STUDENT MANUAL

SQL Querying: Fundamentals

SQL Querying: Fundamentals

SQL Querying: Fundamentals

Part Number: 094005
Course Edition: 1.0

Acknowledgements

PROJECT TEAM

Author	*Media Designer*	*Content Editor*
Rozanne Whalen	Alex Tong	Joe McElveney

Notices

DISCLAIMER

TRADEMARK NOTICES

SQL Querying: Fundamentals

About This Course

Many organizations use databases to store their most critical information: the information that manages their day-to-day operations. After the data is stored in databases, however, it is useless unless you can retrieve it for further business analysis. One example of a database management system is Microsoft SQL Server 2012. The language you use to retrieve information from SQL Server 2012 databases is the Structured Query Language (SQL). This course, *SQL Querying: Fundamentals*, 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, and 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 are highly recommended:

- *Using Microsoft® Windows® 8.1*
- *Microsoft® Windows® 8 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 on-screen.
- Retrieve data from multiple tables.
- Export the results of a query.

The LogicalCHOICE Home Screen

The LogicalCHOICE Home screen is your entry point to the LogicalCHOICE learning experience, of which this course manual is only one part. Visit the LogicalCHOICE Course screen both during and after class to make use of the world of support and instructional resources that make up the LogicalCHOICE experience.

Log-on and access information for your LogicalCHOICE environment will be provided with your class experience. On the LogicalCHOICE Home screen, you can access the LogicalCHOICE Course screens for your specific courses.

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

- eBook: an interactive electronic version of the printed book for your course.
- LearnTOs: brief animated components that enhance and extend the classroom learning experience.

Depending on the nature of your course and the choices of your learning provider, the LogicalCHOICE Course screen may also include access to elements such as:

- The interactive eBook.
- 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.
- Checklists with useful post-class reference information.
- Any course files you will download.
- The course assessment.
- Notices from the LogicalCHOICE administrator.
- Virtual labs, for remote access to the technical environment for your course.
- Your personal whiteboard for sketches and notes.
- Newsletters and other communications from your learning provider.
- Mentoring services.
- A link to the website of your training provider.
- The LogicalCHOICE store.

Visit your LogicalCHOICE 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 practice the guidelines and procedures as well as to solidify your understanding of the informational material presented in the course. Procedures and guidelines are presented in a concise fashion along with activities and discussions. Information is provided for reference and reflection in such a way as 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 LogicalCHOICE 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.

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.

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 these visual cues:

Icon	*Description*
	A **Note** provides additional information, guidance, or hints about a topic or task.
	A **Caution** helps make 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.
	**LearnTO** notes show you where an associated LearnTO is particularly relevant to the content. Access LearnTOs from your LogicalCHOICE Course screen.
	**Checklists** provide job aids you can use after class as a reference to performing skills back on the job. Access checklists from your LogicalCHOICE Course screen.
	**Social** notes remind you to check your LogicalCHOICE Course screen for opportunities to interact with the LogicalCHOICE community using social media.
	**Notes Pages** are intentionally left blank for you to write on.

1 Executing a Simple Query

Lesson Time: 45 minutes

Lesson Objectives

In this lesson, you will connect to the SQL Server database and execute a simple query. You will:

- Connect to the database using SQL Server Management Studio.
- Query the database.
- Save a query for future use.
- Modify an existing query.

Lesson Introduction

In this course, you will query a SQL Server 2012 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.

A typical database can be a vast collection of information. Imagine a scenario where you have to maintain a database for a publishing company. This database might contain information about books, customers, orders placed, representatives, and sales. This information cannot be put to effective use unless you know how to retrieve it. Becoming familiar with the basic language used to communicate with databases enables you to retrieve, add, and update data.

TOPIC A

Connect to the SQL Database

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

Organizations use databases to store large amounts of data such as inventory, sales, and customer information. 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.

Databases

A *database* is a collection of 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. Some databases are created by default when you install a database application such as Microsoft® SQL Server® 2012, and you can create and customize other databases to suit your business needs.

