Read Me First!

Congratulations on selecting CHOICE courseware! This quick guide will give you access to key instructor resources so you can present the best possible CHOICE learning experience.

The CHOICE Instructor’s Edition
This Instructor’s Edition is a crucial tool that provides you with all the course-specific technical and setup information, delivery notes, and instructional material that you need as a training professional to deliver an excellent learning experience to your students.

The CHOICE Facilitator’s Guide
Before presenting this or any CHOICE course, make sure you explore the CHOICE Facilitator’s Guide for critical information about virtual and blended course delivery techniques and the CHOICE instructional philosophy and learning experience. Get access to the CHOICE Facilitator’s Guide through the CHOICE Facilitator’s Course.

The CHOICE Facilitator’s Course
You can find the CHOICE Facilitator’s Guide and other great resources for delivering Logical Operations courseware products through the free CHOICE Facilitator’s Course. To access the free Facilitator’s Course, simply:

1. Visit www.lo-choice.com
2. Enter access key LCSYB24YEF
3. Download and explore the CHOICE Facilitator’s Guide!

The Logical Operations Instructor Community
Logical Operations leads a very active online community and resource center where instructors from all over the world share their ideas, interact, and engage with each other. Join the community from the CHOICE Facilitator’s Course, or search www.linkedin.com for the “Logical Operations Instructor Community” group to join the conversation. Again, congratulations on your choice—the right choice—the Logical CHOICE!
SQL Querying: Fundamentals (Second Edition)
Acknowledgements

PROJECT TEAM

<table>
<thead>
<tr>
<th>Authors</th>
<th>Media Designer</th>
<th>Content Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rozanne Whalen</td>
<td>Brian J. Sullivan</td>
<td>Joe McElveney</td>
</tr>
<tr>
<td>Brian S. Wilson</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Logical Operations wishes to thank the Logical Operations Instructor Community, and in particular Sally Howard and Gary Leenhouts, for their instructional and technical expertise during the creation of this course.

Notices

DISCLAIMER

While Logical Operations, Inc. takes care to ensure the accuracy and quality of these materials, we cannot guarantee their accuracy, and all materials are provided without any warranty whatsoever, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose. The name used in the data files for this course is that of a fictitious company. Any resemblance to current or future companies is purely coincidental. We do not believe we have used anyone's name in creating this course, but if we have, please notify us and we will change the name in the next revision of the course. Logical Operations is an independent provider of integrated training solutions for individuals, businesses, educational institutions, and government agencies. The use of screenshots, photographs of another entity's products, or another entity's product name or service in this book is for editorial purposes only. No such use should be construed to imply sponsorship or endorsement of the book by nor any affiliation of such entity with Logical Operations. This courseware may contain links to sites on the Internet that are owned and operated by third parties (the "External Sites"). Logical Operations is not responsible for the availability of, or the content located on or through, any External Site. Please contact Logical Operations if you have any concerns regarding such links or External Sites.

TRADEMARK NOTICES

Logical Operations and the Logical Operations logo are trademarks of Logical Operations, Inc. and its affiliates.

Microsoft® SQL Server® 2017 and Microsoft® SQL Server® Management Studio 2017 are registered trademarks of Microsoft Corporation in the U.S. and other countries; the SQL Server 2017 products and services discussed or described may be trademarks of Microsoft Corporation.

All other product names and services used throughout this course may be common law or registered trademarks of their respective proprietors.

Copyright © 2017 Logical Operations, Inc. All rights reserved. Screenshots used for illustrative purposes are the property of the software proprietor. This publication, or any part thereof, may not be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, storage in an information retrieval system, or otherwise, without express written permission of Logical Operations, 3535 Winton Place, Rochester, NY 14623, 1-800-456-4677 in the United States and Canada, 1-585-350-7000 in all other countries. Logical Operations’ World Wide Web site is located at www.logicaloperations.com.

This book conveys no rights in the software or other products about which it was written; all use or licensing of such software or other products is the responsibility of the user according to terms and conditions of the owner. Do not make illegal copies of books.
# SQL Querying: Fundamentals (Second Edition)

## Lesson 1: Executing a Simple Query

- **Topic A: Connect to the SQL Database**
  - Page 2
- **Topic B: Query a Database**
  - Page 14
- **Topic C: Save a Query**
  - Page 23
- **Topic D: Modify and Execute a Saved Query**
  - Page 25

## Lesson 2: Performing a Conditional Search

- **Topic A: Search Using One or More Conditions**
  - Page 36
- **Topic B: Search for a Range of Values and NULL Values**
  - Page 56
- **Topic C: Search Data Based on String Patterns**
  - Page 63

## Lesson 3: Working with Functions

- **Topic A: Perform Date Calculations**
  - Page 72
- **Topic B: Calculate Data Using Aggregate Functions**
  - Page 82
- **Topic C: Manipulate String Values**
  - Page 88

## Lesson 4: Organizing Data

- **Topic A: Sort Data**
  - Page 100
Welcome to the Instructor
Welcome and congratulations on your choice to use the finest materials available on the market today for expert-facilitated learning in any presentation modality. You can utilize the SQL Querying: Fundamentals (Second Edition) curriculum to present world-class instructional experiences whether:

- Your students are participating with you in the classroom or virtually.
- You are presenting in a continuous event or in an extended teaching plan, such as an academic semester.
- Your presentation takes place synchronously with the students or asynchronously.
- Your students have physical courseware or are using digital materials.
- You have any combination of these instructional dimensions.

To make the best use of the SQL Querying: Fundamentals (Second Edition) materials in any or all of these dimensions, be sure to review the contents of the CHOICE Facilitator’s Guide for an orientation to all of the components of the CHOICE experience.

Preparing to Present the CHOICE Experience
Effectively presenting the information and skills in this course requires adequate preparation in any presentation modality. As such, as an instructor, you should familiarize yourself with the content of the entire course, including its organization and instructional approaches. You should review each of the student activities, exercises, and Mastery Builders so you can facilitate them during the learning event. Also, make sure you review the tips for presenting in the different dimensions; these instructor tips are available as notes in the margins of your Instructor’s Edition.

In addition to the curriculum itself, Microsoft® PowerPoint® slides, data files, and other course-specific support material may be available by downloading the files from the CHOICE Course screen. Be sure to obtain the course files prior to your learning event and make sure you distribute them to your students.
Course Facilitator Icons

Throughout the Instructor’s Edition, you may see various instructor-focused icons that provide suggestions, answers to problems, and supplemental information for you, the instructor.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🎨</td>
<td>A <strong>display slide</strong> note provides a prompt to the instructor to display a specific slide from the provided PowerPoint files.</td>
</tr>
<tr>
<td>📣</td>
<td><strong>Content delivery tips</strong> provide guidance for specific delivery techniques you may want to utilize at particular points in the course, such as lectures, whiteboard sketching, or performing your own demonstrations for the class.</td>
</tr>
<tr>
<td>🛑</td>
<td><strong>Managing learning interactions</strong> provide suggested places to interact with the class as a whole. You might poll the class with closed-ended questions, check comprehension with open-ended questions, conduct planned discussion activities, or take notes and questions from the group to “park” and address at a later point in the class.</td>
</tr>
<tr>
<td>🕵️‍♂️</td>
<td><strong>Monitoring learner progress</strong> notes suggest when you might want to monitor individual students as they perform activities, or have private sidebar conversations with specific individual participants.</td>
</tr>
<tr>
<td>🎓</td>
<td><strong>Engaging learners</strong> notes suggest opportunities to involve the students in active ways with the course presentation, such as enabling them to demonstrate their work to the class as a whole, or checking in on the logistics of the presentation.</td>
</tr>
<tr>
<td>🔗</td>
<td><strong>Incorporating other assets</strong> notes suggest when and how to include other types of media, such as visiting CHOICE social media sites, accessing specific web resources, or utilizing media assets provided with the course, such as Logical Operations’ Spotlights.</td>
</tr>
</tbody>
</table>

Digital Software Updates

The software vendor may at any time deploy software updates digitally, resulting in changes that may not be reflected dynamically in this course. Stay up to date with product updates and be ready to adapt the material to any changes in the user interface.

Course-Specific Technical Requirements

Hardware

For this course, you will need one computer for each student and one for the instructor. Each computer will need the following minimum hardware configurations:

- 2 GHz or faster 64-bit (x64) processor
- 4 gigabytes (GB) RAM
- 40 GB available hard disk space
- CD-ROM drive (if installing any software from a CD-ROM)
- Keyboard and mouse (or other pointing device)
- 1024 x 768 (or higher) resolution monitor recommended
- Network cards and cabling for local network access
- Internet access (contact your local network administrator)
- Printer (optional) or an installed printer driver
- Projection system to display the instructor's computer screen
Software

To prepare a student or instructor system for the class, install the following software according to the instructions provided. You will need the following software for each student and instructor computer:

- Microsoft® Windows® 10 64-bit with sufficient licenses.
  Windows 10 requires activation unless you have volume-licensing agreements. There is a grace period for activation. If the duration of your class will exceed the activation grace period (for example, if you are teaching the class over the course of an academic semester), you should activate the installations at some point before the grace period expires. Otherwise, the operating system may stop working before the class ends.
- Microsoft® SQL Server® 2017 Express. (When this course was written, the installation file for this free software was available for download from [https://www.microsoft.com/en-us/sql-server/sql-server-downloads](https://www.microsoft.com/en-us/sql-server/sql-server-downloads).)
- Microsoft® SQL Server® Management Studio (SSMS). (When this course was written, the installation file for this free software was available for download from [https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms](https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms).)

Setting Up the Course

For each student and the instructor:

1. Prepare a Windows 10 computer.
   a. Install Microsoft Windows 10 with a local administrator account. You will need to provide the student or instructor using the computer with a local administrator account, so create a separate user account for that purpose if you do not want to provide them with the main administrator account on the computer.
   b. Download and install any available Windows updates.
   c. Restart the computer and log in using the local administrator account.

2. Change how Windows updates are delivered to avoid Windows updates during the class.
   a. Select the Start button.
   b. Select Settings.
   c. Select Update & Security.
   d. In the Update settings group, select Advanced options.
   e. Select the Delivery Optimization link.
   f. Select the option to Allow downloads from other PCs.
   g. Close the Settings app.

3. Ensure Windows 10 displays file extensions and hidden items.
   a. Open File Explorer.
   b. On the ribbon, select the View tab.
   c. On the ribbon, in the Show/hide group, check the File name extensions and Hidden items check boxes.
   d. Close File Explorer.

4. Install SQL Server 2017 Express.
   a. Download SQL Server 2017 Express from the Microsoft website.
   b. Right-click the installation file for SQL Server 2017 Express that you just downloaded from the Microsoft website, and select Run as administrator.
   c. If you are prompted by the User Account Control message box to allow the installation of SQL Server 2017 Express Edition, select Yes.
   d. In the SQL Server 2017 Express Edition dialog box, select the Basic installation option.
e. Select Accept to accept the license terms.
f. When prompted for the install location, select Install to use the default location.
g. When the installer shows the message Installation has completed successfully!, select Close, and select Yes to confirm.
h. Restart the computer, and log back in to Windows with the local administrator account.

5. Install SQL Server Management Studio.
   a. Download Microsoft SQL Server Management Studio (SSMS) from the Microsoft website.
   b. Right-click the installation file for SQL Server Management Studio that you just downloaded from the Microsoft website, and select Run as administrator.
   c. If you are prompted by the User Account Control message box to allow the installation of SQL Server Management Studio, select Yes.
   d. At the Welcome page, select Install.
   e. When installation completes, select Close.

6. Install the course data files.
   a. From the course Files tile on CHOICE, download the 094018Files<ver>.zip file to any location that is convenient for you.
      
      Note: The default download folder will vary depending on your browser version and local settings.

   b. Go to the download folder and locate and unzip (extract) the 094018Files<ver>.zip file. This will create a local 094018Files<ver> folder. It contains separate subfolders with your course overheads and the course data files.
   c. Open the 094018Files<ver> folder.
   d. Move or copy the 094018Data folder to the root of your C drive. This installs the data files to the location specified in the course activities.
      
      Note: If you prefer to place the data files in a different location, simply inform your students that they will be accessing the files from that alternate path.

   The data files folder contains all the lesson-specific subfolders and data files you will need to run this course. There is a separate folder with the starter files for each lesson or Mastery Builder, and there may be a Solutions folder with completed files students can use to check their results.

7. Connect to the database server in SQL Server Management Studio.
   a. Use the student login account to log in to Windows.
   b. Show the Windows Start menu. Locate the SQL Server Management Studio tile. Right-click the tile, select More, and select Run as administrator.
   c. In the Connect to Server dialog box, in the Server type drop-down list, verify that Database Engine is selected.
   d. In the Server name drop-down list, select the down arrow, and select <Browse for more...>.
   e. Before Database Engine, select + to expand the list, showing available database engines.
   f. Select the database engine identified by the computer's name followed by SQLEXPRESS, and select OK.
   g. In the Authentication drop-down list, make sure that Windows Authentication is selected.
   h. Select Connect to connect to the server.

8. Attach the Pub1 database provided in the course setup files.
   a. In the Object Explorer pane (left side of the application), make sure the top icon is expanded so you can see the Databases folder. Right-click the Databases folder, and select Attach.
   b. Select Add.
c. In the left pane, select the \C:\094018Data folder. You may need to scroll and expand folders in order to see this folder.

d. In the right pane, select Pub1.mdf.

e. Select OK to return to the Attach Databases dialog box.

f. Select OK to return to the main window of the SQL Server Management Studio.

g. In the Object Explorer pane, expand the Databases folder and verify that the Pub1 database is shown.

Note: This course is often delivered as the first day in a two-day series, along with SQL Querying: Advanced (Second Edition). The software setup for these two courses is identical. If you plan to deliver these courses back-to-back, at this point you should install the course data files for SQL Querying: Advanced (Second Edition), using the steps shown in Step 8 to attach the Pub2 database from the \C:\094019Data folder. If you do this, then no additional setup steps will be required for the SQL Querying: Advanced (Second Edition) (094019) course.


Presentation Planners

The lesson durations given in the course content are estimates based on a typical class experience. The following planners show examples of how the content could be presented in either a continuous flow or separately across a multi-session seminar series. Your presentation flow may vary based on a number of factors, including the size of the class, whether students are in specialized job roles, whether you plan to incorporate Spotlights or other assets from the CHOICE Course screen into the course, and so on. Use the samples and blank planners to determine how you will conduct the class to meet the needs of your own situation.

Continuous Presentation: Model Class Flow

This planner provides a sample presentation flow based on one 8-hour day of training with breaks and lunch factored in.

<table>
<thead>
<tr>
<th>Section</th>
<th>Duration</th>
<th>Day Planner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Welcome and Introductions</td>
<td>0:30</td>
<td>8:30 - 9:00</td>
</tr>
<tr>
<td>Lesson 1: Executing a Simple Query</td>
<td>0:45</td>
<td>9:00 - 9:45</td>
</tr>
<tr>
<td>Lesson 2: Performing a Conditional Search</td>
<td>0:30</td>
<td>9:45 - 10:15</td>
</tr>
<tr>
<td>BREAK</td>
<td>0:15</td>
<td>10:15 - 10:30</td>
</tr>
<tr>
<td>Lesson 2: Performing a Conditional Search (Cont.)</td>
<td>1:45</td>
<td>10:30 - 11:45</td>
</tr>
<tr>
<td>Lesson 3: Working with Functions</td>
<td>0:30</td>
<td>11:45 - 12:15</td>
</tr>
<tr>
<td>LUNCH</td>
<td>0:30</td>
<td>12:15 - 12:45</td>
</tr>
<tr>
<td>Lesson 3: Working with Functions (Cont.)</td>
<td>0:30</td>
<td>12:45 - 1:15</td>
</tr>
<tr>
<td>Lesson 4: Organizing Data</td>
<td>1:30</td>
<td>1:15 - 2:45</td>
</tr>
<tr>
<td>BREAK</td>
<td>0:15</td>
<td>2:45 - 3:00</td>
</tr>
<tr>
<td>Lesson 5: Retrieving Data from Multiple Tables</td>
<td>1:30</td>
<td>3:00 - 4:30</td>
</tr>
<tr>
<td>Lesson 6: Exporting Query Results</td>
<td>0:30</td>
<td>4:30 - 5:00</td>
</tr>
</tbody>
</table>

Continuous Presentation: Your Class Flow
Use this planner to plan the flow of your own training day based on the needs of your students, the schedule for your own day, and/or any other modifications you choose.

<table>
<thead>
<tr>
<th>Section</th>
<th>Duration</th>
<th>Day Planner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Welcome and Introductions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lesson 1: Executing a Simple Query</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lesson 2: Performing a Conditional Search</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lesson 3: Working with Functions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lesson 4: Organizing Data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lesson 5: Retrieving Data from Multiple Tables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lesson 6: Exporting Query Results</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Non-continuous Presentation: Model Class Flow**

This planner provides a sample presentation flow based on separate sessions presented over multiple days or weeks.

<table>
<thead>
<tr>
<th>Session Number</th>
<th>Material Covered</th>
<th>Session Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>Welcome and Introductions</td>
<td>1:15</td>
</tr>
<tr>
<td></td>
<td>Lesson 1: Executing a Simple Query</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lesson 2: Performing a Conditional Search</td>
<td>1:45</td>
</tr>
<tr>
<td>Two</td>
<td>Lesson 3: Working with Functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lesson 4: Organizing Data</td>
<td>2:30</td>
</tr>
<tr>
<td>Three</td>
<td>Lesson 5: Retrieving Data from Multiple Tables</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lesson 6: Exporting Query Results</td>
<td>2:00</td>
</tr>
</tbody>
</table>

**Non-continuous Presentation: Your Class Flow**
Use this planner to plan how you will present the course content based on the needs of your students, your conventions for the number and length of sessions, and any other modifications you choose.

<table>
<thead>
<tr>
<th>Session Number</th>
<th>Material Covered</th>
<th>Session Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
About This Course

Organizations typically store their most critical information—the information used to manage day-to-day operations—within a database. The ability to retrieve and analyze this information is essential to the functioning of the organization. Structured Query Language (SQL) is the primary language used to accomplish such tasks. Essentially, SQL is the language you use to interact with a database.

The ability to write SQL is an essential job skill for those who need to manage large volumes of data, produce reports, mine data, or combine data from multiple sources. Even if someone else on your team creates reports for you, having a fundamental understanding of SQL querying will help you ask the right questions and know what you're looking for in your data analysis tools.

This course, *SQL Querying: Fundamentals (Second Edition)*, will teach you to use SQL as a tool to retrieve the information you need from databases.

Course Description

Target Student

This course is intended for individuals with basic computer skills, familiar with concepts related to database structure and terminology, who want to use SQL to query databases.

Course Prerequisites

Basic end-user computer skills and some familiarity with database terminology and structure are required. Completion of one of the following Logical Operations courses or equivalent knowledge and skill is highly recommended:

- Using Microsoft® Windows® 10
- Microsoft® Windows® 10: Transition from Windows® 7

Course Objectives

In this course, you will compose SQL queries to retrieve desired information from a database.

You will:

- Connect to the SQL Server database and execute a simple query.
- Include a search condition in a simple query.
- Use various functions to perform calculations on data.
- Organize the data obtained from a query before it is displayed onscreen.
- Retrieve data from multiple tables.
- Export the results of a query.
The CHOICE Home Screen

Logon and access information for your CHOICE environment will be provided with your class experience. The CHOICE platform is your entry point to the CHOICE learning experience, of which this course manual is only one part.

On the CHOICE Home screen, you can access the CHOICE Course screens for your specific courses. Visit the CHOICE Course screen both during and after class to make use of the world of support and instructional resources that make up the CHOICE experience.

Each CHOICE Course screen will give you access to the following resources:

- **Classroom**: A link to your training provider's classroom environment.
- **eBook**: An interactive electronic version of the printed book for your course.
- **Files**: Any course files available to download.
- **Checklists**: Step-by-step procedures and general guidelines you can use as a reference during and after class.
- **Spotlights**: Brief animated videos that enhance and extend the classroom learning experience.
- **Assessment**: A course assessment for your self-assessment of the course content.
- **Social media resources** that enable you to collaborate with others in the learning community using professional communications sites such as LinkedIn or microblogging tools such as Twitter.

Depending on the nature of your course and the components chosen by your learning provider, the CHOICE Course screen may also include access to elements such as:

- LogicalLABS, a virtual technical environment for your course.
- Various partner resources related to the courseware.
- Related certifications or credentials.
- A link to your training provider's website.
- Notices from the CHOICE administrator.
- Newsletters and other communications from your learning provider.
- Mentoring services.

Visit your CHOICE Home screen often to connect, communicate, and extend your learning experience!

How to Use This Book

As You Learn

This book is divided into lessons and topics, covering a subject or a set of related subjects. In most cases, lessons are arranged in order of increasing proficiency.

The results-oriented topics include relevant and supporting information you need to master the content. Each topic has various types of activities designed to enable you to solidify your understanding of the informational material presented in the course. Information is provided for reference and reflection to facilitate understanding and practice.

Data files for various activities as well as other supporting files for the course are available by download from the CHOICE Course screen. In addition to sample data for the course exercises, the course files may contain media components to enhance your learning and additional reference materials for use both during and after the course.

Checklists of procedures and guidelines can be used during class and as after-class references when you’re back on the job and need to refresh your understanding.

At the back of the book, you will find a glossary of the definitions of the terms and concepts used throughout the course. You will also find an index to assist in locating information within the instructional components of the book. In many electronic versions of the book, you can click links on key words in the content to move to the associated glossary definition, and on page references in
the index to move to that term in the content. To return to the previous location in the document after clicking a link, use the appropriate functionality in your PDF viewing software.

As You Review
Any method of instruction is only as effective as the time and effort you, the student, are willing to invest in it. In addition, some of the information that you learn in class may not be important to you immediately, but it may become important later. For this reason, we encourage you to spend some time reviewing the content of the course after your time in the classroom.

As a Reference
The organization and layout of this book make it an easy-to-use resource for future reference. Taking advantage of the glossary, index, and table of contents, you can use this book as a first source of definitions, background information, and summaries.

Course Icons
Watch throughout the material for the following visual cues.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>📝</td>
<td>A Note provides additional information, guidance, or hints about a topic or task.</td>
</tr>
<tr>
<td>🚨</td>
<td>A Caution note makes you aware of places where you need to be particularly careful with your actions, settings, or decisions so that you can be sure to get the desired results of an activity or task.</td>
</tr>
<tr>
<td>🔍</td>
<td>Spotlight notes show you where an associated Spotlight is particularly relevant to the content. Access Spotlights from your CHOICE Course screen.</td>
</tr>
<tr>
<td>📚</td>
<td>Checklists provide job aids you can use after class as a reference to perform skills back on the job. Access checklists from your CHOICE Course screen.</td>
</tr>
<tr>
<td>🌐</td>
<td>Social notes remind you to check your CHOICE Course screen for opportunities to interact with the CHOICE community using social media.</td>
</tr>
</tbody>
</table>
Executing a Simple Query

Lesson Time: 45 minutes

Lesson Introduction

In this course, you will query an SQL database using fundamental query techniques. Simple queries form the basis of building your querying skills. In this lesson, you will begin by composing and executing a simple SQL statement to retrieve information from a database. You will then modify and save a query so that you can use it later.

Lesson Objectives

In this lesson, you will:
• Connect to the database using SQL Server Management Studio.
• Query the database.
• Save a query for future use.
• Modify an existing query.
TOPIC A

Connect to the SQL Database

In this lesson, you will execute a simple query to retrieve information. To do so, you must first connect to the server that contains the database. In this topic, you will access the database that contains the appropriate information.

Databases

A typical database can be a vast collection of information. Organizations use databases to store large amounts of data such as inventory, sales, and customer information. For example, a publishing company might have a database containing information about books, customers, orders placed, representatives, and sales.

For optimal performance, organizations typically store these databases on servers located either on the corporate network or even stored in the cloud. When you need to retrieve information from the databases, you do so by first connecting to the server and then using a querying tool to select the data you want to view.

Figure 1–1: Objects in a database.

Information within a database is organized within various objects. An example of a database object is a table. You use tables to store the information within a database. You can search, retrieve, manipulate, and delete the information in tables.

<table>
<thead>
<tr>
<th>Book Code</th>
<th>Book Title</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>A001</td>
<td>Long Journey</td>
<td>19.95</td>
<td>A</td>
</tr>
<tr>
<td>A002</td>
<td>Manager’s Manual</td>
<td>20.5</td>
<td>D</td>
</tr>
<tr>
<td>A003</td>
<td>History of the Greek Isles</td>
<td>33.95</td>
<td>D</td>
</tr>
</tbody>
</table>
Some databases are created by default when you install a database application such as Microsoft SQL Server 2017, and you can create and customize other databases to suit your business needs.

### Tables

One type of object found in a database is a table. A table is a collection of related information arranged in rows and columns. Information about each item in the collection is displayed as a row. Columns contain the same category of information for every item in the table. A table has a header row that identifies the category of information stored in each column.

You use tables to store the information contained in the database. For example, in a database for a publishing company, you might find a table that contains information about the books the company has published. Each of the rows in the table represents a title that the company publishes. The columns in the table contain the information the company wants to track for each book: the book's part number, title, development cost, author, and so on.
Client/Server Architecture for SQL Databases

SQL database systems are often deployed in a client/server architecture. This means that the services for storing, managing, and presenting data are distributed among two or more computers. As the name implies, there are two tiers in the client/server architecture:

- The **client** tier typically presents data and commands in a user-friendly format that is meant for humans—through a desktop computer application or a mobile app, for example. End users interact with the client tier.
- The **server** tier works directly with the database, managing data on behalf of the many clients that share access to the database. The server tier is managed by system operators/administrators. End users do not interact directly with it.

The following diagram and table describe various components involved in a client/server architecture.

---

**Figure 1-3: A table displaying information about books.**

---

**Client/Server Architecture for SQL Databases**

SQL database systems are often deployed in a client/server architecture. This means that the services for storing, managing, and presenting data are distributed among two or more computers. As the name implies, there are two tiers in the client/server architecture:

- The **client** tier typically presents data and commands in a user-friendly format that is meant for humans—through a desktop computer application or a mobile app, for example. End users interact with the client tier.
- The **server** tier works directly with the database, managing data on behalf of the many clients that share access to the database. The server tier is managed by system operators/administrators. End users do not interact directly with it.

The following diagram and table describe various components involved in a client/server architecture.
Figure 1–4: A database server connected to three clients.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database server</td>
<td>Databases are typically hosted on a server. A server is a computer that provides one or more services to other computers on a network. Servers are high-performance computers designed to provide services to multiple users simultaneously. This server may be located at your company or in the cloud.</td>
</tr>
<tr>
<td>Database application</td>
<td>A database application such as Microsoft SQL Server 2017 runs on the database server to manage communication between the server and clients.</td>
</tr>
<tr>
<td>Network</td>
<td>The server and clients communicate data and commands across a network connection. The client may be located on the same network as the server, or it may connect to the server from a remote location, such as across the Internet.</td>
</tr>
<tr>
<td>Client</td>
<td>End users get their database work done indirectly through a client, which is typically a computer or mobile device (such as a smartphone or tablet) running an operating system such as Microsoft® Windows®, Android, iOS, MacOS, or Linux.</td>
</tr>
<tr>
<td>Client application</td>
<td>A software application running on the client provides a user interface through which users can enter, view, and modify data, typically through a user-friendly interface that presents data in tables or entry forms, with buttons, text boxes, and other controls. In the background, the client application uses a database language (such as SQL) to communicate with the database server, translating the end user's requests into instructions that the database server can understand.</td>
</tr>
</tbody>
</table>
Item | Description
--- | ---
User | A client application may provide different levels of access to different types of users. For example, in a bookstore's database, content administrators may have more control over content (such as book prices and descriptions) than salespeople do. And salespeople may have access to book data that end users viewing book information on a website don't have (such as the cost to produce and print a book).

Other Architectures for SQL Databases

There are other architecture models besides the two-tiered client/server architecture. For example, an application might use an additional tier that processes the business logic of the application separate from the database server and the client. Yet another tier of software and hardware may prepare the presentation of the application (the user interface, essentially).

Web and database servers in a three-tier architecture.

Web application servers obtain dynamic data content from the database server and add static web content, such as HTML code, graphics, and other documents to present the data in an attractive format, and provide a user interface that users can experience through a web browser.

In some organizations, rather than installing database servers directly on the organization's network, administrators may choose to deploy database servers that are hosted by third-party providers on
the Internet. The term "cloud-based computing" refers to the act of connecting to and using servers that are hosted on the Internet ("in the cloud") instead of on a local network.

One reason network administrators elect to use cloud-based database servers is to minimize the cost of installing and maintaining the servers themselves. Hosting the servers on the Internet also makes them accessible to clients located anywhere in the world. Another reason is that services like Amazon Web Services and Google Cloud Platform can provide you with access to massive amounts of storage and processing power in their data centers, enabling you to process huge data sets that might be impractical to process using your local servers.

SQL

Structured Query Language (SQL) is the language that a client application uses to communicate with an SQL database. SQL consists of commands that create and manage databases and tables, as well as retrieve, delete, and modify information in those tables. It is a standard language that works with a wide variety of different database applications, so it has been among the most desirable of IT skills for years, and will continue to be in demand for data analysts, web developers, application developers, and others.

