marks are involved and you do not take into consideration the three-valued logic.

Consider the following apparently intuitive query that is supposed to return customers who did not place orders.

```sql
SELECT custid, companyname 
FROM Sales.Customers 
WHERE custid NOT IN(SELECT O.custid 
                    FROM Sales.Orders AS O);
```

With the current sample data in the `Orders` table in the `TSQL2012` database, the query seems to work the way you expect it to; and indeed, it returns two rows for the two customers who did not place orders.

<table>
<thead>
<tr>
<th>custid</th>
<th>companyname</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Customer DTDMN</td>
</tr>
<tr>
<td>57</td>
<td>Customer WVAXS</td>
</tr>
</tbody>
</table>
Next, run the following code to insert a new order to the Orders table with a NULL customer ID.

```sql
INSERT INTO Sales.Orders
    (custid, empid, orderdate, requireddate, shippeddate, shipperid,
    freight, shipname, shipaddress, shipcity, shipregion,
    shippostalcde, shipcountry)
VALUES(NULL, 1, '20090212', '20090212',
    '20090212', 1, 123.00, N'abc', N'abc', N'abc',
    N'abc', N'abc', N'abc');
```

Run the query that is supposed to return customers who did not place orders again.

```sql
SELECT custid, companyname
FROM Sales.Customers
WHERE custid NOT IN(SELECT O.custid
                      FROM Sales.Orders AS O);
```

This time, the query returns an empty set. Keeping in mind what you’ve read in the section about NULL marks in Chapter 2, “Single-Table Queries,” try to explain why the query returns an empty set. Also try to think of ways to get customers 22 and 57 in the output, and in general, to figure out best practices you can follow to avoid such problems, assuming that there is a problem here.

Obviously, the culprit in this story is the NULL customer ID that was added to the Orders table and is now returned among the known customer IDs by the subquery.

Let’s start with the part that behaves the way you expect it to. The IN predicate returns TRUE for a customer who placed orders (for example, customer 85), because such a customer is returned by the subquery. The NOT operator is used to negate the IN predicate; hence, the NOT TRUE becomes FALSE, and the customer is not returned by the outer query. This means that when a customer ID appears in the Orders table, you can tell for sure that the customer placed orders, and therefore you don’t want to see it in the output. However, when you have a NULL customer ID in the Orders table, you can’t tell for sure whether a certain customer ID does not appear in Orders, as explained shortly.

The IN predicate returns UNKNOWN (the truth value UNKNOWN like the truth values TRUE and FALSE) for a customer such as 22 that does not appear in the set of known customer IDs in Orders. The IN predicate returns UNKNOWN for such a customer, because comparing it with all known customer IDs yields FALSE, and comparing it with the NULL in the set yields UNKNOWN. FALSE OR UNKNOWN yields UNKNOWN. As a more tangible example, consider the expression 22 NOT IN (1, 2, NULL). This expression can be rephrased as NOT 22 IN (1, 2, NULL). You can expand the last expression to NOT (22 = 1 OR 22 = 2 OR 22 = NULL). Evaluate each individual expression in the parentheses to its truth value and you get NOT (FALSE OR FALSE OR UNKNOWN), which translates to NOT UNKNOWN, which evaluates to UNKNOWN.

The logical meaning of UNKNOWN here before you apply the NOT operator is that it can’t be determined whether the customer ID appears in the set, because the NULL could represent that customer ID as well as anything else. The tricky part is that negating the UNKNOWN with the NOT operator still yields UNKNOWN, and UNKNOWN in a query filter is filtered out. This means that in a case where it is unknown whether a customer ID appears in a set, it is also unknown whether it doesn’t appear in the set.
In short, when you use the \texttt{NOT IN} predicate against a subquery that returns at least one \texttt{NULL}, the outer query always returns an empty set. Values from the outer table that are known to appear in the set are not returned because the outer query is supposed to return values that do not appear in the set. Values that do not appear in the set of known values are not returned because you can never tell for sure that the value is not in the set that includes the \texttt{NULL}.

So, what practices can you follow to avoid such trouble?

First, when a column is not supposed to allow \texttt{NULL} marks, it is important to define it as \texttt{NOT NULL}. Enforcing data integrity is much more important than many people realize.

Second, in all queries that you write, you should consider all three possible truth values of three-valued logic (\texttt{TRUE}, \texttt{FALSE}, and \texttt{UNKNOWN}). Think explicitly about whether the query might process \texttt{NULL} marks, and if so, whether the default treatment of \texttt{NULL} marks is suitable for your needs. When it isn’t, you need to intervene. For example, in the example we’ve been working with, the outer query returns an empty set because of the comparison with \texttt{NULL}. If you want to check whether a customer ID appears in the set of known values and ignore the \texttt{NULL} marks, you should exclude the \texttt{NULL} marks—either explicitly or implicitly.

One way to explicitly exclude the \texttt{NULL} marks is to add the predicate \texttt{O.custid IS NOT NULL} to the subquery, like this.

\begin{verbatim}
SELECT custid, companyname
FROM Sales.Customers
WHERE custid NOT IN(SELECT O.custid
    FROM Sales.Orders AS O
    WHERE O.custid IS NOT NULL);
\end{verbatim}

You can also exclude the \texttt{NULL} marks implicitly by using the \texttt{NOT EXISTS} predicate instead of \texttt{NOT IN}, like this.

\begin{verbatim}
SELECT custid, companyname
FROM Sales.Customers AS C
WHERE NOT EXISTS
    (SELECT *
        FROM Sales.Orders AS O
        WHERE O.custid = C.custid);
\end{verbatim}

Recall that unlike \texttt{IN}, \texttt{EXISTS} uses two-valued predicate logic. \texttt{EXISTS} always returns \texttt{TRUE} or \texttt{FALSE} and never \texttt{UNKNOWN}. When the subquery stumbles into a \texttt{NULL} in \texttt{O.custid}, the expression evaluates to \texttt{UNKNOWN} and the row is filtered out. As far as the \texttt{EXISTS} predicate is concerned, the \texttt{NULL} cases are eliminated naturally, as though they weren’t there. So \texttt{EXISTS} ends up handling only known customer IDs. Therefore, it’s safer to use \texttt{NOT EXISTS} than \texttt{NOT IN}.

When you’re done experimenting, run the following code for cleanup.

\begin{verbatim}
DELETE FROM Sales.Orders WHERE custid IS NULL;
\end{verbatim}
Substitution Errors in Subquery Column Names

Logical bugs in your code can sometimes be very elusive. In this section, I describe an elusive bug that has to do with an innocent substitution error in a subquery column name. After explaining the bug, I provide best practices that can help you avoid such bugs in the future.

The examples in this section query a table called MyShippers in the Sales schema. Run the following code to create and populate this table.

```sql
IF OBJECT_ID('Sales.MyShippers', 'U') IS NOT NULL
  DROP TABLE Sales.MyShippers;
CREATE TABLE Sales.MyShippers
  (
    shipper_id INT          NOT NULL,
    companyname NVARCHAR(40) NOT NULL,
    phone       NVARCHAR(24) NOT NULL,
    CONSTRAINT PK_MyShippers PRIMARY KEY(shipper_id)
  );
INSERT INTO Sales.MyShippers(shipper_id, companyname, phone)
VALUES(1, N'Shipper GVSUA', N'(503) 555-0137'),
       (2, N'Shipper ETYNR', N'(425) 555-0136'),
       (3, N'Shipper ZHISN', N'(415) 555-0138');
```

Consider the following query, which is supposed to return shippers who shipped orders to customer 43.

```sql
SELECT shipper_id, companyname
FROM Sales.MyShippers
WHERE shipper_id IN
  (SELECT shipper_id
   FROM Sales.Orders
   WHERE custid = 43);
```

This query produces the following output.

```
shipper_id  companyname
----------- ---------------
1           Shipper GVSUA
2           Shipper ETYNR
3           Shipper ZHISN
```

Apparently, only shippers 2 and 3 shipped orders to customer 43, but for some reason, this query returned all shippers from the MyShippers table. Examine the query carefully and also the schemas of the tables involved, and see if you can explain why.

It turns out that the column name in the Orders table holding the shipper ID is not called shipper_id; it is called shipperid (no underscore). The column in the MyShippers table is called shipper_id with an underscore. The resolution of nonprefixed column names works in the context of a subquery from the current/inner scope outward. In our example, SQL Server first looks for the column shipper_id in the Orders table. Such a column is not found there, so SQL Server looks for it in the outer table in the query, MyShippers. Because one is found, it is the one used.
You can see that what was supposed to be a self-contained subquery unintentionally became a correlated subquery. As long as the Orders table has at least one row, all rows from the MyShippers table find a match when comparing the outer shipper ID with a query that returns the very same outer shipper ID for each row from the Orders table.

Some might argue that this behavior is a design flaw in standard SQL. However, it’s not that the designers of this behavior in the ANSI SQL committee thought that it would be difficult to detect the “error;” rather, it’s an intentional behavior designed to allow you to refer to column names from the outer table without needing to prefix them with the table name, as long as those column names are unambiguous (that is, as long as they appear only in one of the tables).