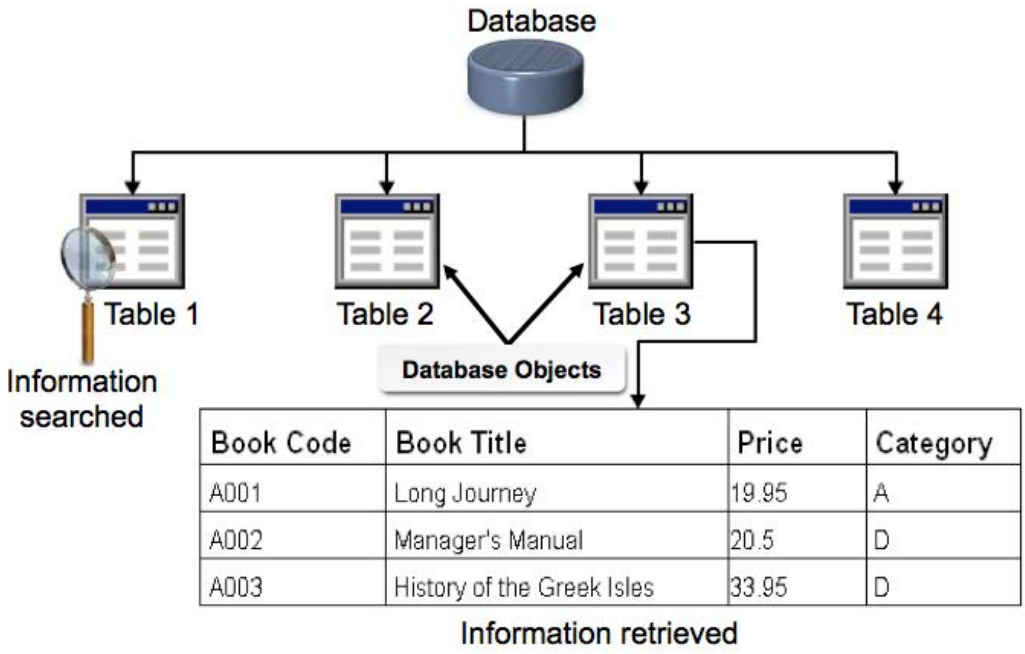


Figure 1-1: The Books table.

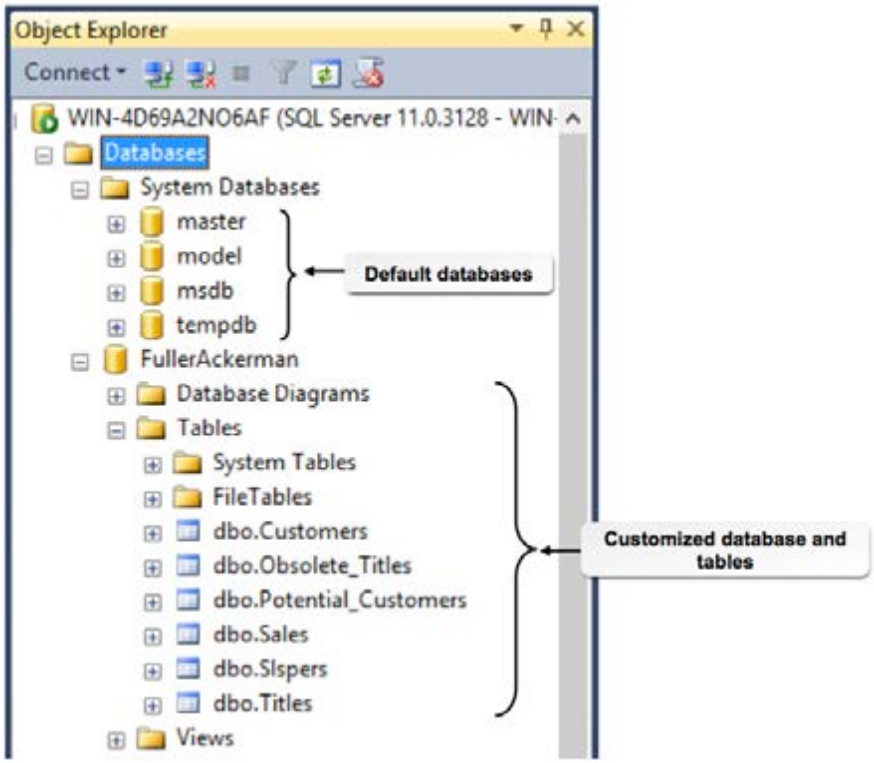
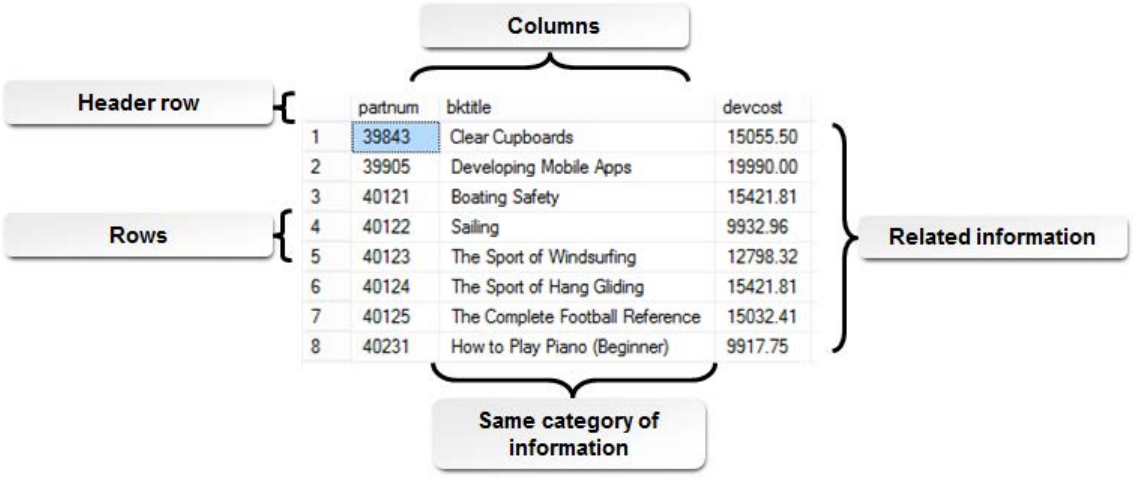