![Figure 1-6: A typical SQL exchange between a client and server.]

Note: You may hear SQL pronounced as "ess queue ell" or "sequel," depending on whom you talk to. Both pronunciations are in common use. The original language was named SEQUEL (Structured English Query Language) but had to be changed to SQL due to a trademark conflict. When the language was standardized by the American National Standards Institute (ANSI) in 1986, the official pronunciation was identified as "ess queue ell." However, in some development circles, the original "sequel" pronunciation is preferred. In general, this course assumes the "S-Q-L" pronunciation (as in "an SQL database"), and "sequel" when referring specifically to the Microsoft SQL Server product (as in "a SQL Server implementation").

SQL Command Groups

SQL is made up of three major command groups, described in the following table.

<table>
<thead>
<tr>
<th>Command Group</th>
<th>Description of This Command Group</th>
<th>Example Commands</th>
</tr>
</thead>
</table>
| Data Manipulation Language (DML) | • Views, changes, and manipulates data within a table  
• Includes commands to select, update, and insert data in a table, and delete data from a table  
• Typically used by data analysts, report authors, and programmers writing client applications | SELECT  
UPDATE  
INSERT  
DELETE |
Command Group  |  Description of This Command Group  |  Example Commands
--- | --- | ---
Data Definition Language (DDL)  |  • Creates and defines the database and objects within it  
• Includes commands to create and delete tables  
• Typically used by database administrators and programmers  |  CREATE TABLE  
DROP TABLE  
CREATE VIEW  
DROP VIEW  

Data Control Language (DCL)  |  • Controls access to the data in a database  
• Includes commands to grant and revoke database privileges  
• Typically used by database administrators and programmers  |  REVOKE  
GRANT  

### Ways SQL Is Used

Depending on your role, there are various ways you might use SQL. For example, many organizations collect data such as sales or manufacturing data, market research, logistics, and so forth. Data analysts transform this raw data into **information** that organizations can use to make informed decisions. This involves sorting, filtering, and formatting database content to produce reports or data sets that are ready for further processing in other software tools.

![Figure 1-7: Direct interaction with a database server.](image)

A data analyst might work with a database using tools such as Microsoft SQL Server Management Studio, typing **SQL statements** directly to extract information from a database, or saving a series of SQL statements in a script that can be run repeatedly (at the end of every month, for example).

Some organizations might have a separate database administrator role—a role responsible for setting up and managing databases. Like the data analyst, a database administrator might work in a tool such as Microsoft SQL Server Management Studio, entering SQL statements directly to create and manage databases and tables.

---

**Direct Interaction**

Point out that this is how students will interact with SQL in this course, entering interactive SQL commands in the SQL Server Management Studio.
End users of client applications don’t have to manually enter SQL statements, as the client application constructs SQL statements on their behalf. The desktop, mobile, or web application developers who create client applications may work with SQL a bit differently than data analysts or administrators, creating applications in a programming language such as Java, C#, VB.NET, C++, PHP, or Python, and using that language to construct SQL statements dynamically. For example, a client application might use data entered by the user into a search text field in a client application as the basis for creating an SQL statement that it sends to the database server. When the results are returned to the application from the database server, the application displays them.

The Query Editor Window

One of the tools you can use to access a Microsoft SQL Server database is Microsoft SQL Server Management Studio (SSMS). Within SSMS, you’ll primarily use the Query Editor window to execute queries. The Query Editor window consists of two panes. The top pane is the editor pane, where you enter SQL statements. The bottom section contains the Results pane, which displays the results of queries, and the Messages tab, which displays information about the query that you execute.
Figure 1–9: The Query Editor window consisting of a statement and results.

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Connect to a Database.
ACTIVITY 1–1
Connecting to a Database

Before You Begin
As part of course setup, Microsoft SQL Server 2017 and SQL Server Management Studio have been installed, and the Pub1 database (located in C:\094018Data\Pub1.mdf) has been connected to the server.

Scenario
Fuller & Ackerman is a publishing company. They store organizational data in a SQL Server 2017 database called Pub1. This database contains information about customers, books published by the company, obsolete titles, sales transactions, and details of sales representatives working for the company.

In this activity, you'll prepare to start working with the database in SQL by doing the following:
• Connecting to the server
• Making sure the Pub1 database is attached to the server
• Entering the `USE` command to identify which database your SQL commands apply to

For information on tables and column names in the Pub1 database, refer to the table structure in Appendix A.

Note: Activities may vary slightly if the software vendor has issued digital updates. Your instructor will notify you of any changes.

1. Launch SQL Server Management Studio and connect to the server.
   a) On the Start screen, select Microsoft SQL Server Management Studio.

Activity: Connecting to a Database

Notify students of any changes to activities based on digital software updates issued by the software vendor.
b) In the Connect to Server dialog box, select Connect to connect to the server.

The database server has already been set up for you as part of the course setup.

2. Verify that the Pub1 database is attached to the server.
   a) In Object Explorer, before the Databases folder icon, select +.
   b) Before the System Databases folder icon, select +.

Various sample databases were included in the SQL Server installation. The database you'll be using for the course, Pub1, has also been attached to the server as part of the course setup.

3. Establish the context for SQL commands.
   a) On the Standard toolbar, select New Query to open the Query Editor window.
   b) In the editor pane, type:

```
USE Pub1
```
c) On the SQL Editor toolbar, select **Execute**.

![SQL Editor toolbar with 'Execute' selected]

Since multiple databases are available, the system needs to know which database your SQL statements apply to. The **USE** command identifies the target of your SQL statements.

Point out that students should not be concerned if the titles in their tabs and windows differ from those shown in the screen images.

d) Notice that Pub1 is now selected in the **Available Databases** drop-down list.

![SQL Editor with 'Pub1' selected in Available Databases]

You have successfully logged in to SQL Server and selected the Pub1 database as the context for subsequent SQL statements that you will add to the script.
TOPIC B

Query a Database

When you are connected to a database, you can access the information stored in its tables. One way you access table information is by viewing it. The command you use to view information is the `SELECT` command. In this topic, you will retrieve information from a database by using `SELECT` statements.

Queries

A query is an SQL statement that requests information from tables present in a database. When the database server receives a query, it determines which table or tables contain the requested data, and which columns and rows should be returned, based on the data requested in the query.

The SELECT Command

You construct a query in SQL by using the `SELECT` command. The order of column names you include in the `SELECT` statement determines which columns are shown in the output, and the order in which they will appear. If you specify more than one column name, you must use commas to separate the column names.

Figure 1-10: A query statement formed from the SQL SELECT command.

Query Processing

SQL was designed so that it would read somewhat like a sentence in English. As with English, there is a certain order in which you must type the various parts of an SQL statement.
However, it is helpful to understand that the order in which you type the parts of an SQL statement is not necessarily the order in which a database server processes them. For example, consider how the simple `SELECT` statement shown in this figure is processed.

![Query processing diagram](image)

**Figure 1-11: How a query is processed by the database server.**

If you think about it, this order of operations is logical. The server begins with the full set of data in the table specified in the `FROM` part of the statement. It then reduces that dataset to only those items matching the `WHERE` part of the statement. Finally, it returns only the columns that were requested in the `SELECT` part of the statement.

**Syntax**

The rules for how you are expected to put together an SQL statement are called the *syntax*. Syntax is the expected form of a command, including its various parts (called clauses) and placeholders for actual elements that you use with the command.

![Syntax illustration](image)

**Figure 1-12: Syntax illustration.**

Clauses used in a statement should all appear in the precise order specified in the syntax. For example, the syntax of the `SELECT` command requires that you specify the columns you want to retrieve first, and then the table from which you want to retrieve the information next. Some clauses are required (such as `SELECT` and `FROM`), but other clauses (such as `WHERE`, `GROUP BY`, `HAVING`, and
ORDER BY) are optional. Choose which optional clauses to include based on the task you want the statement to accomplish.

Guidelines for Constructing SQL Statements

Following are guidelines for constructing SQL statements. Note that there are minor differences among different implementations of SQL, some of which are configurable within the database. To construct a standard SQL statement:

• Separate words in the statement as follows:
  • Provide at least one space or line break between any two words.
  • Enter extra spaces or line breaks if you like, and the statement will still execute correctly.
  • Type SQL keywords in upper- or lowercase (although all caps is typical). For example, SELECT, Select, and select are interpreted as the same command.
• Type table, column, and index names that:
  • Begin with a letter.
  • Limit length to 30 characters (up to 128 in SQL Server).
  • Include no spaces. (Spaces are permitted in SQL Server.)
• The capitalization of table and column names does not have to exactly match the capitalization of those names in the database.
• Enclose literal strings, text, and literal dates with single quotes (’).

Select All Columns

When you want to display all of the columns of a table in the output, you can provide an asterisk (*) instead of column names.
Figure 1-13: A SELECT clause to display all columns and records from the Titles table.

Note: Do not use `SELECT *` indiscriminately, particularly if you are using SQL in production code (in an application you are developing). `SELECT *` may return more data than you actually need, which could lead to unnecessary performance bottlenecks. If the column order changes in the database itself, it will change the order in which results are returned by a `SELECT *` query.

Optional Clauses of the SELECT Statement

SQL includes a number of optional clauses that you can use with the `SELECT` statement to further identify the data you want to retrieve from a table. SQL requires that you include these optional clauses in a specific order. The following table describes some of the optional clauses for the `SELECT` statement and their usage.

<table>
<thead>
<tr>
<th>Optional Clause</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHERE</td>
<td>A clause that enables you to request only certain rows from a table. For example, you might use a <code>WHERE</code> clause when querying a customer table to retrieve a list of only the customers who have a particular sales representative.</td>
</tr>
<tr>
<td>GROUP BY</td>
<td>A clause that uses a column identifier to organize the data in the result set into groups.</td>
</tr>
<tr>
<td>HAVING</td>
<td>A clause that you use in conjunction with the <code>GROUP BY</code> clause in order to specify which groups to include in the results.</td>
</tr>
<tr>
<td>ORDER BY</td>
<td>A clause that enables you to sort query results by one or more columns and in ascending or descending order.</td>
</tr>
</tbody>
</table>

Syntax Notation for the SQL SELECT Command

When you refer to SQL help and documentation, syntax notation will be represented as follows.
SELECT colname1[, colname2, colname3 ...]
FROM tablename

The syntax notation models the pattern for constructing a complete statement from a particular command, such as `SELECT`. Parameters enclosed within square brackets are optional. You must specify at least one column name in the `SELECT` statement or use the asterisk to request all columns in the table. Additional column names are optional.

**Note:** As a reminder, the commands and keywords in SQL are not case sensitive. As a convention in this course, you'll find the keywords capitalized in syntax and examples to differentiate them from object names.

**Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Query a Database.**
ACTIVITY 1–2
Querying a Database

Before You Begin
You have launched SQL Server Management Studio and connected to the server. You have attached the Pub1 database to the server. You have started entering a query, beginning with the `USE Pub1` statement to specify the database your SQL statements will apply to.

Scenario
The sales manager at Fuller & Ackerman Publishing would like to be able to give a list of all book titles and prices to a newly hired sales representative. You have been asked to prepare this list. You know that the information is contained in the Pub1 database but aren't familiar with the tables it contains. You will create a query to return the requested data.

1. Identify the table that contains information about book titles published by Fuller & Ackerman Publishing.
   a) In Microsoft SQL Server Management Studio, in the Object Explorer pane, make sure the Databases folder is expanded.
      You can select the + button to the left of the Databases object to expand the outline at that point.
   b) In the Databases folder, expand Pub1 and Tables.

   Tables in the Pub1 database are listed. Each table contains a set of information related to book customers, out-of-print (obsolete) book titles, sales, salespersons, current book titles, and so forth.
c) In the Databases folder, expand dbo.Titles and Columns to view the columns contained in the Titles table.

- dbo.Titles
  - Columns
    - partnum (nvarchar(3), null)
    - bktitle (nvarchar(40), null)
    - devcost (money, null)
    - sprice (money, null)
    - pubdate (smalldatetime, null)

d) Verify that the Columns folder contains the column bktitle (nvarchar(40), null).

Fuller & Ackerman stores the titles of its current books in this column.

2. Enter a query to display book titles, and run the query.

a) In the Query Editor pane, type a SELECT statement on two lines, as shown:

```
USE Pub1
SELECT * FROM Titles
```

b) Observe the color coding.

Keywords are displayed in blue. This can help you to spot typos as you enter commands. Keywords you spell incorrectly will not be blue.

c) On the SQL Editor toolbar, select Execute or press F5 to execute the query.

d) In the Results pane, observe that the part number, book title, development cost, sale price, and publishing date are displayed for each book in the Titles table.

<table>
<thead>
<tr>
<th>partnum</th>
<th>bktitle</th>
<th>devcost</th>
<th>sprice</th>
<th>pubdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>39843</td>
<td>Clear Cupboards</td>
<td>15055.50</td>
<td>49.95</td>
<td>2016-08-19 00:00:00</td>
</tr>
<tr>
<td>39905</td>
<td>Developing Mobile Apps</td>
<td>19990.00</td>
<td>45.00</td>
<td>2017-01-01 00:00:00</td>
</tr>
<tr>
<td>40121</td>
<td>Boating Safety</td>
<td>15421.81</td>
<td>36.50</td>
<td>2017-05-18 00:00:00</td>
</tr>
<tr>
<td>40122</td>
<td>Sailing</td>
<td>9393.26</td>
<td>29.15</td>
<td>2017-05-03 00:00:00</td>
</tr>
<tr>
<td>40123</td>
<td>The Sport of Windsurfing</td>
<td>12789.32</td>
<td>38.50</td>
<td>2016-07-13 00:00:00</td>
</tr>
<tr>
<td>40124</td>
<td>The Sport of Hang Gliding</td>
<td>15421.81</td>
<td>49.68</td>
<td>2017-06-01 00:00:00</td>
</tr>
<tr>
<td>40125</td>
<td>The Complete Football Reference</td>
<td>15032.41</td>
<td>49.99</td>
<td>2016-08-03 00:00:00</td>
</tr>
<tr>
<td>40231</td>
<td>How to Play Piano (Beginner)</td>
<td>9917.75</td>
<td>25.00</td>
<td>2016-06-11 00:00:00</td>
</tr>
</tbody>
</table>

Since you specified `*`, all of the columns in the table are returned by the server.

3. Scroll down to view all of the returned rows.
4. Revise the `SELECT` statement to return just two columns.

a) In the Query Editor pane, select the asterisk in the `SELECT` statement and press Delete.

   After you delete the asterisk, the insertion point is located where the asterisk had been, as shown.

   ![SQL Editor with Query](image)

b) Type `title`

   ![SQL Editor with Query](image)

   - As you start typing, the editor's autocomplete feature provides a list of objects that are available in this context.
   - Since the Titles table is provided in the `FROM` clause, columns from the Titles table (and other objects that would be appropriate in this context) are listed.
   - You can select items from this list to avoid typing and to ensure you spell object names correctly.
   - The Titles table is highlighted in the list.

   ![SQL Editor with Query](image)

c) Press Tab.

   - The highlighted object (Titles) is inserted into the query.
   - Alternatively, you can double-click an item in the list to select it.
d) Type .

• When you type the dot, fields within the Titles table are listed.
• The bktitle column is highlighted in the list.

Note: If you close the autocomplete list, you can reopen it by retyping the characters that trigger it to appear. If you have already completed an SQL clause, you may have to delete a few characters and retype them to get it to reappear.

e) Press Tab or double-click bktitle.

The bktitle column name is added into the editor pane.
f) Type directly or use the autocomplete feature to finish typing the SELECT statement as shown.

```
SELECT Titles.bktitle, Titles.slprice
FROM Titles
```

5. Execute the revised query.
   a) Execute the query.

Note: Press F5 or select the Execute button to execute the query. Note that if you have a portion of the query highlighted, SQL Server Management Studio will only execute the highlighted portion.

b) In the Results pane, observe that only the book title and sale price are displayed for each book in the Titles table.
TOPIC C

Save a Query

Sometimes SQL queries might be long, and even complex. After you have created and executed them, you might want to reuse these SQL queries for other purposes. For this reason, Microsoft SQL Server Management Studio enables you to save your queries in SQL script files. By saving your queries, you can cut down on the amount of time required for inputting the query when you need to re-run a similar query.

SQL Script Files

SQL Server Management Studio offers you the ability to save the queries that you create. When you save a query, SSMS saves the query in a script file with the extension .sql. The advantage to saving queries is that doing so enables you to re-run queries simply by opening the saved script and selecting the Execute button.

SSMS provides a default name for queries when you save them. The default name is SQLQuery#.sql, where # is a unique number assigned by SSMS. You can replace this name with a name of your choosing. Saving a query with a more meaningful name than the default name makes it easier for you to find the script file later.

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Save a Query.
ACTIVITY 1–3
Saving a Query

Before You Begin
You have launched SQL Server Management Studio and connected to the server. You have attached the Pub1 database to the server, and have entered a query to select the book title and sales price columns of data from the Titles table.

Scenario
You have created a query to retrieve information about books from the Titles table. You then realize that most of the people in the organization frequently request information that this query retrieves. Instead of retyping this query each time someone requests a list of titles, it will be helpful if you save this query as an SQL file.

1. Save the query and close the Query Editor window.
   a) On the Standard toolbar, select the Save button.
   b) In the Save File As dialog box, navigate to the C:\094018Data\Executing a Simple Query folder.
   c) In the File name text box, change the file name to My Titles Query
   d) In the Save as type drop-down list, verify that the SQL Files (*.sql) option is selected.
   e) Select the Save button to save the query.
   f) Select File→Close to close the saved query.

2. True or False? A saved query is automatically named after the database table name being queried.
   □ True
   ☑ False
TOPIC D

Modify and Execute a Saved Query

After you have saved a query, you may need to revise it. Modifying an existing query, especially if it is long and complex, is faster than retyping the query. Thus, the ability to modify a saved query and then execute it saves you time.

Data Types

![Figure 1-14: A table structure displaying various column types.]

When you create a table, you must specify the data type of each column. The **data type** is the classification of data into groups based on their characteristics, such as how many characters the value can contain, and what type of value it contains (integer, decimal, text characters, and so forth). The data type determines what calculations you can perform with that data.

All values entered into an SQL database can be classified into one of the data types available in Microsoft SQL Server 2017. The data type of the value entered into a column must match the data type the database administrator assigned to that column.

**Note:** To further explore data types, check out the Spotlight on **SQL Data Types** presentation from the Spotlight tile on the CHOICE Course screen.
### Data Types Available in SQL Server

Various data types supported on SQL Server are listed in the following table. Note that data types differ among various SQL implementations, so refer to the documentation for the specific server you are using to determine the supported data types.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Used to Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>bigint</td>
<td>The integer (whole numbers) data range from (-2^{63}) through (2^{63} - 1) with a storage size of 8 bytes.</td>
</tr>
<tr>
<td>int</td>
<td>The integer (whole numbers) data range from (-2^{31}) through (2^{31} - 1) with a storage size of 4 bytes.</td>
</tr>
<tr>
<td>smallint</td>
<td>The integer data range from (-2^{15}) through (2^{15} - 1) with a storage size of 2 bytes.</td>
</tr>
<tr>
<td>tinyint</td>
<td>The integer data range from 0 through 255 with a storage size of 1 byte.</td>
</tr>
<tr>
<td>bit</td>
<td>The integer data type that can take a value of 1, 0, or NULL.</td>
</tr>
<tr>
<td>decimal</td>
<td>Fixed precision and scale numeric data from (-10^{38} + 1) through (10^{38} - 1).</td>
</tr>
<tr>
<td>numeric</td>
<td>Functionally equivalent to a decimal.</td>
</tr>
<tr>
<td>money</td>
<td>The monetary data value range from (-922,337,203,685,477.5808) through (+922,337,203,685,477.5807), with accuracy to a ten-thousandth of a monetary unit with a storage size of 8 bytes.</td>
</tr>
<tr>
<td>smallmoney</td>
<td>The monetary data value range from (-214,748.3648) through (+214,748.3647), with accuracy to a ten-thousandth of a monetary unit with a storage size of 4 bytes.</td>
</tr>
<tr>
<td>float</td>
<td>Floating precision number data with the following valid values: (-1.79E + 308) through (-2.23E - 308), 0 and (2.23E + 308) through (1.79E + 308).</td>
</tr>
<tr>
<td>real</td>
<td>Floating precision number data with the following valid values: (-3.40E + 38) through (-1.18E - 38), 0 and (1.18E - 38) through (3.40E + 38).</td>
</tr>
<tr>
<td>datetime</td>
<td>Date and time data from January 1, 1753, through December 31, 9999, with an accuracy of three-hundredths of a second, or 3.33 milliseconds.</td>
</tr>
<tr>
<td>smalldatetime</td>
<td>Date and time data from January 1, 1900, through June 6, 2079, with an accuracy of one minute.</td>
</tr>
<tr>
<td>char</td>
<td>Fixed-length non-Unicode character data with a maximum storage size of 8,000 bytes.</td>
</tr>
<tr>
<td>varchar</td>
<td>Variable-length non-Unicode data with a maximum storage size of 8,000 bytes.</td>
</tr>
<tr>
<td>varchar(max)</td>
<td>Variable-length non-Unicode data with a maximum storage size of (2^{31} - 1) bytes.</td>
</tr>
<tr>
<td>nchar</td>
<td>Fixed-length Unicode data with a maximum length of 4,000 characters.</td>
</tr>
<tr>
<td>nvarchar</td>
<td>Variable-length Unicode data with a maximum length of 4,000 characters. syssname is a system-supplied user-defined data type that is functionally equivalent to nvarchar (128) and is used to reference database object names.</td>
</tr>
<tr>
<td>nvarchar(max)</td>
<td>Variable-length Unicode data with a maximum length of (2^{31} - 1) bytes.</td>
</tr>
<tr>
<td>binary</td>
<td>Fixed-length binary data with a maximum length of 8,000 bytes.</td>
</tr>
<tr>
<td>varbinary</td>
<td>Variable-length binary data with a maximum length of 8,000 bytes.</td>
</tr>
<tr>
<td>varbinary(max)</td>
<td>Variable-length binary data with a maximum length of (2^{31} - 1) bytes.</td>
</tr>
<tr>
<td>Data Type</td>
<td>Used to Store</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cursor</td>
<td>A reference to a cursor.</td>
</tr>
<tr>
<td>sql_variant</td>
<td>A data type that stores values of various SQL Server–supported data types,</td>
</tr>
<tr>
<td></td>
<td>except for text, ntext, timestamp, and sql_variant.</td>
</tr>
<tr>
<td>table</td>
<td>A special data type used to store a result set for later processing.</td>
</tr>
<tr>
<td>timestamp</td>
<td>A database-wide unique number that gets updated every time a row gets updated.</td>
</tr>
<tr>
<td>uniqueidentifier</td>
<td>A globally unique identifier (GUID).</td>
</tr>
<tr>
<td>xml</td>
<td>A built-in data type that stores the XML documents and fragments in a SQL</td>
</tr>
<tr>
<td></td>
<td>Server database. The stored representation of the xml data type cannot exceed 2</td>
</tr>
<tr>
<td></td>
<td>GB.</td>
</tr>
</tbody>
</table>

**Note:** The text, ntext, and image data types were replaced by varchar(max), nvarchar(max), and varbinary(max) in the SQL Server 2008 version.

## Stored Procedures

A *stored procedure* is essentially an SQL script file that you save as a database object. The advantage to saving a query as a stored procedure is that SQL Server compiles stored procedures in advance. Compiling the stored procedure saves an execution plan as part of the stored procedure, which makes the stored procedure run faster when you execute it.

SQL Server 2017 includes many system-created stored procedures. These stored procedures are installed by default when you install SQL Server. The system stored procedures have names that begin with *sp_* followed by a descriptive name. For example, the stored procedure *sp_help* followed by a table name such as *Titles* enables you to view the structure of that table. Here's the syntax:

```sql
sp_help Titles
```

**Note:** To further explore stored procedures, check out the Spotlight on *System Stored Procedures* presentation from the Spotlight tile on the CHOICE Course screen.

## Comments

A *comment* is a word or statement, entered in the *Query Editor* window, that is not meant for SQL Server to execute when it runs the query. You use comments to provide explanations about a query or to temporarily disable parts of an SQL statement. You indicate a single line comment by preceding it with two hyphens. Everything following the two hyphens is ignored in the remainder of that line. You can specify multiple line comments by enclosing the lines within the /* and */ characters.

If your students include web developers, point out that using stored procedures can help prevent a significant type of web security vulnerability called SQL injection.

The Spotlight on *System Stored Procedures* presentation is available from the Spotlight tile on the CHOICE Course screen. You may choose to include it in your instructional plans, or you can remind students about the tile and the supplemental information it contains.
Figure 1-15: Various types of comments in SQL.

- Comment explains the code
  ```
  -- Display the table structure
  SP_HELP Titles
  ```

- Inline comment
  ```
  SELECT custname, state -- not city
  FROM Customers
  WHERE city = 'Rochester'
  ```

- Statement “commented out” so it won’t execute
  ```
  /* SELECT custname, state, city
  FROM Customers
  WHERE city = 'Rochester' */
  ```

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Work with Saved Queries.
ACTIVITY 1–4
Modifying a Saved Query

Before You Begin
You have entered a query to select all columns of data from the Titles table, and you have saved it as C:\094018Data\Executing a Simple Query\My Titles Query.sql.

Scenario
You created a query to produce a list of book titles and prices for the new sales representative. After retrieving the information, you realize that it might be helpful to provide a product ID or part number of some sort, as some books may have similar names.

1. Display the structure of the Titles table.
   a) On the Standard toolbar, select the New Query button to open the editor pane.
   b) From the Available Databases drop-down list, select Pub1.
   c) In the editor pane, type sp_help titles
   d) Execute the query.

Briefly explain the results of the query so that students understand the structure of the Titles table.

Structural information regarding the Titles table is shown.
2. Based on the scenario, which column should you add to your `SELECT` statement?
   
   A: The `partnum` field will provide the information you’re looking for.

3. Close the editor pane without saving the `sp_help` query.
   
   a) Select `File→Close`.
   
   b) When you’re prompted to save the query, select `No`.

4. Open the `My Titles Query.sql` file.
   
   a) Select `File→Open→File`.
   
   b) In the `Open File` dialog box, navigate to the `C:\094018Data\Executing a Simple Query` folder.
   
   c) Select the `My Titles Query.sql` file and select `Open` to open the query file.

5. Modify the `SELECT` statement to display only the necessary columns and verify the query works.
   
   a) Position the insertion point immediately after the `SELECT` keyword, as shown.

   
   ![SELECT statement example](image)

   b) Press `Space`, and type `Titles.partnum`.

   The completed query is:

   ```sql
   USE Pub1
   SELECT Titles.partnum, Titles.bktitle, Titles.sprice
   FROM Titles
   ```

   c) Execute the query.
   
   d) Observe the results.

   ![Query results](image)

   92 rows are returned. The query results include the part number.

6. Add a comment to the query.
   
   a) Position the insertion point at the end of the `USE Pub1` statement, and press `Enter` twice to add two new lines.
b) Type two dashes to start a comment line, and finish typing the comment as shown.

![Comment in SQL Server Management Studio]

c) Observe that the comment is shown in green in SQL Server Management Studio.

7. Save the modified query.
   a) Select File→Save My Titles Query.sql As.
   b) In the File name text box, change the file name to *My Titles and Partnum Query* and then select Save to save the modified query as My Titles and Partnum Query.sql.
      If you omit the .sql file name extension, it is added automatically when you save the file.
   c) Close the Query Editor window, but leave SQL Server Management Studio running.
ACTIVITY 1–5
Executing a Saved Query

Before You Begin
You have saved your modified query statement in C:\094018Data\Executing a Simple Query\My Titles and Partnum Query.sql.

Scenario
The sales manager wants an updated list of published book titles. You remember that you have saved the query to retrieve book titles along with their sale price and part number.

1. Open the saved SQL file.
   a) Select File→Open→File.
   b) If necessary, in the Open File dialog box, navigate to the C:\094018Data\Executing a Simple Query folder.
   c) Select the My Titles and Partnum Query.sql file and select Open to open the query file.
   d) On the SQL Editor toolbar, from the Available Databases drop-down list, select Pub1.

2. Execute the query.
   a) Execute the query.
   b) In the Results pane, observe that the part number along with the book title and sale price are displayed.
   c) Close the Query Editor window.
Summary

In this lesson, you began working with SQL Server by connecting to a database and executing some simple queries. By using the `SELECT` statement to retrieve information from tables, you can gather exactly the information that you need from practically any SQL database.

As an employee, how often might you connect to the database while working?

A: Answers will vary, but may include: there might be instances where you will retrieve information regarding your day-to-day business tasks from the server and implement the same in your course of work. Therefore, it becomes important for you to access information from the database. To do so, you should initially connect to the database. Therefore, every time you require information from the server, you will connect to the database.

Which SQL Server command group would you use the most in your current job? Why?

A: Answers will vary, but may include: Data Manipulation Language (DML) is the SQL command group you'll use most often because it's the language for retrieving information from the database. In a database, you might update data quite often according to the requirements. DML enables you to insert, update, or delete information from the database.

Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.
Performing a Conditional Search

Lesson Time: 1 hour, 45 minutes

Lesson Introduction
At this point, you have connected to a server, written a simple query and executed it, and saved that query to a file. With simple queries, you typically retrieve all the information in a table. In a production environment, however, you rarely need to view all the information in a table. Instead, you want to retrieve only the rows in a table that meet certain criteria. In this lesson, you will add criteria to your SELECT statements so that you can perform more sophisticated queries.