This problem is more common in environments that do not use consistent attribute names across tables. Sometimes the names are only slightly different, as in this case—shipperid in one table and shipper_id in another. That’s enough for the bug to manifest itself.

You can follow a couple of best practices to avoid such problems—one to implement in the long run, and one that you can implement in the short run.

In the long run, your organization should as a policy not underestimate the importance of using consistent attribute names across tables. In the short run, of course, you don’t want to start changing existing column names, which could break application code.

In the short run, you can adopt a very simple practice—prefix column names in subqueries with the source table alias. This way, the resolution process only looks for the column in the specified table, and if no such column is there, you get a resolution error. For example, try running the following code.

```
SELECT shipper_id, companyname
FROM Sales.MyShippers
WHERE shipper_id IN
  (SELECT O.shipper_id
   FROM Sales.Orders AS O
   WHERE O.custid = 43);
```

You get the following resolution error.

```
Msg 207, Level 16, State 1, Line 4
Invalid column name 'shipper_id'.
```

After getting this error, you of course can identify the problem and correct the query.

```
SELECT shipper_id, companyname
FROM Sales.MyShippers
WHERE shipper_id IN
  (SELECT O.shipperid
   FROM Sales.Orders AS O
   WHERE O.custid = 43);
```
This time, the query returns the expected result.

<table>
<thead>
<tr>
<th>shipper_id</th>
<th>companyname</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Shipper ETYNR</td>
</tr>
<tr>
<td>3</td>
<td>Shipper ZHISN</td>
</tr>
</tbody>
</table>

When you’re done, run the following code for cleanup.

```sql
IF OBJECT_ID('Sales.MyShippers', 'U') IS NOT NULL
DROP TABLE Sales.MyShippers;
```

**Conclusion**

This chapter covered subqueries. It discussed self-contained subqueries, which are independent of their outer queries, and correlated subqueries, which are dependent on their outer queries. Regarding the results of subqueries, I discussed scalar and multivalued subqueries. I also provided a more advanced section as optional reading, in which I covered returning previous and next values, using running aggregates, and dealing with misbehaving subqueries. Remember to always think about the three-valued logic and the importance of prefixing column names in subqueries with the source table alias.

The next chapter focuses on table subqueries, also known as table expressions.

**Exercises**

This section provides exercises to help you familiarize yourself with the subjects discussed in this chapter. The sample database *TSQL2012* is used in all exercises in this chapter.

1

Write a query that returns all orders placed on the last day of activity that can be found in the *Orders* table.

- Tables involved: *Sales.Orders*
- Desired output:

<table>
<thead>
<tr>
<th>orderid</th>
<th>orderdate</th>
<th>custid</th>
<th>empid</th>
</tr>
</thead>
<tbody>
<tr>
<td>11077</td>
<td>2008-05-06 00:00:00.000</td>
<td>65</td>
<td>1</td>
</tr>
<tr>
<td>11076</td>
<td>2008-05-06 00:00:00.000</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>11075</td>
<td>2008-05-06 00:00:00.000</td>
<td>68</td>
<td>8</td>
</tr>
<tr>
<td>11074</td>
<td>2008-05-06 00:00:00.000</td>
<td>73</td>
<td>7</td>
</tr>
</tbody>
</table>
2 (Optional, Advanced)

Write a query that returns all orders placed by the customer(s) who placed the highest number of orders. Note that more than one customer might have the same number of orders.

- Tables involved: Sales.Orders

- Desired output (abbreviated):

<table>
<thead>
<tr>
<th>custid</th>
<th>orderid</th>
<th>orderdate</th>
<th>empid</th>
</tr>
</thead>
<tbody>
<tr>
<td>71</td>
<td>10324</td>
<td>2006-10-08 00:00:00.000</td>
<td>9</td>
</tr>
<tr>
<td>71</td>
<td>10393</td>
<td>2006-12-25 00:00:00.000</td>
<td>1</td>
</tr>
<tr>
<td>71</td>
<td>10398</td>
<td>2006-12-30 00:00:00.000</td>
<td>2</td>
</tr>
<tr>
<td>71</td>
<td>10440</td>
<td>2007-02-10 00:00:00.000</td>
<td>4</td>
</tr>
<tr>
<td>71</td>
<td>10452</td>
<td>2007-02-20 00:00:00.000</td>
<td>8</td>
</tr>
<tr>
<td>71</td>
<td>10510</td>
<td>2007-04-18 00:00:00.000</td>
<td>6</td>
</tr>
<tr>
<td>71</td>
<td>10555</td>
<td>2007-06-02 00:00:00.000</td>
<td>6</td>
</tr>
<tr>
<td>71</td>
<td>10603</td>
<td>2007-07-18 00:00:00.000</td>
<td>8</td>
</tr>
<tr>
<td>71</td>
<td>10607</td>
<td>2007-07-22 00:00:00.000</td>
<td>5</td>
</tr>
<tr>
<td>71</td>
<td>10612</td>
<td>2007-07-28 00:00:00.000</td>
<td>1</td>
</tr>
<tr>
<td>71</td>
<td>10627</td>
<td>2007-08-11 00:00:00.000</td>
<td>8</td>
</tr>
<tr>
<td>71</td>
<td>10657</td>
<td>2007-09-04 00:00:00.000</td>
<td>2</td>
</tr>
<tr>
<td>71</td>
<td>10678</td>
<td>2007-09-23 00:00:00.000</td>
<td>7</td>
</tr>
<tr>
<td>71</td>
<td>10700</td>
<td>2007-10-10 00:00:00.000</td>
<td>3</td>
</tr>
<tr>
<td>71</td>
<td>10711</td>
<td>2007-10-21 00:00:00.000</td>
<td>5</td>
</tr>
<tr>
<td>71</td>
<td>10713</td>
<td>2007-10-22 00:00:00.000</td>
<td>1</td>
</tr>
<tr>
<td>71</td>
<td>10714</td>
<td>2007-10-22 00:00:00.000</td>
<td>5</td>
</tr>
<tr>
<td>71</td>
<td>10722</td>
<td>2007-10-29 00:00:00.000</td>
<td>8</td>
</tr>
<tr>
<td>71</td>
<td>10748</td>
<td>2007-11-20 00:00:00.000</td>
<td>3</td>
</tr>
<tr>
<td>71</td>
<td>10757</td>
<td>2007-11-27 00:00:00.000</td>
<td>6</td>
</tr>
<tr>
<td>71</td>
<td>10815</td>
<td>2008-01-05 00:00:00.000</td>
<td>2</td>
</tr>
<tr>
<td>71</td>
<td>10847</td>
<td>2008-01-22 00:00:00.000</td>
<td>4</td>
</tr>
<tr>
<td>71</td>
<td>10882</td>
<td>2008-02-11 00:00:00.000</td>
<td>4</td>
</tr>
<tr>
<td>71</td>
<td>10894</td>
<td>2008-02-18 00:00:00.000</td>
<td>1</td>
</tr>
<tr>
<td>71</td>
<td>10941</td>
<td>2008-03-11 00:00:00.000</td>
<td>7</td>
</tr>
<tr>
<td>71</td>
<td>10983</td>
<td>2008-03-27 00:00:00.000</td>
<td>2</td>
</tr>
<tr>
<td>71</td>
<td>10984</td>
<td>2008-03-30 00:00:00.000</td>
<td>1</td>
</tr>
<tr>
<td>71</td>
<td>11002</td>
<td>2008-04-06 00:00:00.000</td>
<td>4</td>
</tr>
<tr>
<td>71</td>
<td>11030</td>
<td>2008-04-17 00:00:00.000</td>
<td>7</td>
</tr>
<tr>
<td>71</td>
<td>11031</td>
<td>2008-04-17 00:00:00.000</td>
<td>6</td>
</tr>
<tr>
<td>71</td>
<td>11064</td>
<td>2008-05-01 00:00:00.000</td>
<td>1</td>
</tr>
</tbody>
</table>

(31 row(s) affected)
3

Write a query that returns employees who did not place orders on or after May 1, 2008.

- Tables involved: `HR.Employees` and `Sales.Orders`
- Desired output:

<table>
<thead>
<tr>
<th>empid</th>
<th>FirstName</th>
<th>lastname</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Judy</td>
<td>Lew</td>
</tr>
<tr>
<td>5</td>
<td>Sven</td>
<td>Buck</td>
</tr>
<tr>
<td>6</td>
<td>Paul</td>
<td>Suurs</td>
</tr>
<tr>
<td>9</td>
<td>Zoya</td>
<td>Dolgopyatova</td>
</tr>
</tbody>
</table>

4

Write a query that returns countries where there are customers but not employees.