Figure 1-2: Default and customized databases.

Tables

One of the objects a database contains is tables. You use tables to store the information contained in the database. 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.

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.



	partnum	bktitle	devcost
1	39843	Clear Cupboards	15055.50
2	39905	Developing Mobile Apps	19990.00
3	40121	Boating Safety	15421.81
4	40122	Sailing	9932.96
5	40123	The Sport of Windsurfing	12798.32
6	40124	The Sport of Hang Gliding	15421.81
7	40125	The Complete Football Reference	15032.41
8	40231	How to Play Piano (Beginner)	9917.75

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

Servers

Database administrators install database applications such as Microsoft SQL Server 2012 on servers. 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. Users do not work on servers directly. Instead, users use computers and software applications that access the data stored on the servers. Servers manage resources and provide services to users that access information contained in the server.

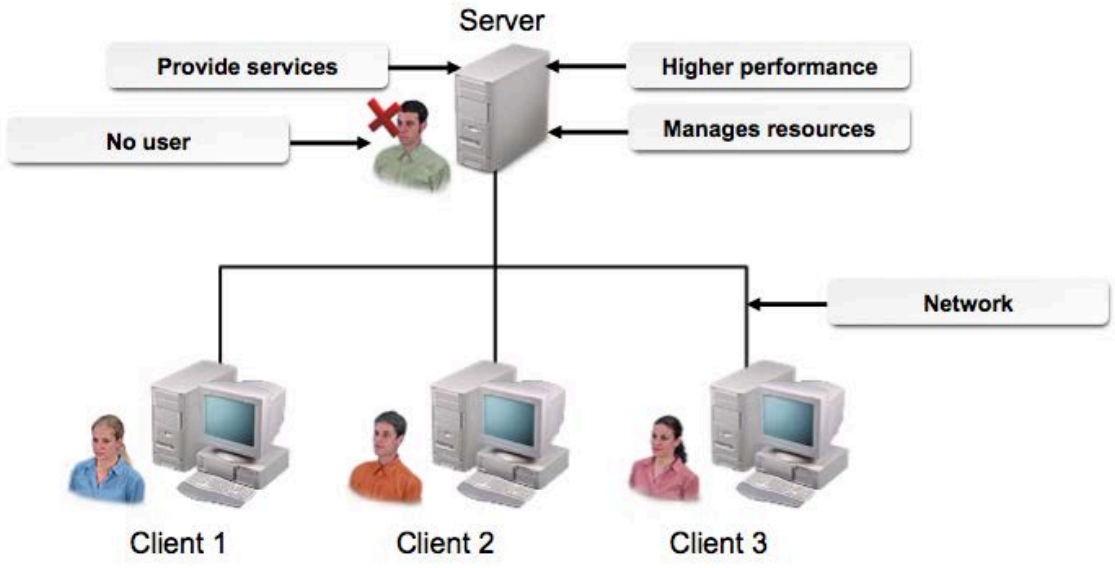


Figure 1-4: A server connected to three clients.

Clients

A *client* is a computer consisting of the interface that enables users to request the services of a server and display the results returned by the server. The client computer can be located on the same network as the server or could connect to the server remotely, typically via the Internet. Client computers typically run an operating system such as Microsoft® Windows®.