Lesson Objectives
In this lesson, you will:
- Use one or more simple search conditions to retrieve the desired output.
- Retrieve records based on a range of values and NULL values.
- Search for patterns in a table.
TOPIC A

Search Using One or More Conditions

The most basic of conditional searches use one or more criteria to query a table. In this topic, you will use one or more search conditions to retrieve rows from tables.

Conditional searches enable you to limit the number of rows returned by a SELECT statement. The benefit to limiting the number of rows in a query's output is that you reduce the amount of work the SQL Server must perform to return a query's results. Reducing the amount of work for the server helps to improve its performance.

Conditions

Most production tables consist of hundreds or even thousands of records or more. Executing a query that retrieves all columns and all rows from such large tables can severely affect the performance of a server. In an environment with large tables, you can improve the performance of your queries and avoid negatively affecting a server's performance by including conditional search criteria in your SELECT queries.

A condition is a search criterion you use in a SELECT statement to retrieve or manipulate specific information. You can include more than one search criterion in a SELECT statement so that you can retrieve the exact information you need. You use search criteria to compare information in a column to a specific value. You can also perform calculations on numeric columns before comparing information.

In the figure, you see a SELECT statement that queries the Sales table in the Pub1 database. The SELECT statement retrieves the ordnum (order number), sldate (sales date), qty (quantity), partnum (part number), and repid (sales representative ID) for all orders in which the repid column contains the value NO2. Thus, this SELECT statement enables you to see all the orders generated by sales representative NO2.

```
SELECT ordnum, sldate, qty, partnum, repid
FROM sales
WHERE repid = 'NO2'
```

Figure 2-1: A SELECT statement displaying a single search condition.

The Conditional Search Process

SQL Server uses a conditional search to retrieve only selected records (or rows) from a table. To process a conditional search:

1. You enter the SELECT statement along with a condition in the Query Editor window. To perform a conditional search, you must include a WHERE clause as part of the SELECT statement.
2. SQL Server searches the table using the condition present in the WHERE clause.
3. SQL Server returns the rows that match the condition in the WHERE clause.
4. The server displays the retrieved rows in the Results pane of the Query Editor window.
The WHERE Clause

The **WHERE clause** is a clause you use to specify a search condition in an SQL statement. The WHERE clause contains an expression or column name followed by an operator, and then an expression or value that SQL Server needs to compare with one or more columns in the table. You can include more than one condition in the WHERE clause.

The syntax of the WHERE clause used in the SELECT statement is:

```
SELECT colname1[, colname2, colname3 ...]
FROM tablename
WHERE condition
```

In the figure, the WHERE clause includes a single condition. The condition specifies that SQL Server should return only those rows in the Titles table in which the slprice column has a value greater than 100.

```
SELECT partnum, bktitle, slprice
FROM titles
WHERE slprice > 100
```

![Figure 2-3: A SELECT statement containing a single condition.](image)

This next figure shows a SELECT statement in which the WHERE clause has two conditions. Because these two conditions are separated by the **AND** operator, both conditions must be true in order for SQL Server to display a row in the output. In other words, SQL Server includes a row in the output only when the column repid contains the value N02 and the column qty is greater than or equal to 400.

```
SELECT ordnum, sldate, qty, partnum, repid
FROM sales
WHERE repid = 'N02' AND qty >= 400
```

![Figure 2-4: A SELECT statement displaying multiple conditions.](image)
Operators

Operators are symbols or words used in expressions that manipulate values or make comparisons. They are mostly used between a word and a value in a search condition for a WHERE clause. You can use operators to perform calculations, compare values, and match patterns. In the following figure, the WHERE clause contains two conditions: `slprice > 40` and `partnum > 1000`. This query returns all books where the sale price is greater than $40 and the part number is greater than 1000.

![Figure 2-5: A SELECT statement displaying multiple conditional operators.]

As another example, the following figure displays a SELECT statement in which the WHERE clause searches for all books with a sale price greater than or equal to 30. SQL Server thus searches through the Titles table and returns only those rows in which the sale price is greater than or equal to 30.

![Figure 2-6: A SELECT statement displaying operators used in a WHERE clause.]

Operators Used in SQL

There are eight key categories of operators used in Microsoft® SQL Server® 2017.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic operators</td>
<td>Perform mathematical operations on two expressions of numeric data.</td>
</tr>
<tr>
<td>Assignment operators</td>
<td>Establish the relationship between a column heading and the expression that defines values for the column.</td>
</tr>
<tr>
<td>Compound operators</td>
<td>Perform mathematical operations on two numeric expressions by combining one operator with another.</td>
</tr>
<tr>
<td>Bitwise operators</td>
<td>Perform bit manipulations between two expressions of the integer data type.</td>
</tr>
<tr>
<td>Comparison operators</td>
<td>Test whether two expressions are the same, greater than, or less than.</td>
</tr>
<tr>
<td>Logical operators</td>
<td>Test for the truth of a condition. Return a Boolean data type with a value of TRUE or FALSE.</td>
</tr>
<tr>
<td>String concatenation operators</td>
<td>Allow string concatenation.</td>
</tr>
</tbody>
</table>
Comparison Operators

Comparison operators are symbols you use in a WHERE clause to compare two expressions or values.

In SQL Server, you use comparison operators in conditions as part of a WHERE clause. You cannot use comparison operators with columns that use the text, ntext, or image data types.

The output of a comparison operator is one of three values: TRUE, FALSE, or UNKNOWN.

- If the output of a comparison operator is TRUE, SQL Server displays the row in the results of a query.
- If the output is FALSE, SQL Server does not display the row in the query results.
- If the output is UNKNOWN, the results may vary. This may occur, for example, in some WHERE conditions involving NULL values.

Comparison Operators and Their Descriptions

There are nine comparison operators used in SQL Server 2017.

<table>
<thead>
<tr>
<th>Comparison Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to (not SQL-92 standard)</td>
</tr>
<tr>
<td>!&lt;</td>
<td>Not less than (not SQL-92 standard)</td>
</tr>
<tr>
<td>!&gt;</td>
<td>Not greater than (not SQL-92 standard)</td>
</tr>
</tbody>
</table>

Arithmetic Operators

Arithmetic operators are symbols used to perform mathematical operations on numeric expressions. You can also use the plus (+) and minus (-) operators to perform arithmetic operations.

Figure 2–7: A SELECT statement displaying the comparison operator.
on datetime and smalldatetime values. In the following figure, you see a SELECT statement that uses the plus operator to add 20 dollars to the sale price of books; you might use such a statement if your organization planned to increase the cost of books by a certain dollar amount. The WHERE clause then restricts the output to only those books where the sale price plus 20 dollars is greater than 50 dollars.

**Figure 2-8: A SELECT statement displaying an arithmetic operator.**

**Arithmetic Operators Used in SQL**

There are five arithmetic operators used in SQL.

<table>
<thead>
<tr>
<th>Arithmetic Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>%</td>
<td>Returns the integer remainder of a division</td>
</tr>
</tbody>
</table>

**Column Aliasing**

A column alias is a name you assign for SQL Server to use as a column heading in the output. You can assign a column alias to any column in a table and SQL Server will display the alias in place of the default column heading in the result set. Using an alias enables you to provide more descriptive headings for the columns in a table. The alias can contain any alphanumeric characters along with a few special characters, such as $ or #. If you want the alias to contain a space, period, or exclamation point, you must enclose it in single or double quotes.

By default, SQL Server displays the column name in the heading of the result set for a SELECT statement. If the column does not have a column name because it is the result of a calculation, SQL Server displays the heading "No column name." You make the output of a query more meaningful when you provide a column alias for any calculations in the SELECT statement.
Alternative Ways to Define an Alias

The syntax for defining an alias provides for some variations, as shown in the following figure.

Logical Operators

Logical operators are operators that test the truth of a condition. Logical operators, like comparison operators, return a value of either TRUE or FALSE. You may use multiple logical operators in a WHERE clause to test for more than one condition. For each row, the condition or combination of conditions in the clause is evaluated. If the result returns TRUE, then that row is returned by the database when the query is evaluated.
<table>
<thead>
<tr>
<th><strong>Logical Operator</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>TRUE if all of a set of comparisons are TRUE.</td>
</tr>
<tr>
<td>AND</td>
<td>TRUE if both Boolean expressions are TRUE.</td>
</tr>
<tr>
<td>ANY</td>
<td>TRUE if any one of a set of comparisons is TRUE.</td>
</tr>
<tr>
<td>BETWEEN</td>
<td>TRUE if the operand is within a range.</td>
</tr>
<tr>
<td>EXISTS</td>
<td>TRUE if a subquery contains any rows.</td>
</tr>
<tr>
<td>IN</td>
<td>TRUE if the operand is equal to one of a list of expressions.</td>
</tr>
<tr>
<td>LIKE</td>
<td>TRUE if the operand matches a pattern.</td>
</tr>
<tr>
<td>OR</td>
<td>TRUE if either Boolean expression is TRUE.</td>
</tr>
<tr>
<td>SOME</td>
<td>TRUE if some of a set of comparisons are TRUE.</td>
</tr>
<tr>
<td>NOT</td>
<td>Inverts the logical value returned by the other operators.</td>
</tr>
</tbody>
</table>

This list of SQL Server logical operators is typical. Refer to your database's documentation for a specific list of supported logical operators.

**Note:** Boolean is a type of expression with two possible values, "true" and "false."

### The AND, OR, and NOT Operators

The **AND**, **OR**, and **NOT** operators are the most commonly used logical operators. The **AND** and **OR** operators are used to combine the result of two or more Boolean expressions. The **AND** operator returns TRUE when both expressions are TRUE, while the **OR** operator returns TRUE when either of the expressions is TRUE. The **NOT** operator is used to negate a Boolean expression.

**Note:** Spaces are necessary between words, but optional where there is a symbol, such as a comparison or arithmetic operator.

### Syntax of Commonly Used Logical Operators

- `boolean_expression1 AND boolean_expression2`
- `boolean_expression1 OR boolean_expression2`
- `[ NOT ] boolean_expression`

### OR Example

This figure shows a **WHERE** clause with two search conditions separated by the **OR** operator. For this example, SQL Server will return all rows in the Customers table in which the value in the state column is either Massachusetts or California.
**AND Example**

In the following figure, the *WHERE* clause consists of multiple search conditions. The *AND* logical operator connects the two search conditions, and both the conditions must be true in order for SQL Server to include a row in the output of a query. In this example, for SQL Server to include a row in the output, the customer must live in the state of Texas and the city of Houston.

```
SELECT custnum, city, state, custname
FROM customers
WHERE state = 'TX' AND city = 'Houston'
```

*Figure 2–12: Using the AND operator in SQL querying.*

**NOT Example**

This figure shows you an example of the *NOT* logical operator. In this example, SQL Server will return all rows in the Customers table in which the city column does not contain the value Ryebrook. In other words, this query gives you a list of all customers who don't live in the city of Ryebrook.

```
SELECT custnum, city, state
FROM customers
WHERE city <> 'Ryebrook'
```

*Figure 2–13: Using the NOT operator in SQL querying.*
AND and NOT Combined Example
In this example, the \texttt{WHERE} clause consists of two conditions separated by the \texttt{AND NOT} operators. This syntax enables SQL Server to retrieve all rows in the Customers table where the customers live in New York but not the city of Ryebrook.

\begin{verbatim}
SELECT custnum, city, state, custname
FROM customers
WHERE NOT city = 'Ryebrook'
\end{verbatim}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure2-13.png}
\caption{Using NOT operators in SQL.}
\end{figure}

\textbf{Operator Precedence}
When you provide multiple conditions in a complex \texttt{WHERE} clause, you might assume that they are performed from left to right, in the order in which they were typed. However, the operators you use to chain these various conditions together may have different precedence levels, which determine the order in which SQL Server executes them. You can use parentheses to override the order in which SQL Server executes operators.

If two operators in an expression have the same operator precedence level, they are evaluated left to right based on their position in the expression.

Operators are listed here in their order of precedence. Items at the top are performed first.

<table>
<thead>
<tr>
<th>Operator Level</th>
<th>Operator Precedence in Each Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+ (Positive), - (Negative), ~ (Bitwise NOT)</td>
</tr>
<tr>
<td>2</td>
<td>* (Multiply), / (Division), % (Modulo)</td>
</tr>
<tr>
<td>3</td>
<td>+ (Add), + (Concatenate), - (Subtract)</td>
</tr>
<tr>
<td>4</td>
<td>=, &gt;, &lt;, &gt;=, &lt;=, &lt;&gt;, !=, !=, !&gt;</td>
</tr>
<tr>
<td>5</td>
<td>^ (Bitwise Exclusive OR), &amp; (Bitwise AND),</td>
</tr>
<tr>
<td>6</td>
<td>NOT</td>
</tr>
<tr>
<td>7</td>
<td>AND</td>
</tr>
</tbody>
</table>
### Parentheses

You can provide parentheses to override the normal operator precedence, as shown in these examples.

In the top example, since AND has precedence over OR, the AND operation is performed first—even though it appears after the OR within the statement.

In the bottom example, the normal order is overridden by parentheses. The OR operation within the parentheses is performed before the AND operation.

**Figure 2-15: Parentheses override the normal order of operations.**

### Comparisons Involving NULL

NULL means that a value is unknown. NULL provides a way for the user to represent that the data value is unknown, not applicable, or intended to be added later. NULL can be useful for representing unknown values, but it can make comparisons a bit awkward because NULL values are unknown and therefore can’t really be compared.

In the default configuration for SQL Server, when a value in a comparison operation is NULL, it is treated differently from an empty or zero value. It is important to understand how NULL values are dealt with in comparison operations so your queries produce the results you intend to produce.
DO NOT DUPLICATE
FOR EVALUATION USE ONLY
(C) 2017 Logical Operations, Inc.

Figure 2-16: A comparison involving a NULL value.

Consider the query shown in the previous figure as an example of how comparison operators function when a NULL value is involved. In this example, suppose the Titles table contains a row with an slprice value of 40 and a devcost value of NULL. Because of the NULL value in devcost, this row will not be returned in the query result, even though the value of slprice is more than 35.

The first expression in the WHERE clause (slprice > 35) returns TRUE. The second expression (devcost > 15000) returns NULL.

The following table shows the results for various combinations of expressions and operations.

<table>
<thead>
<tr>
<th>Expression 1</th>
<th>Expression 2</th>
<th>Result of AND Operation</th>
<th>Result of OR Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>NULL</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>FALSE</td>
<td>NULL</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

In the table, you can see that an AND operation performed on TRUE and NULL returns FALSE. Only rows in which the WHERE clause evaluates as TRUE are included in the result.

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Search Using One or More Conditions.
ACTIVITY 2–1
Searching Using a Simple Condition

Scenario
The sales department has asked you to provide the sales representatives with a list of all books with a price of $50 or more. They have also noticed an increase in demand for books on sailing. For this reason, the sales department has also asked you to confirm that a book named "Sailing" is available.

1. Launch SQL Server Management Studio, open a new query editor, and select the Pub1 database.
   a) If SQL Server Management Studio is not running, launch it and connect to the server.
   b) On the Standard toolbar, select New Query to open the Query Editor window.
   c) On the SQL Editor toolbar, from the Available Databases drop-down list, ensure that the Pub1 database is selected.

2. List books for which the sale price is greater than or equal to $50.
   a) In the editor pane, press Enter to skip a line and enter the FROM clause as shown.

   • By typing the FROM clause first, you provide a clue to inform SQL Server Management Studio which table you're querying.
   • This enables the autocomplete feature to display suggested column names that you type in clauses such as SELECT.

   b) Move the insertion point to the first line. Type SELECT pa

   The partnum column is selected.

Activity: Searching Using a Simple Condition

SQL code snippets for the course activities are stored as snippets.sql in the Solutions folder for each lesson's course data. You can use these to save yourself some typing as you lead students through the activities. Or you may want to direct students to these files if they are having trouble keeping up with the typing in class.
c) Press **Tab**.
   The partnum column name is inserted.

d) Finish the **SELECT** clause, and enter the **WHERE** clause as shown.

   ![SQL query example]

   ```sql
   SELECT partnum, bktitle, slprice
   FROM Titles
   WHERE slprice >= 50
   ```

  e) Observe the column names you entered in the **SELECT** and **WHERE** clauses.
     - It is not necessary to specify the table with each field (fully qualified names such as
       `Titles.partnum`, `Titles.bktitle`, and so forth) because the query refers only to the Titles table.
     - When you create more complex queries that use multiple tables, you'll need to use fully qualified
       names.

  f) Press **F5** or select the **Execute** button to execute the query.

g) In the **Results** pane, observe the book titles for which the sale price is greater than or equal to $50.

   ![Results pane example]

   Three books are returned.

3. List the details of the book that has "Sailing" as the book title.
   a) Revise the last line of your query as shown.

      ![Revised SQL query example]

      ```sql
      SELECT partnum, bktitle, slprice
      FROM Titles
      WHERE bktitle = 'Sailing'
      ```

   b) Execute the query.
   c) In the **Results** pane, observe that a book with the title "Sailing" is displayed.

   ![Results pane example for Sailing]
ACTIVITY 2–2
Comparing Calculated Column Values

Before You Begin
SQL Server Management Studio is running. A query editor window is open, and the Pub1 database is selected.

Scenario
Your boss has come up with an idea to promote the sales of the company by offering discounts for certain books. The plan is to provide 7% off on all books for which the sale price is greater than $45 after the discount. You need to identify the book titles that will be included in the discount sale.

1. Display the normal prices of all books.
   a) In the editor pane, delete the WHERE clause.
   b) Execute the query.
   c) In the Results pane, observe that books are shown with their normal prices.

2. Display all books with a discounted price.
   a) In the editor pane, add the text as shown.
   b) Execute the query.
   c) In the Results pane, observe that books are shown with their discounted prices.
c) In the Results pane, observe that books are now shown with discounted prices.

![Results pane showing discounted prices](image)

The third column no longer shows the normal sale price, but rather a calculated value, so no label is shown for the column.

3. Provide the calculated column with an alias.
   a) In the editor pane, add the text as shown.

   ```sql
   SELECT partnum, bktile, slprice - slprice * 0.07 AS discounted_price
   FROM Titles
   ```

   This gives the calculated column an alias.
   b) Execute the query.
   c) In the Results pane, observe that the discounted prices now have a column name.

![Results pane with discounted prices](image)

4. Show only books whose discounted price is at least $45.
   a) In the editor pane, add the `WHERE` clause as shown.

   ```sql
   SELECT partnum, bktile, slprice - slprice * 0.07 AS discounted_price
   FROM Titles
   WHERE slprice - slprice * 0.07 >= 45
   ```

   This limits the results to books with a sale price more than $45 after the discount.
   b) Execute the query.
c) In the **Results** pane, observe the books with discounted prices greater than $45.

<table>
<thead>
<tr>
<th>partnum</th>
<th>title</th>
<th>discounted_price</th>
</tr>
</thead>
<tbody>
<tr>
<td>39843</td>
<td>Clear Cupboards</td>
<td>46.453500</td>
</tr>
<tr>
<td>40124</td>
<td>The Sport of Hang Gliding</td>
<td>46.202400</td>
</tr>
<tr>
<td>40125</td>
<td>The Complete Football Reference</td>
<td>46.490700</td>
</tr>
<tr>
<td>40455</td>
<td>More Home Repairs Made Easy</td>
<td>46.490700</td>
</tr>
<tr>
<td>40482</td>
<td>The Complete Auto Repair Guide</td>
<td>47.420700</td>
</tr>
<tr>
<td>40933</td>
<td>North American History</td>
<td>46.500000</td>
</tr>
<tr>
<td>40895</td>
<td>The History of Baseball</td>
<td>65.090700</td>
</tr>
<tr>
<td>40897</td>
<td>Mythologies of the World</td>
<td>46.453500</td>
</tr>
</tbody>
</table>
ACTIVITY 2–3
Searching for Rows Using Multiple Conditions

Before You Begin
SQL Server Management Studio is running. A query editor window is open, and the Pub1 database is selected.

Scenario
The sales manager has information about a book exhibition that is to be held in Ryebrook, New York. He wants a list of customers in New York and the customers in the city of Ryebrook so that he can notify them of Fuller & Ackerman Publishing's participation in the book exhibition. The sales manager wants to run a promotional sale in New York and Massachusetts. He wants the sales representative with ID S01 to be in charge of the sale, and to contact the customers in those states.

1. List all customers who are from the state of New York.
   a) Open a new Query Editor window, but do not close the existing window.
   b) In the new Query Editor window, enter the statement in the editor pane as shown.

   ```sql
   SELECT city, state, custname
   FROM Customers
   WHERE state = 'NY'
   ```

   c) Execute the query.
   d) In the Results pane, observe that SQL Server displays a list containing six customers from New York State.

2. List all customers who live in either the state of Massachusetts or New York.
a) Revise the WHERE clause as shown.

```
SELECT city, state, custname
FROM Customers
WHERE state = 'NY' OR state = 'MA'
```

b) Execute the query.

c) In the Results pane, observe that SQL Server lists a total of seven customers from the states of New York and Massachusetts.

```
<table>
<thead>
<tr>
<th>city</th>
<th>state</th>
<th>custname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rochester</td>
<td>NY</td>
<td>CK Music!</td>
</tr>
<tr>
<td>Rybrook</td>
<td>NY</td>
<td>Friendly Books</td>
</tr>
<tr>
<td>Cambridge</td>
<td>MA</td>
<td>Frank's Music Shop</td>
</tr>
<tr>
<td>Buffalo</td>
<td>NY</td>
<td>Prince's Pets</td>
</tr>
<tr>
<td>Rybrook</td>
<td>NY</td>
<td>Toujours Tours</td>
</tr>
<tr>
<td>Rybrook</td>
<td>NY</td>
<td>One Stop Bookshop</td>
</tr>
<tr>
<td>Syracuse</td>
<td>NY</td>
<td>The Corner Bookstore</td>
</tr>
</tbody>
</table>
```

Adding an **OR** condition increased the number of matches from 6 to 7.

3. Attempt to list customers whose sales representative ID is S01 and who are either from Massachusetts or from New York.

a) Edit your query to read:

```
SELECT city, state, custname,repid
FROM Customers
WHERE state = 'NY' OR state = 'MA' AND repid = 'S01'
```

b) Execute the query.

Point out to students that the representative ID S01 is "ess zero one."
c) In the Results pane, observe that a list of seven customers from Massachusetts or New York are displayed.

<table>
<thead>
<tr>
<th>city</th>
<th>state</th>
<th>custname</th>
<th>repid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rochester</td>
<td>NY</td>
<td>CK Music!</td>
<td>S01</td>
</tr>
<tr>
<td>Ryebrook</td>
<td>NY</td>
<td>Friendly Books</td>
<td>S03</td>
</tr>
<tr>
<td>Cambridge</td>
<td>MA</td>
<td>Frank’s Music Shop</td>
<td>S01</td>
</tr>
<tr>
<td>Buffalo</td>
<td>NY</td>
<td>Prince’s Pets</td>
<td>S01</td>
</tr>
<tr>
<td>Ryebrook</td>
<td>NY</td>
<td>Toujours Tours</td>
<td>S01</td>
</tr>
<tr>
<td>Ryebrook</td>
<td>NY</td>
<td>One Stop Bookshop</td>
<td>W02</td>
</tr>
<tr>
<td>Syracuse</td>
<td>NY</td>
<td>The Corner Bookstore</td>
<td>E01</td>
</tr>
</tbody>
</table>

d) Consider how the query produced these results.

```
SELECT city, state, custname, repid
FROM Customers
WHERE state = 'NY' OR state = 'MA' AND repid = 'S01'
```

- **Produces 6 matching rows**
- **Produces 1 matching row**

- AND operations take precedence over OR operations.
- As written, the WHERE clause produces two groups of results. The first group is those where `state = 'NY'`. The second group is those where `state = 'MA'` AND `repid = 'S01'`.
- To apply the `repid = 'S01'` condition to both states, you need to enclose the OR operation within parentheses so it is performed before AND.

4. Revise the query so the OR operation is performed before AND.

a) Add parentheses in the WHERE clause as shown.

```
SELECT city, state, custname, repid
FROM Customers
WHERE (state = 'NY' OR state = 'MA') AND repid = 'S01'
```

First the list of all customers will be reduced to those in NY or MA. Then the list will be further reduced to those where the repid is S01.
b) Execute the query, and observe the results.

<table>
<thead>
<tr>
<th>city</th>
<th>state</th>
<th>custname</th>
<th>repid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rochester</td>
<td>NY</td>
<td>CK Music!</td>
<td>S01</td>
</tr>
<tr>
<td>Cambridge</td>
<td>MA</td>
<td>Frank's Music Shop</td>
<td>S01</td>
</tr>
<tr>
<td>Buffalo</td>
<td>NY</td>
<td>Prince's Pets</td>
<td>S01</td>
</tr>
<tr>
<td>Ryebrook</td>
<td>NY</td>
<td>Toujours Tours</td>
<td>S01</td>
</tr>
</tbody>
</table>

A list of four customers who are either from the state of Massachusetts or New York is displayed. All have a repid of S01.
TOPIC B

Search for a Range of Values and NULL Values

You have retrieved records from a table based on conditions. There are times when you may need to retrieve information from a database based on a specified range of values. In this topic, you will search for records based on a range of values.

The BETWEEN...AND Operator

The output of a query depends on the condition you use to retrieve information. If there is a table that contains a list of book titles and their prices, and you need only the list of books whose price range is between $40 and $50, then instead of using two conditions, you can use an operator to retrieve the records that fall in that range.

The BETWEEN...AND operator is a logical operator that searches for rows where one or more columns contain a value within a range of values. You specify the start value of the range after the BETWEEN keyword and the end value after the AND keyword in the WHERE clause of an SQL statement. You can use the logical operator NOT to retrieve records that fall outside a specified range.

In the following figure, the WHERE clause specifies that SQL Server should retrieve all rows in the Titles table where the slprice column is less than $50 or greater than $55.

![Figure 2-17: Using the BETWEEN.. AND operator in SQL.](Image)

The BETWEEN...AND operator is the equivalent of using >= and <= operators to frame a condition. Using the BETWEEN...AND operator rather than the mathematical symbols makes it easier to read and understand the SQL statement.

Syntax of the BETWEEN...AND Operator

The syntax of the BETWEEN...AND operator is:
The IN Operator

The **IN operator** is a logical operator that checks whether a column value or expression matches a list of values. You use the **IN** operator in a **WHERE** clause between the column name and the list of values to be matched. You specify the list of values within parentheses, separated by commas. If you use text in the list of values, you must enclose it within single quotes. The data type of values in the list must match the data type of the column or expression.

In the following figure, the **WHERE** clause uses the **IN** operator followed by a list of values in parentheses. In this example, SQL Server retrieves all rows in which the state value is California, New York, or Massachusetts.

### Syntax of the IN Operator

The syntax of the **IN** operator is:

```
expression [ NOT ] IN ( expression [ value1, value2, ... ] )
```

### The NULL Value

**NULL** is a value that SQL Server stores in a column when the value of the column is either unknown or undefined. When you view table information, SQL Server displays the word "NULL" in the columns that contain the NULL value. NULL is not the same as zero, blank, or a zero-length character string. If you attempt to compare NULL values, they will not be equal because the value of each NULL is unknown.
The IS NULL Clause

The **IS NULL clause** is a clause that tests for the NULL value in a column. You can use the **IS** NULL clause in a **WHERE** clause after the expression or column name you want to check for NULL values. You can use the **NOT** operator between the **IS** and **NULL** keywords to check for values in a column that are not NULL.

In the following figure, you see an example of using the **IS NULL** clause as part of a **WHERE** clause in a query. In this example, SQL Server returns those books that have a sale price between $35 and $45 and a development cost that is unknown.

```
SELECT bktile, slprice, devcost
FROM titles
WHERE slprice BETWEEN 35 AND 45 AND devcost IS NULL
```

**Figure 2-20: Use of the IS NULL clause.**

The following figure provides you with an example of how to use the **IS NOT NULL** clause. In this example, SQL Server returns all books that have a sale price between $35 and $45 and have a defined development cost.
Syntax for the IS NULL Clause
The syntax for the IS NULL clause is:

```
expression IS [NOT] NULL
```

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Search for a Range of Values and NULL Values.
ACTIVITY 2–4
Searching for a Range of Values and NULL Values

Before You Begin
SQL Server Management Studio is running. A query editor window is open, and the Pub1 database is selected.

Scenario
The manager of the sales department plans to provide discounts to customers for books that are priced between $35 and $70. He has asked you to generate a list of these books. In addition, he would like you to provide him with a list of all books that do not have a defined development cost. Finally, the sales manager has asked you to provide a list of books that have a defined development cost so that he can analyze the profitability of each book.

1. Generate a list of all titles with a sale price between $35 and $70.
   a) Switch to the query tab with the Titles query and modify the `SELECT` and `WHERE` clauses as shown here:

   ```sql
   SELECT bkttitle, slprice
   FROM Titles
   WHERE slprice BETWEEN 35 AND 70
   ```

   Note: If you closed the editor containing the Titles query, enter this query on a new query tab, making sure you select the Pub1 database.

   b) Execute the query.
   c) In the Results pane, observe that a list of 38 book titles with a sale price between $35 and $70 is displayed.
d) Scroll down to view all rows, and observe that the BETWEEN . . . AND operator includes the values at the start and end of the range. For example, books selling for exactly $35 are included in the list.