- Tables involved: `Sales.Customers` and `HR.Employees`
- Desired output:

<table>
<thead>
<tr>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argentina</td>
</tr>
<tr>
<td>Austria</td>
</tr>
<tr>
<td>Belgium</td>
</tr>
<tr>
<td>Brazil</td>
</tr>
<tr>
<td>Canada</td>
</tr>
<tr>
<td>Denmark</td>
</tr>
<tr>
<td>Finland</td>
</tr>
<tr>
<td>France</td>
</tr>
<tr>
<td>Germany</td>
</tr>
<tr>
<td>Ireland</td>
</tr>
<tr>
<td>Italy</td>
</tr>
<tr>
<td>Mexico</td>
</tr>
<tr>
<td>Norway</td>
</tr>
<tr>
<td>Poland</td>
</tr>
<tr>
<td>Portugal</td>
</tr>
<tr>
<td>Spain</td>
</tr>
<tr>
<td>Sweden</td>
</tr>
<tr>
<td>Switzerland</td>
</tr>
<tr>
<td>Venezuela</td>
</tr>
</tbody>
</table>

(19 row(s) affected)
Write a query that returns for each customer all orders placed on the customer’s last day of activity.

- Tables involved: Sales.Orders

- Desired output:

<table>
<thead>
<tr>
<th>custid</th>
<th>orderid</th>
<th>orderdate</th>
<th>empid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11011</td>
<td>2008-04-09 00:00:00.000</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>10926</td>
<td>2008-03-04 00:00:00.000</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>10856</td>
<td>2008-01-28 00:00:00.000</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>11016</td>
<td>2008-04-10 00:00:00.000</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>10924</td>
<td>2008-03-04 00:00:00.000</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>87</td>
<td>11025</td>
<td>2008-04-15 00:00:00.000</td>
<td>6</td>
</tr>
<tr>
<td>88</td>
<td>10935</td>
<td>2008-03-09 00:00:00.000</td>
<td>4</td>
</tr>
<tr>
<td>89</td>
<td>11066</td>
<td>2008-05-01 00:00:00.000</td>
<td>7</td>
</tr>
<tr>
<td>90</td>
<td>11005</td>
<td>2008-04-07 00:00:00.000</td>
<td>2</td>
</tr>
<tr>
<td>91</td>
<td>11044</td>
<td>2008-04-23 00:00:00.000</td>
<td>4</td>
</tr>
</tbody>
</table>

(90 row(s) affected)

Write a query that returns customers who placed orders in 2007 but not in 2008.

- Tables involved: Sales.Customers and Sales.Orders

- Desired output:

<table>
<thead>
<tr>
<th>custid</th>
<th>companyname</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Customer KIDPX</td>
</tr>
<tr>
<td>23</td>
<td>Customer WVFAF</td>
</tr>
<tr>
<td>33</td>
<td>Customer FVXPQ</td>
</tr>
<tr>
<td>36</td>
<td>Customer LVJSO</td>
</tr>
<tr>
<td>43</td>
<td>Customer UISOJ</td>
</tr>
<tr>
<td>51</td>
<td>Customer PVDZC</td>
</tr>
<tr>
<td>85</td>
<td>Customer ENQZT</td>
</tr>
</tbody>
</table>

(7 row(s) affected)
7 (Optional, Advanced)

Write a query that returns customers who ordered product 12.


- Desired output:

<table>
<thead>
<tr>
<th>custid</th>
<th>companyname</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>Customer DVFMB</td>
</tr>
<tr>
<td>39</td>
<td>Customer GLLAG</td>
</tr>
<tr>
<td>71</td>
<td>Customer LCOUJ</td>
</tr>
<tr>
<td>65</td>
<td>Customer NYUHS</td>
</tr>
<tr>
<td>44</td>
<td>Customer OXFRU</td>
</tr>
<tr>
<td>51</td>
<td>Customer PVDZC</td>
</tr>
<tr>
<td>86</td>
<td>Customer SNXOJ</td>
</tr>
<tr>
<td>20</td>
<td>Customer THHDP</td>
</tr>
<tr>
<td>90</td>
<td>Customer XBBVR</td>
</tr>
<tr>
<td>46</td>
<td>Customer XPNIK</td>
</tr>
<tr>
<td>31</td>
<td>Customer YJCBX</td>
</tr>
<tr>
<td>87</td>
<td>Customer ZHYOS</td>
</tr>
</tbody>
</table>

(12 row(s) affected)

8 (Optional, Advanced)

Write a query that calculates a running-total quantity for each customer and month.

- Tables involved: Sales.CustOrders

- Desired output:

<table>
<thead>
<tr>
<th>custid</th>
<th>ordermonth</th>
<th>qty</th>
<th>runqty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2007-08-01 00:00:00.000</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>1</td>
<td>2007-10-01 00:00:00.000</td>
<td>41</td>
<td>79</td>
</tr>
<tr>
<td>1</td>
<td>2008-01-01 00:00:00.000</td>
<td>17</td>
<td>96</td>
</tr>
<tr>
<td>1</td>
<td>2008-03-01 00:00:00.000</td>
<td>18</td>
<td>114</td>
</tr>
<tr>
<td>1</td>
<td>2008-04-01 00:00:00.000</td>
<td>60</td>
<td>174</td>
</tr>
<tr>
<td>2</td>
<td>2006-09-01 00:00:00.000</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2007-08-01 00:00:00.000</td>
<td>18</td>
<td>24</td>
</tr>
<tr>
<td>2</td>
<td>2007-11-01 00:00:00.000</td>
<td>10</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>2008-03-01 00:00:00.000</td>
<td>29</td>
<td>63</td>
</tr>
<tr>
<td>3</td>
<td>2006-11-01 00:00:00.000</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>2007-04-01 00:00:00.000</td>
<td>30</td>
<td>54</td>
</tr>
<tr>
<td>3</td>
<td>2007-05-01 00:00:00.000</td>
<td>80</td>
<td>134</td>
</tr>
<tr>
<td>3</td>
<td>2007-06-01 00:00:00.000</td>
<td>83</td>
<td>217</td>
</tr>
<tr>
<td>3</td>
<td>2007-09-01 00:00:00.000</td>
<td>102</td>
<td>319</td>
</tr>
<tr>
<td>3</td>
<td>2008-01-01 00:00:00.000</td>
<td>40</td>
<td>359</td>
</tr>
</tbody>
</table>

... 

(636 row(s) affected)
Solutions

This section provides solutions to the exercises in the preceding section.

1

You can write a self-contained subquery that returns the maximum order date from the `Orders` table. You can refer to the subquery in the `WHERE` clause of the outer query to return all orders that were placed on the last day of activity. Here’s the solution query.

```
USE TSQL2012;

SELECT orderid, orderdate, custid, empid
FROM Sales.Orders
WHERE orderdate =
  (SELECT MAX(O.orderdate) FROM Sales.Orders AS O);
```

2

This problem is best solved in multiple steps. First, you can write a query that returns the customer or customers who placed the highest number of orders. You can achieve this by grouping the orders by customer, ordering the customers by `COUNT(*)` descending, and using the `TOP(1) WITH TIES` option to return the IDs of the customers who placed the highest number of orders. If you don’t remember how to use the `TOP` option, refer to Chapter 2. Here’s the query that solves the first step.

```
SELECT TOP (1) WITH TIES O.custid
FROM Sales.Orders AS O
GROUP BY O.custid
ORDER BY COUNT(*) DESC;
```

This query returns the value 71, which is the customer ID of the customer who placed the highest number of orders, 31. With the sample data stored in the `Orders` table, only one customer placed the maximum number of orders. But the query uses the `WITH TIES` option to return all IDs of customers who placed the maximum number of orders, in case there are more than one.

The next step is to write a query against the `Orders` table returning all orders where the customer ID is in the set of customer IDs returned by the solution query for the first step.

```
SELECT custid, orderid, orderdate, empid
FROM Sales.Orders
WHERE custid IN
  (SELECT TOP (1) WITH TIES O.custid
   FROM Sales.Orders AS O
   GROUP BY O.custid
   ORDER BY COUNT(*) DESC);
```
3
You can write a self-contained subquery against the Orders table that filters orders placed on or after May 1, 2008 and returns only the employee IDs from those orders. Write an outer query against the Employees table returning employees whose IDs do not appear in the set of employee IDs returned by the subquery. Here's the complete solution query.

```sql
SELECT empid, FirstName, lastname
FROM HR.Employees
WHERE empid NOT IN
  (SELECT O.empid
   FROM Sales.Orders AS O
   WHERE O.orderdate >= '20080501');
```

4
You can write a self-contained subquery against the Employees table returning the country attribute from each employee row. Write an outer query against the Customers table that filters only customer rows where the country does not appear in the set of countries returned by the subquery. In the SELECT list of the outer query, specify DISTINCT country to return only distinct occurrences of countries, because the same country can have more than one customer. Here's the complete solution query.

```sql
SELECT DISTINCT country
FROM Sales.Customers
WHERE country NOT IN
  (SELECT E.country FROM HR.Employees AS E);
```

5
This exercise is similar to Exercise 1, except that in that exercise, you were asked to return orders placed on the last day of activity in general; in this exercise, you were asked to return orders placed on the last day of activity for the customer. The solutions for both exercises are similar, but here you need to correlate the subquery to match the inner customer ID with the outer customer ID, like this.

```sql
SELECT custid, orderid, orderdate, empid
FROM Sales.Orders AS O1
WHERE orderdate =
  (SELECT MAX(O2.orderdate)
   FROM Sales.Orders AS O2
   WHERE O2.custid = O1.custid)
ORDER BY custid;
```