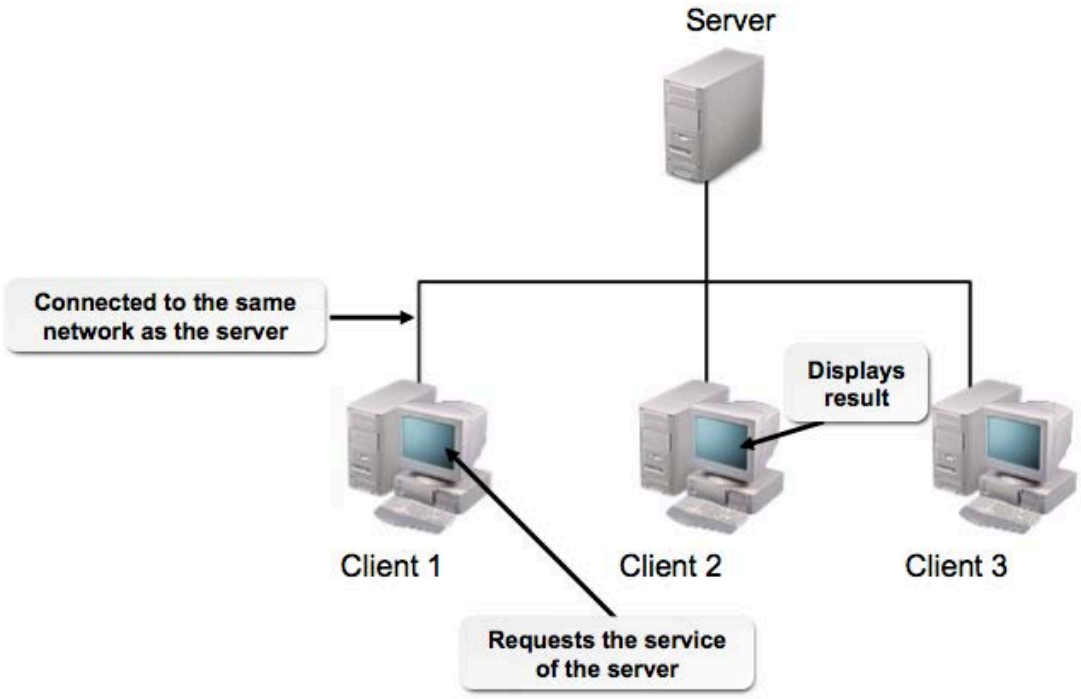


Figure 1-5: Clients connected to a server.

The Client/Server Architecture

The client/server architecture is the architecture of a computer network consisting of a server and one or more clients with processing functions distributed between the server and the client. The client/server architecture is also referred to as two-tier architecture. Microsoft SQL Server 2012 uses the client/server architecture in which multiple clients connect to one or more SQL servers to transfer data over a network.

Tiered Architecture

The *tiered architecture* helps in the usage and maintenance of a database by providing various views to different levels of users. With the tiered architecture, application developers can reuse specific blocks of code or write development programs. It enables the server to respond efficiently to client requests and also allows end users to access a database irrespective of the database design.

Cloud-Based Computing

In some organizations, network administrators install database servers directly on the organization's network. In other organizations, network administrators choose to implement database servers that are hosted by third-party providers on the Internet. One reason network administrators elect to use database servers that are hosted on the Internet 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.

A term you'll hear frequently to describe the Internet is the cloud. The term "cloud-based computing" thus refers to the act of connecting to and using servers that are hosted on the Internet instead of on your local network.

SQL

Structured Query Language (SQL) is the language you use to communicate with a SQL database. SQL consists of commands that you can use to retrieve, delete, and modify information in a database's

tables. SQL is made up of three major language components: Data Manipulation Language (DML), Data Definition Language (DDL), and Data Control Language (DCL).

You use the DML commands to view, change, and manipulate data within a table. DML includes commands to select, update, and insert data in a table, and delete data from a table. The DDL commands enable you to create and define the database and objects within it. DDL includes the commands to create and delete tables. DDL is typically used by database administrators and programmers. The DCL commands are used to control access to the data in a database. It includes commands to grant and revoke database privileges. As with DDL commands, DCL commands are also typically used by database administrators and programmers.

SQL Language Components

SQL language has several commands that enable you to create, query, update, and maintain a database. Some examples of SQL commands are listed in the following table.

DML Commands	*DDL Commands*	*DCL Commands*
SELECT	CREATE TABLE	GRANT
UPDATE	DROP TABLE	REVOKE
INSERT	CREATE VIEW	
DELETE	DROP VIEW	

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 pane contains the **Results** pane, which displays the results of queries, and the **Messages** tab, which displays information about the query that you execute.