2. List the titles that do not have the development cost recorded.
   a) Edit the query so that it reads as follows:

   ```sql
   SELECT bkttitle, slprice, devcost
   FROM Titles
   WHERE devcost IS NULL
   ```

   - Add the devcost column to the output.
   - Change the WHERE clause to show those books for which no devcost value has been entered.

   b) Execute the query.
   c) In the Results pane, observe that the list includes five book titles whose development cost is not defined.

3. Modify the query to list titles that have a numerical development cost.
a) Edit the query to retrieve all rows that have a value in the devcost column:

```
SELECT bktitle, slprice, devcost
FROM Titles
WHERE devcost IS NOT NULL
```

b) Execute the query.

c) In the Results pane, observe that a list of 87 titles that have a numerical development cost is displayed.

The list contains no books with a NULL devcost value.
TOPIC C

Search Data Based on String Patterns

Other types of search conditions you might use when querying a table include pattern matching and wildcard characters. Search conditions using pattern matching and wildcard characters enable you to perform searches, such as all titles that contain the word "sailing" in them or all customers whose last names begin with "C." In this topic, you will use pattern matching and wildcard characters in search conditions.

Wildcard Characters

Sometimes you won't want to search tables using exact matches. Perhaps you want to see a list of customers whose ZIP Codes begin with "151" and end in any two numbers. Or you might want to retrieve all books that begin with the letter "T." Pattern matching by using a combination of letters or numbers and wildcard characters provides you with greater flexibility when querying a table.

A wildcard character is a special character you can use in a search condition to represent certain characters. You can insert it anywhere within a search pattern to locate column values in records that contain a known sequence of characters without having to enter the entire string of characters, or when the entire set of characters is not known. There are four wildcard characters in SQL. Some of them substitute for a single character, while others substitute for an unlimited number of characters. You can use more than one wildcard character in an expression.

The following figure provides you with an example of a search condition in which a wildcard is used. In this example, the percent (%) wildcard is used to represent any number of characters. This query therefore returns all books with titles that begin with the letter "C."

```
SELECT bktitle, slprice, partnum
FROM titles
WHERE bktitle LIKE 'c%'
```

![Figure 2-22: Wildcard in SQL querying.](image)

SQL Server supports four wildcard characters for matching patterns as part of a WHERE clause in an SQL statement.

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>Any string of zero or more characters.</td>
</tr>
<tr>
<td>_</td>
<td>Any single character.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Any single character within the specified range.</td>
</tr>
<tr>
<td>[^]</td>
<td>Any single character not within the specified range.</td>
</tr>
</tbody>
</table>
Pattern Matching

Pattern matching is a method of searching for column values in a table with the help of patterns known to contain a specific combination of text or numeric characters. You can use a pattern to search for a single character or a combination of characters, and can include one or more wildcard characters. Pattern matching tests whether the specified pattern exists anywhere within the value in the specified column. Pattern matching uses the LIKE operator followed by the pattern enclosed within single quotes. Characters used in the pattern are not case sensitive.

Figure 2-23: Pattern matching in SQL.

In this example, the WHERE clause uses a search condition in which several characters are enclosed within the percent signs (‘%art%’). This query returns all books with titles that contain the letters "art" anywhere in the title. In other words, this query would return a book with the title Art Classics and also a book with the title of Smart Cooking.

You may need to combine different wildcards to obtain the results you need. In the following figure, you see a SELECT statement that searches for all customers who have an "o" as the second character in their name. The underscore character (_) is the wildcard for a single character, and the percent character (%) is the wildcard for any number of characters. With this example, a match would be found whether the customer name is "Colleen's Craft Supplies" or the more unusual "Kolleen's Kraft Supplies."

Figure 2-24: Wildcard used in the Titles table.

Syntax of the LIKE Operator

The syntax of the LIKE operator is:

```
expression [ NOT ] LIKE 'pattern'
```

The pattern should be enclosed within single quotes.

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Retrieve Data Based on String Patterns.
ACTIVITY 2–5
Retrieving Data Based on String Patterns

Before You Begin
SQL Server Management Studio is running. A query editor window is open, and the Pub1 database is selected.

Scenario
A customer has told a salesperson that he wants a list of all books that have information about art. After selecting the first book from the list of book titles, the customer wants to select a second book that he’s seen before. He thinks the title of the book started with the character A, M, or C. After seeing the list of books, he realizes that the search has to be extended for book titles whose characters range from A to G. Another customer wants to know if his name and customer ID are still available in the database. He was a customer of this company a long time ago and does not remember his customer ID; however, he remembers that it was a four-digit number, with the last digit being either 1 or 9.

1. Display the books that have the characters "art" in the title.
   a) Revise the SELECT and WHERE clauses as shown.

   ```sql
   SELECT bktitle, partnum, sprice
   FROM Titles
   WHERE bktitle LIKE '%art%'
   ```

   b) Execute the query.
   c) In the Results pane, observe the titles that have been returned.

   - SQL Server displays a list of five books with the characters "art" in their titles.
   - Some titles contain the word "Art."
   - Other titles contain "art" as part of another word, such as Starting a Small Garden and Starting a Greenhouse.

2. Display the details of books for which the titles begin with A, M, or C.

   Ask students if they can think of other ways to successfully query book titles about art.
a) Edit the pattern in the `WHERE` clause as shown.

```
SELECT bktitle, partnum, slprice
FROM Titles
WHERE bktitle LIKE '[A-M]%'
```

b) Execute the query.

c) In the **Results** pane, observe that SQL Server displays 16 books with titles that begin with A, M, or C.

---

3. Modify the query to display the details of books for which the titles begin with the characters from A to G.

a) Edit the pattern in the `WHERE` clause as shown:

```
SELECT bktitle, partnum, slprice
FROM Titles
WHERE bktitle LIKE '[A-G]%'
```

b) Execute the query.
c) In the **Results** pane, observe that books in the Results all have titles beginning with the characters from A to G.

<table>
<thead>
<tr>
<th>bktitle</th>
<th>partnum</th>
<th>slprice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Cupboards</td>
<td>39843</td>
<td>49.95</td>
</tr>
<tr>
<td>Developing Mobile Apps</td>
<td>39905</td>
<td>45.00</td>
</tr>
<tr>
<td>Boating Safety</td>
<td>40121</td>
<td>36.50</td>
</tr>
<tr>
<td>Flower Arranging</td>
<td>40323</td>
<td>20.00</td>
</tr>
<tr>
<td>English Gardens</td>
<td>40326</td>
<td>40.00</td>
</tr>
<tr>
<td>Basic Home Electronics</td>
<td>40457</td>
<td>32.29</td>
</tr>
<tr>
<td>Guide to Stereo Equipment</td>
<td>40471</td>
<td>39.98</td>
</tr>
<tr>
<td>Guide to Video Equipment</td>
<td>40472</td>
<td>39.98</td>
</tr>
<tr>
<td>Creating Toys in Wood</td>
<td>40521</td>
<td>23.79</td>
</tr>
<tr>
<td>Cross-stitching for Special Occasions</td>
<td>40522</td>
<td>20.00</td>
</tr>
<tr>
<td>Furniture Refinishing</td>
<td>40526</td>
<td>39.99</td>
</tr>
<tr>
<td>Furniture Upholstery</td>
<td>40527</td>
<td>46.95</td>
</tr>
<tr>
<td>All Kinds of Knitting</td>
<td>40561</td>
<td>26.98</td>
</tr>
<tr>
<td>Calligraphy</td>
<td>40569</td>
<td>25.25</td>
</tr>
<tr>
<td>Conversational Italian</td>
<td>40614</td>
<td>35.00</td>
</tr>
<tr>
<td>Conversational German</td>
<td>40624</td>
<td>35.00</td>
</tr>
<tr>
<td>Conversational French</td>
<td>40634</td>
<td>35.00</td>
</tr>
<tr>
<td>Conversational Japanese</td>
<td>40644</td>
<td>35.00</td>
</tr>
<tr>
<td>Conversational Chinese</td>
<td>40654</td>
<td>35.00</td>
</tr>
<tr>
<td>Decorating Wedding Cakes</td>
<td>40713</td>
<td>44.45</td>
</tr>
<tr>
<td>Chocolate Lovers Cookbook</td>
<td>40714</td>
<td>25.95</td>
</tr>
<tr>
<td>eCommerce: The Future of the Inte...</td>
<td>39908</td>
<td>30.00</td>
</tr>
<tr>
<td>eCommerce: The Future of the Inte...</td>
<td>39908</td>
<td>30.00</td>
</tr>
</tbody>
</table>

4. List the customer details from the Customers table for those who have a four-digit customer number.
   a) Switch to the query tab with the Customers query and revise it as shown. (Alternatively, you can type this query in a new query window.)

```sql
SELECT custnum, custname, city
FROM Customers
WHERE custnum LIKE '_____'
```

**Note:** The single quotes in the `WHERE` clause contain four underscores. On most keyboards, the underscore character is located on the right end of the top row of keys (approximately above the letter P). Hold **Shift** while pressing this key to get the underscore.

b) Execute the query.
c) In the Results pane, observe that SQL Server displays five customers who have four-digit customer numbers.

5. Modify the query to list customers with a four-digit customer number with the last digit being either 1 or 9.
   a) Revise the last line of the query as shown.
   
   ```sql
   SELECT custnum, custname, city
   FROM Customers
   WHERE custnum LIKE '__[19]'
   ```

   b) Execute the query.
   c) In the Results pane, observe that SQL Server displays the two customers who have four-digit customer numbers with the last digit being either 1 or 9.

6. What `WHERE` clause would match cities that do not begin with "H", but do have a second letter of "o"?
   A: `WHERE city like '(^[^H]o)'`

7. What `WHERE` clause would match cities that do not contain the letters A, B, C, or D?
   A: This can be accomplished in various ways, such as `WHERE city NOT like '%[a-d]%'` and `WHERE city NOT like '%[abcd]%'`

8. Clean up the workspace.
a) Close all **Query Editor** windows without saving the queries.
Summary

In this lesson, you narrowed the scope of your queries by performing conditional searches. Depending on the size of an organization, its production databases can consist of hundreds or thousands of rows. Executing `SELECT` statements without any search conditions retrieves all of this data and can significantly degrade the performance of the database servers. Using search conditions enables you to narrow down the scope of your queries to select only those rows you need for further business analysis.

Why do you use a condition in a query?

A: Answers will vary, but may include: conditions are used to retrieve specific information from the database. They make information retrieval easier and faster. Conditions become an important criterion in querying when the user requires only a specific part of the information.

What are the operators that you will use when you have more than one condition to be included in a query?

A: Answers will vary, but may include: you will use logical operators as well as arithmetic operators very often when more than one condition is included in a query. Depending upon the requirement, various operators like compound assignment and comparison operators can also be used.

Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.
Working with Functions

Lesson Time: 1 hour

Lesson Introduction

You have retrieved specific data from a table by using a WHERE clause. But you can do more than just display data that is in a table. You can perform calculations and other operations on data and present it in a desired format. In this lesson, you will use various functions to perform calculations on data to obtain meaningful output from the database.

Lesson Objectives

In this lesson, you will:
• Perform date calculations.
• Calculate data using aggregate functions.
• Manipulate string values in a query.
TOPIC A

Perform Date Calculations

The data present in a table can be of any number of data types, including dates and times. But dates and times may not be stored exactly the way you want to present them. Fortunately, SQL provides tools you can use to format and perform calculations on dates and times.

Date Storage

Microsoft® SQL Server® 2017 stores date information in a table using the data types datetime and smalldatetime. Dates and times are actually stored as numbers, with each day as a whole number and the time as a decimal. This makes it feasible to sort dates and use date arithmetic, such as calculating the number of days, years, or minutes between two date values. When SQL Server retrieves this information, you might want to view it in a specific date and time format other than its default format.

Functions

A function is a SQL Server object with a specified name and optional parameters that operates as a single logical unit. The parameters the function accepts can be column names or values. If the function requires multiple input parameters, you must separate the parameters with commas. The function performs a designated action and returns a result.

The following figure shows a SELECT statement that contains two functions: DATEADD() and GETDATE(). The first function, DATEADD(), enables you to add days, months, or years to an existing date. In this example, the function DATEADD(month, 3, pubdate) adds three months to the current value in the pubdate column for each row. (You might run into this scenario if you had to push back the publication date of all books published by the company.) The second function, GETDATE(), simply displays today's date and time.

![Figure 3-1: A SELECT statement displaying functions.](image)

Types of Functions

Functions can be classified as built-in functions, which cannot be modified by users, and user-defined functions, which can be created and modified by users. There are three types of functions: rowset functions, which are used to reference tables in an SQL statement; aggregate functions, which operate on a collection of values but return a single value; and scalar functions, which operate
on a single value and then return a single value. Scalar functions can be used in any valid
eexpressions. The following table describes the categories of functions SQL Server includes.

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration functions</td>
<td>Return information about configuration settings.</td>
</tr>
<tr>
<td>Cursor functions</td>
<td>Return information about the status of a cursor.</td>
</tr>
<tr>
<td>Date and time functions</td>
<td>Manipulate datetime and smalldatetime values.</td>
</tr>
<tr>
<td>Mathematical functions</td>
<td>Perform trigonometric, geometric, and other numeric operations.</td>
</tr>
<tr>
<td>Metadata functions</td>
<td>Return information on the attributes of databases and database objects.</td>
</tr>
<tr>
<td>Security functions</td>
<td>Return information about users and roles.</td>
</tr>
<tr>
<td>String functions</td>
<td>Manipulate char, varchar, nchar, nvarchar, binary, and varbinary values.</td>
</tr>
<tr>
<td>System functions</td>
<td>Operate or report on various system-level options and objects.</td>
</tr>
<tr>
<td>System statistical functions</td>
<td>Return information regarding the performance of the database server.</td>
</tr>
<tr>
<td>Text and image functions</td>
<td>Manipulate text and image values.</td>
</tr>
</tbody>
</table>

**Date Functions**

You use *date functions* to perform calculations on date columns that contain a date and time input value. These functions return a string, numeric, or date and time value. In date functions, if datetime or smalldatetime values are used, they are enclosed within single quotes.

In the following figure, you see a `SELECT` statement that uses the `DATEADD()` and `DATEDIFF()` functions. As mentioned earlier, the `DATEADD(month, 3, pubdate)` function adds three months to the value in the `pubdate` column. The `DATEDIFF(year, pubdate, '12-31-2017')` function displays the difference in years between the book’s publication date (`pubdate`) and the date `12-31-2017`. 
To demonstrate SQL Server's "day zero," today's date/time, and the number value behind today's date/time, execute this command:

```
SELECT CAST(0 AS datetime),
       GETDATE(),
       CAST(GETDATE() AS float)
```

Also point out that this is an example of a query that does not use a FROM clause.

![Figure 3-2: A SELECT statement displaying date functions.](image)

**Note:** The SET DATEFORMAT statement can be used to set the order of the dateparts (month/day/year) for entering datetime and smalldatetime values.

Date functions are scalar functions that perform operations on date and time values. The following table describes several of the date functions included with SQL Server's Transact-SQL. Other SQL databases may provide different functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATEADD(datepart, number, date)</td>
<td>Returns a new datetime value based on adding an interval to the specified date.</td>
</tr>
<tr>
<td>DATEDIFF(datepart, startdate, enddate)</td>
<td>Returns the number of date and time boundaries crossed between two specified dates.</td>
</tr>
<tr>
<td>DATENAME(datepart, date)</td>
<td>Returns a character string representing the datepart of the specified date. The datepart can be month, date, or year.</td>
</tr>
<tr>
<td>DATEPART(datepart, date)</td>
<td>Returns an integer representing the specified datepart of the specified date.</td>
</tr>
<tr>
<td>DAY(date)</td>
<td>Returns an integer representing the day datepart of the specified date.</td>
</tr>
<tr>
<td>GETDATE()</td>
<td>Returns the current system date and time of the SQL Server computer in the format specified for datetime values.</td>
</tr>
<tr>
<td>GETUTCDATE()</td>
<td>Returns the datetime value representing the current UTC (Universal Time Coordinate or Greenwich Mean Time) time. The current UTC time is derived from the current local time and the time zone setting in the operating system of the computer on which SQL Server is running.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MONTH(date)</td>
<td>Returns an integer that represents the month part of a specified date.</td>
</tr>
<tr>
<td>YEAR(date)</td>
<td>Returns an integer that represents the year part of a specified date.</td>
</tr>
</tbody>
</table>

**Date and Time Data Types**

SQL Server supports several different date and time data types. The database administrator chooses from these data types when designing and creating a table. The date and time data types vary based on the precision of the date and time values they store in the table. There are six data types for storing date and time information: date, datetime, datetime2, datetimeoffset, smalldatetime, and time.

**Data Type**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>This data type enables you to store dates in the format 'YYYY-MM-DD.'</td>
</tr>
<tr>
<td></td>
<td>Columns that use the DATE data type use three storage bytes. Supported</td>
</tr>
<tr>
<td></td>
<td>values are 0001-01-01 through 9999-12-31. The accuracy of this data type</td>
</tr>
<tr>
<td></td>
<td>is exactly one day.</td>
</tr>
<tr>
<td>TIME</td>
<td>The TIME data type enables you to store values in the 'hh:mm:ss.nnnn'</td>
</tr>
<tr>
<td></td>
<td>format. TIME supports the data range of 00:00:00.0000000 through 23:59:59.9999999.</td>
</tr>
<tr>
<td></td>
<td>The accuracy of this data type is 100 nanoseconds.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>The DATETIME data type enables you to store dates in the format 'YYYY-MM-DD</td>
</tr>
<tr>
<td></td>
<td>hh:mm:ss.nnnnn.' DATETIME uses eight bytes to store its information.</td>
</tr>
<tr>
<td></td>
<td>Supported year values are from 1753 through 9999. The DATETIME data type</td>
</tr>
<tr>
<td></td>
<td>has the accuracy of one thousandth of a second.</td>
</tr>
</tbody>
</table>
### Data Type Description

**DATETIME2**

The DATETIME2 data type also represents the date value 'YYYY-MM-DD hh:mm:ss.nnnnnn.' Values stored in the DATETIME2 data type consume six to eight bytes of storage space. DATETIME2 supports dates of January 1, 001 through December 31, 9999. The accuracy of this data type is 100 nanoseconds. Thus, the data stored in columns that use the DATETIME2 format is more accurate than that of the DATETIME format.

**DATETIMEOFFSET**

DATETIMEOFFSET represents a date value in the format 'YYY-MM-DD hh:mm:ss.nnnnnn [+ | - ] hh:mm.' It takes 8 to 10 bytes to store this information and supports the data range of January 1, 001 through December 31, 9999. The accuracy of this data type is 100 nanoseconds.

**SMALLDATETIME**

The SMALLDATETIME data type enables you to configure a column to store dates from 01-01-1900 to 12-31-2079 and times from 00:00:00 to 23:59:59. SQL Server uses four bytes to store the contents of a SMALLDATETIME column. The accuracy of the SMALLDATETIME data type is one minute.

### Conversion Between Date Types

Dates may not always be stored in the format you need to present in your output. The `CONVERT()` and `CAST()` functions are useful for converting between data types.

The `CONVERT()` function enables you to provide a numeric style value that identifies the formatting that will be applied to the value. `CAST()` is a bit more straightforward, converting between two data types, but not providing style options. The `CONVERT()` function is the more powerful of the two functions, but it is also specific to SQL Server. In general, it is best to use the more standard `CAST()` function whenever it can meet your needs, and use `CONVERT()` only when `CAST()` doesn't provide the formatting you require.

Following are some examples showing the `CAST()` function being used to convert dates to other data types.

-- Show date as a floating point number
SELECT CAST(pubdate AS float) FROM Titles
-- Example from output: 42599

-- Show date as a datetime
SELECT CAST(pubdate AS datetime) FROM Titles
-- Example from output: 2016-08-19 00:00:00.000

-- Show date as a date
SELECT CAST(pubdate AS date) FROM Titles
-- Example from output: 2016-08-19

Following are examples showing the `CONVERT()` function being used to convert date values into a varchar containing various date formats, which are specified by a particular style value.

-- Show date as a varchar containing date and time format
SELECT CONVERT(varchar, pubdate, 100) FROM Titles
-- Example from output: Aug 19 2016 12:00AM

-- Show date as a varchar containing short date-only format
SELECT CONVERT(varchar, pubdate, 6) FROM Titles
-- Example from output: 19 Aug 16

-- Show date as a varchar containing date in slash format with year first
SELECT CONVERT(varchar, pubdate, 111) FROM Titles
-- Example from output: 2016/08/19

-- Show date as a varchar containing date in slash format with year last
SELECT CONVERT(varchar, pubdate, 100) FROM Titles
-- Example from output: 08/19/2016

-- Show date as a varchar containing date in dash format with year last
SELECT CONVERT(varchar, pubdate, 110) FROM Titles
-- Example from output: 08-19-2016

Syntax of CAST()
The syntax of the CAST() function is:
CAST ( expression AS data_type [ ( length ) ] )

Syntax of CONVERT()
The syntax of the CONVERT() function is:
CONVERT ( data_type [ ( length ) ] , expression [ , style ] )
A list of style codes for CONVERT() is provided at https://docs.microsoft.com/en-us/sql/t-sql/functions/cast-and-convert-transact-sql.

The DATEPART() Function
The DATEPART() function is a date function that specifies the part of the date you want SQL Server to return, such as the year, month, day, and hour. You'll find that you will use DATEPART() as an input parameter for many of the date functions. You can use an abbreviation in date functions instead of entering the full name of the datepart.

Following is the list of abbreviations of DATEPART().

<table>
<thead>
<tr>
<th>Datepart</th>
<th>Abbreviations</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>yy, yyyy</td>
</tr>
<tr>
<td>quarter</td>
<td>qq, q</td>
</tr>
<tr>
<td>month</td>
<td>mm, m</td>
</tr>
<tr>
<td>dayofyear</td>
<td>dy, y</td>
</tr>
<tr>
<td>day</td>
<td>dd, d</td>
</tr>
<tr>
<td>week</td>
<td>wk, ww</td>
</tr>
<tr>
<td>weekday</td>
<td>dw</td>
</tr>
<tr>
<td>hour</td>
<td>hh</td>
</tr>
<tr>
<td>minute</td>
<td>mi, n</td>
</tr>
<tr>
<td>second</td>
<td>ss, s</td>
</tr>
<tr>
<td>millisecond</td>
<td>ms</td>
</tr>
</tbody>
</table>

Using SMALLDATETIME in Date Functions
The SMALLDATETIME data type is accurate only to the minute. So, when you use a date function on a column that uses the SMALLDATETIME data type, SQL Server always returns zeros for the seconds and milliseconds.
Nested Functions

When you nest a function within another function, the innermost function is evaluated first. Its output becomes an input argument for the outer function. Be sure your inner function is providing the correct data type to the outer function. For example, the query shown in the following figure returns the number of years that have transpired since the book was published.

![Query diagram](image)

**Figure 3-4: The GETDATE() function nested within the DATEDIFF() function.**

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Perform Date Calculations.
ACTIVITY 3–1
Performing Date Calculations

Scenario
Fuller & Ackerman Publishing is celebrating its 25th year in the marketplace. Management has decided to release a "golden oldies" collection of books published in the company's early years. The details of the old books are available in the "Obsolete_Titles" table. The information required is a list of books that were released between 1994 and 2003, the exact year they were released, and the age of each book.

1. Launch SQL Server Management Studio, open a new query editor, and select the Pub1 database.
   a) If SQL Server Management Studio is not running, launch it and connect to the server.
   b) On the Standard toolbar, select New Query to open the Query Editor window.
   c) On the SQL Editor toolbar, from the Available Databases drop-down list, ensure that the Pub1 database is selected.

2. Identify the column name that contains the date information in the Obsolete_Titles table.
   a) In the editor pane, type:
      `sp_help obsolete_titles`
   b) Execute the query.

   ![SqlQuery1.sql - B...OOPS\BW, Wilson (53)](Image)

   c) In the Results pane, observe that the pubdate column has `smalldatetime` as its data type.
   d) In the editor pane, delete the query `sp_help obsolete_titles`.

SQL code snippets for the course activities are stored as snippets.sql in the Solutions folder for each lesson's course data. You can use these to save yourself some typing as you lead students through the activities. Or you may want to direct students to these files if they are having trouble keeping up with the typing in class.
3. True or False? You can use the `DATEPART` function to extract the year from the published date of the book.

☑ True
☐ False

4. Enter a query to list the details of obsolete book titles, along with the year in which they were published, and the age of the book.

a) In the editor pane, enter the following query.

```sql
SELECT bktitle, pubdate,
    DATEPART(YEAR, pubdate) AS year,
    DATEDIFF(YEAR, pubdate, GETDATE()) AS age
FROM Obsolete_Titles
WHERE pubdate BETWEEN '1/1/1994' and '12/31/2003'
```

- The `SELECT` clause can be broken into multiple lines.
- It is common (but not universal) practice to break the lines as shown, indenting and starting each new line with the comma used to separate each column.
- This approach enables you to delete various lines of the `SELECT` statement without having to delete the comma from the previous line.
- It is also common practice to place the commas at the end of the line.
- The query functions the same way, regardless of which way you position the commas.

b) Execute the query.

c) In the Results pane, observe that only books published in the specified date range are displayed.

<table>
<thead>
<tr>
<th>bktitle</th>
<th>pubdate</th>
<th>year</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>When Birds Do not Fly</td>
<td>2001-04-07</td>
<td>2001</td>
<td>16</td>
</tr>
<tr>
<td>Hammer and Nails</td>
<td>1997-09-10</td>
<td>1997</td>
<td>20</td>
</tr>
<tr>
<td>Taking a Walk Alone</td>
<td>1999-03-01</td>
<td>1999</td>
<td>18</td>
</tr>
<tr>
<td>Learning to Diet</td>
<td>1997-09-15</td>
<td>1997</td>
<td>20</td>
</tr>
<tr>
<td>Wonderful Thoughts and Marvellous Dreams</td>
<td>1994-02-10 ISO 8601</td>
<td>1994</td>
<td>23</td>
</tr>
<tr>
<td>Clear Cupboards</td>
<td>1999-08-19</td>
<td>1999</td>
<td>18</td>
</tr>
</tbody>
</table>

Note: Depending on the current year (when you work through this activity), your results for age will differ from those shown here.

5. Modify the query to use just the year.
a) In the editor pane, comment out the `WHERE` clause as shown.

```
-- WHERE pubdate BETWEEN '1/1/1994' and '12/31/2003'
```

b) In the editor pane, add a new `WHERE` clause as shown.

```
WHERE DATEPART(YEAR, pubdate) BETWEEN 1994 AND 2003
```

This version of the `WHERE` clause examines just the year part of the date.

c) Execute the query.

d) In the Results pane, observe that you get the same result as before.

<table>
<thead>
<tr>
<th>bktile</th>
<th>pubdate</th>
<th>year</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>When Birds Do not Fly</td>
<td>2001-04-07 00:00:00</td>
<td>2001</td>
<td>16</td>
</tr>
<tr>
<td>Hammer and Nails</td>
<td>1997-09-10 00:00:00</td>
<td>1997</td>
<td>20</td>
</tr>
<tr>
<td>Taking a Walk Alone</td>
<td>1999-03-01 00:00:00</td>
<td>1999</td>
<td>18</td>
</tr>
<tr>
<td>Learning to Diet</td>
<td>1997-09-15 00:00:00</td>
<td>1997</td>
<td>20</td>
</tr>
<tr>
<td>Wonderful Thoughts and Marvellous Dreams</td>
<td>1994-02-10 00:00:00</td>
<td>1994</td>
<td>23</td>
</tr>
<tr>
<td>Clear Cupboards</td>
<td>1999-08-19 00:00:00</td>
<td>1999</td>
<td>18</td>
</tr>
<tr>
<td>Y2K Why Worry?</td>
<td>2000-01-01 00:00:00</td>
<td>2000</td>
<td>17</td>
</tr>
</tbody>
</table>

When you need to deal with whole years, you can use this approach.
TOPIC B

Calculate Data Using Aggregate Functions

SQL Server includes a number of functions that enable you to perform summary calculations on the numerical data present in a table. For example, you might use the \texttt{SUM} aggregate function to calculate the total of a column. In this topic, you will use aggregate functions to perform calculations on numeric columns.

Aggregate Functions

An aggregate function is a function that performs calculations on a set of values and returns a single value. The function is composed of two parts: a name that gives an indication of the calculation SQL Server performs, followed by values or references to the values, enclosed in parentheses. When SQL Server executes the query with the aggregate function, the result contains a single row with the summary information. Aggregate functions usually ignore NULL values.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{aggregate_functions.png}
\caption{A SELECT statement using aggregate functions.}
\end{figure}

This figure shows a \texttt{SELECT} statement that uses the \texttt{COUNT()} and \texttt{AVG()} aggregate functions. The \texttt{COUNT()} function enables SQL Server to count the values in a column. The \texttt{AVG()} function calculates the average of all the values in the column.