You're not comparing the outer row's order date with the general maximum order date, but instead with the maximum order date for the current customer.
You can solve this problem by querying the Customers table and using EXISTS and NOT EXISTS predicates with correlated subqueries to ensure that the customer placed orders in 2007 but not in 2008. The EXISTS predicate returns "TRUE" only if at least one row exists in the Orders table with the same customer ID as in the outer row, within the date range representing the year 2007. The NOT EXISTS predicate returns "TRUE" only if no row exists in the Orders table with the same customer ID as in the outer row, within the date range representing the year 2008. Here's the complete solution query.

```sql
SELECT custid, companyname
FROM Sales.Customers AS C
WHERE EXISTS
  (SELECT *
   FROM Sales.Orders AS O
   WHERE O.custid = C.custid
   AND O.orderdate >= '20070101'
   AND O.orderdate < '20080101')
AND NOT EXISTS
  (SELECT *
   FROM Sales.Orders AS O
   WHERE O.custid = C.custid
   AND O.orderdate >= '20080101'
   AND O.orderdate < '20090101');
```

You can solve this exercise by nesting EXISTS predicates with correlated subqueries. You write the outermost query against the Customers table. In the WHERE clause of the outer query, you can use the EXISTS predicate with a correlated subquery against the Orders table to filter only the current customer's orders. In the filter of the subquery against the Orders table, you can use a nested EXISTS predicate with a subquery against the OrderDetails table that filters only order details with product ID 12. This way, only customers who placed orders that contain product 12 in their order details are returned. Here's the complete solution query.

```sql
SELECT custid, companyname
FROM Sales.Customers AS C
WHERE EXISTS
  (SELECT *
   FROM Sales.Orders AS O
   WHERE O.custid = C.custid
   AND EXISTS
     (SELECT *
      FROM Sales.OrderDetails AS OD
      WHERE OD.orderid = O.orderid
      WHERE OD.ProductID = 12));
```
8

When I need to solve querying problems, I often find it useful to rephrase the original request in a more technical way so that it will be more convenient to translate the request to a T-SQL query. To solve the current exercise, you can first try to express the request “return a running total quantity for each customer and month” differently—in a more technical manner. For each customer, return the customer ID, month, the sum of the quantity for that month, and the sum of all months less than or equal to the current month. The rephrased request can be translated to the following T-SQL query quite literally.

```
SELECT custid, ordermonth, qty,
   (SELECT SUM(O2.qty)
      FROM Sales.CustOrders AS O2
      WHERE O2.custid = O1.custid
      AND O2.ordermonth <= O1.ordermonth) AS runqty
FROM Sales.CustOrders AS O1
ORDER BY custid, ordermonth;
```
This page intentionally left blank
Symbols

1NF (first normal form), 7
2NF (second normal form), 8
* (asterisk)
  performance, 41
  SELECT lists of subqueries, 139
\ (backslash), named instances, 14
[<Character>-<Character>] wildcard, 72
[Character List or Range>] wildcard, 73
, (comma), 37, 265
{} curly brackets, set theory, 3
" (double quotes), 64
@@@identity function, 254
[<List of Characters>] wildcard, 72
@params, 360
() parentheses
  column aliases in CTEs, 164
derived tables, 157
functions, 80
precedence, 52
% (percent) wildcard, 71
+ (plus sign) operator, 64
; (semicolon)
  MERGE, 272
  statements, 21, 29
' (single quotes), 64
.sql script files, 385
@stmt, 360
_ (underscore) wildcard, 72

A

ABC flavors, 12
access, views using permissions, 169
accounts
  creating user accounts on SQL Server, 376
  Windows Azure platform account, 376
AFTER INSERT trigger, 367
after trigger, 367
aggregates
  aggregation phase and pivoting data, 224
  functions
    NULL, 35
  running aggregates, 141, 350
  window functions, 220
aliases
  column aliases, 159
  columns, 38, 42
  expressions and attributes, 37
  external column aliasing
    views, 169
ALL
  set operators, 192
  UNION ALL operator, 196
all-at-once operations
  about, 59
  UPDATE, 266
Alt button, 392
ALTER DATABASE, 64
alternate keys, 7
ALTER SEQUENCE, 258
ALTER TABLE
  identity property, 255
  LOCK_ESCALATION, 302
A-Mark, 6
Analysis Services, BISM, 11
anchor members, defined, 167
AND operator, 51, 274
ANSI (American National Standards Institute), SQL, 2
ANSI SQL-89 syntax
  cross joins, 101
  inner joins, 105
ANSI SQL-92 syntax

ANSI SQL-92 syntax
  cross joins, 100
  inner joins, 103
appliance flavor, 12
APPLY operator, 178–181, 306
arguments
  CTEs, 165
derived tables, 161
arithmetic operators, 51
arrays, 1NF, 8
AS, inline aliasing, 160
assignment SELECT, 340
assignment UPDATE, 269
asterisk (*)
  performance, 41
  SELECT lists of subqueries, 139
atomicity, attributes, 7
attributes
  atomicity, 7
  blocking_session_id attribute, 308
  expressions, 36
  filtering in outer joins, 115
  foreign key constraints, 23
  nullability, 20
  set theory, 4
autonumbering, assignment UPDATE, 269

B
backslash (\), named instances, 14
bag, 3
batches, 341–345
  GO, 344
  statements that cannot be combined in the
    same batch, 343
  as a unit of parsing, 342
  as a unit of resolution, 344
  variables, 343
BEGIN, 346
BEGIN TRAN, 297
BETWEEN, 50
BISM (Business Intelligence Semantic Model), 11
blockers, terminating, 308
blocking. See locks and blocking
blocking_session_id attribute, 308
boundaries, transactions, 297
box flavor, 13
BULK INSERT, 252
Business Intelligence Semantic Model (BISM), 11
C
  caching, sequence objects, 257
  candidate keys
    3NF, 8
    about, 7
Cantor, George, set theory, 3
Cartesian products
  cross joins, 99
  inner joins, 103
CASE expressions
  about, 53
  pivoting data, 225
CAST function, 81, 138
catalog views, 88
CATCH blocks, 371
calendar data, 61–73
  collation, 62
  data types, 61
  LIKE predicate, 71
  operators and functions, 64–71
character data types, 51
CHARINDEX function, 67
check constraints, 24
CHECK, @@identity and SCOPE_IDENTITY, 255
CHECK OPTION option, 174
CHOOSE function, 55
clauses, defined, 29
close world assumption (CWA), 5
coding style, 21
collation
  character data, 62
  property, 16
COLUMNPROPERTY function, 90
columns
  aliases
    assigning, 159
    CTEs, 164
    query example, 38
    referencing within a SELECT clause, 42
  asterisk in column names, 41
  attributes in set theory, 4
  external column aliasing, 169
  identity property, 255
  INSERT VALUES, 248
  ordinal position
    in SQL, 41
    in T-SQL, 43
INDEX

prefixes, 101
substitution errors in subquery column names, 145
table expressions, 158
comma (,), 37, 265
COMMIT TRAN, 297
comparison operators, 51
compatibility, lock modes, 300
composite constraints, 22
composite joins, 106
compostable DML, 285
compression, 62
concatenating
strings, 64, 362
user input, 359
CONCAT function, 64
currency, 297–338
deadlocks, 323–325
exercises, 326–338
isolation levels, 309–323
READ COMMITTED isolation level, 311
READ COMMITTED SNAPSHOT isolation level, 321
READ UNCOMMITTED isolation level, 310
REPEATABLE READ isolation level, 313
row versioning, 316–322
SERIALIZABLE isolation level, 314
SNAPSHOT isolation level, 317–319
summary of isolation levels, 323
locks and blocking, 300–309
locks, 300
troubleshooting blocking, 303–309
transactions, 297–300
conflict detection, SNAPSHOT isolation level, 319
consistency, defined, 298
constraints
about, 6
check constraints, 24
data integrity, 22
default constraints, 24
foreign key constraints, 23
primary keys, 22
contained databases, 17
CONTINUE, 347
CONVERT function, 77, 81
correlated subqueries
about, 136–139
defined, 129
tables, 179
COUNT, outer joins, 118
CREATE SEQUENCE, 257
CREATE TABLE
about, 20
identity property, 255
ordinal position of columns, 41
CROSS APPLY, 178
cross joins, 99–103
ANSI SQL-89 syntax, 101
ANSI SQL-92 syntax, 100
self cross joins, 101
tables of numbers, 102
CTEs (common table expressions), 163
arguments, 165
column aliases, 164
multiple references, 166
recursive CTEs, 166–168
CUBE subclause, grouping sets, 234
curly brackets {}, set theory, 3
current date and time functions, 80
CURRENT_TIMESTAMP function, 80
cursors
about, 348–352
defined, 43
CWA (close world assumption), 5

data. See character data
DATABASEPROPERTYEX function, 90
databases
collation, 63
general engines, installing, 377–384
installing the sample database, 385
SQL Server, 15–18
triggers, 368
data compression, 62
Data Control Language (DCL), defined, 2
Data Definition Language (DDL)
defined, 2
triggers, 368
data integrity, 22–25
check constraints, 24
default constraints, 24
foreign key constraints, 23
primary key constraints, 22
DATALENGTH function, 67