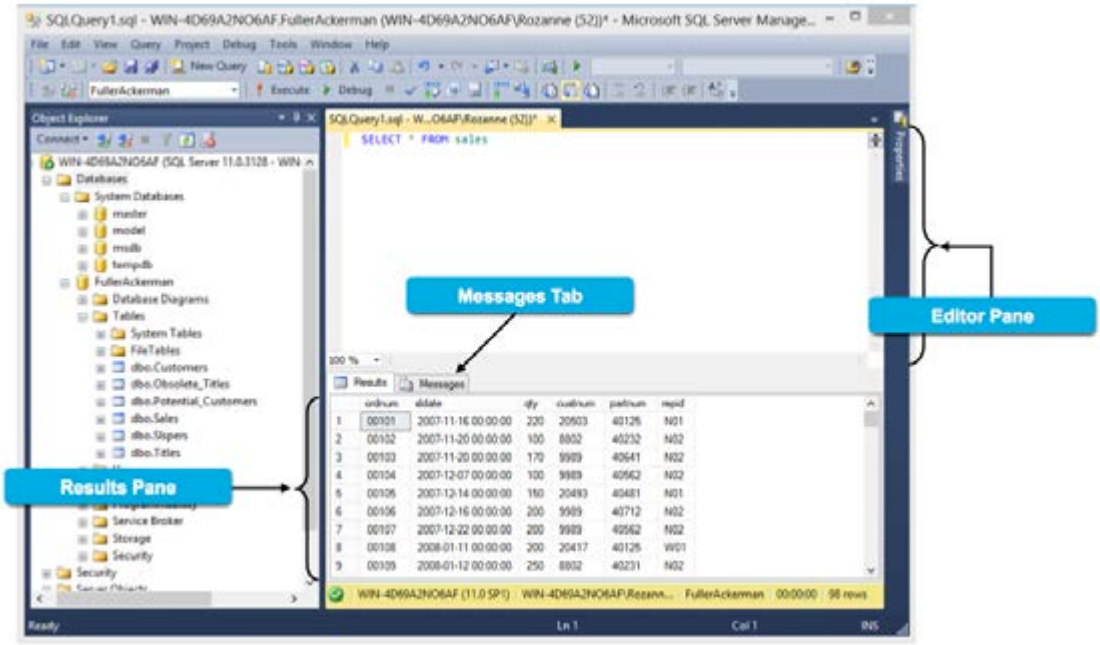


Figure 1-6: The Query Editor window consisting of a statement and results.

	Access the Checklist tile on your LogicalCHOICE course screen for reference information and job aids on How to Connect to a Database

ACTIVITY 1-1
Connecting to a Database

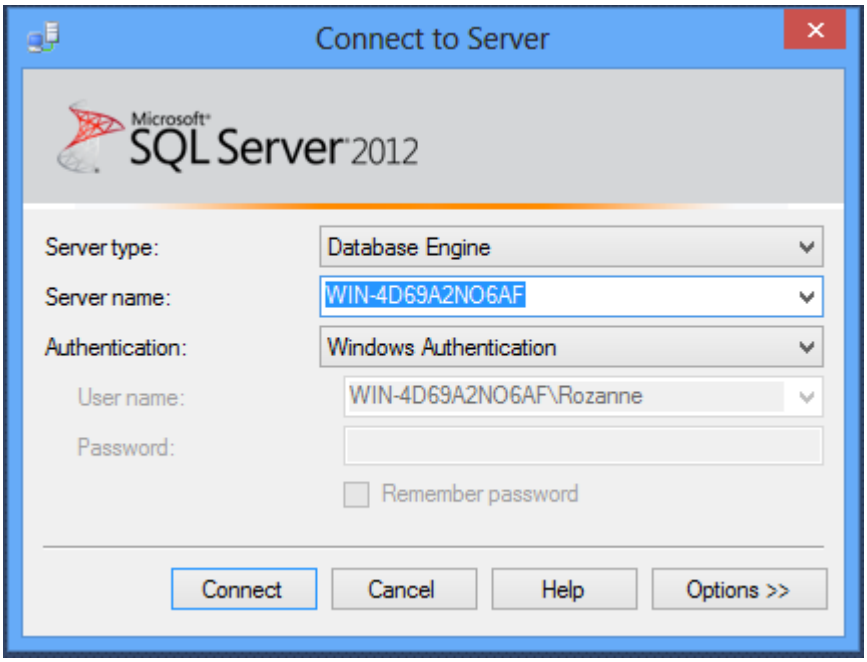
Scenario

You are an employee at Fuller & Ackerman Publishing. Organizational data is stored and maintained in a SQL Server 2012 database called FullerAckerman. This database contains information about customers, titles that are published by the company, obsolete titles, sales transactions, and details of sales representatives working for the company. Your day-to-day activities require you to retrieve information from this database. For information on tables and column names in the FullerAckerman 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 **SQL Server Management Studio**.

   b) In the **Connect to Server** dialog box, in the **Server type** drop-down list, verify that **Database Engine** is selected.
   c) In the **Server name** drop-down list, verify that the name of the server is automatically selected. In this case, the server name is the same as your computer's name.
   d) In the **Authentication** drop-down list, verify that **Windows Authentication** is selected.