All aggregate functions operate on a collection of values but return a single, summarizing value.

\begin{table}
\centering
\begin{tabular}{|l|l|}
\hline
Aggregate Function & Description \\
\hline
\texttt{AVG(expression)} & Returns the average of values in a column. The column can contain only numeric data. Ignores NULL values (adds the non-NULL values and divides by the count of non-NULL values). \\
\texttt{COUNT(expression)}, \texttt{COUNT(*)} & Returns a count of values in a column (if you specify a column name as an expression) or of all rows in a table or group (if you specify \texttt{*}). \texttt{COUNT(expression)} ignores NULL values, but \texttt{COUNT(*)} includes them in the count. \\
\hline
\end{tabular}
\end{table}

\texttt{MAX(expression)} & Returns the highest value in a column (last value alphabetically for text data types). Ignores NULL values.
<table>
<thead>
<tr>
<th>Aggregate Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN(expression)</td>
<td>Returns the lowest value in a column (first value alphabetically for text data types). Ignores NULL values.</td>
</tr>
<tr>
<td>SUM(expression)</td>
<td>Returns the total of values in a column. The column can contain only numeric data. Ignores NULL values.</td>
</tr>
</tbody>
</table>

**Keywords**

In the following figure, both `SELECT` and `FROM` are SQL keywords you use when writing a `SELECT` statement.

![Figure 3-6: A SELECT statement displaying keywords.](image)

A *keyword* in SQL Server is a word that is reserved for defining, manipulating, and accessing data. When you enter keywords in the Query Editor window, Microsoft® SQL Server® Management Studio (SSMS) displays them in color.

```sql
SELECT custname, custnum, city
FROM customers
```

**Figure 3-6: A SELECT statement displaying keywords.**

Because a keyword has a predefined meaning in SQL, if used outside the predetermined context, you must enclose it within double quotes. For example, if the `SELECT` statement uses the SQL keyword "from" as an alias for the custnum column in the output, you must enclose it in double quotes so it won’t be confused with the `FROM` keyword.

**Note:** Keywords can also be used as identifiers or names of databases or database objects, such as tables, columns, and views.

**The DISTINCT Keyword**

You use the `DISTINCT` keyword to eliminate duplicate values in a list of values.
DISTINCT is an optional keyword you can use in two contexts: after the SELECT keyword, or inside SUM(), AVG(), and COUNT() functions. SELECT DISTINCT will return unique rows of output.

In the SUM, AVG, and COUNT functions, you can use the DISTINCT keyword to eliminate duplicate values before performing calculations. The DISTINCT keyword is always used with column names and not with arithmetic expressions.

The SELECT statement in the following figure provides you with a list of all the unique cities in which Fuller & Ackerman Publishing’s customers live.

![Figure 3-8: A SELECT clause with the DISTINCT keyword.](image)

Within the following query, the DISTINCT keyword removes any duplicate values before they are processed by the COUNT function.

```sql
SELECT DISTINCT city, state
FROM customers;
```
DISTINCT in a COUNT Function

SELECT COUNT(*) AS # Rows,
      COUNT(city) AS # Cities,
      COUNT(DISTINCT city) AS #_Unique_Cities
FROM Customers

Figure 3-9: DISTINCT removes duplicate values returned by a COUNT function.

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Calculate Data Using Aggregate Functions.
ACTIVITY 3–2
Calculating Data Using Aggregate Functions

Before You Begin
SQL Server Management Studio is running. A query editor window is open, and the Pub1 database is selected.

Scenario
Management wants to expand Fuller & Ackerman Publishing. They have decided that as part of their planning, they need to determine the amount invested in the titles published in the past and then estimate the investments they need to make for the growth of the company. They have decided to use the sales figures from the year 2017 for this analysis.
In order to perform their analysis, management would like you to retrieve the following information:
• The titles released in 2017.
• A count of the titles released, the total development cost for all books, and the average development cost for a title.

1. List all titles that were developed in 2017.
   a) Revise all lines of the query as shown.

      ```sql
      SELECT * 
      FROM Titles 
      WHERE DATEPART(YEAR, pubdate) = 2017
      ```

   b) Execute the query.
   c) In the Results pane, observe that SQL Server displays all books that were developed in 2017.

2. Display the total number of titles released, the total cost of development for all titles, and the average development cost for a title in the year 2017.
a) Revise the `SELECT` clause as shown. (You do not have to revise the `FROM` and `WHERE` clauses.)

```sql
SELECT COUNT(bktitle) AS title_count,
       SUM(devcost) AS devcost_total,
       AVG(devcost) AS devcost_average
FROM Titles
WHERE DATEPART(YEAR, pubdate) = 2017
```

Three output columns are specified, each produced by an aggregate function, and named by an alias.

b) Execute the query.

c) In the Results pane, observe that the count of titles, the sum of the development cost, and the average cost of development for a title are displayed.

<table>
<thead>
<tr>
<th>title_count</th>
<th>devcost_total</th>
<th>devcost_average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>588349.97</td>
<td>11100.9428</td>
</tr>
</tbody>
</table>

3. Find the number of unique prices.

a) Modify the `SELECT` clause as shown to observe `COUNT(*)` and `COUNT(DISTINCT).`

```sql
SELECT COUNT(*) AS #_rows,
       COUNT(bktitle) AS title_count,
       COUNT(sprice) AS price_count,
       COUNT(DISTINCT sprice) AS distinct_price_count,
       COUNT(devcost) AS devcost_count,
       SUM(devcost) AS devcost_total,
       AVG(devcost) AS devcost_average
FROM Titles
WHERE DATEPART(YEAR, pubdate) = 2017
```

b) Execute the query.

c) In the Results pane, compare the price_count to the distinct_price_count.

<table>
<thead>
<tr>
<th>#_rows</th>
<th>title_count</th>
<th>price_count</th>
<th>distinct_price_count</th>
<th>devcost_count</th>
<th>devcost_total</th>
<th>devcost_average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>56</td>
<td>56</td>
<td>31</td>
<td>53</td>
<td>588349.97</td>
<td>11100.9428</td>
</tr>
</tbody>
</table>

While there are 56 different prices, there are only 31 distinct prices.

Note: NULL devcost values are ignored (not counted) by the aggregate functions. Duplicate values are ignored (counted only once) when using `COUNT(DISTINCT column)`.
TOPIC C

Manipulate String Values

You have used functions to manipulate dates and process numeric values. SQL also includes string functions, which enable you to manipulate text.

Strings

A string is a collection of characters that you cannot use in an arithmetic calculation. SQL Server uses char, varchar, text, nchar, nvarchar, and ntext to store strings. The text and ntext data types were removed in SQL Server 2008. You can use varchar(max) and nvarchar(max) instead for very long string data.

The characters you store in these data types can be uppercase or lowercase letters, numerals, and special characters such as the "at" sign (@), ampersand (&), and exclamation point (!) in any combination. When you use string values in an expression, you must enclose them within single quotes. When SQL Server compares strings, it ignores the case of the strings.

Figure 3-10: A SELECT statement in which the WHERE clause uses a string.

String Functions

String functions are functions that perform an operation on a string input value and return a string or a numeric value. The following table describes key string functions in SQL Server.

<table>
<thead>
<tr>
<th>String Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII (character_expression)</td>
<td>Returns the ASCII code value of the leftmost character of a character expression.</td>
</tr>
<tr>
<td>CHAR (integer_expression)</td>
<td>Converts an integer ASCII code to a character.</td>
</tr>
<tr>
<td>LEFT (character_expression, integer_expression)</td>
<td>Returns the part of a character string starting at a specified number of characters from the left.</td>
</tr>
<tr>
<td>RIGHT (character_expression, integer_expression)</td>
<td>Returns the part of a character string starting at a specified number of characters from the right.</td>
</tr>
<tr>
<td>LEN (string_expression)</td>
<td>Returns the number of characters, rather than the number of bytes, of the given string expression, excluding any trailing blanks.</td>
</tr>
<tr>
<td>String Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>STR</strong> <em>(float_expression [, length [, decimal ] ])</em></td>
<td>Returns character data converted from numeric data.</td>
</tr>
<tr>
<td><strong>LOWER</strong> <em>(character_expression)</em></td>
<td>Returns a character expression after converting uppercase character data to lowercase.</td>
</tr>
<tr>
<td><strong>UPPER</strong> <em>(character_expression)</em></td>
<td>Returns a character expression with lowercase character data converted to uppercase.</td>
</tr>
<tr>
<td><strong>LTRIM</strong> <em>(character_expression)</em></td>
<td>Returns a character expression after removing leading blanks.</td>
</tr>
<tr>
<td><strong>RTRIM</strong> <em>(character_expression)</em></td>
<td>Returns a character string after truncating all trailing blanks.</td>
</tr>
<tr>
<td><strong>REPLACE</strong> <em>(character_expression)</em></td>
<td>Replaces all occurrences of the second given string expression in the first string expression with a third expression.</td>
</tr>
<tr>
<td><strong>REVERSE</strong> <em>(character_expression)</em></td>
<td>Returns the reverse of a character expression.</td>
</tr>
<tr>
<td><strong>REPLICATE</strong> <em>(character_expression, integer_expression)</em></td>
<td>Repeats a character expression for a specified number of times.</td>
</tr>
<tr>
<td><strong>SPACE</strong> <em>(integer_expression)</em></td>
<td>Returns a string of repeated spaces.</td>
</tr>
<tr>
<td><strong>STUFF</strong> <em>(character_expression, start, length, character_expression)</em></td>
<td>Deletes a specified length of characters and inserts another set of characters at a specified starting point.</td>
</tr>
<tr>
<td><strong>SUBSTRING</strong> <em>(expression, start, length)</em></td>
<td>Returns part of a character, binary, text, or image expression.</td>
</tr>
<tr>
<td><strong>UNICODE</strong> <em>(n&quot;character_expression&quot;)</em></td>
<td>Returns the integer value, as defined by the Unicode standard, for the first character of the input expression.</td>
</tr>
<tr>
<td><strong>NCHAR</strong> <em>(integer_expression)</em></td>
<td>Returns the Unicode character associated with the given integer code, as defined by the Unicode standard.</td>
</tr>
<tr>
<td><strong>SOUNDEX</strong> <em>(character_expression)</em></td>
<td>Returns a four-character (SOUNDEX) code to evaluate the similarity of two strings.</td>
</tr>
<tr>
<td><strong>DIFFERENCE</strong> <em>(character_expression, character_expression)</em></td>
<td>Returns the difference between the SOUNDEX values of two character expressions as an integer.</td>
</tr>
<tr>
<td><strong>QUOTENAME</strong> <em>(&quot;character_string&quot; [, &quot;quote_character&quot; ])</em></td>
<td>Returns a Unicode string with the delimiters added to make the input string a valid Microsoft SQL Server delimited identifier.</td>
</tr>
<tr>
<td><strong>PATINDEX</strong> <em>(&quot;%pattern%&quot;, expression)</em></td>
<td>Returns the starting position of the first occurrence of a pattern in a specified expression, or zeros if the pattern is not found, on all valid text and character data types.</td>
</tr>
</tbody>
</table>
### String Function

<table>
<thead>
<tr>
<th>String Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CHARINDEX</strong> (expression1, expression2 [, start_location])</td>
<td>Returns the starting position of the specified expression in a character string.</td>
</tr>
</tbody>
</table>

In the following figure, the `SELECT` statement uses the `LEN()` function to determine the number of characters in each customer’s name. You might use this information to help you decide whether to use a customer’s name as their login name for your website or to make sure the string is long enough before using `SUBSTRING()` or `CHARINDEX()` functions.

![Figure 3-11: A SELECT statement displaying a string function.](image)

### Case Conversion Functions

**Case conversion functions** are functions that convert the case of a string. The `LOWER` function accepts uppercase characters as the input and converts them to lowercase. The `UPPER` function accepts lowercase characters as the input and converts them to uppercase. The input parameter you provide for the case conversion function can be a value or a column name.

The following figure shows you the usage of the `UPPER()` and `LOWER()` case functions and their output. The `UPPER(custname)` function converts the value in the `custname` column to all uppercase letters. Likewise, the `LOWER(custname)` function converts the value in the `custname` column to all lowercase letters.
Figure 3-12: A SELECT statement consisting of case conversion functions.

Note: In the editor pane, you can select a literal text string and convert it by pressing Ctrl+Shift+U to convert to uppercase and Ctrl+Shift+L to convert to lowercase.

Column Leading and Trailing Spaces

Leading and trailing spaces are spaces that are present when the data stored in a char or nchar (fixed length) column is less than the maximum number of characters that the column can contain. Spaces inserted before the value are called leading spaces, whereas those inserted at the end of the value are called trailing spaces.

Even when data is stored in variable length columns such as varchar, which doesn't allow leading or trailing spaces, the query output sent to the Microsoft SQL Server Management Studio Results pane usually includes trailing spaces. You'll sometimes want to trim those spaces to make your output look better.

Figure 3-13: Leading and trailing spaces.
The Trim Functions

The trim functions enable you to remove the leading and trailing blank spaces that are part of a string of characters. You can use the LTRIM function to remove blank spaces before the value in a column and the RTRIM function to remove blank spaces after the value in a column. The TRIM function trims both ends of the value. The trim functions work only on string values.

In the following example, LTRIM(lname) removes any leading spaces from the contents of the lname field. The RTRIM(fname) function removes any trailing spaces from the fname field.

![Figure 3-14: A SELECT statement displaying trim functions.](image)

The SUBSTRING Function

SUBSTRING is the function you use to extract characters from any part of a given string. The SUBSTRING function accepts three input parameters. The first parameter can be a character string, binary string, text, an image, a column, or an expression that includes a column. You cannot use aggregate functions as expressions. The second parameter is an integer that specifies where you want the substring to begin. The third parameter is a positive integer that specifies the number of characters or bytes you want SQL Server to return.

In the following figure, you see an example of the use of the SUBSTRING() function. In the SUBSTRING(custname, 5, 10) function, SQL Server will return the string of characters that begins with the fifth character of the customer name and continues for up to 10 characters. For example, if the value in the custname field is "Chloe Community Gallery and Workshop," the SUBSTRING(custname, 5, 10) function returns the string "e Communit".

The LEFT() and RIGHT() functions return characters from the beginning or end of a string. For example, to extract three characters from the beginning of the customer name, you can use SUBSTRING(custname, 1, 3) or LEFT(custname, 3). To extract six characters from the end of the customer name, you can use SUBSTRING(custname, LEN(custname)-5, 6) or RIGHT(custname, 6).
Data Types Supported by SUBSTRING

The output of the SUBSTRING function can be character data if the expression is one of the supported character data types, binary data if the expression is one of the supported binary data types, or a string if the expression is the same type as the given expression.

Concatenation

Concatenation is the process of combining two string expressions into one string expression. Use the + (string concatenation) operator to concatenate two expressions. Both expressions must be of the same data type, or you must be able to convert one expression to the data type of the other expression.

In the following figure, the SELECT statement concatenates the sales representatives' last names with their first names (lname+fname). In the output, however, the values in the lname and fname columns include trailing spaces, which are included in the results producing a gap between the last name and the first name.
Figure 3-16: A SELECT statement displaying the concatenation operation.

You can resolve the problem of the extra spaces between the last and first names by applying the `TRIM()` function to the source values.

You might also want to use string concatenation to insert separating characters between two expressions, such as a comma and a space. The following figure shows a version of the query with the formatting improved in this way.

Figure 3-17: Concatenating column values and static values.

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Manipulate String Values.
ACTIVITY 3–3
Manipulating String Values

Before You Begin
SQL Server Management Studio is running. A query editor window is open, and the Pub1 database is selected.

Scenario
Your manager has asked for a list of customers in a format that can be easily used for addressing envelopes. You will prepare a list with three columns:

• Customer name
• Customer's street address
• City, state, and ZIP Code on a single line, with city and state separated by commas

1. List the names of customers and their addresses from the Customers table. Be sure to remove any trailing spaces from the output.
   a) In the editor pane, select the entire query and press Delete.
   b) Type a new query as shown.

   ```sql
   SELECT custname, address AS street, city + ', ' + state + ' ' + zipcode AS citystatezip
   FROM Customers
   ```

   c) Execute the query.
d) In the Results pane, observe the list of customer names and addresses.

- The city, state, and ZIP Code are displayed in a single column, instead of each component in its own column.
- Unfortunately, the city data is output with trailing spaces, which have been incorporated into the citystatezip column.
- You’ll need to trim the city values before they are concatenated into citystatezip.

2. Revise the query to trim the city values.
   a) Revise the query as shown, applying the **TRIM** function to city.

   ```sql
   SELECT custname, address AS street, 
   TRIM(city) + ', ' + state + ', ' + zipcode AS citystatezip
   FROM Customers
   ```

   b) Execute the query, and observe the results.

   The gaps following the city names have been eliminated.

3. Clean up the workspace.
a) Close all Query Editor windows without saving the queries.
Summary

In this lesson, you used functions to perform calculations on data. Functions enable you to perform tasks without having to write in-depth programs yourself. For example, aggregate functions enable you to count the number of rows that satisfy a search condition, which is something you might use if you wanted to know how many customers purchased a particular item.

Mention some instances where you would use string functions. Why would you use them?

A: Answers will vary, but may include: string functions can be used in a wide variety of instances. One example might occur when you want to remove leading or trailing spaces from the contents of a column. In this scenario, you could use the RTRIM() and LTRIM() functions.

Which function would you use most often? Why?

A: Answers will vary, but one of them might include the AVG() function. This function enables you to calculate the average of the values in a particular column.

Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

Encourage students to use the social networking tools provided on the CHOICE Course screen to follow up with their peers after the course is completed for further discussion and resources to support continued learning.
Organizing Data

Lesson Time: 1 hour, 30 minutes

Lesson Introduction

Retrieving data is the main purpose of most SQL queries. Organizing the data that appears in the result set helps you identify the information that you need instead of searching for it among the retrieved data. In this lesson, you will sort and group data so that the required output is displayed.

Lesson Objectives

In this lesson, you will:

• Sort the query output to display the result in a specific order.
• Rank data.
• Group the data displayed in the output.
• Filter grouped data.
• Summarize grouped data.
• Use PIVOT and UNPIVOT operators.
TOPIC A

Sort Data

One of the simplest and most straightforward organizational techniques you can employ is a basic sort. In this topic, you will sort the output of a query based on one or more columns.

Data Sorting

*Sorting* is a method of arranging the rows displayed in the output of a query in either ascending or descending order based on one or more column values. You can perform multiple levels of sorting with a given set of rows.

**Figure 4–1:** A table displaying the sort in ascending order by sale price.

**Figure 4–2:** A table displaying the sort in descending order by sale price.

The ORDER BY Clause

You use the **ORDER BY** clause to sort the rows in a query's result set in a specific sort order. To use this clause, enter `ORDER BY` followed by the column name on which you want to sort the output, and then the optional keyword `ASC` for ascending order or `DESC` for descending order. (By default, SQL Server sorts the output in ascending order if you don't specify an order.)

SQL Server treats the NULL values present in the columns as the lowest values. The **ORDER BY** clause, when used, is entered at the end of the SQL **SELECT** statement.

```
[ ORDER BY [colname1, colname2,...] [ ASC | DESC ] ]
```
Note: SQL Server cannot sort columns that contain ntext, text, or image data types.

You can perform multiple levels of sorting by specifying column names, one after the other, and separating them with commas.

In the following figure, you see a SELECT statement that includes an ORDER BY clause. The ORDER BY clause specifies that SQL Server should first sort the output in ascending order by the slprice column. Then, when there are two or more books that have the same price, SQL Server should sort the output by the partnum column in descending order.

![Figure 4-3: A SELECT statement displaying the ORDER BY clause along with the resultant table.](image)

Note: There is no limit to the number of columns in the ORDER BY clause, but when all these column sizes are added, the maximum capacity is 8,060 bytes.

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Sort Data.
ACTIVITY 4-1
Sorting Data

Scenario
The management of Fuller & Ackerman Publishing has set up a team to analyze their profit and loss. They require information about book titles and their prices, listing books in descending order of price. They also want the number of books that need to be sold to break even.

1. Launch SQL Server Management Studio, open a new query editor, and select the Pub1 database.
   a) If SQL Server Management Studio is not running, launch it and connect to the server.
   b) On the Standard toolbar, select New Query to open the Query Editor window.
   c) On the SQL Editor toolbar, from the Available Databases drop-down list, ensure that the Pub1 database is selected.

2. List the book titles along with their sale price for all book titles in the Titles table. Sort this list in descending order of sale price and in ascending order of book title.
   a) Enter the query as shown.

   ![SQLQuery1.sql - B...GOPS\8Wilson (51)]* - ☑
   SELECT bttitle, slprice
   FROM Titles
   ORDER BY slprice DESC, bttitle

   b) Execute the query.
   c) In the Results pane, observe that 92 book titles are displayed with their sale price displayed in descending order, and that books with the same price are sorted in ascending alphabetical order.

3. Write a query to show the break-even quantity for sales of each book.
a) Revise the `SELECT` and `ORDER BY` clauses, and add a `WHERE` clause as shown.

```
SELECT bktitle, slprice, devcost, devcost/slprice AS breakeven_point
FROM Titles
WHERE devcost IS NOT NULL
ORDER BY devcost/slprice
```

- The break-even point is the number of books that must be sold to recoup the development cost for a book.
- Divide the development cost by sale price to determine how many books must be sold to match the development cost.
- The results are sorted by the break-even calculated value.

b) Execute the query.

c) In the Results pane, observe the results.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Conversational Chinese</td>
<td>35.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>The History of Baseball</td>
<td>69.99</td>
<td>13187.06</td>
</tr>
<tr>
<td>3</td>
<td>Furniture Upholstery</td>
<td>46.95</td>
<td>13136.26</td>
</tr>
<tr>
<td>4</td>
<td>Conversational German</td>
<td>35.00</td>
<td>9972.89</td>
</tr>
<tr>
<td>5</td>
<td>Recipes From India</td>
<td>38.89</td>
<td>11105.15</td>
</tr>
<tr>
<td>6</td>
<td>Basic Home Electronics</td>
<td>32.29</td>
<td>9274.17</td>
</tr>
<tr>
<td>7</td>
<td>Minor Home Repairs Made Easy</td>
<td>45.95</td>
<td>12588.89</td>
</tr>
<tr>
<td>8</td>
<td>Furniture Refinishing</td>
<td>39.99</td>
<td>11642.95</td>
</tr>
<tr>
<td>9</td>
<td>Recipes From Italy</td>
<td>39.99</td>
<td>11772.73</td>
</tr>
</tbody>
</table>

Each book title is shown along with its sale price, development cost, and the number of copies that must be sold to recoup the development cost.
TOPIC B

Rank Data

You have sorted data, which helps to reveal relationships between the values in different rows. You can provide additional insights through SQL's various ranking functions.

The Ranking Functions

*Ranking functions* are functions that sequentially number the rows in a result set based on the partitioning and ordering of the rows. Depending on the ranking function you use in the query, some of the rows might get the same rank value as other rows. A ranking function is always followed by the *OVER* clause, which determines the partitioning and ordering of the rows before SQL Server applies a ranking function. The *OVER* clause is supported by the *PARTITION BY* clause, which determines how rows are grouped for ranking, and the *ORDER BY* clause, which determines the order of rows within each partition.

The following query uses the ranking functions to accomplish these tasks:

- The *RANK()* function ranks the rows by the values in the `qty` column.
- The *DENSE_RANK()* function ranks the rows by the values in the `qty` column but without gaps in the ranking values.
- The *NTILE()* function distributes the rows in the output into the specified number of groups (five in the example).
- The *ROW_NUMBER()* function assigns a unique row number to each row in the output.

![Figure 4-4: A SELECT statement displaying ranking functions.](image)

**Syntax of a Ranking Function**

The syntax of a ranking function is:

```sql
Ranking Function () OVER ([PARTITION BY value_expression, ... [n]] ORDER BY <column> [ASC | DESC], ... [n]))
```
The RANK Function

The RANK function is a ranking function that returns a ranking value for each row in a result set. The rank values returned by the RANK function are not continuous. If two or more rows of a table have the same value, SQL Server assigns them the same rank value. In such a case, the ranking value increases as specified by the ORDER BY clause.

In the following figure, you see a SELECT statement that displays the repid, qty, and custnum columns in the output. In addition, SQL Server assigns a ranking to the rows based on the quantity (qty) column. Notice that SQL Server assigns a tie to rows in which the quantity is the same.

**Figure 4–5: A SELECT statement displaying the RANK function.**

Syntax of the RANK Function

The syntax of the RANK function is:

```
RANK () OVER (< partition_by_clause> [order_by_clause])
```

The DENSE_RANK Function

The DENSE_RANK function is a ranking function that performs a task similar to that of the RANK function, but it does not produce gaps in the rank values. Instead, this function consecutively ranks each unique ORDER BY value.
Figure 4-6: A SELECT query displaying the DENSE_RANK function.

Syntax of the DENSE_RANK Function

The syntax of the \texttt{DENSE\_RANK} function is:

\begin{verbatim}
DENSE\_RANK () OVER ([< partition_by_clause>] <order_by_clause>)
\end{verbatim}

The \texttt{ROW\_NUMBER} Function

The \texttt{ROW\_NUMBER} function is a ranking function that uses an \texttt{ORDER BY} clause and a unique partition value to return a result set, which consists of sequential numbers for each row set. The row number is subject to change according to the rows in the output.

Figure 4-7: A SELECT statement displaying the \texttt{ROW\_NUMBER} function.
Syntax of the ROW_NUMBER Function

The syntax of the ROW_NUMBER function is:

```
ROW_NUMBER () OVER ( [ < partition_by_clause> ] < order_by_clause> )
```

The NTILE Function

The NTILE function is a ranking function that divides the rows in each partition of a result set into a specified number of groups based on a given value and ranks them according to the partition. The NTILE function contains an integer expression as its main argument, which specifies the number of groups into which each partition will be divided. The rows in the result set will be divided evenly among the partitions, but when the number of rows in the result set does not divide exactly into the number of partitions, the rows are distributed in such a way that the larger groups appear first in the result set.

```
SELECT repid, qty, custnum,
       NTILE (integer_expression) OVER ( [ < partition_by_clause> ] < order_by_clause> ) AS 'Ntile'
FROM sales
```

**Figure 4-8: A SELECT statement displaying the NTILE function.**

Syntax of the NTILE Function

The syntax of the NTILE function is:

```
NTILE (integer_expression) OVER ( [ < partition_by_clause> ] < order_by_clause> )
```

The TOP n Keyword

You can use the TOP n keyword with an ORDER BY clause to select a specific number or percentage of rows in the output. For example, you might use the TOP n keyword to select the top 10 sales orders (by quantity) in the Sales table. Note that you must include an ORDER BY clause in the SELECT statement for the TOP n keyword to work properly.

In the following query, SQL Server lists rows from the Sales table in descending order by quantity. Because the query specifies TOP 10 in the SELECT statement, SQL Server lists only the first 10 rows in the sorted output.
Figure 4–9: A SELECT statement using the TOP n keyword.

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Rank Data.
ACTIVITY 4-2  
Ranking Data

Before You Begin
SQL Server Management Studio is running. A query editor window is open, and the Pub1 database is selected.

Scenario
Management of Fuller & Ackerman Publishing has set up a team to analyze the performance of sales representatives based on their sales during 2017. Even though they have a report containing the total sales quantities for 2017, they want to list the representatives based on the sales made by them during that year. You would like to use ranking functions to list the representatives based on the quantity of books they sold in 2017.

1. Show all sales results for 2017.
   a) Enter a query to show sales results for 2017.

   
   ```sql
   SELECT repid, qty, custnum
   FROM Sales
   WHERE DATEPART(YEAR, sldate) = 2017
   ```

   b) Execute the query and observe the results.

   There are 94 rows in the results.

   Next, you will add a column that partitions the sales representatives into five ranking groups, based on their sales quantities.

2. Add a custom field based on the RANK function.
a) Revise the query, inserting a new second line as shown.

```
SELECT repid, qty, custnum,
       RANK() OVER(PARTITION BY repid ORDER BY qty DESC) AS 'Rank'
FROM Sales
WHERE DATEPART(year, sdate) = 2017
```

b) Execute the query.

c) Scroll through the results, and observe how they are partitioned and ordered.

- Because you partitioned by repid, rows for the same repid are clustered together.
- Because you ordered by qty, rows within each partition are shown in order by the quantity sold.
- The Rank column shows the rank order of each sales quantity. For example, 500 is the top ranked sales quantity (rank of 1), and 0 is the lowest ranked sales quantity (rank of 22).

3. Add a custom field based on the DENSE_RANK function.

a) Revise the query, inserting a new third line as shown.

```
SELECT repid, qty, custnum,
       RANK() OVER(PARTITION BY repid ORDER BY qty DESC) AS 'Rank',
       DENSE_RANK() OVER(PARTITION BY repid ORDER BY qty DESC) AS 'Dense Rank'
FROM Sales
WHERE DATEPART(year, sdate) = 2017
```

b) Execute the query.
c) Compare the numbering of the Dense Rank column to the Rank column.

Rows are ranked in the same order, but `DENSE_RANK` does not skip ranking values when there are multiple rows with the same rank, making it easier to understand the ranking.

4. Divide the rows into five groups and add row numbers.
   a) Revise the query, inserting a new fourth and fifth line as shown.

b) Execute the query.
c) Examine the values shown in the Row Number column.