D
data life cycle

data life cycle, 9–12
  BISM, 11
  DM, 12
  DW, 10
  OLTP, 10
Data Manipulation Language. See DML
data mart, defined, 10
data mining (DM), 12
Data Mining Extensions (DMX), defined, 12
data modification, 247–296
deleting data, 261–264
  DELETE, 262
  DELETE based on joins, 263
  TRUNCATE, 263
exercises and solutions, 287–296
inserting data, 247–261
  BULK INSERT, 252
  identity property and sequence object, 252–261
  INSERT EXEC, 250
  INSERT SELECT, 249
  INSERT VALUES, 247
  SELECT INTO, 251
merging data, 270–274
OUTPUT, 280–287
  compostable DML, 285
  DELETE, 282
  INSERT, 280
  MERGE, 284
  UPDATE, 283
table expressions, 274–277
  TOP and OFFSET-FETCH, 277–279
update data, 264–270
  assignment UPDATE, 269
  UPDATE, 265
  UPDATE based on joins, 267
data staging area (DSA), ETL process, 11
data types
  character data, 61
data and time data, 73
  precedence, 52, 74
  scalar expressions, 51
  set operators, 191
data warehouse (DW), 10
  DATEADD function, 83
data and time data, 73–87
data types, 73
  functions, 80–87
  literals, 74–78
  sequences, 113
  DATEDIFF function, 84, 126
  DATENAME function, 86
  DATEPART function, 85
  DATETIMEOFFSET, SWITCHOFFSET function, 83
  DAY function, 85
  DBCC CHECKIDENT, 256
  DB_NAME function, 305
  DCL (Data Control Language), defined, 2
  DDL (Data Definition Language)
    defined, 2
    triggers, 368
  DEADLOCK_PRIORITY, 323
deadlocks, concurrency, 323–325
declarative data integrity, 22
  DECLARE, 339, 356
defaults
  constraints, 24
  default instance, 14
  isolation levels, 301, 310, 321
  lock timeout value, 309
delete. See also TRUNCATE
  DELETE, 261–264
    about, 262
    based on joins, 263
    with DML triggers, 367
    when enabling snapshot-based isolation levels, 316
    OUTPUT, 282
delimiting identifier names, 30
derived tables, 157–163
  arguments, 161
  column aliases, 159
  multiple references, 162
  nesting, 161
dimension tables, snowflake dimension, 11
  DISTINCT
    distinct, defined in set theory, 3
    distinct set operators
      EXCEPT distinct set operator, 198
      INTERSECT distinct set operator, 195
      UNION distinct set operator, 193
  DM (data mining), 12
  DML (Data Manipulation Language)
    compostable DML, 285
    triggers, 367
foreign key constraints

DMV (dynamic management view), locks, 304
DMX (Data Mining Extensions), defined, 12
documentation
  installing, 377–384
  SQL Server Books Online, 393–396
double quotes ("), 64
downloading
  source code, 385
  SQL Server, 376
DROP, 263
DSA (data staging area), ETL process, 11
duplicates
  INTERSECT distinct set operator, 195
  rows, 39
  UNION distinct set operator, 193
durability, defined, 298
DW (data warehouse), 10
dynamic management view (DMV), locks, 304
dynamic SQL, 359–362
  EXEC, 359
  PIVOT, 361
  sp_executesql, 360

E

elements, order in set theory, 3
ELSE, CASE expressions, 53
embedded subqueries, 130
ENCRYPTION option, 172
END, 346
Entity Relationship Modeling (ERM), normalization, 7
EOMONTH function, 87
equi joins, 107
ERM (Entity Relationship Modeling), normalization, 7
error handling
  programmable objects, 370–374
  stored procedures, 364
ERROR_LINE function, 371
ERROR_MESSAGE function, 371
ERROR_NUMBER function, 371
ERROR_PROCEDURE function, 371
ERROR_SEVERITY function, 371
ERROR_STATE function, 371
ESCAPE character, 73
ETL process, 11
EVENTDATA function, 368
EXCEPT operator, 198–200
  EXCEPT ALL multiset operator, 199
  EXCEPT distinct set operator, 198

exclusive lock mode
  about, 300
  lock compatibility, 301
EXEC, 359
exercises and solutions
  beyond the fundamentals of querying, 239–246
  concurrency, 326–338
  data modification, 287–296
  joins, 120–128
  set operators, 204–210
  single-table queries, 91–98
  subqueries, 147–156
  table expressions, 182–190
EXISTS
  correlated subqueries, 138
  INTERSECT distinct set operator, 195
  using, 154
expressions. See also table expressions
  attributes, 36
  CASE expressions, 53
  table expressions, 274–277
  vector expressions, 268
extensions, SQL, 3
external column aliasing, views, 169
external forms
  column aliases in CTEs, 164
  column aliases in general, 160

F
FALSE, 55, 345
FETCH, OFFSET-FETCH, 47
file extensions, databases, 18
filegroups, 18
filters
  attributes in outer joins, 115
  data using predicates, 4
  date ranges, 79
  OFFSET-FETCH filter, 47
  TOP filter, 44–47
first normal form (1NF), 7
FIRST_VALUE function, 218
flavors, ABC flavors, 12
flow elements, 345–348
  IF ... ELSE, 345
  WHILE, 346
foreign key columns, NULL, 24
foreign key constraints
  about, 23
  TRUNCATE, 263
FORMAT function

FORMAT function, 71
forms
  external forms, 160
    column aliases in CTEs, 164
  inline aliasing form, 160
four-valued predicate logic, 6
framing
  aggregate functions, 221
  window functions, 213
FROM
  about, 29
  cross joins, 100
  DELETE, 262
  DELETE based on joins, 264
derived tables, 162
  multi-join queries, 116
  multiple references in CTEs, 166
table UDFs, 362
FROMPARTS function, 87
FULL, outer joins, 110
functions
  @identity function, 254
  aggregate functions, 35
  CAST function, 138
  CHARINDEX function, 67
  CHOOSE function, 55
  COALESCE function, 66
  COLUMNPROPERTY function, 90
  CONCAT function, 64
  CONVERT function, 77
  CURRENT_TIMESTAMP function, 80
  DATABASEPROPERTYEX function, 90
  DATETIME function, 67
date and time functions, 80–87
DB_NAME function, 305
ERROR_LINE function, 371
ERROR_MESSAGE function, 371
ERROR_NUMBER function, 371
ERROR_PROCEDURE function, 371
ERROR_SEVERITY function, 371
ERROR_STATE function, 371
EVENTDATA function, 368
FIRST_VALUE function, 218
FORMAT function, 71
GETDATE function, 80
GROUPING and GROUPING_ID functions, 236–238
IDENT_CURRENT function, 254
IIF function, 55
ISNULL function, 55
LAG function, 217, 243
LAST_VALUE function, 218
LEAD function, 217, 243
LEFT and RIGHT functions, 66
LEN function, 67
LOWER function, 70
LTRIM function, 70
NEWID function, 363
NEXT VALUE FOR function, 258, 281
NTILE function, 215
OBJECT_DEFINITION function, 173
OBJECT_NAME function, 305
OBJECTPROPERTY function, 90
ORDER BY and window functions, 196
PATINDEX function, 68
RAND function, 363
ranking functions, 215
REPLACE function, 68
REPLICATE function, 69
ROW_NUMBER function, 48, 196, 215, 276
RTRIM function, 70
SCHEMA_NAME function, 88
SCOPE_IDENTITY function, 254, 280
STUFF function, 70
SUBSTRING function, 66
SYSDATETIME function, 25, 80
SYSDATETIMEOFFSET function, 80
sys.dm_exec_sql_text function, 306
system stored procedures and functions, 89
SYSUTCDATETIME function, 80
TRY_CAST function, 81
TRY_CONVERT function, 81
TRY_PARSE function, 81
UDFs, 362
UPPER function, 70
window functions
  about, 48
    aggregates, 220, 352
    offset window functions, 217–219
  ORDER BY, 196
  ranking, 214–217
G
GETDATE function, 80
getting started, 375–396
downloading source code, 385
installing the sample database, 385
Microsoft SQL Server Books Online, 393–396
<table>
<thead>
<tr>
<th>SQL Server, 376–384</th>
</tr>
</thead>
<tbody>
<tr>
<td>creating user accounts, 376</td>
</tr>
<tr>
<td>installing prerequisites, 377</td>
</tr>
<tr>
<td>installing the database engine, documentation and tools, 377–384</td>
</tr>
<tr>
<td>obtaining SQL Server, 376</td>
</tr>
<tr>
<td>SQL Server Management Studio, 387–393</td>
</tr>
<tr>
<td>Windows Azure SQL Database, 375</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GETUTCDATE function, 80</th>
</tr>
</thead>
<tbody>
<tr>
<td>global temporary tables, 355</td>
</tr>
<tr>
<td>GO, 342, 344</td>
</tr>
<tr>
<td>granularity, data warehouses, 11</td>
</tr>
<tr>
<td>GROUP BY</td>
</tr>
<tr>
<td>about, 32–35</td>
</tr>
<tr>
<td>pivoting, 226</td>
</tr>
<tr>
<td>grouping phase, pivoting data, 224</td>
</tr>
<tr>
<td>grouping sets, 232–238</td>
</tr>
<tr>
<td>CUBE subclause, 234</td>
</tr>
<tr>
<td>GROUPING and GROUPING_ID functions, 236–238</td>
</tr>
<tr>
<td>GROUPING SETS subclause, 234</td>
</tr>
<tr>
<td>ROLLUP subclause, 235</td>
</tr>
</tbody>
</table>