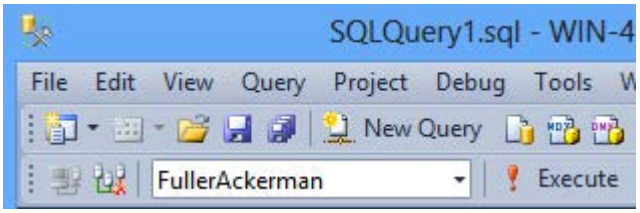


e) Select **Connect** to connect to the server.

2. Connect to the FullerAckerman database.
   a) On the **Standard** toolbar, select **New Query** to open the **Query Editor** window.
   b) In the **Editor** pane, type:

```
Use FullerAckerman
```

   c) On the **SQL Editor** toolbar, select **Execute**.
   d) In the **Messages** pane, observe that the message **Command(s) completed successfully** displays. Notice that FullerAckerman now displays in the **Available Databases** drop-down list. You have successfully logged in to SQL Server and connected to the FullerAckerman database.



   e) In the **Query Editor** window, select the **Close** button.
   f) When you are prompted to save changes, select **No**.

TOPIC B

Query a Database

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

When information is stored in a database's tables, organizations want to view this information to perform tasks such as generating mailing lists or developing reports to analyze their customers' purchasing habits. The only command you use in SQL Server to view the contents of tables in a database is the SELECT statement. Mastering the use of the SELECT statement will thus enable you to find and retrieve the information you need.

Syntax

Syntax is the expected form of a command, including any clauses and placeholders for actual elements that you use with the command. Clauses used in an instruction should all appear in the precise order specified in the syntax. For example, you use the SELECT command to retrieve information from a table. 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.

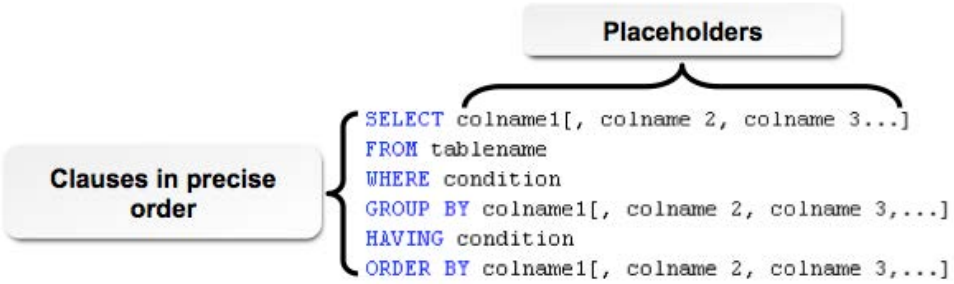


Figure 1-7: Syntax illustration.

SQL Statements

A *SQL statement* is a command written in SQL using the appropriate syntax. You must use the required clauses and keywords in the command's syntax structure when writing a SQL statement. You can also include optional clauses. For example, you can optionally use the WHERE clause with a SELECT statement to display only specific rows in a table instead of all rows. Notice that the syntax of the SELECT statement requires you to place the WHERE clause after the FROM clause.

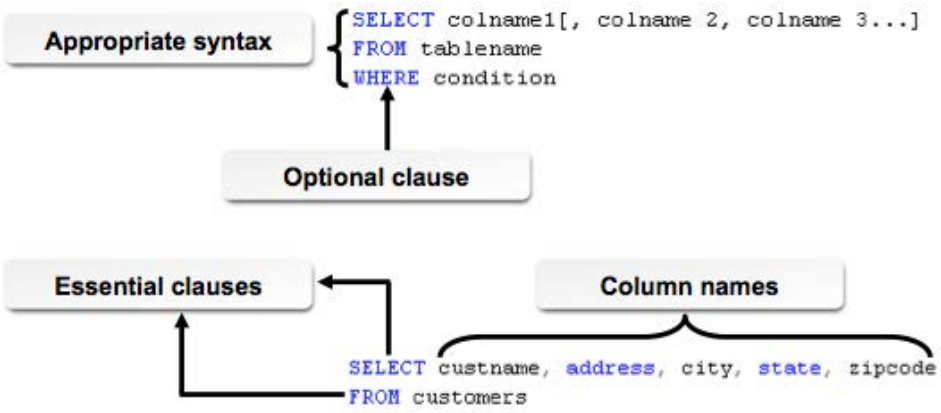


Figure 1-8: The syntax and example of a SQL statement.