- While row numbers seem redundant in the numbered table provided in the Results pane, it can be useful to have row numbers directly in the SQL output.
- If you plan to present the data in a table (on a website or in a mobile app, for example) that users can sort (by selecting a column header, for example), it may be useful to have a row number showing the original sort order.

d) On the ribbon, select the Results to Text button to switch to plain text output. Alternatively, you can press Ctrl+T.
e) Execute the query.
f) In the Results pane, scroll and observe the text-only results.

- This is how a custom application or reporting tool would normally receive the SQL results.
- SQL Server Management Studio adds its own row numbers, but you may have situations where you need to add them directly to the content.

5. Examine the Ntile values.

a) On the ribbon, select the Results to Grid button to switch back to grid output. Alternatively, you can press Ctrl+G.

The next time you run a query, it will be shown in grid format.
b) Execute the query.
c) Scroll and observe the Ntile column data.

The values produced by Ntile break the ranked values up into the number of groups you specify (five, in this query). So rows with an Ntile value of 1 are in the top fifth of rankings. Those with an Ntile value of 2 are in the second fifth, and so on.
TOPIC C

Group Data

There are times when you need to summarize or "roll up" groups of data, rather than show the detail of every row. For example, you might want to show a summary of total sales for each sales rep, rather than list each sale separately. SQL provides various ways to accomplish this.

Groups

A group is a collection of two or more records combined into one unit based on the values in one or more columns. Summarized records present in each row of the group are listed together as a single row in the output. SQL Server does not sort the groups in any order, but does sort the records within the group in ascending order.

Figure 4–10: A table displaying grouped records.

The GROUP BY Clause

GROUP BY is a clause you use to group two or more rows displayed in the output based on one or more columns.

The GROUP BY clause is followed by a column or a non-aggregate expression that references a column.

Any columns you specify directly in the SELECT statement (that is, those that you do not use in an aggregating function) must also be included in the GROUP BY clause. In most cases, you use the GROUP BY clause to enable you to perform a calculation on the group.

In the following figure, SQL Server groups the rows in the sales table by sales representative ID. It then calculates the total quantity of the sales for the group. In other words, this query enables you to determine the total sales quantities for each sales representative.
Figure 4-11: A SELECT statement using the GROUP BY clause.

If you do not include an ORDER BY clause in the query, SQL Server returns the groups for the GROUP BY clause in no particular order. To retrieve a sorted output, include the ORDER BY clause in the query.

Syntax of the GROUP BY Clause

The syntax of the GROUP BY clause is:

```
[ GROUP BY [group_by_expression1, group_by_expression2,..] ]
```

Specifications for Using the GROUP BY Clause

A GROUP BY clause must comply with certain specifications when written as part of a SELECT statement.

<table>
<thead>
<tr>
<th>Specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multilevel groups</td>
<td>You can form multilevel groups by entering columns separated by commas. The number of levels is limited by the size of the data stored in the column, aggregate columns, and aggregate values involved in the query.</td>
</tr>
<tr>
<td>Aggregate functions</td>
<td>If you use aggregate functions in the SELECT clause, SQL Server calculates the summary values after the groups are formed.</td>
</tr>
<tr>
<td>Non-aggregate lists</td>
<td>When a non-aggregate list of values is entered in the SELECT clause, each of the non-aggregate values must be included in the GROUP BY list.</td>
</tr>
<tr>
<td>NULL values</td>
<td>If the column on which the group is formed contains NULL values, they are all placed in a single group.</td>
</tr>
<tr>
<td>Column aliases</td>
<td>The column alias used in the SELECT clause cannot be used to specify a grouping column.</td>
</tr>
</tbody>
</table>
Query Grouping Sets

Grouping sets allow you to define multiple groupings within a single query. These grouping sets are introduced as extensions to the GROUP BY clause. These extensions can include the CUBE and ROLLUP subclauses and the GROUPING_ID function. However, these grouping sets define their own purpose, without which a single query defines only one grouping set in the GROUP BY clause. The grouping set statement generates a result that is equivalent to the result set generated by using the GROUP BY, ROLLUP, or CUBE operation. These grouping sets show better performance because they execute a single query for multiple groupings.

```
SELECT custnum, repid, SUM(qty) AS total_sales
FROM Sales
WHERE DATEPART(year, sldate) = 2012 AND
DATEPART(MONTH,sldate) BETWEEN 1 AND 6
GROUP BY GROUPING SETS((custnum,repid),(repid))
```

Figure 4–12: A SELECT statement displaying grouping sets.

The GROUPING_ID function helps you identify those grouping sets that each result row belongs to. In order to execute this function, you must provide all attributes that are involved in the grouping set as the input. The result of the GROUPING_ID function is an integer result represented as a bitmap, in which each bit represents a different attribute. Therefore, this function produces a unique integer for each of the grouping sets.

```
SELECT GROUPING_ID(ordnum,custnum)
FROM sales
GROUP BY CUBE(ordnum, custnum);
SELECT GROUPING_ID(ordnum,custnum)
FROM sales
GROUP BY ROLLUP(ordnum, custnum);
```

Example of the GROUPING SETS Subclause

The following code illustrates the use of the GROUPING SETS subclause.

```
SELECT partnum,bktitle
FROM titles
GROUP BY GROUPING SETS ((partnum,bktitle));
```

CUBE and ROLLUP Subclauses

The CUBE and ROLLUP subclauses are shortcuts to the predefined GROUPING SETS specifications. More precisely, they are abbreviations to the GROUPING SETS subclause. Each of these subclauses has unique operations associated with it. The CUBE subclause generates all possible grouping sets obtained from the elements listed in parentheses. This also includes the empty grouping set.
However, the result obtained from the `CUBE` subclause is large because the results are exponential in number. The `CUBE` subclause is beneficial for complex data analysis.

```
CUBE(custid,repid,bktitle);
```

is equivalent to

```
GROUPING SETS((custid,repid,bktitle),(custid,bktitle),
(repid,bktitle),(custid,repid),(custid),(repid),(bktitle),())
```

**Figure 4-13: The CUBE and GROUPING SETS subclauses.**

The `ROLLUP` subclause, on the other hand, produces a hierarchical series of grouping sets. The `ROLLUP` subclause is used with the `ROLLUP` keyword to specify the hierarchy of grouping attributes. This subclause returns \( n+1 \) grouping sets for \( n \) elements in a hierarchical manner.

```
ROLLUP(custid,repid,bktitle)
```

is equivalent to

```
GROUPING SETS((custid,repid,bktitle),(custid,repid),(custid),())
```

**Figure 4-14: The ROLLUP and GROUPING SETS subclauses.**

**Example of the CUBE Subclause**
The given example code illustrates the CUBE subclause. Consider the columns from the Pub1 database.

```
CUBE(custid,repid,tsales);
```

is equivalent to

```
GROUPING SETS((custid,repid,tsales),(custid,tsales),(repid,tsales),
(custid,repid),(custid),(repid),(tsales),())
```

**Example of the ROLLUP Subclause**
The given example code illustrates the ROLLUP subclause. Consider the columns from the Pub1 database.

```
ROLLUP(custid,repid,tsales)
```

is equivalent to

```
GROUPING SETS((custid,repid,tsales),(custid,repid),(custid),())
```

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Group Data.
ACTIVITY 4–3
Grouping Data

Before You Begin
SQL Server Management Studio is running. A query editor window is open, and the Pub1 database is selected.

Scenario
You must summarize the total sales made by each representative over a period of one year.

1. Show all 2017 sales results.
   a) Delete the four rows of calculated columns so the query appears as shown.

   ```sql
   SELECT repid, qty, custnum
   FROM Sales
   WHERE DATEPART(YEAR, sldate) = 2017
   ``

   b) Add an `ORDER BY` clause as shown.

   ```sql
   SELECT repid, qty, custnum
   FROM Sales
   WHERE DATEPART(YEAR, sldate) = 2017
   ORDER BY repid
   ```

   This will keep the rows for each rep sorted together.
c) Execute the query and scroll as you examine the result.

<table>
<thead>
<tr>
<th>Results</th>
<th>Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>repid</td>
<td>qty</td>
</tr>
<tr>
<td>20</td>
<td>E01</td>
</tr>
<tr>
<td>21</td>
<td>E01</td>
</tr>
<tr>
<td>22</td>
<td>E01</td>
</tr>
<tr>
<td>23</td>
<td>E02</td>
</tr>
<tr>
<td>24</td>
<td>E02</td>
</tr>
<tr>
<td>25</td>
<td>E02</td>
</tr>
<tr>
<td>26</td>
<td>E02</td>
</tr>
<tr>
<td>27</td>
<td>E02</td>
</tr>
<tr>
<td>28</td>
<td>E02</td>
</tr>
<tr>
<td>29</td>
<td>E02</td>
</tr>
<tr>
<td>30</td>
<td>E01</td>
</tr>
<tr>
<td>31</td>
<td>E01</td>
</tr>
</tbody>
</table>

- There are multiple rows for each sales representative.
- Now you need to modify the query so a single summary row is shown for each sales representative.

Note: The image in step 1c shows part of the results you'll see. You'll need to scroll to see all rows in the results.

2. Attempt to group the results into a single row for each sales representative.
   a) Change ORDER BY to GROUP BY as shown.

   ```sql
   SELECT repid, qty, custnum
   FROM Sales
   WHERE DATEPART(year, sldate) = 2017
   GROUP BY repid
   ```

   b) Execute the query and examine the result.

   An error message is shown.

   ```
   Msg 8120, Level 16, State 1, Line 1
   Column 'Sales.qty' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.
   ```
   - This message may not make the problem clear to you.
   - In this case, you're trying to group the results for each sales rep into a single row, and qty contains different values for each row.
   - You need to summarize qty somehow—using one of the aggregate functions (like SUM), for example.
   - No problem was reported for repid because you're grouping by that column, so every row in that group contains the same repid.
   - An error message wasn't shown for the custnum column (which has the same problem) because the error produced by the qty column was serious enough to stop the query before the custnum column could be evaluated.
3. Add an aggregate function.
   a) Revise the first line of the query as shown.

   ```sql
   SELECT repid, COUNT(DISTINCT custnum) AS #_Cust
   FROM Sales
   WHERE DATEPART(year, sldate) = 2017
   GROUP BY repid
   ```

   • Enclosing custnum in a `COUNT` function returns the number of unique customers sold to by each sales rep.
   • `COUNT(ordnum)` and `COUNT(custnum)` would each return the same result: the number of sales rows per sales rep.

   b) Execute the query and observe the results.

<table>
<thead>
<tr>
<th>repid</th>
<th>#_Cust</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>E01</td>
</tr>
<tr>
<td>2</td>
<td>E02</td>
</tr>
<tr>
<td>3</td>
<td>N01</td>
</tr>
<tr>
<td>4</td>
<td>N02</td>
</tr>
<tr>
<td>5</td>
<td>S01</td>
</tr>
<tr>
<td>6</td>
<td>S02</td>
</tr>
<tr>
<td>7</td>
<td>S03</td>
</tr>
<tr>
<td>8</td>
<td>W01</td>
</tr>
</tbody>
</table>

   • Results for eight reps are shown.
   • Results are grouped for each sales rep, showing the number of unique customers they sold to during the year.
TOPIC D

Filter Grouped Data

When you group data, all rows that form part of that group are listed together. You can further filter grouped data by adding conditions. In this topic, you will filter grouped data using aggregate functions.

The HAVING Clause

HAVING is a clause you can use to specify a search condition based on an aggregate value. (Use a WHERE clause if you want to search based on one of the columns in the SELECT statement.) You use the HAVING clause with the GROUP BY clause. After SQL Server groups and aggregates the data, it applies the conditions in the HAVING clause.

In the following figure, the GROUP BY repid clause specifies that SQL Server should group the output rows based on the sales representative IDs. SQL Server then calculates the total sales for each sales representative based on the aggregate function SUM(qty) in the SELECT statement. Finally, the HAVING clause restricts the output to only the sales representatives who have sold a total quantity of 2000 or more.

Both the HAVING clause and the WHERE clause enable you to filter the results SQL Server displays in the output of a query. But these clauses differ in what they can filter. The WHERE clause enables you to filter data coming in to the query based on any of the columns in the SELECT statement. In contrast, the HAVING clause can filter based only on the output of an aggregate function.

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Filter Grouped Data.
ACTIVITY 4–4
Filtering Grouped Data

Before You Begin
SQL Server Management Studio is running. A query editor window is open, and the Pub1 database is selected.

Scenario
Fuller & Ackerman Publishing wants to increase by 10 percent the production of books that have sold 2000 copies or more. The publishing department requires the part numbers of these books so that they can increase the production of the specified books. As an incentive, management decided to provide a bonus to representatives who have sold 2000 copies. The financial department requires the IDs of these representatives to credit the bonus into their salary account.

1. Add a HAVING clause to show only representatives who sold at least 2000 books.
   a) Revise the query, adding a HAVING clause as the last line of the query, as shown.

   ```sql
   SELECT rapid, COUNT(DISTINCT custnum) AS #_Cust
   FROM Sales
   WHERE DATEPART(year, sldate) = 2017
   GROUP BY rapid
   HAVING SUM(qty) >= 2000
   ```

   • The database server will process the HAVING clause after FROM, WHERE, GROUP BY, and any aggregating functions such as the SUM function in the SELECT clause.
   • Only rows with a total sales quantity greater than 2000 will be shown.

2. Execute the query and examine the results.
   a) Execute the query.
   b) Examine the results.

   • Only five reps had annual sales totals greater than 2000.
• Before you added the `HAVING` clause, eight reps were shown.
• If you wanted to sort the results by the number of customers sold to, you could add `ORDER BY #_Cust` as the final clause of the `SELECT` statement.
TOPIC E

Summarize Grouped Data

After you have grouped data, you may find that creating additional subgroups in the grouped data will be helpful. In this topic, you will summarize grouped data.

The CUBE and ROLLUP Operators

CUBE and ROLLUP are operators that are used to display summary rows along with the rows displayed by the GROUP BY clause. You enter the CUBE or ROLLUP operator after the GROUP BY clause. In the result, the left column value of the summary row is displayed as NULL and the right column value contains the summary value.

When you use the CUBE operator, the number of columns listed in the GROUP BY clause determines the number of summary rows displayed in the output. A summary row is returned for every group and subgroup in the output. So, the number of rows in the output is the same, regardless of the order in which grouping columns are specified.

![Figure 4-16: A SELECT statement displaying the CUBE operator.]

When you use the ROLLUP operator, groups are summarized in hierarchical order, from the lowest level in the group to the highest. The group hierarchy is determined by the order in which grouping columns are specified. Changing the order of grouping columns can affect the number of rows displayed in the output.
The ROLLUP Operator

```
SELECT repid, custnum, SUM(qty) AS Annual_Total
FROM Sales
WHERE DATENAME(MONTH, sldate)='March'
GROUP BY repid, custnum WITH ROLLUP
ORDER BY repid, custnum
```

**Figure 4–17: A SELECT statement displaying the ROLLUP operator.**

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Summarize Grouped Data.
**ACTIVITY 4–5**

**Summarizing Grouped Data**

**Before You Begin**
SQL Server Management Studio is running. A query editor window is open, and the Pub1 database is selected.

**Scenario**
To break down how each sales representative is making their sales quotas for the year, you’ll summarize each sales representative’s sales to each customer.

**1. Display the representative IDs and total sales made by each representative.**
   a) Delete the **HAVING** clause, and revise the **SELECT** and **GROUP BY** clauses as shown.

   ```sql
   SELECT repid, custnum, SUM(qty) AS annual_total
   FROM Sales
   WHERE DATEPART(year, sldate) = 2017
   GROUP BY repid, custnum
   ```

   These changes will:
   • Display the custnum column, as well as a sum of sales quantities in the annual_total column.
   • Add a subgroup by custnum to the **GROUP BY** clause.
b) Execute the query, and observe 17 rows with subtotals for each repid and custnum.

2. Roll up the results.
   a) Add **WITH ROLLUP** to the **GROUP BY** clause as shown.

   ```sql
   SELECT repid, custnum, SUM(qty) AS annual_total
   FROM Sales
   WHERE DATEPART(year, sldate) = 2017
   GROUP BY repid, custnum **WITH ROLLUP**
   ```

   The query had already been grouped by repid and subgrouped by custnum. With these revisions, the query will also display a rollup to show the grand total of sales made by all representatives.

   b) Execute the query.
c) Examine the query results, scrolling down to view all of the rows.

<table>
<thead>
<tr>
<th>repid</th>
<th>custnum</th>
<th>annual_total</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>S01</td>
<td>20309</td>
</tr>
<tr>
<td>14</td>
<td>S01</td>
<td>20506</td>
</tr>
<tr>
<td>15</td>
<td>S01</td>
<td>20557</td>
</tr>
<tr>
<td>16</td>
<td>S01</td>
<td>21133</td>
</tr>
<tr>
<td>17</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>18</td>
<td>S02</td>
<td>20482</td>
</tr>
<tr>
<td>19</td>
<td>S02</td>
<td>NULL</td>
</tr>
<tr>
<td>20</td>
<td>S03</td>
<td>20330</td>
</tr>
<tr>
<td>21</td>
<td>S03</td>
<td>20512</td>
</tr>
<tr>
<td>22</td>
<td>S03</td>
<td>21151</td>
</tr>
<tr>
<td>23</td>
<td>S03</td>
<td>NULL</td>
</tr>
<tr>
<td>24</td>
<td>W01</td>
<td>20417</td>
</tr>
<tr>
<td>25</td>
<td>W01</td>
<td>NULL</td>
</tr>
<tr>
<td>26</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

- The results show the IDs of representatives and the customers to whom they have sold books, along with the total sales to each customer.
- A summary row (with a custnum of NULL) totals the sales for each representative ID.
- The last row (repid and custnum of NULL) shows the grand total of sales made by all representatives.
TOPIC F

Use PIVOT and UNPIVOT Operators

You have summarized data using the CUBE and ROLLUP operators. After you have summarized data, you may wish to rotate column values into multiple columns so you can use aggregate functions on any columns in the output. You might also want to rotate those columns into column values again. In this topic, you will use PIVOT and UNPIVOT relational operators to accomplish such tasks. Pivoted output is sometimes called cross-tabulation or spreadsheet-style output.

The PIVOT and UNPIVOT Operators

The PIVOT operator is a relational operator that is used to rearrange the related columns and values of a table in the output of a query.

The PIVOT relational operator rotates unique values from one column of a table into multiple columns in the output. This rotation enables SQL Server to perform aggregate functions on the columns and display the resultant data in a pivoted table.

Syntax of the PIVOT Operator

The syntax of the PIVOT operator is:

```sql
SELECT [non-pivoted column] AS <column name>, ...[last pivoted column] AS <column name>
FROM (SELECT query that produces data) AS <alias for the source query>
PIVOT (<aggregate function> (column being aggregated)
FOR [column that contains the values that will become column headers] IN ([first pivoted column], ....[last pivoted column]))
AS <alias for the pivot table>
<optional ORDER BY clause>
```
The **UNPIVOT** operator does the opposite of the **PIVOT** operator by rotating multiple columns into the values of a single column. The **UNPIVOT** operator does not perform the exact reverse of the **PIVOT** operator, because the **UNPIVOT** operator rearranges pivoted column values into columns having new headings and new data.

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Use the PIVOT and UNPIVOT Operators.
ACTIVITY 4–6
Using PIVOT

Before You Begin

SQL Server Management Studio is running. A query editor window is open, and the Pub1 database is selected.

Scenario

You are looking for patterns of monthly sales by sales representatives over the period of January through June. You will write a query to summarize the quantities sold by each sales representative each month. You will pivot the data to display each month in a separate column.

Figure 4–19: Table showing six months of total sales for each representative.

1. Create a simple query to produce the columns of data you’ll need.
   a) Revise the query as shown.

   ```sql
   SELECT LEFT(DATENAME(month, sldate), 3) AS mo, qty, repid
   FROM Sales
   ```
b) Execute the query, and observe the results.

<table>
<thead>
<tr>
<th>mo</th>
<th>qty</th>
<th>repid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nov</td>
<td>220</td>
<td>N01</td>
</tr>
<tr>
<td>Nov</td>
<td>100</td>
<td>N02</td>
</tr>
<tr>
<td>Nov</td>
<td>170</td>
<td>N02</td>
</tr>
<tr>
<td>Dec</td>
<td>100</td>
<td>N02</td>
</tr>
<tr>
<td>Dec</td>
<td>150</td>
<td>N01</td>
</tr>
<tr>
<td>Dec</td>
<td>200</td>
<td>N02</td>
</tr>
<tr>
<td>Jan</td>
<td>200</td>
<td>W01</td>
</tr>
<tr>
<td>Jan</td>
<td>250</td>
<td>N02</td>
</tr>
<tr>
<td>Jan</td>
<td>250</td>
<td>S03</td>
</tr>
<tr>
<td>Jan</td>
<td>100</td>
<td>N02</td>
</tr>
</tbody>
</table>

Each sale is shown with:
- The month in which the sale occurred (shown as just the first three characters of the month)
- The quantity of items sold
- The representative who sold them

You would like to show the total sales quantity for each representative each month, rather than showing each individual sale. You can accomplish this using a \textit{Pivot} relational operator. But first you have to prepare the results of this query to be used as the source for another query, which you will use to actually perform the pivot operation.

2. Prepare the query to serve as the input source for another query.
   a) Highlight both lines of the query, and press \textbf{Tab} once to indent them.
   b) Insert lines before and after the original query, to wrap it in a new outer query, as shown:

   ```sql
   SQLQuery1.sql - B...GOPSP\BWilson (52)*)
   
   SELECT * FROM (SELECT LEFT(DATENAME(month, sldate), 3) AS mo, qty, repid
   FROM Sales)
   AS source
   ```

   - The new outer query uses your previous query as its \textit{FROM} source.
   - The outer query selects all columns from the source table, and doesn't apply a \textit{WHERE} filter, so at this point, it should simply return the same results as the inner query.
   - You've used an alias to name the results of this query as \textit{source}.
   - The inner query is a subquery and must be enclosed by parentheses.
   c) Execute the query.

   The results are the same as the previous results. At this point, the outer query doesn't add much, but you will now apply a pivot operation to change the way the sales data is presented in the query results.

3. Pivot the query results.
a) Apply a pivot operation to the outer query as shown.

The pivot:
- Uses the `SUM` aggregation function to total the `qty` values for each month.
- Specifies `mo` (the month) as the column that contains values to be used as column headers.
- Specifies the first six months as the columns to be pivoted.
- Is given the alias `pivoted` to name the resulting (pivoted) table.

b) Execute the query, and examine the results.

The data has been pivoted. Months are now used as column headers.

4. Exit SQL Server Management Studio without saving the query.

Note: If you want to review the query later, you can save it to a file.
Summary

In this lesson, you organized the data obtained from a query. By using the ability to organize data, you can easily find the information you need in your queries' result sets.

When and why might you enable sorting options?
A: Answers will vary, but may include: sorting can be enabled to get a listing of data in ascending or descending order to facilitate data analysis or to view the results alphabetically.

When might you group the data in a table?
A: Answers will vary, but may include: grouping data will facilitate better analysis, such as viewing the total sales of a company grouped region-wise or city-wise.

Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

Encourage students to use the social networking tools provided on the CHOICE Course screen to follow up with their peers after the course is completed for further discussion and resources to support continued learning.
Retrieving Data from Multiple Tables

Lesson Time: 1 hour, 30 minutes

Lesson Introduction

Up to now, you have been retrieving and organizing data from a single table using various clauses and functions in SQL. In this lesson, you will retrieve data from multiple tables.

Lesson Objectives

In this lesson, you will:

• Use the UNION operator to combine results from two queries into a single output.
• Compare results of two queries to get distinct values.
• Retrieve data by joining tables.
TOPIC A

Combine the Results of Two Queries

Sometimes, you might need to pull information from multiple different tables and combine the results in a single, unified result set. One way to accomplish this is using the UNION operator.

The UNION Operator

The UNION operator enables you to combine the results of two or more queries into a single output. You enter the UNION operator between two SELECT SQL statements. The number of columns in each SELECT query must be identical. In each query, the data type of the respective columns must be compatible. By default, when you use the UNION operator, SQL Server removes duplicate rows from the result set. To display these duplicate rows, use the ALL keyword after the UNION operator.

In the following figure, you see a query that combines the results of two SELECT statements to generate a single output. This query provides you with a list of all books from both the Titles and Obsolete_Titles tables in a single result set. Note that the syntax meets the following requirements:

- Both SELECT statements specify the same number of columns, and the columns have the same data types.
- The ALL keyword after the UNION operator specifies that SQL Server should list all rows in both tables, including duplicates.

![Figure 5-1: Two SELECT statements using the UNION ALL operator.](image)

Syntax of the UNION Operator

The syntax of the UNION operator is:
SELECT column_name(s) 
FROM table1 UNION [ALL] 
SELECT column_name(s) FROM table2;

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Combine the Results of Two Queries.
ACTIVITY 5-1
Combining the Results of Two Queries

Scenario
To clinch a business deal, management wants an alphabetical list of all books that have been published from the day the company came into existence. Some of the books that were published are currently out of print, and this information is found in the Obsolete_Titles table. The other books published are in the Titles table.

1. Launch SQL Server Management Studio, open a new query editor, and select the Pub1 database.
   a) If SQL Server Management Studio is not running, launch it and connect to the server.
   b) On the Standard toolbar, select New Query to open the Query Editor window.
   c) On the SQL Editor toolbar, from the Available Databases drop-down list, ensure that the Pub1 database is selected.

2. List the part number, book title, and publishing date of all books in the Titles table.
   a) Enter the query as shown.

```
SELECT partnum, bktile, pubdate 
FROM Titles
```

   b) Execute the query, and examine the results.

   92 books are listed in three columns.

3. List the part number, book title, and publishing date of all books in the Obsolete_Titles table.
   a) Press Enter twice to insert a blank line after the previous query.

```
SELECT partnum, bktile, pubdate 
FROM Obsolete_Titles
```
b) Type a second query as shown.