**H**

<table>
<thead>
<tr>
<th>HAVING, 36</th>
</tr>
</thead>
<tbody>
<tr>
<td>heaps, Windows Azure SQL Database, 251</td>
</tr>
<tr>
<td>hints, table hints and isolation levels, 310</td>
</tr>
<tr>
<td>HOLDLOCK, 310</td>
</tr>
</tbody>
</table>

**I**

<table>
<thead>
<tr>
<th>IDENT_CURRENT function, 254</th>
</tr>
</thead>
<tbody>
<tr>
<td>identifiers, delimiting names of, 30</td>
</tr>
<tr>
<td>IDENTITY_INSERT, 255</td>
</tr>
<tr>
<td>identity property, 252–261</td>
</tr>
<tr>
<td>IF ... ELSE, 345</td>
</tr>
<tr>
<td>IF statement, 20</td>
</tr>
<tr>
<td>IIF function, 55</td>
</tr>
<tr>
<td>I-Mark, 6</td>
</tr>
<tr>
<td>IMPLICIT_TRANSACTIONS, 297</td>
</tr>
<tr>
<td>IN</td>
</tr>
<tr>
<td>self-contained multivalued subqueries, 132</td>
</tr>
<tr>
<td>static queries, 361</td>
</tr>
<tr>
<td>subqueries, 143</td>
</tr>
<tr>
<td>increments, sequence objects, 257</td>
</tr>
<tr>
<td>information schema views, 89</td>
</tr>
<tr>
<td>inline aliasing form, 160</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>inline TVFs</th>
</tr>
</thead>
<tbody>
<tr>
<td>about, 176</td>
</tr>
<tr>
<td>views, 169</td>
</tr>
<tr>
<td>inner joins, 103–106</td>
</tr>
<tr>
<td>ANSI SQL-89 syntax, 105</td>
</tr>
<tr>
<td>ANSI SQL-92 syntax, 103</td>
</tr>
<tr>
<td>inner join safety, 105</td>
</tr>
<tr>
<td>input parameters, inline table-valued functions, 176</td>
</tr>
<tr>
<td>INSERT</td>
</tr>
<tr>
<td>DML triggers, 367</td>
</tr>
<tr>
<td>OUTPUT, 280</td>
</tr>
<tr>
<td>inserting data, 247–261</td>
</tr>
<tr>
<td>BULK INSERT, 252</td>
</tr>
<tr>
<td>identity property and sequence object, 252–261</td>
</tr>
<tr>
<td>INSERT EXEC, 250</td>
</tr>
<tr>
<td>INSERT SELECT, 249</td>
</tr>
<tr>
<td>INSERT VALUES, 247</td>
</tr>
<tr>
<td>SELECT INTO, 251</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>installing</th>
</tr>
</thead>
<tbody>
<tr>
<td>sample database, 385</td>
</tr>
<tr>
<td>SQL Server, 376–396</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>default instances, 14</td>
</tr>
<tr>
<td>named instances, 14</td>
</tr>
<tr>
<td>SQL Server, 14</td>
</tr>
<tr>
<td>instead of trigger, 367</td>
</tr>
<tr>
<td>integers, sequences of, 102</td>
</tr>
<tr>
<td>integrity, referential integrity, 23</td>
</tr>
<tr>
<td>International Organization for Standardization (ISO), SQL, 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INTERSECT operator, 194–197</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERSECT ALL multiset operator, 195–197</td>
</tr>
<tr>
<td>INTERSECT distinct set operator, 195</td>
</tr>
<tr>
<td>ISDATE function, 86</td>
</tr>
<tr>
<td>ISNULL function, 55</td>
</tr>
<tr>
<td>ISO (International Organization for Standardization), SQL, 2</td>
</tr>
<tr>
<td>isolation, defined, 298</td>
</tr>
<tr>
<td>isolation levels, 309–323</td>
</tr>
<tr>
<td>READ COMMITTED isolation level, 311</td>
</tr>
<tr>
<td>READ COMMITTED SNAPSHOT isolation level, 311</td>
</tr>
<tr>
<td>READ UNCOMMITTED isolation level, 310</td>
</tr>
<tr>
<td>REPEATABLE READ isolation level, 313</td>
</tr>
<tr>
<td>row versioning, 316–322</td>
</tr>
<tr>
<td>SERIALIZABLE isolation level, 314</td>
</tr>
<tr>
<td>SNAPSHOT isolation level, 317–319</td>
</tr>
<tr>
<td>summary of isolation levels, 323</td>
</tr>
</tbody>
</table>
Joins, 99–128
- Composite joins, 106
- Cross joins, 99–103
  - ANSI SQL-89 syntax, 101
  - ANSI SQL-92 syntax, 100
- Self cross joins, 101
- Tables of numbers, 102
- Exercises and solutions, 120–128
- Inner joins, 103–106
  - ANSI SQL-89 syntax, 105
  - ANSI SQL-92 syntax, 103
- Inner join safety, 105
- Multi-join queries, 109
- Non-equi joins, 107
- Outer joins, 110–119
  - COUNT, 118
  - Filtering attributes, 115
  - Missing values, 113
  - Multi-join queries, 116
- Versus subqueries, 133

Keys
- Alternate keys, 7
- Candidate keys, 7, 8
- Constraints, 3
- Foreign key constraints, 23
- Primary key constraints, 22
- Surrogate keys, 252

L
- LAG function, 217, 243
- Language independence, 2
- Languages, date and time formats, 75
- LAST_VALUE function, 218
- LATERAL, 178
- .ldf file extension, 18
- LEAD function, 217, 243
- LEFT function, 66
- LEFT keyword, outer joins, 110
- LEN function, 67
- LIKE predicate
  - about, 71
  - Character strings and specified patterns, 50

Literals
- Data types, 61
- Date and time data, 74–78
- Local temporary tables, 353
- Lock compatibility
  - About, 301
  - Requested modes, 302
- LOCK_ESCALATION, 302
- Locks and blocking, 300–309
  - Locks, 300
  - Troubleshooting blocking, 303–309
- LOCK_TIMEOUT
  - About, 308
  - Default value, 309
- Log Data File, 18
- Logical operators, 51
- Logical phases, circumventing unsupported logical phases, 202
- Logical query processing
  - About, 27
  - Defined, 99
- Logic, predicate logic, 4
- Logon
  - SQL Server authenticated logon, 17
  - Windows authenticated logon, 17
- Lost updates, 314
- LOWER function, 70
- LTRIM function, 70

M
- Master databases, 16
- Master Data File, 18
- MAX, 62
- Maximum values, 257
- .mdf file extension, 18
- MERGE
  - About, 270–274
  - OUTPUT, 284
- Metadata, 88–90
  - Catalog views, 88
  - Information schema views, 89
  - System stored procedures and functions, 89
- Microsoft .NET routines, 362
- SQL Server prerequisites, 377
- Microsoft SQL Azure. See Windows Azure SQL Database
Microsoft SQL Server. See SQL Server
Microsoft SSMS
about, 393–396
loading SQL Server Books Online, 395
minimum values, 257
mirrored pairs, non-equi joins, 108
missing values
about, 6
outer joins, 113
model databases, 16
modes, locks, 300
MOLAP, 11
MONTH function, 85
msdb databases, 16
multi-join queries
about, 109
outer joins, 116
multiset operators
EXCEPT ALL multiset operator, 199
INTERSECT ALL multiset operator, 195–197
UNION ALL multiset operator, 192
multiset tables, 3
multivalued subqueries, examples, 132

N
named instances, 14
names
column names, 145
table columns in table expressions, 158
temporary tables, 353
namespaces, schemas, 19
natural joins, defined, 107
.ndf file extension, 18
nesting
derived tables, 161
queries, 129
.NET
routines, 362
SQL Server prerequisites, 377
NEWID function, 363
NEXT VALUE FOR function, 258, 281
next values, returning, 140
N (National), 51
NOCOUNT, 262
NOLOCK, 310
non-equi joins, 107
normalization, 7–9