Queries

A *query* is a `SELECT` SQL statement that requests information from tables present in a database. A query requires the column and table names to generate the output. You can include optional clauses in a query to retrieve specific information.

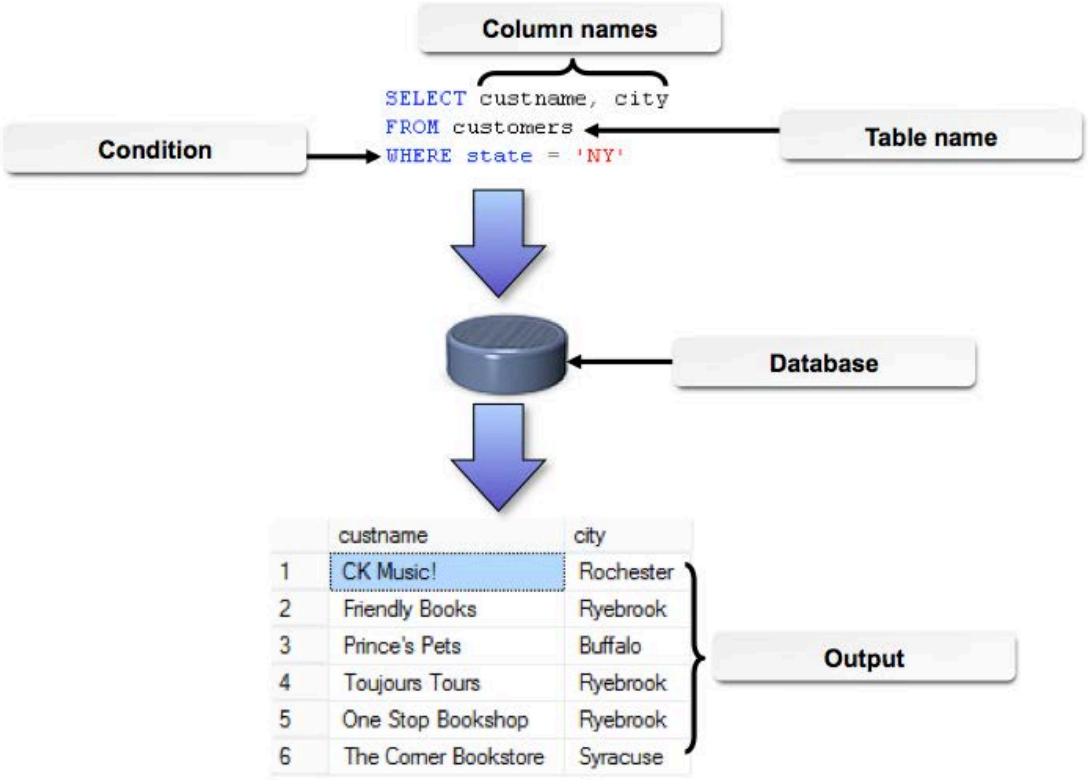


Figure 1-9: A query to retrieve records from the Customers table.

The SELECT Statement

A *SELECT statement* is a SQL statement or query you use to retrieve information from a table. A simple `SELECT` statement consists of two parts: a `SELECT` clause that includes all of the column

names that you want to display in the output, and a FROM clause that identifies the table name from which you want to retrieve the columns.

The order of column names you include in the SELECT statement determines the order in which SQL Server displays the columns in the output. If you specify more than one column name, you must use commas to separate the column names. When you want to display all of the columns of a table in the output, you can use an asterisk (*) instead of the column names.

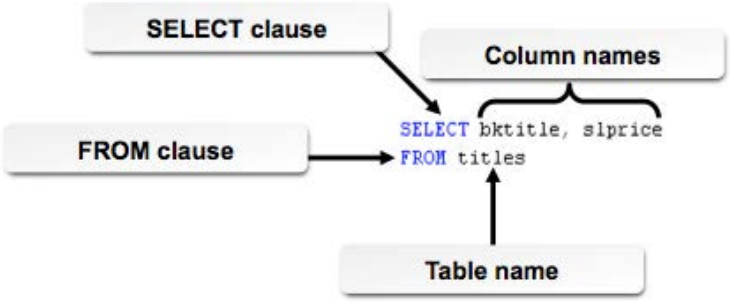


Figure 1-10: A SELECT clause that displays selected columns for all records in the Titles table.