```
SELECT partnum, bktitle, pubdate
FROM Titles
```

```
SELECT partnum, bktitle, pubdate
FROM Obsolete_Titles
```

c) Execute the queries.
Both queries run, and their results are shown in two separate output tables.
d) Observe that there is a duplicate entry.

- There are 99 rows in total between the two tables.
- The `Clear Cupboards` book appears in the current Titles table and the Obsolete_Titles table.
- The partnum and bktitle for the duplicate entries are the same, but the pubdate values are different.
e) Click within the top results table.

The top table (Titles) contains 92 rows.
4. Create a combined book list.
   a) In the editor pane, select the blank line between the two `SELECT` statements and type `UNION` to combine the results of the two tables.
   
   ```sql
   SELECT partnum, bktitle, pubdate
   FROM Titles
   UNION
   SELECT partnum, bktitle, pubdate
   FROM Obsolete_Titles
   ```
   
   b) Execute the query.
   c) Examine the results.

   - SQL Server displays the combined output of both tables (99 rows).
   - The `Clear Cupboards` book appears in the list twice.
   - Although the `UNION` operator should only show one copy of any duplicate rows, it is showing two copies of `Clear Cupboards` because the rows are not identical. They have different `pubdate`. If you didn't show `pubdate`, the two rows would be identical.

5. Eliminate the `pubdate` column, and run the query again.
   a) In the editor pane, remove the `pubdate` column from both `SELECT` clauses.
   
   ```sql
   SELECT partnum, bktitle
   FROM Titles
   UNION
   SELECT partnum, bktitle
   FROM Obsolete_Titles
   ```

   - The `Clear Cupboards` book appears in the list only once, as expected.
b) Execute the query.
c) Scroll and examine the results to determine whether *Clear Cupboards* still appears twice.

The *Clear Cupboards* book now appears only once. Without the different dates in the *pubdate* column, the data for both copies is now exactly the same. The *UNION* operator automatically removes duplicates.

6. Use the *UNION ALL* operator to show all combined rows, including duplicates.

a) In the editor pane, change *UNION* to *UNION ALL*.

b) Execute the query.
c) Scroll and examine the results to determine whether both copies of *Clear Cupboards* are shown.

Once again, there are two copies of the *Clear Cupboards* book (rows 1 and 98).
TOPIC B

Compare the Results of Two Queries

You have used the UNION operator to combine the results of two queries. But what if you want to retrieve distinct values from the result set of either of the queries or from both queries? In this topic, you will compare results to get distinct values in the output.

The EXCEPT and INTERSECT Operators

The EXCEPT and INTERSECT operators enable SQL Server to compare the rows identified by each SELECT statement and use this comparison to generate the query's result set. With the EXCEPT operator, SQL Server selects all rows in the first table except those that have matching rows in the second table. In contrast, with the INTERSECT operator, SQL Server selects only those rows in the first table that it finds in the second table.

Figure 5-2: Two SELECT statements using the EXCEPT operator.

The EXCEPT operator enables you to combine two SELECT statements so that SQL Server retrieves all rows from the first table that are not found in the second table. You might use this type of query to retrieve all books in the Titles table except those that are listed in the Obsolete_Titles table. Use this syntax for the EXCEPT operator:

```
SELECT column1[, column2, ...]
FROM table1
EXCEPT
SELECT column1[, column2, ...]
FROM table2
```

Figure 5-3: Two SELECT statements with the INTERSECT operator.

You use the INTERSECT operator to return the rows that two queries have in common. For example, if you want to find out whether any of the books in the Titles table are also in the Obsolete_Titles table, you can use two SELECT statements separated by the INTERSECT operator to retrieve this list. Here is the syntax for the INTERSECT operator:

```
SELECT column1[, column2, ...]
FROM table1
INTERSECT
SELECT column1[, column2, ...]
FROM table2
```
SELECT column1[, column2, ...]
FROM table2

As with the **UNION** operator, both **EXCEPT** and **INTERSECT** require that your **SELECT** statements specify the same number of columns. In addition, the data types of those columns must be compatible.

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Compare the Results of Two Queries.
ACTIVITY 5–2
Comparing Results of Two Queries

Before You Begin
SQL Server Management Studio is running. A query editor window is open, and the Pub1 database is selected.

Scenario
You are planning to place an order for books that are listed in the Titles table. However, you've been told by the salesperson that this order also contains some obsolete titles, which are listed in the Obsolete_Titles table. You want to ensure that you do not sell obsolete titles. So, you decide to first check for obsolete titles listed in the Titles table and then list the titles that are not obsolete.

1. Retrieve any records in the Titles table that are also in the Obsolete_Titles table.
   a) In the editor pane, change UNION ALL to INTERSECT.
   b) Execute the query.
   c) In the Results pane, verify that the book title Clear Cupboards is displayed. There is one obsolete title still in the Titles table.

2. Retrieve a list of books from the Titles table that are not included in the Obsolete_Titles table.
   a) In the editor pane, change INTERSECT to EXCEPT.
   b) Execute the query.
   c) In the Results pane, observe that SQL Server displays all records from the Titles table that are not in the Obsolete_Titles table. There are a total of 91 rows.
TOPIC C

Retrieve Data by Joining Tables

In most databases, you'll find that the information you need is distributed across multiple tables, not contained in a single table. You can obtain this distributed information by combining the output from multiple tables. In this topic, you will use SQL Server's ability to join tables to retrieve data from more than one table.

Joins

A join is a method of combining data from two or more tables into one result set based on a condition or a column that is common to both tables. There are four types of joins:

- Cross joins
- Inner joins
- Outer joins
- Self joins

Figure 5-4: An output table generated by joining Table 1 and Table 2.

Cross Joins

The cross join is a join in which SQL Server combines each row in one table with each row from the second table. In the SQL statement, you enter the CROSS JOIN keyword between the two table names you are joining together. The output of a cross join is sometimes called a Cartesian product because SQL Server joins every row in the first table with every row of the second table. The total
number of records displayed in the output is the number of rows in the first table multiplied by the number of rows in the second table.

**Note:** If you add a `WHERE` clause to a cross join, SQL Server treats it as an inner join.

![Figure 5-5: A cross join combining the values in the Colors and Sizes tables.](image)

**Figure 5-5: A cross join combining the values in the Colors and Sizes tables.**

**Note:** You'll find that cross joins are rarely used in a production environment because of their limited usefulness.

### Syntax of a Cross Join

The syntax of a cross join is:

```sql
SELECT colname1, [colname2, ...] 
FROM tablename1 CROSS JOIN tablename2
```

### Inner Joins

An **inner join** is a join that displays records from two tables that have equivalent values in one or more columns. SQL Server compares the values of the joined columns based on the equivalent comparison operator. The output of an inner join query consists of only the rows that have matching values in both tables.

**Note:** The most commonly used joins in SQL Server are inner joins.

In the following example, the `SELECT` statement displays the custnum column from the Sales table and the fname and lname columns from the sales representative (Slspers) table. The query then uses the `FROM` and the `INNER JOIN` clauses to join the two tables together (`FROM sales INNER JOIN slspers`). Finally, the `ON` clause specifies the column that the two tables have in common: repid. Notice that because the repid column exists in both tables, SQL Server requires you to identify the
table for each repid column by preceding the column name with the table name followed by a period.

Figure 5-6: The Sales table with an inner join to the Slspers table on the repid column.

When you create queries that retrieve rows from multiple tables, you will encounter situations where the tables have columns with the same name. For example, the column partnum exists in both the Titles and Sales tables in the Pub1 database. If you reference the column partnum in a query that joins these two tables, you must identify the table from which you want SQL Server to pull the column's information. You identify the table by preceding the column with the name of the table followed by a period.

SELECT custnum, fname, lname, Slspers.repid
FROM Sales
INNER JOIN Slspers ON Sales.repid = Slspers.repid
ORDER BY custnum

Figure 5-7: Qualified column names.

Syntax of an Inner Join

The syntax of an inner join is:

SELECT colname1, [colname2, ...]
FROM tablename1 INNER JOIN tablename2
ON tablename1.column = tablename2.column

Note: An inner join is also known as an equi-join.
Outer Joins

Another type of join supported in SQL Server is an outer join. Unlike an inner join in which SQL Server displays the matching rows between two tables, with an outer join, SQL Server displays all rows in one table regardless of whether there's a matching row in the other table. For example, you might want a list of all books in the Titles table and their associated sales from the Sales table regardless of whether each book has any sales. In other words, it's possible that you have a book in Titles for which there are no sales. An outer join enables you to identify those books. SQL Server supports three types of outer joins: left, right, or full.

**Figure 5-8: The results of a left outer join.**

You use a left outer join to display all rows from the table on the left of the FROM clause regardless of whether it has matching rows in the table on the right. Here is the syntax for a left outer join:

```sql
SELECT column1, column2, ...
FROM table1 LEFT OUTER JOIN table2
ON table1.column = table2.column
```

**Note:** You can typically rewrite a left outer join as a right outer join, and vice versa.

You use a right outer join to accomplish the opposite of a left outer join: display all the rows from the table on the right side of the FROM clause whether there are any matching rows in the table on the left side or not. Use this syntax for a right outer join:

```sql
SELECT column1, column2, ...
FROM table1 RIGHT OUTER JOIN table2
ON table1.column = table2.column
```

A full outer join query returns all rows from both tables in the join condition regardless of whether there are matching values present in either table. This type of query has limited usefulness and you'll rarely use it. Use this syntax to perform a full outer join:

```sql
SELECT column1, column2, ...
FROM table1 FULL OUTER JOIN table2
ON table1.column = table2.column
```
The Table Alias Name

A table alias name is an alternative name you give to a table so that you can use it to refer to that table in place of the table name. You typically use a table alias to avoid having to type a long table name in an SQL statement or when you want to refer to the same table as two different tables in the same query. You specify the table alias after the table name in the FROM clause of a query. To access a column in a table using the table alias, enter the table alias, followed by a period, and then the column name.

In the following example, the FROM clause assigns the table alias name of "c" to the Potential_Customers table. The alias is used in the SELECT clause as shorthand to refer to the table.

Figure 5-9: A SELECT statement displaying the table alias names.

Note: Optionally, you can omit the AS keyword between the table name and table alias.

Self Joins

A self join is a join that joins a table to itself. In the SQL statement, you use two table aliases in the join condition to identify the table. You then enter the INNER JOIN keywords in the FROM clause between the table names and aliases. In a self join, both table names are the same, but the table alias names are different. You specify the join condition in the ON clause.

The following figure shows an example of a self join. The Potential_Customers table contains a list of all potential customers and their assigned customer numbers. In addition, each potential customer has a customer number in the referredby column that identifies which customer referred them to Fuller & Ackerman Publishing.
Multiple Table Joins

It's possible for you to join more than two tables together in a single query by using multiple join conditions. You do this when you want to retrieve columns of information from more than two tables and have those columns appear in a single result set. For example, consider the Pub1 database. Consider the structure of the following tables, as shown in the figure.

![Figure 5-10: A code sample using a self join.](image-url)
This figure shows the structure of the Sales, Customers, and Titles tables. Notice that the Sales table and Customers table have the custnum column in common. Likewise, the Sales and Titles tables have the partnum column in common.

The relationships between these three tables make it possible for you to write a query such as the following, which retrieves a list of sales that contains the order number and quantity from the Sales table, the book title from the Titles table, and the customer name from the Customers table.

```sql
SELECT ordnum, qty, bktitle, custname
FROM Sales INNER JOIN Customers
ON Sales.custnum = customers.custnum
INNER JOIN Titles
ON Sales.partnum = Titles.partnum
```

Be sure to list each table name just once in the FROM clause. It doesn’t matter which table you list first. The ON clauses make it clear which table.column to relate to the other table.column.

The following figure shows the output of the multiple table join query. Notice that the result set contains columns from the Sales, Titles, and Customers tables.
Figure 5–12: Results of the multiple table join query.

The Spotlight on Join Types presentation is available from the Spotlight tile on the CHOICE Course screen. You may choose to include it in your instructional plans, or you can remind students about the tile and the supplemental information it contains.

Note: To further explore joins, check out the Spotlight on Join Types presentation from the Spotlight tile on the CHOICE Course screen.

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Retrieve Data by Joining Tables.
ACTIVITY 5–3
Retrieving Data by Performing an Inner Join

Before You Begin
SQL Server Management Studio is running. A query editor window is open, and the Pub1 database is selected.

Scenario
Management has asked you to provide a list of all sales along with the name of the sales representative responsible for each sale. For each sale, they would like to know the order number, the sale date, and the quantity sold, in addition to the name of the sales representative. They have asked that you sort the output in descending order by the quantity sold.

1. List all sales orders and sales reps in two queries.
   a) In the query editor, delete the current query, and type two new queries as shown.

```
SELECT ordnum, sldate, qty, repid
FROM Sales

SELECT repid, fname, lname
FROM Slsers
```
b) Execute the queries, and examine the results.

<table>
<thead>
<tr>
<th>ordnum</th>
<th>sldate</th>
<th>qty</th>
<th>repid</th>
</tr>
</thead>
<tbody>
<tr>
<td>00101</td>
<td>2017-11-16 00:00:00</td>
<td>220</td>
<td>N01</td>
</tr>
<tr>
<td>00102</td>
<td>2017-11-20 00:00:00</td>
<td>100</td>
<td>N02</td>
</tr>
<tr>
<td>00103</td>
<td>2017-11-20 00:00:00</td>
<td>170</td>
<td>N02</td>
</tr>
<tr>
<td>00104</td>
<td>2017-12-07 00:00:00</td>
<td>150</td>
<td>N02</td>
</tr>
<tr>
<td>00105</td>
<td>2017-12-16 00:00:00</td>
<td>200</td>
<td>N02</td>
</tr>
<tr>
<td>00106</td>
<td>2017-12-16 00:00:00</td>
<td>200</td>
<td>N02</td>
</tr>
<tr>
<td>00107</td>
<td>2017-12-16 00:00:00</td>
<td>200</td>
<td>N02</td>
</tr>
<tr>
<td>00108</td>
<td>2017-12-16 00:00:00</td>
<td>200</td>
<td>N02</td>
</tr>
</tbody>
</table>

The repid column appears in both tables. You can use it to join data from the two tables.

2. Enter an inner join query to retrieve a list of all sales along with the sales representatives' names.

a) Delete the second query, and add the fname and lname columns to the first query as shown.

```sql
SELECT ordnum, sldate, qty, repid, fname, lname
FROM Sales
```

- The fname and lname columns are highlighted by SQL Server Management Studio because they are not contained in the Sales table.
- These two columns are from the Slspers table.

b) Add an INNER JOIN clause to join data from the Slspers table to the Sales table.

```sql
SELECT ordnum, sldate, qty, repid, fname, lname
FROM Sales
INNER JOIN Slspers ON Sales.repid = Slspers.repid
```

- Now that two tables are involved, you have to specify the table names for columns that appear in both tables, to clarify which source you are referring to.
- The repid column is highlighted in the SELECT clause because you need to specify which table it belongs to.
c) In the `SELECT` clause, add the table name for repid as shown.

```
SELECT ordnum, sldate, qty, Sales.repid, fname, lname 
FROM Sales 
INNER JOIN Slspers ON Sales.repid = Slspers.repid
```

Qualifying repid with either Sales or Slspers gives the same results.

d) Add the `ORDER BY` clause to sort the output in alphabetical order by sales representative name.

```
SELECT ordnum, sldate, qty, Sales.repid, fname, lname 
FROM Sales 
INNER JOIN Slspers ON Sales.repid = Slspers.repid 
ORDER BY qty DESC
```

e) Execute the query.

f) In the **Results** pane, scroll down to examine the records.

- You have joined data from two different tables.
- The results are shown in descending order by the quantity sold.
ACTIVITY 5–4
Retrieving Data by Performing an Outer Join

Before You Begin
SQL Server Management Studio is running. A query editor window is open, and the Pub1 database is selected.

Scenario
You have been asked to provide a list of all books published by Fuller & Ackerman and their total sales (if any). You will use a left outer join query to produce this list.

1. Display all columns in the Titles and Sales tables.
   a) In the query editor, delete the current query, and type two new queries as shown.

   ```sql
   SELECT * FROM Titles
   SELECT * FROM Sales
   ```

   b) Execute the query, and examine the results to determine how you will join the two tables.

   - The partnum column will enable you to join the tables.
   - You’ll pull the sales from the Sales table, and use partnum to join the book titles from Titles into the results.

2. Determine if any rows exist in one table, but not the other.
a) Revise the queries as shown.

These revisions:
• Show only partnum columns in both SELECT clauses.
• Use the EXCEPT operator to show only the values that exist in one table but not the other.

b) Execute the query, and examine the results.

37 books, including part number 39843, do not exist in both tables.
You need to include even books that haven't sold any copies in your output. So you will use an outer join.

3. Write an outer join query, joining on partnum.

a) In the query editor, delete the current query, and enter a new query as shown.
b) Execute the query, and examine the results.

```
<table>
<thead>
<tr>
<th>bktitle</th>
<th>qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Cupboards</td>
<td>NULL</td>
</tr>
<tr>
<td>Developing Mobile Apps</td>
<td>NULL</td>
</tr>
<tr>
<td>Boating Safety</td>
<td>110</td>
</tr>
<tr>
<td>Sailing</td>
<td>250</td>
</tr>
<tr>
<td>The Sport of Windsurfing</td>
<td>160</td>
</tr>
<tr>
<td>The Sport of Hang Gliding</td>
<td>NULL</td>
</tr>
<tr>
<td>The Complete Football Reference</td>
<td>220</td>
</tr>
<tr>
<td>The Complete Football Reference</td>
<td>200</td>
</tr>
<tr>
<td>The Complete Football Reference</td>
<td>220</td>
</tr>
<tr>
<td>How to Play Piano (Beginner)</td>
<td>250</td>
</tr>
</tbody>
</table>
```

- All titles in the left table (Titles) were included, even those not in the Sales table (because no sales were recorded for those titles).
- The qty values for titles not in the Sales table are shown as NULL, since there is no value to provide.
- Separate rows are shown where there have been separate orders for the same title, such as *The Complete Football Reference*.

4. Group the results by book title, and aggregate the quantities sold.

a) In the query editor, revise the query as shown.

```
SELECT bktitle, SUM(qty) AS total_qty
FROM Titles LEFT OUTER JOIN Sales
ON Titles.partnum = Sales.partnum
GROUP BY bktitle
```

After these changes, the query will:
- Use the `SUM` function to aggregate the qty values.
- Use a `GROUP BY` clause to summarize results for each book title.

b) Execute the query.
c) In the **Results** pane, scroll down to view the results.

<table>
<thead>
<tr>
<th>bkttitle</th>
<th>total_qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taking Care of Your Rabbit</td>
<td>NULL</td>
</tr>
<tr>
<td>The Art of Oil Painting</td>
<td>1000</td>
</tr>
<tr>
<td>The Art of Pen and Ink Drawing</td>
<td>100</td>
</tr>
<tr>
<td>The Art of Water Painting</td>
<td>100</td>
</tr>
<tr>
<td>The Complete Auto Repair Guide</td>
<td>250</td>
</tr>
<tr>
<td>The Complete Football Reference</td>
<td>640</td>
</tr>
<tr>
<td>The Complete Guide to Flowers</td>
<td>NULL</td>
</tr>
<tr>
<td>The Complete Guide to Vegetables</td>
<td>490</td>
</tr>
<tr>
<td>The History of Baseball</td>
<td>170</td>
</tr>
<tr>
<td>The Mayan Civilization</td>
<td>1850</td>
</tr>
</tbody>
</table>

- Books that have had multiple orders, such as *The Complete Football Reference*, have been aggregated to a single row.
- The total sales quantity for all books with multiple orders is shown.

You may wish to demonstrate HAVING `SUM(qty) IS NOT NULL` to show only those books that did not sell.

If you have time, consider showing the three-table query example from the Multiple Table Joins section.
ACTIVITY 5–5
Retrieving Data from a Single Table Using a Self Join

Before You Begin
SQL Server Management Studio is running. A query editor window is open, and the Pub1 database is selected.

Scenario
You have been asked to provide a list of all potential customers and the customers that referred them. Since both types of customers come from the same table, you will use a self join query to produce the list.

1. Enter a self join query to retrieve a list of potential customers and the customers that referred them to Fuller & Ackerman Publishing.
   a) In the query editor, delete the current query, and type a new query as shown.

   ![Query Editor Screenshot]

   b) Execute the query, and observe the results.
   ![Customer List]

   In this list of customers, some customers have been referred by other customers.
   **•** The referredby column shows the customer number of the person who referred each customer. For example, customer 31001 (Empire Books) was referred by customer 31004 (The Family Sing Center).
   **•** You can join this table to itself to show the name of each customer, with another column showing the name of the customer who referred them.

2. Revise the query to show the name of the referredby customer.
a) Add an alias so you can refer to results of the `SELECT` query.

```
SELECT *
FROM Potential_Customers AS custlist
```

This will enable you to refer to columns coming from this result when you expand on this query using a join.

b) Add two more lines to the query as shown.

```
SELECT *
FROM Potential_Customers AS custlist
INNER JOIN potential_customers AS referredlist
ON custlist.referredby = referredlist.custnum
```

• With the `INNER JOIN` operator, you have named the right side of the join as `referredlist`.
• In the last line, you are matching the `referredby` column of `custlist` to the `custname` column of `referredlist`.

c) Execute the query and examine the results.

```
<table>
<thead>
<tr>
<th>custnum</th>
<th>referredby</th>
<th>custname</th>
<th>address</th>
<th>city</th>
<th>state</th>
<th>zipcode</th>
<th>phone</th>
<th>referrednum</th>
<th>referredby</th>
<th>custname</th>
</tr>
</thead>
<tbody>
<tr>
<td>31003</td>
<td>31004</td>
<td>Empire Books</td>
<td>911 Empire Blvd</td>
<td>Cincinnati</td>
<td>OH</td>
<td>45203</td>
<td>851</td>
<td>31003</td>
<td>31004</td>
<td>Book Publishers, Inc</td>
</tr>
<tr>
<td>31002</td>
<td>31005</td>
<td>Book Publishers, Inc</td>
<td>3 Park Ave</td>
<td>Wilson</td>
<td>NC</td>
<td>27803</td>
<td>902</td>
<td>31002</td>
<td>31005</td>
<td>Hand Loved Craft Supplies</td>
</tr>
<tr>
<td>31003</td>
<td>31002</td>
<td>Bill Rings</td>
<td>15 Main St</td>
<td>Syracuse</td>
<td>NY</td>
<td>13206</td>
<td>901</td>
<td>31003</td>
<td>31002</td>
<td>Book Publishers, Inc</td>
</tr>
<tr>
<td>31004</td>
<td>31005</td>
<td>The Family Sing Center</td>
<td>224 Philips Ave</td>
<td>Santa Ana</td>
<td>CA</td>
<td>92704</td>
<td>503</td>
<td>31004</td>
<td>31005</td>
<td>Book Publishers, Inc</td>
</tr>
<tr>
<td>31005</td>
<td>31004</td>
<td>Hand Loved Craft Supplies</td>
<td>57 Underwood Blvd</td>
<td>Troy</td>
<td>MI</td>
<td>48024</td>
<td>382</td>
<td>31005</td>
<td>31004</td>
<td>The Family Sing Center</td>
</tr>
</tbody>
</table>
```

All columns from `custlist` and `referredlist` are shown in the combined results. Multiple columns with the same name might be confusing. You could rename some of the columns using aliases in the `SELECT` clause. Also, you don't need all of these columns for your report—just the names of the customer and who referred them.

3. Show specific columns from each side of the join.
   a) Revise the `SELECT` clause as shown.

```
SELECT custlist.custname, referredlist.custname AS referred_by
FROM Potential_Customers AS custlist
INNER JOIN potential_customers AS referredlist
ON custlist.referredby = referredlist.custnum
```
b) Execute the query and examine the results.

<table>
<thead>
<tr>
<th>custnum</th>
<th>custname</th>
<th>referred_by</th>
</tr>
</thead>
<tbody>
<tr>
<td>31001</td>
<td>Empire Books</td>
<td>The Family Sing Center</td>
</tr>
<tr>
<td>31002</td>
<td>Book Publishers, Inc.</td>
<td>Hand Loved Craft Supplies</td>
</tr>
<tr>
<td>31003</td>
<td>BxB Fitness</td>
<td>Book Publishers, Inc.</td>
</tr>
<tr>
<td>31004</td>
<td>The Family Sing Center</td>
<td>Book Publishers, Inc.</td>
</tr>
<tr>
<td>31005</td>
<td>Hand Loved Craft Supplies</td>
<td>The Family Sing Center</td>
</tr>
</tbody>
</table>

The completed query produces a list of all potential customers and the customers that referred them.

4. Exit SQL Server Management Studio without saving the query.

**Note:** If you want to review the query later, you can save it to a file.
Summary

In this lesson, you executed queries that retrieved data from multiple tables. By using the **UNION**, **EXCEPT**, **INTERSECT**, and **JOIN** operators in your **SELECT** statements, you can retrieve the exact information you need from data split across multiple tables and display it in a single, united result set.

**When might you join one table to another? Why?**

**A:** Answers will vary, but may include: you join one table to another when data in one table is dependent upon the data in another table. Sometimes, the primary key in one instance is used to retrieve data from the other table based on the relationship between the columns in the table.

**When might you join a table to itself? Why?**

**A:** Answers will vary, but may include: joining a table to itself is done when you want to retrieve information from the table based on some condition that is satisfied within that table itself. For example, if you want to retrieve information of all employees belonging to the same country within the table, then you use a self join.

**Note:** Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.

Encourage students to use the social networking tools provided on the CHOICE Course screen to follow up with their peers after the course is completed for further discussion and resources to support continued learning.
Exporting Query Results

Lesson Time: 30 minutes

Lesson Introduction

So far in this course, you have queried tables and viewed the results in the Results pane in Microsoft® SQL Server® Management Studio (SSMS). SQL professionals would have no problem with viewing and understanding such results. End users, on the other hand, typically don’t have SSMS installed on their computers. For this reason, you need to be able to provide users with data in formats they can use in the applications they do have available to them. In this lesson, you will obtain query results in formats that users can use in other applications and tools.

Lesson Objectives

In this lesson, you will:
• Generate a text file.
• Generate an XML file.
TOPIC A

Generate a Text File

One of the ways in which you can distribute the results of your queries to other users is by saving those results in text files. Users can then import those results into an application such as Microsoft® Excel® for further manipulation. In this topic, you will save the result set of queries into text files.

Text Data Formats

Microsoft® SQL Server® 2017 enables you to save the results of a query into two text formats. These formats differ based on how SQL Server separates the data that is stored in the columns in the result set. The two formats are:

- Tab delimited values. In this format, SQL Server separates the values in the columns of the result set with tabs.
- Comma delimited values. SQL Server separates the values in the columns of the result set with commas. By default, SQL Server saves result sets in the comma delimited value format.

You choose between the two data formats supported by SQL Server based on the application in which a user plans to use the result set. For example, Microsoft Excel can open and use the data in a comma delimited text file.

Note: To further explore the use of text data files, check out the Spotlight on Exporting and Using Text Data Files presentation from the Spotlight tile on the CHOICE Course screen.

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Save the Query Results.
ACTIVITY 6–1
Saving the Query Results

Data File
C:\094018Data\Exporting Query Results\Commissions.sql

Scenario
You have a query that generates a report on sales commissions for all sales representatives for an entire year. You will export the report to a CSV format, so it can be imported into Excel for further analysis.

1. Launch SQL Server Management Studio, open a new query editor, and select the Pub1 database.
   a) If SQL Server Management Studio is not running, launch it and connect to the server.
   b) Select File→Open→File. Navigate to C:\094018Data\Exporting Query Results, and select Commissions.sql.
   c) Select Open.
      The query is loaded into the query editor.
   d) On the SQL Editor toolbar, from the Available Databases drop-down list, ensure that the Pub1 database is selected.
   e) Examine the query.

   ![SQL Query](Commissions.sql)

   This query combines numerous techniques that you have used, including joins, aliases, sorting, functions, and concatenation.
1. Execute the query.

The query provides summary information for each sales rep, including commission rate, total sales, and commission for the year.

2. Save the query results in the comma delimited format.
   a) In the Results pane, right-click the blank area and select Save Results As.
   b) In the Save Grid Results dialog box, navigate to the C:\094018Data\Exporting Query Results folder.
   c) In the File name text box, type Annual Sales Commissions.
   d) Verify that CSV (Comma delimited) (*.csv) is selected in the Save as type drop-down list.
   e) Select Save to save the results as a comma delimited text file.
   f) Minimize the SQL Server Management Studio application window.

3. Open the Annual Sales Commissions.csv file and view its content.
   a) In File Explorer, navigate to the C:\094018Data\Exporting Query Results folder.

Consider having a student volunteer to explain how the query works step by step.
b) Double-click Annual Sales Commissions.csv to open the file. If Excel is installed on your computer, the CSV file will open in Excel.

![Excel spreadsheet](image)

Note: You may need to drag the dividers between column headings to adjust column widths so all the data is visible.

c) If you are prompted to select an application with which to open the file, select Notepad. If you open the file in Notepad, you can see that commas are used to separate the columns. String values are enclosed in double quotes.

![Notepad window](image)

d) Exit the application (Excel or Notepad) that you used to examine the CSV file.

e) Close any File Explorer windows that you opened.
TOPIC B

Generate an XML File

In addition to saving result sets in text formats, you can also save the output of queries in the XML format. Files in XML format can be used by a variety of applications, particularly web-based applications. In this topic, you will save query results in an XML file.

XML

XML stands for eXtensible Markup Language. XML is used to create custom markups, thereby allowing users to "mark up" or define their own elements of data within a document. Users define these elements through the use of tags. The main purpose of XML is to facilitate the sharing of structured data among applications and across the Internet. XML is easily readable because it uses self-descriptive tags. XML is similar to HTML in its coding format, but unlike HTML, which is used to display data, XML is used to define data.

The FOR XML Clause

The **FOR XML** clause is a clause that you can use to return the results of a query in XML format. The FOR XML clause requires one or all of four modes to return results.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>Generates a single element per row in the rowset with a generic identifier as the element tag.</td>
</tr>
<tr>
<td>AUTO</td>
<td>Returns query results in a simple XML tree.</td>
</tr>
<tr>
<td>EXPLICIT</td>
<td>Defines the shape of the resulting XML tree. It requires a specific format for the resulting rowset that is generated.</td>
</tr>
<tr>
<td>PATH</td>
<td>Generates an element wrapper for each row in a rowset.</td>
</tr>
</tbody>
</table>

In the following example, the **SELECT** statement selects all sales representatives whose IDs begin with the letter "N." The **ORDER BY** clause sorts the result set by the representative IDs. The **FOR XML AUTO, TYPE, ELEMENTS** clause specifies that SQL Server should generate the query's result set in XML with a single element per row for each salesperson.
Figure 6–1: A FOR XML clause illustration.

Syntax of the FOR XML Clause

The syntax of the FOR XML clause is:

```sql
SELECT (columnname1,columnname2,....) FROM table name WHERE (condition) ORDER BY (expression) FOR XML mode
```

Access the Checklist tile on your CHOICE Course screen for reference information and job aids on How to Generate an XML File.
ACTIVITY 6–2
Generating an XML File

Before You Begin
SQL Server Management Studio is running. A query editor window is open, and the Pub1 database is selected. The Commissions.sql file is open in the query editor.

Scenario
Your team uses a cloud-based application to perform analysis on your sales. The application requires that you upload input data in an XML format. You will export the annual sales commissions report to XML.

1. Revise the query to produce XML output:
   a) Add a new last line to the query as shown.

   ```sql
   -- Sales commission report
   SELECT report.repId,
   TRIM(report.fname) + ' ' + report.lname AS repname,
   CAST(MAX(report.commrate) * 100 AS varchar) + '%' AS commrate,
   '$' + CONVERT(varchar, SUM(Sales.qty * Titles.slprice), 1) AS sales,
   '$' + CONVERT(varchar, CONVERT(money, SUM(report.commrate * Sales.qty)), 1) AS daily,
   MAX(DATEPART(YEAR, Sales.sldate)) AS Year
   FROM Slipers as report
   INNER JOIN Sales ON report.repId = Sales.repId
   INNER JOIN Titles ON Sales.partnum = Titles.partnum
   WHERE DATEPART(YEAR, Sales.sldate) = 2017
   GROUP BY report.repId,
   report.fname,
   report.lname
   ORDER BY SUM(Sales.qty * Titles.slprice) desc, report.lname, report.fname
   FOR XML AUTO, TYPE, ELEMENTS
   ```

   The `FOR XML AUTO, TYPE, ELEMENTS` clause causes SQL Server to return the results of the query in the XML format.
   - The `AUTO` parameter requests that XML be displayed in a nested format in which each sales representative's information appears indented below the sales representative's ID.
   - The `TYPE` parameter sets the data type of the results to XML.
   - The `ELEMENTS` parameter requests that SQL Server return the column information for each sales representative as XML elements.

2. Produce the XML report.
a) Execute the query.

The results are formatted as XML. A link is provided so you can open the XML output as a file.

b) In the Results pane, select the hyperlink.

The XML that was generated opens in an XML editor.

c) In the XML Editor window, observe that all column values are displayed within their respective column names as XML elements.

d) Select File→Save xmlresult1.xml As to save the XML file.

e) In the Save File As dialog box, navigate to the C:\094018Data\Exporting Query Results folder.

f) In the File name text box, type Commission Report

g) In the Save as type drop-down list, make sure that XML Files is selected.

h) Select Save to save the XML file.

3. Exit SQL Server Management Studio without saving your changes to the query.
Summary

In this lesson, you exported query results in both the text and XML formats. Exporting result sets in different formats enables you to easily work with your data in other applications.

For what purposes will you save query results?
A: Answers will vary, but may include: query results can be saved to analyze data, and to enable clients to view the results in different applications.

How is the ability to generate an XML file beneficial?
A: Answers will vary, but may include: XML format reports can be accessed by many different applications. XML data is also easily readable because it uses self-descriptive tags.

Note: Check your CHOICE Course screen for opportunities to interact with your classmates, peers, and the larger CHOICE online community about the topics covered in this course or other topics you are interested in. From the Course screen you can also access available resources for a more continuous learning experience.
Course Follow-Up

Organizations use databases and the tables within them to store mission-critical information. When you need to retrieve information from these databases, you must use the Structured Query Language (SQL). Using SQL enables you to perform business analysis tasks such as identifying sales trends, targeting customers for marketing purposes, and verifying whether an inventory item is in stock.

In this course, you used SQL as a tool to retrieve information from a database and its tables. You retrieved all information in tables, and then you developed queries to retrieve specific information using conditions. You learned to sort, group, and filter the results of queries. You also joined tables to extract information present in multiple tables. Then, you exported query result sets as text and XML files.

What's Next?

*SQL Querying: Advanced (Second Edition)* is the next course in this series. This course covers advanced querying concepts such as creation of tables and indexes.

You are encouraged to explore SQL querying fundamentals further by actively participating in any of the social media forums set up by your instructor or training administrator through the Social Media tile on the CHOICE Course screen.
The database used in this book, called the Pub1 database, is being used in a hypothetical book publishing company called Fuller & Ackerman Publishing. The following tables constitute the Pub1 database.

- The Customers table describes each of Fuller & Ackerman Publishing’s customers.
- The Sales table describes each book sale.
- The Slspers table describes each sales representative working at Fuller & Ackerman Publishing.
- The Titles table describes each book produced by Fuller & Ackerman Publishing.
- The Obsolete Titles table describes all books that are out of print.
- The Potential Customers table describes any possible new customers for Fuller & Ackerman Publishing.

The Customers Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>custnum</td>
<td>nvarchar</td>
<td>10</td>
<td>The customer number for each client. Each customer is assigned a unique customer number.</td>
</tr>
<tr>
<td>referredby</td>
<td>nvarchar</td>
<td>10</td>
<td>The customer number of the client who referred this potential customer to Fuller &amp; Ackerman Publishing.</td>
</tr>
<tr>
<td>custname</td>
<td>nvarchar</td>
<td>60</td>
<td>The customer’s name, or business name.</td>
</tr>
<tr>
<td>address</td>
<td>nvarchar</td>
<td>50</td>
<td>The customer’s street address.</td>
</tr>
<tr>
<td>city</td>
<td>nvarchar</td>
<td>40</td>
<td>The city in which the customer resides.</td>
</tr>
<tr>
<td>state</td>
<td>nvarchar</td>
<td>4</td>
<td>The state in which the customer resides.</td>
</tr>
<tr>
<td>zipcode</td>
<td>nvarchar</td>
<td>24</td>
<td>The state’s ZIP Code.</td>
</tr>
<tr>
<td>repid</td>
<td>nvarchar</td>
<td>6</td>
<td>The customer’s sales representative’s identification number.</td>
</tr>
</tbody>
</table>

The Sales Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ordnum</td>
<td>nvarchar</td>
<td>10</td>
<td>The order number for each book sale. Each sales order is assigned a unique order number.</td>
</tr>
<tr>
<td>sldate</td>
<td>smalldatetime</td>
<td>4</td>
<td>The date of sale.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Data Type</td>
<td>Length</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>qty</td>
<td>int</td>
<td>4</td>
<td>The number of books ordered.</td>
</tr>
<tr>
<td>custnum</td>
<td>nvarchar</td>
<td>10</td>
<td>The customer number for the customer purchasing books.</td>
</tr>
<tr>
<td>partnum</td>
<td>nvarchar</td>
<td>10</td>
<td>The part number of the book being ordered.</td>
</tr>
<tr>
<td>repid</td>
<td>nvarchar</td>
<td>6</td>
<td>The sales representative responsible for the sale.</td>
</tr>
</tbody>
</table>

The Slspers Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>repid</td>
<td>nvarchar</td>
<td>6</td>
<td>The identification number for each salesperson. Each sales representative is assigned a unique identification number.</td>
</tr>
<tr>
<td>fname</td>
<td>nvarchar</td>
<td>20</td>
<td>The first name of the sales representative.</td>
</tr>
<tr>
<td>lname</td>
<td>nvarchar</td>
<td>40</td>
<td>The last name of the sales representative.</td>
</tr>
<tr>
<td>commrate</td>
<td>float</td>
<td>8</td>
<td>The sales representative's commission rate.</td>
</tr>
</tbody>
</table>

The Titles Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>partnum</td>
<td>nvarchar</td>
<td>10</td>
<td>The part number for each book published by Fuller &amp; Ackerman. Each book is assigned a unique part number.</td>
</tr>
<tr>
<td>bktitle</td>
<td>nvarchar</td>
<td>80</td>
<td>The title of the book.</td>
</tr>
<tr>
<td>devcost</td>
<td>money</td>
<td>8</td>
<td>The development cost of the book.</td>
</tr>
<tr>
<td>slprice</td>
<td>money</td>
<td>8</td>
<td>The sale price of the book.</td>
</tr>
<tr>
<td>pubdate</td>
<td>smalldatetime</td>
<td>4</td>
<td>The date when the book was published.</td>
</tr>
</tbody>
</table>

The Obsolete_Titles Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>partnum</td>
<td>nvarchar</td>
<td>10</td>
<td>The part number for each book considered obsolete.</td>
</tr>
<tr>
<td>bktitle</td>
<td>nvarchar</td>
<td>80</td>
<td>The title of the obsolete book.</td>
</tr>
<tr>
<td>devcost</td>
<td>money</td>
<td>8</td>
<td>The development cost of the book.</td>
</tr>
<tr>
<td>slprice</td>
<td>money</td>
<td>8</td>
<td>The price of the book.</td>
</tr>
<tr>
<td>pubdate</td>
<td>smalldatetime</td>
<td>4</td>
<td>The date when the book was published.</td>
</tr>
</tbody>
</table>
The Potential_Customers Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>custnum</td>
<td>nvarchar</td>
<td>10</td>
<td>A unique number assigned for the potential customer.</td>
</tr>
<tr>
<td>referredby</td>
<td>nvarchar</td>
<td>10</td>
<td>The customer number of the client who referred this potential customer to Fuller &amp; Ackerman Publishing.</td>
</tr>
<tr>
<td>custname</td>
<td>nvarchar</td>
<td>60</td>
<td>The potential customer's name, or business name.</td>
</tr>
<tr>
<td>address</td>
<td>nvarchar</td>
<td>50</td>
<td>The potential customer's street address.</td>
</tr>
<tr>
<td>city</td>
<td>nvarchar</td>
<td>40</td>
<td>The city in which the potential customer resides.</td>
</tr>
<tr>
<td>state</td>
<td>nvarchar</td>
<td>4</td>
<td>The state in which the potential customer resides.</td>
</tr>
<tr>
<td>zipcode</td>
<td>nvarchar</td>
<td>24</td>
<td>The potential customer’s ZIP Code.</td>
</tr>
<tr>
<td>repid</td>
<td>nvarchar</td>
<td>6</td>
<td>The identification number of the sales representative in the potential customer’s area.</td>
</tr>
</tbody>
</table>
Mastery Builders

Mastery Builders are provided for certain lessons as additional learning resources for this course. Mastery Builders are developed for selected lessons within a course in cases when they seem most instructionally useful as well as technically feasible. In general, Mastery Builders are supplemental, optional unguided practice and may or may not be performed as part of the classroom activities. Your instructor will consider setup requirements, classroom timing, and instructional needs to determine which Mastery Builders are appropriate for you to perform, and at what point during the class. If you do not perform the Mastery Builders in class, your instructor can tell you if you can perform them independently as self-study, and if there are any special setup requirements.
Mastery Builder 1–1
Executing a Simple Query

Activity Time: 15 minutes

Scenario
Information about customers, book titles, sales representatives, and sales is stored in separate tables in the Pub1 database. You must query the database and retrieve information requested by functional managers and executives in the company for various business needs. The information the managers have requested includes:

- Lists of all customers, titles, sales representatives, and sales.
- A mailing list that consists of the customer name and address information.
- A list with the book title, part number, and sale price.
- A list of sales representatives’ names and ID numbers.
- A list of all orders, the part number ordered, and the sale quantity.

1. If SQL Server Management Studio is not running, launch it and connect to the server. Open a new query editor, and from the Available Databases list, select Pub1.

2. In the Object Explorer pane, explore the tables included in the Pub1 database and their structures.

3. Write and execute SELECT statements to view all columns and all rows in each of the Customers, Titles, Slspers, and Sales tables.

4. Write and execute a query that retrieves these columns from the Customers table: custname, address, city, state, and zipcode.

5. Write and execute a query that retrieves the bktitle, partnum, and slprice columns from the Titles table.

6. Write and execute a query that retrieves the repid, fname, and lname columns from the Slspers table.

7. Write and execute a query to retrieve the ordnum, partnum, and qty columns from the Sales table.

8. Exit Microsoft SQL Server Management Studio without saving your statements.
Mastery Builder 2–1
Performing a Conditional Search

Activity Time: 15 minutes

Scenario
A sales analysis revealed that most of the books Fuller & Ackerman Publishing sells are priced between $10 and $30. So the sales manager decides to increase the inventory level for books in this price range. Now, you need to list the book title, part number, and sale price of these books. The sales manager also wants information about the representatives who were hired in the past six months. These representatives were assigned IDs that began with either E or N. The sales manager wants their details along with the list of sales made by them. She also wants the information about the sale quantity made by these representatives if the sale quantity is 400 or above in a single sale.

Note: The file C:\094018Data\Performing a Conditional Search\Solution\Performing a Conditional Search.sql contains one possible solution for these tasks. Try to perform these tasks on your own before you refer to the solution file.

1. If SQL Server Management Studio is not running, launch it and connect to the server. Open a new query editor, and from the Available Databases list, select Pub1.

2. In the Object Explorer pane, explore the tables included in the Pub1 database and their structures.

3. List the book title, part number, and sale price of books that are priced between $10 and $30. Execute the query.
   You should see 47 rows in your results, presented like this.

<table>
<thead>
<tr>
<th>bktitle</th>
<th>partnum</th>
<th>slprice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sailing</td>
<td>40122</td>
<td>29.15</td>
</tr>
<tr>
<td>How to Play Piano (Beginner)</td>
<td>40231</td>
<td>25.00</td>
</tr>
<tr>
<td>How to Play Piano (Intermediate)</td>
<td>40232</td>
<td>20.50</td>
</tr>
<tr>
<td>How to Play Piano (Advanced)</td>
<td>40233</td>
<td>20.50</td>
</tr>
<tr>
<td>How to Play Piano (Professional)</td>
<td>40234</td>
<td>25.00</td>
</tr>
<tr>
<td>How to Play Guitar (Beginner)</td>
<td>40251</td>
<td>25.00</td>
</tr>
</tbody>
</table>

4. Show sales made by representatives whose IDs start with either E or N. Execute the query.
   You should see 63 rows in your results, presented like this.
5. Add another condition to the `WHERE` clause in the query you wrote for Step 4. Include only those sales where the quantity was at least 400. Execute the query. This should reduce the results to 7 rows.

6. Close the **Query Editor** window without saving your changes.
Mastery Builder 3–1
Working with Functions

Activity Time: 15 minutes

Scenario
A customer is considered a big buyer if he or she purchases 400 or more books in a single purchase. The sales team wants to identify customers who are big buyers and list the sales made to them during the first six months of 2017. You are also asked to list the total quantity of books sold to these customers and the total number of such sales. There are some customer records in the table with a four-digit customer ID. The human resources manager wants the list of customers with a four-digit customer ID so that she can update the database.

Note: The file C:\094018Data\Working with Functions\Solution\Working with Functions.sql contains one possible solution for these tasks. Try to perform these tasks on your own before you refer to the solution file.

1. If SQL Server Management Studio is not running, launch it and connect to the server. Open a new query editor, and from the Available Databases list, select Pub1.

2. List all columns from the Sales table that occurred during the first six months of 2017, and execute the query.

   87 rows should be returned in this format.

<table>
<thead>
<tr>
<th>ordnum</th>
<th>sldate</th>
<th>qty</th>
<th>custnum</th>
<th>partnum</th>
<th>repid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00108</td>
<td>200</td>
<td>20417</td>
<td>40125</td>
<td>W01</td>
</tr>
<tr>
<td>2</td>
<td>00109</td>
<td>250</td>
<td>8802</td>
<td>40231</td>
<td>N02</td>
</tr>
<tr>
<td>3</td>
<td>00110</td>
<td>250</td>
<td>20330</td>
<td>40482</td>
<td>S03</td>
</tr>
<tr>
<td>4</td>
<td>00111</td>
<td>100</td>
<td>9989</td>
<td>40551</td>
<td>N02</td>
</tr>
<tr>
<td>5</td>
<td>00112</td>
<td>120</td>
<td>8802</td>
<td>40251</td>
<td>N02</td>
</tr>
<tr>
<td>6</td>
<td>00113</td>
<td>400</td>
<td>20417</td>
<td>40614</td>
<td>W01</td>
</tr>
</tbody>
</table>

3. Add to the first query by entering another search condition to check that the sale quantity is greater than or equal to 400, and list the output.

   10 rows should be returned in this format.

<table>
<thead>
<tr>
<th>ordnum</th>
<th>sldate</th>
<th>qty</th>
<th>custnum</th>
<th>partnum</th>
<th>repid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00113</td>
<td>400</td>
<td>20417</td>
<td>40614</td>
<td>W01</td>
</tr>
<tr>
<td>2</td>
<td>00132</td>
<td>400</td>
<td>20503</td>
<td>40581</td>
<td>N01</td>
</tr>
<tr>
<td>3</td>
<td>00134</td>
<td>500</td>
<td>9989</td>
<td>40552</td>
<td>N02</td>
</tr>
<tr>
<td>4</td>
<td>00145</td>
<td>500</td>
<td>20181</td>
<td>40633</td>
<td>E02</td>
</tr>
<tr>
<td>5</td>
<td>00164</td>
<td>500</td>
<td>20181</td>
<td>40321</td>
<td>E02</td>
</tr>
</tbody>
</table>
4. Modify the existing query to list the sum of quantity and count of rows if the sale quantity is greater than or equal to 400 for the sales made during the first six months of 2017, and execute the query.

   The following result should be returned.

<table>
<thead>
<tr>
<th>sum_qty</th>
<th>count_qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4700</td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

5. List the details of customers who have a four-digit customer ID.

   The following result should be returned.

<table>
<thead>
<tr>
<th>custnum</th>
<th>reflcode</th>
<th>custname</th>
<th>address</th>
<th>city</th>
<th>state</th>
<th>zipcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>9960</td>
<td>Pretty Gardens</td>
<td>283 Jacobson St.</td>
<td>Quebec</td>
<td>CA</td>
<td>H8R 1C3</td>
</tr>
<tr>
<td>2000</td>
<td>9959</td>
<td>Marty's Books</td>
<td>43 West Nash Street</td>
<td>Wilson</td>
<td>NC</td>
<td>27853</td>
</tr>
<tr>
<td>3000</td>
<td>9958</td>
<td>The Corner Bookstore</td>
<td>36 North Miller Avenue</td>
<td>Syracuse</td>
<td>NY</td>
<td>13206</td>
</tr>
<tr>
<td>4000</td>
<td>9957</td>
<td>Advertising &amp; Graphic Design</td>
<td>2008 Delta Ave.</td>
<td>Cincinnati</td>
<td>OH</td>
<td>45208</td>
</tr>
<tr>
<td>5000</td>
<td>9969</td>
<td>National Learners</td>
<td>39 Gallimore Dairy Road</td>
<td>Greensboro</td>
<td>NC</td>
<td>27409</td>
</tr>
</tbody>
</table>

6. Close the Query Editor window without saving changes.
Mastery Builder 4–1
Organizing Data

Activity Time: 15 minutes

Scenario
The sales manager decided to analyze customer demands of titles in the company. To do this, she needs the total sales of each book present in the database so as to prepare a handout listing the books in descending order of sales. She also wants the list of representatives who have sold 2000 books or more, sorted in ascending order of total sales quantity.

Note: The file C:\094018Data\Organizing Data\Solution\Organizing Data.sql contains one possible solution for these tasks. Try to perform these tasks on your own before you refer to the solution file.

1. If SQL Server Management Studio is not running, launch it and connect to the server. Open a new query editor, and from the Available Databases list, select Pub1.

2. List the part number and sum of quantity from the Sales table and execute the query. Group by part number and order by the sum of quantity in descending order. Eliminate any NULL values in the quantity, and execute the query. You should get 55 rows.

3. Write a new query to list the representatives who have sold more than 2000 books, eliminating NULL values. Sort the list based on the sum of quantity in ascending order and execute the query. You should get five rows.

4. Close the Query Editor window without saving changes.
Mastery Builder 5–1
Retrieving Data from Multiple Tables

Activity Time: 15 minutes

Scenario
Management asks you to provide the following information:

• It is the holiday season and marketing has decided to send out discount coupons to both existing and potential customers. Provide a list that contains the names and addresses of all customers and potential customers.
• For an organizational internal audit, the auditor wants to review all sales and the names of the representatives responsible for each sale. The auditor asks you to provide a list of sales by order number, customer name, book title, sales representative name, and quantity sold.

Note: The file C:\094018Data\Retrieving Data from Multiple Tables\Solution\Retrieving Data from Multiple Tables.sql contains one possible solution for these tasks. Try to perform these tasks on your own before you refer to the solution file.

1. If SQL Server Management Studio is not running, launch it and connect to the server. Open a new query editor, and from the Available Databases list, select Pub1.

2. Write a query to retrieve the customer name and address from the customers in both the Customers and Potential_Customers tables. Display the results in a single result set.
   The query should produce 31 rows of data in this format.

<table>
<thead>
<tr>
<th>custname</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Advertising &amp; Graphic Design</td>
<td>2008 Delta Ave., Cincinnati, OH - 45208</td>
</tr>
<tr>
<td>2 Book Publishers, Inc.</td>
<td>3 Park Ave., Wilson, NC - 27893</td>
</tr>
<tr>
<td>3 Brainy Learning Center</td>
<td>615 Richmond Ave., Houston, TX - 77042</td>
</tr>
<tr>
<td>4 BxB Fitness</td>
<td>15 Main St., Syracuse, NY - 13206</td>
</tr>
<tr>
<td>5 Cards for All Occasions</td>
<td>99 South Downing, Westchester, IL - 60154</td>
</tr>
<tr>
<td>6 CK Music!</td>
<td>#1149 Blossom Road, Rochester, NY - 14610</td>
</tr>
<tr>
<td>7 Empire Books</td>
<td>811 Empire Blvd., Cincinnati, OH - 45209</td>
</tr>
</tbody>
</table>

3. Write a query that lists sales by order number, customer name, book title, sales representative name, and quantity sold.
   The query should produce 98 rows of data in this format.
4. Close the *Query Editor* window without saving changes.
Mastery Builder 6–1
Exporting Query Results

Activity Time: 15 minutes

Scenario
Your sales manager wants a list of book titles along with their sale price and development cost for those books where the development cost is not NULL. The sales manager wants to receive this list in a file he can use in Excel and also a web-based application that requires XML data.

Note: The file C:\094018Data\Exporting Query Results\Solution\Exporting Query Results.sql contains one possible solution for these tasks. Try to perform these tasks on your own before you refer to the solution file.

1. If SQL Server Management Studio is not running, launch it and connect to the server. Open a new query editor window, and from the Available Databases list, select Pub1.

2. Display book titles along with their respective development cost and sale price. Eliminate NULL values in development cost, and list book titles in ascending order.

3. Execute the query.

4. Save the result as My Books.csv

5. Add options to display the result as an XML document.

6. Save the result as My Books.xml
Glossary

**aggregate function**
A function that performs calculations on a set of values and returns a single value.

**AND operator**
A logical operator that returns TRUE only if both conditions are true.

**arithmetic operators**
Symbols used to perform mathematical calculations.

**BETWEEN...AND operator**
An operator that searches for an inclusive range of values specified by the start and end values.

**case conversion functions**
Functions that you can use to convert the case of a string.

**character extraction**
The process of extracting certain characters from a string value.

**client**
A computer that has applications to use the services provided by the server.

**column alias**
A meaningful name assigned to the column heading when the output is displayed.

**comment**
A non-executable set of words or statements describing the intent of code.

**comparison operators**
Symbols used to compare two expressions or values.

**concatenation**
A process of combining two string expressions into one string expression.

**condition**
A search criterion used to retrieve or manipulate specific information.

**cross join**
A join that displays one row for every possible pairing of rows from two tables.

**CUBE operator**
An operator that displays summary rows along with rows displayed by the GROUP BY clause.

**data type**
An attribute that determines the type of data that is stored in each column of a table.

**database**
Data organized and stored on a computer that can be searched and retrieved by a computer program.

**date function**
A function used to perform calculations on date columns that contain date and time information.

**DATEPART() function**
A date function that you use to specify the part of the date you want SQL Server to
return, such as the year, month, day, and hour.

**DENSE_RANK**
A ranking function that performs the same task as the RANK function, but assigns consecutive rank values for each row within a specified partition in a result set.

**DISTINCT keyword**
A keyword used to eliminate duplicate values in a list of values.

**FOR clause used with the XML option**
A clause used to return query results as an XML option.

**function**
A piece of code with a specified name and optional parameters that operates as a single logical unit, performs an action, and returns the result.

**GROUP**
A collection of two or more records combined into one unit based on one or more columns.

**GROUP BY clause**
A clause used to group rows based on grouping columns.

**HAVING clause**
A clause used to specify a search condition for a group or an aggregate value.

**IN operator**
A logical operator used to check that a given value matches any values in a list.

**information**
Data that has been processed, interpreted, organized, structured, or presented in a way that makes it meaningful or useful.

**inner join**
A join that displays records from two tables that have matching values.

**IS NULL clause**
A clause that checks that a NULL value is present.

**join**
A process of combining results obtained from two or more tables into one result and presenting it as the output.

**keyword**
A reserved word used for defining, manipulating, and accessing data.

**leading and trailing spaces**
Spaces that are present in a column when data stored in a column is less than the maximum number of characters that the column can contain.

**logical operator**
An operator that tests for the truth of a condition.

**LOWER**
A function used to convert uppercase characters to lowercase letters.

**LTRIM**
A function used to remove blank spaces before the values in a column.

**NOT operator**
A logical operator that reverses the result of a search condition.

**NTILE**
A ranking function that divides rows in each partition of a result set into a specified number of groups based on a given value and ranks them according to the partition.

**NULL**
A value that can be stored in a column when the value is either unknown or undefined.

**operators**
Symbols or words used in expressions to manipulate values.

**OR operator**
A logical operator that combines the output of two conditions and returns TRUE when either of the conditions is true.
ORDER BY
A clause used to sort rows displayed in the output based on the specified column names.

outer join
A join that selects all rows from one table along with the matching rows from the second table.

pattern matching
A method of searching for records that match a specific combination of characters.

PIVOT
A relational operator used to rotate column values from one column into multiple columns in the result set.

query
A request sent to the database to retrieve information from the database.

RANK
A function that returns a ranking value for each row within a specified partition in a result set.

ranking functions
Functions used to sequentially number the rows in a result set based on partitioning and ordering of the rows.

ROLLUP operator
An operator that displays, in a hierarchical order, summary rows along with the usual rows displayed by the GROUP BY clause.

ROW_NUMBER
One of the ranking functions that use sequential numbering to rank each row in the result set. A ranking function that returns a sequential number for each row within a specified partition in the result set.

RTRIM
A function used to remove blank spaces after the values in a column.

SELECT statement
An SQL statement used to retrieve information from tables present in the database.

self join
A join that relates data in a table to itself.

server
A computer that provides services to other computers on a network.

sorting
A method of arranging column values displayed in the output in either ascending or descending order.

SQL
Structured Query Language) The language you use to communicate with an SQL database. SQL consists of commands that you can use to retrieve, delete, and modify information in a database's tables.

SQL statement
An instruction written using the required syntax in SQL.

stored procedure
A database object that consists of one or more SQL statements. SQL Server compiles stored procedures in advance for optimized performance.

string
A collection of letters, numbers, or other characters in any combination.

string function
A function that performs an operation on a string input value and returns a string or numeric value.

SUBSTRING
A function used to extract characters from a given string.

syntax
The expected form of an instruction with clauses and placeholders for the actual elements that will be used in the instruction.

table
A collection of related information arranged in rows and columns.
table alias
A name provided to a table so that the table can be referred to by the alias name.

trim functions
Functions that enable you to remove the leading and trailing blank spaces that are part of a string of characters.

UNION operator
An operator used to combine the result of two or more queries into a single output.

UNPIVOT
A relational operator used to convert pivoted columns to column values of a single column.

UPPER
A function used to convert lowercase characters to uppercase letters.

WHERE clause
A clause used to include conditions.

wildcard
Characters used to search for patterns within data.
Index

A
aggregate functions 82
AND operator 42
arithmetic operators 39

B
BETWEEN...AND operator 56

C
case conversion functions 90
character extraction 92
clauses
FOR XML 170
GROUP BY 113, 114
HAVING 120
IS NOT NULL 58
IS NULL 58
ORDER BY 100
WHERE 37
cross joins 145
CUBE operator 123
CUBE subclause 115

D
databases
objects 2
overview 2
data sorting 100
data types
date/time 25
overview 25
SUBSTRING-supported 93
date functions
converting 76
overview 73
SMALLDATETIME 77
DATEPART() function 77
default column headings 40
DENSE_RANK function 105
DISTINCT keyword 83

E
EXCEPT operator 142
eXtensible Markup Language, See XML

F
FOR XML clause 170
functions
aggregate 82
case conversion 90
CAST() 76
CONVERT() 76
date 73
DATEPART() 77
DENSE_RANK 105
GROUPING_ID 115
LOWER 90
LTRIM 92
nesting 78
NTILE 107
overview 72
RANK 105
ranking 104
ROW_NUMBER 106, 107
RTRIM 92
string 88
SUBSTRING 92
TRIM 92
types of 72
UPPER 90

G
GROUP BY clause
ordering 114
overview 113
specifications for 114
GROUPING_ID function 115
grouping sets 115
groups 113

H
HAVING clause 120

I
information 8
inner joins 146
IN operator 57
INTERSECT operator 142
IS NOT NULL clause 58
IS NULL clause 58, 59

J
joins
cross 145
inner 146
multiple table 150
outer 148
overview 145
self 149

K
keywords
DISTINCT 83
overview 83
TOP n 107

L
leading and trailing spaces 91
LIKE operator 64
logical operators 41, 42
LOWER function 90

M
multiple conditional operators 38
multiple table joins 150

N
NOT operator 42
NTILE function 107
NULL 45, 57

O
operators
AND 42
arithmetic 39
BETWEEN...AND 56
comparison 39
CUBE 123
EXCEPT 142
IN 57
INTERSECT 142
LIKE 64
logical 41, 42
multiple conditional 38
NOT 42
OR 42
PIVOT 128
precedence 44
ROLLUP 123
single conditional 38
SQL Server 38
UNION 136
UNPIVOT 129
ORDER BY clause 100
OR operator 42
outer joins 148

P
pattern matching 64
PIVOT operator 128

Q
qualified names 147
queries
  overview 14
  processing order 14
  saving 23
Query Editor 9

R
  RANK function 105
  ranking functions 104
  ROLLUP operator 123
  ROLLUP subclause 115
  ROW_NUMBER function 106, 107

S
  SELECT statement
    optional clauses 17
    overview 14
  self joins 149
  servers 4
  single conditional operators 38
  sort 100
  spaces 91
  SQL
    arithmetic operators 40
    command groups 7
    keywords 83
    operators 38
    overview 7
    pronunciation 7
    Server data types 26
    wildcards 63
  stored procedures 27
  string functions 88
  strings 88
  Structured Query Language, See SQL
  subclauses
    CUBE 115
    ROLLUP 115
  SUBSTRING function 92
  syntax
    notation 17
    overview 15

T
  table aliases 149
  tables 3
  text data formats 166
  TOP n keyword 107
  trim functions 92

U
  UNION operator 136
  UNPIVOT operator 129
  UPPER function 90

W
  WHERE 41
  WHERE clause 37
  wildcards 63, 64
  windows
    Query Editor 9

X
  XML 170