NOT EXISTS
EXCEPT distinct set operator, 199
using, 154
NOT IN, 144
Not Master Data File, 18
NOT operator, 51
NTILE function, 215
NULL
aggregate functions, 35
concatenation, 65
foreign key columns, 24
@@identity and SCOPE_IDENTITY, 255
IF ... ELSE, 345
INSERT SELECT, 249
INSERT VALUES, 248
INTERSECT distinct set operator, 195
misbehaving subqueries, 142
multi-join queries, 116
outer joins, 110, 115
single-table queries, 55–59
subqueries, 134, 140
support for, 6
unpivoting, 231
nullability, 20
numbers, cross joins, tables of numbers, 102

O
obfuscated text, 172
OBJECT_DEFINITION function, 173
OBJECT_NAME function, 305
OBJECTPROPERTY function, 90
objects. See also programmable objects
object names and schemas, 19
SCHEMABINDING option, 174
schema-qualifying names of, 29
sequence object, 252–261
set theory, 3
SQL Server, 18
OFFSET clause, 172
OFFSET-FETCH
about, 47
circumventing unsupported logical phases, 203
data modification, 277–279
using, 158, 171
offsets
DATETIMEOFFSET, 74
window functions, 217–219
OLTP (online transactional processing), 10
ON
   ANSI SQL-89 syntax, 105
   ANSI SQL-92 syntax, 103
   multi-join queries, 116
   outer joins, 112
online transactional processing (OLTP), 10
on-premises SQL Server, 13
operations, all-at-once operations, 59
operators
   APPLY operator, 178–181, 306
   arithmetic operators, 51
   comparison operators, 51
   CROSS APPLY operator, 179
   logical operators, 51
   OUTER APPLY operator, 179
   plus sign (+) operator, 64
   precedence rules, 52
   SELECT, 50–53
optimistic concurrency, 301
optimization. See performance
order
   rows in tables, 4
   set elements, 3
   SQL processing of query clauses, 28
ORDER BY
   about, 42
   circumventing unsupported logical phases, 202
cursors, 348
OFFSET-FETCH, 48, 277
set operators, 191
table expressions, 158
TOP, 44, 277
using, 49
views, 170
window functions, 196, 213
ordering
   window ordering and aggregate functions, 221
   windows functions, 212
OR operator, 51
OUTER APPLY, 178
outer joins, 110–119
   about, 110–113
   COUNT, 118
   filtering attributes, 115
   missing values, 113
   multi-join queries, 116
   outer queries, defined, 129
OUTPUT, 280–287
   compostable DML, 285
   DELETE, 282
   INSERT, 280
   MERGE, 284
   UPDATE, 266, 283
   using, 365
OVER
   aggregate windows functions, 220
   sequence objects, 259
   window functions, 48, 211

P
Parallel Data Warehouse (PDW), appliances, 12
parameters, input parameters and inline table-valued functions, 176
parentheses ()
   column aliases in CTEs, 164
derived tables, 157
functions, 80
   precedence, 52
parsing
   batches as unit of parsing, 342
   PARSE function, 81
PARTITION BY, window functions, 48, 213
PATINDEX function, 68
PDW (Parallel Data Warehouse), appliances, 12
PERCENT, TOP, 45
percent (%) wildcard, 71
performance
   aggregates and window functions, 352
   ANSI SQL-89 syntax versus ANSI SQL-92 syntax, 101
   asterisk (*)
      in column names, 41
      in SELECT lists of subqueries, 139
   blocking issues, 303
   DISTINCT in subqueries, 134
dynamic SQL, 359
multiple references in CTEs, 166
row versioning and isolation levels, 316
sp_executesql, 360
stored procedures, 365
table expressions, 160
UPDATE based on joins, 268
WHERE, 32
window functions, 213
permissions
  schema level, 19
views, 169
pessimistic concurrency
  defined, 301
isolation levels, 309
phantom reads, 314
phases
  circumventing unsupported logical phases, 202
defined, 29
physical query processing, 99
PIVOT, 361
pivoting data, 222–228
  native T-SQL PIVOT operator, 225
  standard SQL, 224
plus sign (+) operator, 64
PowerPivot, BISM, 11
precedence
  data types
    defined, 74
    using, 52
  operator precedence rules, 52
  set operators, 200
predicates
  about, 5
LIKE predicate, 71
logic
  about, 4
    two-, three- and four- value logic, 6
SELECT, 50–53
prefixes, columns, 101
previous values, returning, 140
PRIMARY filegroup, 18
primary key constraints, 22
PRINT, 343
procedural data integrity, 22
procedures
  stored procedures, 364
  system stored procedures and functions, 89
programmable objects, 339–374
batches, 341–345
  batches as a unit of resolution, 344
  batches as unit of parsing, 342
GO, 344
  statements that cannot be combined in the same batch, 343
variables, 343
cursors, 348–352
dynamic SQL, 359–362
EXEC, 359
PIVOT, 361
sp_executesql, 360
error handling, 370–374
flow elements, 345–348
IF ... ELSE, 345
WHILE, 346
routines, 362–370
  stored procedures, 364
  triggers, 366–370
UDFs, 362
temporary tables, 353–358
  global temporary tables, 355
  local temporary tables, 353
table types, 357
table variables, 356
variables, 339
properties
  collation property, 16
  defining sets, 4
  identity property, 252–261
propositions, 5
queries. See also single-table queries; subqueries
  logical query processing, 99
  multi-join queries, 109
  multi-join queries using outer joins, 116
  physical query processing, 99
  set operators, 191
query expressions. See table expressions
QUOTED_IDENTIFIER, 64
Q
RAND function, 363
ranges, dates, 79
ranking
  functions, 215
  window functions, 214–217
RDBMSs (relational database management systems),
defined, 1
READ COMMITTED
  about, 301
  isolation level, 311
  using, 303
READCOMMITTEDLOCK, 303
READ COMMITTED SNAPSHOT

about, 301
isolation level, 321
using, 325
READ UNCOMMITTED isolation level, 310
recursive CTEs, 166–168
references
CTEs, 166
derived tables, 162
referenced tables, 23
referencing relations, 7
referential integrity, 23
regular data types, 61
relational database management systems (RDBMSs),
defined, 1
relational model, 4–9
constraints, 6
Edgar F. Codd, 4
missing values, 6
normalization, 7–9
propositions, predicates and relations, 5
SQL, 39
relations
about, 5
referencing relations, 7
variables versus relations, 5
REPEATABLE READ isolation level, 313
REPLACE function, 68
REPLICATE function, 69
resolution, batches as a unit of resolution, 344
resource databases, 16
resource types, lockable, 302
RETURN, 363
returning previous or next values, 140
RIGHT function, 66
ROLLBACK TRAN, 297, 366
rolled backs, temporary tables, 356
ROLLUP subclause, grouping sets, 235
routines, 362–370
stored procedures, 364
triggers, 366–370
DDL triggers, 368
DML triggers, 367
UDFs, 362
ROW_NUMBER function
about, 215
using, 48, 196, 276
rows
constructors, 268
duplicate rows, 39
FIRST_VALUE and LAST_VALUE functions, 219
INTERSECT distinct set operator, 195
phantoms, 314
tuples in set theory, 4
UNION ALL multiset operator, 192
versioning, 316–322
RTRIM function, 70
running aggregates, subqueries, 141

S
sample database, installing, 385
scalar expressions, data types, 51
scalar self-contained subqueries, 135
scalar subqueries, examples, 130
scalar UDFs, 362
scalar variables, 339
SCHEMABINDING option, 174
SCHEMA_NAME function, 88
schema-qualifying object names, 29
schemas
snowflake schema, 11
SQL Server, 18
star schema, 10
SCOPE_IDENTITY function, 254, 280
searching
CASE expressions, 53
SQL Server Books Online, 393
second normal form (2NF), 8
security, stored procedures, 364
SELECT, 27–50, 36–42. See also single-table queries
column aliases, 159
DML, 3
FROM, 29
GROUP BY, 32–35
HAVING, 36
OFFSET-FETCH filter, 47
ORDER BY, 42
SELECT clause, 36–42
TOP filter, 44–47
WHERE, 31
window functions, 48
SELECT INTO, 251
SELECT *, views, 170
self-contained subqueries, 129–135
defined, 129
multivalued subquery examples, 132–135
scalar subquery examples, 129

408 Index
self cross joins, 101
self pairs, non-equi joins, 108
semicolon (;
MERGE, 272
statements, 21, 29
SEQUEL (Structured English QUEry Language), 2
sequences
assignment UPDATE, 269
dates, 113
integers, 102
sequence object, 252–261
SERIALIZABLE isolation level, 314
servers. See SQL Server
SET
UPDATE based on joins, 268
using, 339
SET DEFAULT statement, 24
set diagram, 192
SET NOCOUNT ON, 365
SET NULL statement, 24
set operators, 191–210
circumventing unsupported logical phases, 202
EXCEPT operator, 198
EXCEPT ALL multiset operator, 199
EXCEPT distinct set operator, 198
exercises and solutions, 204–210
INTERSECT operator, 194–197
INTERSECT ALL multiset operator, 195–197
INTERSECT distinct set operator, 195
precedence, 200
UNION operator, 192–194
UNION ALL multiset operator, 192
UNION distinct set operator, 193
set theory, 3
shared lock mode
about, 300
lock compatibility, 301
SharePoint Designer. See Microsoft SharePoint
Designer
SharePoint Workspace. See Microsoft SharePoint
Workspace
short circuits, 60
side effects, UDFs, 363
simple CASE expressions, 53
single quotes (‘), 64
single-table queries, 27–98
all-at-once operations, 59
CASE expressions, 53
character data, 61–73
collation, 62
data types, 61
LIKE predicate, 71
operators and functions, 64–71
date and time data, 73–87
data types, 73
filtering date ranges, 79
functions, 80–87
literals, 74–78
working with date and time separately, 78
exercises and solutions, 91–98
metadata, 88–90
catalog views, 88
information schema views, 89
system stored procedures and functions, 89
NULL, 55–59
predicates and operators, 50–53
SELECT, 27–50
FROM, 29
GROUP BY, 32–35
HAVING, 36
OFFSET-FETCH filter, 47
ORDER BY, 42
SELECT clause, 36–42
TOP filter, 44–47
WHERE, 31
window functions, 48
skipping, OFFSET-FETCH, 47
SMALLDATETIME, 73
SNAPSHOT isolation level, 317–319
snowflake schema, 11
source code, downloading, 385
sp_executesql, 360
sp_helptext, 173
SPID (unique server process ID), 304
spreading phase, pivoting data, 224
sp_sequence_get_range, 260
SQL Azure. See Windows Azure SQL Database
SQL Database. See Windows Azure SQL Database
SQL injection and concatenating user input, 359
SQL Server, 12–19, 376–384
ABC flavors, 12
authenticated logon, 17
creating user accounts, 376
databases, 15, 15–18
installing prerequisites, 377
installing the database engine, documentation
and tools, 377–384
instances of, 14
obtaining SQL Server, 376
schemas and objects, 18
SQL Server Books Online, 393–396
SQL Server Management Studio. See SSMS
SQL (Structured Query Language)
  ANSI SQL-89 syntax
cross joins, 101
inner joins, 105
ANSI SQL-92 syntax
cross joins, 100
inner joins, 103
background, 2
dynamic SQL, 359–362
EXEC, 359
PIVOT, 361
sp_executesql, 360
language independence, 2
logical order of processing query clauses, 28
pivoting data, 224
relational model, 39
unpivoting data, 229–231
SSMS (SQL Server Management Studio)
about, 387–393
loading SQL Server Books Online, 395
star schema, defined, 10
starting values, 257
statements
  semicolon (;), 21
SQL categories, 2
statements that cannot be combined in the same batch, 343
stored procedures, 364
strings, concatenating, 64, 362
Structured Query Language. See SQL
STUFF function, 70
style, coding, 21
subqueries, 129–156
correlated subqueries, 136–139
exercises and solutions, 147–156
limitations of, 212
misbehaving subqueries, 142–147
NULL, 142
substitution errors in subquery column names, 145
returning previous or next values, 140
running aggregates, 141
self-contained subqueries, 129–135
multivalued subquery examples, 132–135
scalar subquery examples, 130
subsets, defining using predicates, 4
substitution errors, subquery column names, 145
SUBSTRING function, 66
surrogate keys, 252
SWITCHOFFSET function, 83
SYSDATETIME function, 25, 80
SYSDATETIMEOFFSET function, 80
sys.dm_exec_connections, 306
sys.dm_exec_sessions, 307
sys.dm_exec_sql_text function, 306
sys.dm_tran_locks view, 306
System R, 2
system stored procedures and functions, 89
SYSUTCDATETIME function, 80
T
table expressions, 157–190
APPLY operator, 178–181
CTEs, 163
  arguments, 165
column aliases, 164
multiple CTEs, 165
multiple references, 166
recursive CTEs, 166–168
data modification, 274–277
derived tables, 157–163
  arguments, 161
column aliases, 159
multiple references, 162
nesting, 161
exercises and solutions, 182–190
inline TVFs, 176
views, 169–176
  options, 172–176
ORDER BY clause, 170
tables. See also derived tables; single-table queries;
temporary tables
columns and prefixes, 101
creating, 20
defined, 5
hints and isolation levels, 310
numbers and cross joins, 102
operators and multi-join queries, 109
order within, 43
referencing and referenced tables, 23
SELECT INTO, 251
temporary tables, 353–358
global temporary tables, 355
local temporary tables, 353
table types, 357
table variables, 356
TRUNCATE, 263
Table-Valued Functions (TVFs), inline TVFs, 176
tempdb
databases, 16
isolation levels based on row versioning, 316
local temporary tables, 353
temporary tables, 353–358
global temporary tables, 355
local temporary tables, 353
table types, 357
table variables, 356
tenants. See also multitenyancy
terminating, blockers, 308
text, obfuscated text, 172
three-valued predicate logic, 6
THROW, 372
ties and tiebreakers, 46
time. See date and time data
TODATETIMEOFFSET function, 83
tools, installing, 377–384
TOP
about, 44–47
circumventing unsupported logical phases, 203
data modification, 277–279
using, 171
transactions, 297–300. See also concurrency
online transactional processing, 10
roll backs and temporary tables, 356
versus batches, 341
triggers, 366–370
  DDL triggers, 368
  DML triggers, 367
troubleshooting
  blocking, 303–309
table expressions, 275
TRUE, 55
TRUNCATE
  about, 263
  DDL, 3
TRY blocks, 371
TRY_CAST function, 81
TRY...CATCH, 370
TRY_CONVERT function, 81
TRY_PARSE function, 81
tuples, set theory, 4
TVFs (Table-Valued Functions), inline TVFs, 176
two-valued predicate logic, 6
types. See also data types
  relations, 6

U
UDFs (user-defined functions), 362
underscore (_') wildcard, 72
Unicode
  data type, 51, 61
  number of bytes, 67
  sp_executesql, 360
UNION ALL
  INSERT SELECT, 249
  unpivoting, 233
UNION operator, 192–194
  UNION ALL multiset operator, 192
  UNION distinct set operator, 193
UNIQUE, 59
unique constraints, defined, 22
unique server process ID (SPID), 304
UNKNOWN
  ELSE, 345
  negating, 143
  NULL, 55, 112
unpivoting data, 228–232
  native T-SQL UNPIVOT operator, 231
  standard SQL, 229–231
UPDATE, 264–270
  about, 265
  assignment UPDATE, 269
  DML triggers, 367
  isolation levels based on row versioning, 316
  lost updates, 314
  OUTPUT, 283
  UPDATE based on joins, 267
UPPER function, 70
user accounts, creating on SQL Server, 376
user-defined functions (UDFs), 362
user input, concatenating, 359
USE statement, 20

V
VALUES
  INSERT VALUES, 248
  UNION ALL operators, 208
values, missing values, 6
VAR, 61
variables
  batches, 343
  programmable objects, 339
  relation variables versus relations, 5
  table variables, 356
vector expressions, 268
VertiPaq, 11
views, 169–176
catalog views, 88
information schema views, 89
options, 172–176
CHECK OPTION, 174
ENCRYPTION, 172
SCHEMABINDING, 174
ORDER BY clause, 170
sys.dm_tran_locks view, 306
Visual Studio. See Microsoft Visual Studio

W
WHEN MATCHED, 273
WHEN MATCHED AND, 274
WHEN NOT MATCHED, 273
WHEN NOT MATCHED BY SOURCE, 273
WHERE
about, 31
DELETE, 262, 263
outer joins, 112, 115
UPDATE based on joins, 267
WHILE, 346
whole, set theory, 3
wildcards, LIKE predicate, 71

window functions, 211–222
about, 48
aggregates, 220, 352
offset window functions, 217–219
ORDER BY, 196
ranking, 214–217
Windows authenticated logon, 17
Windows Azure platform account, 376
Windows Azure SQL Database
about, 13
collation, 63
databases, 20
database triggers, 368
default isolation levels, 301, 310
gine, 13
getting started, 375
global temporary variables, 355
heaps, 251
logical layer, 17
READ COMMITTED, 303
system database master, 16
Windows Live ID, 376
WITH NOCHECK option, 24
WITH statement, CTEs, 163
WITH TIES, 152

Y
YEAR function, 85
About the Author

ITZIK BEN-GAN is a mentor with and co-founder of SolidQ. A SQL Server Microsoft MVP since 1999, Itzik has taught numerous training events around the world focused on T-SQL querying, query tuning, and programming. Itzik is the author of several books about T-SQL. He has written many articles for SQL Server Pro as well as articles and white papers for MSDN and The SolidQ Journal. Itzik’s speaking engagements include Tech-Ed, SQL PASS, SQL Server Connections, presentations to various SQL Server user groups, and SolidQ events. Itzik is a subject-matter expert within SolidQ for its T-SQL related activities. He authored SolidQ’s Advanced T-SQL and T-SQL Fundamentals courses and delivers them regularly worldwide.