You can use additional clauses with the SELECT statement, if necessary. For example, you might use the WHERE clause to select only certain rows in the table based on criteria you specify.

	partnum	bktitle	devcost	slprice	pubdate
1	39843	Clear Cupboards	15055.50	49.95	2012-08-19 00:00:00
2	39905	Developing Mobile Apps	19990.00	45.00	2013-01-01 00:00:00
3	40121	Boating Safety	15421.81	36.50	2013-05-18 00:00:00
4	40122	Sailing	9932.96	29.15	2013-05-03 00:00:00
5	40123	The Sport of Windsurfing	12798.32	38.50	2012-07-13 00:00:00
6	40124	The Sport of Hang Gliding	15421.81	49.68	2013-01-06 00:00:00
7	40125	The Complete Football Reference	15032.41	49.99	2012-08-03 00:00:00
8	40231	How to Play Piano (Beginner)	9917.75	25.00	2012-06-11 00:00:00
9	40232	How to Play Piano (Intermediate)	8565.35	20.50	2012-10-22 00:00:00

Titles

Figure 1-11: A SELECT clause to display all columns and records from the Titles table.

Syntax of the SELECT Statement

The basic syntax of the SELECT statement is:

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

In this syntax, square brackets [...] are used to enclose optional parameters. You must specify at least one column name in the SELECT statement or use the asterisk to request all columns in the table. You can optionally specify more than one column name.

```
SELECT *
FROM tablename
```

In the aforementioned example, the asterisk in the SELECT statement informs SQL Server to retrieve all columns from the table.

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.

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.

Optional Clause	*Purpose*
WHERE	A clause that enables you to request only certain rows from a table. For example, you might use a WHERE clause when querying a customer table to retrieve a list of only the customers who live in Florida.
GROUP BY	A clause that uses a column identifier to organize the data in the result set into groups.
HAVING	A clause that you use in conjunction with the GROUP BY clause in order to specify which groups to include in the results.
ORDER BY	A clause that enables you to sort query results by one or more columns and in ascending or descending order.

Access the Checklist tile on your LogicalCHOICE course screen for reference information and job aids on How to Query a Database

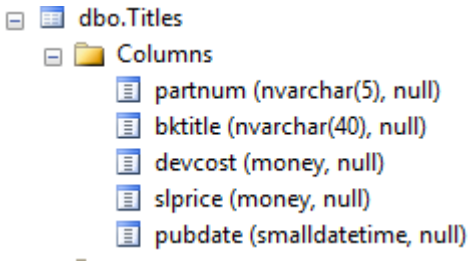
ACTIVITY 1-2
Querying a Database

Scenario

The sales manager at Fuller & Ackerman Publishing would like to be able to give a list of available book titles published by the company to a newly hired sales representative. You have been asked to prepare this list. You know that the information is contained in the FullerAckerman database but aren't familiar with the tables it contains. For information on the tables and column names in the FullerAckerman database, refer to Appendix A.

---

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, expand **Databases**.
   b) In the **Databases** folder, first expand **FullerAckerman**,**Tables**, **dbo.Titles**, and then **Columns** to view the columns contained in the table.



   c) Verify that the **Columns** folder contains the column **bktitle (nvarchar(40), NULL)**. Fuller & Ackerman stores the titles of the books it publishes in this column.

2. Enter a query to display book titles and run the query.
   a) On the **Standard** toolbar, select **New Query** to open a new **Query** pane.
   b) On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, select **FullerAckerman**.
   c) In the **Query Editor** pane, type the following query:

```
SELECT *
FROM titles
```

   d) Observe that the keywords are displayed in blue and the table name is displayed in green.
   e) On the **SQL Editor** toolbar, select **Execute** to execute the query.
   f) 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. You should see a total of 92 rows.

	partnum	bktitle	devcost	slprice	pubdate
1	39843	Clear Cupboards	15055.50	49.95	2012-08-19 00:00:00
2	39905	Developing Mobile Apps	19990.00	45.00	2013-01-01 00:00:00
3	40121	Boating Safety	15421.81	36.50	2013-05-18 00:00:00
4	40122	Sailing	9932.96	29.15	2013-05-03 00:00:00
5	40123	The Sport of Windsurfing	12798.32	38.50	2012-07-13 00:00:00
6	40124	The Sport of Hang Gliding	15421.81	49.68	2013-01-06 00:00:00

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 enables you to save your queries in query 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.

Query Saving

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 LogicalCHOICE course screen for reference information and job aids on How to Save a Query

ACTIVITY 1-3
Saving a Query

Scenario

After some trial and error, 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 a SQL file. For information on tables and column names in the FullerAckerman database, refer to the table structure in Appendix A.

1. Save the query as ***My First SQL*** 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:\094005Data\Executing a Simple Query** folder.
   c) In the **File name** text box, double-click and type ***My First SQL***
   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) In the **Query Editor** window, select the **Close** button to close the window.

2. **True or False? A saved query is automatically named after the database table name being queried.**
   ☐ True
   ☐ False

TOPIC D

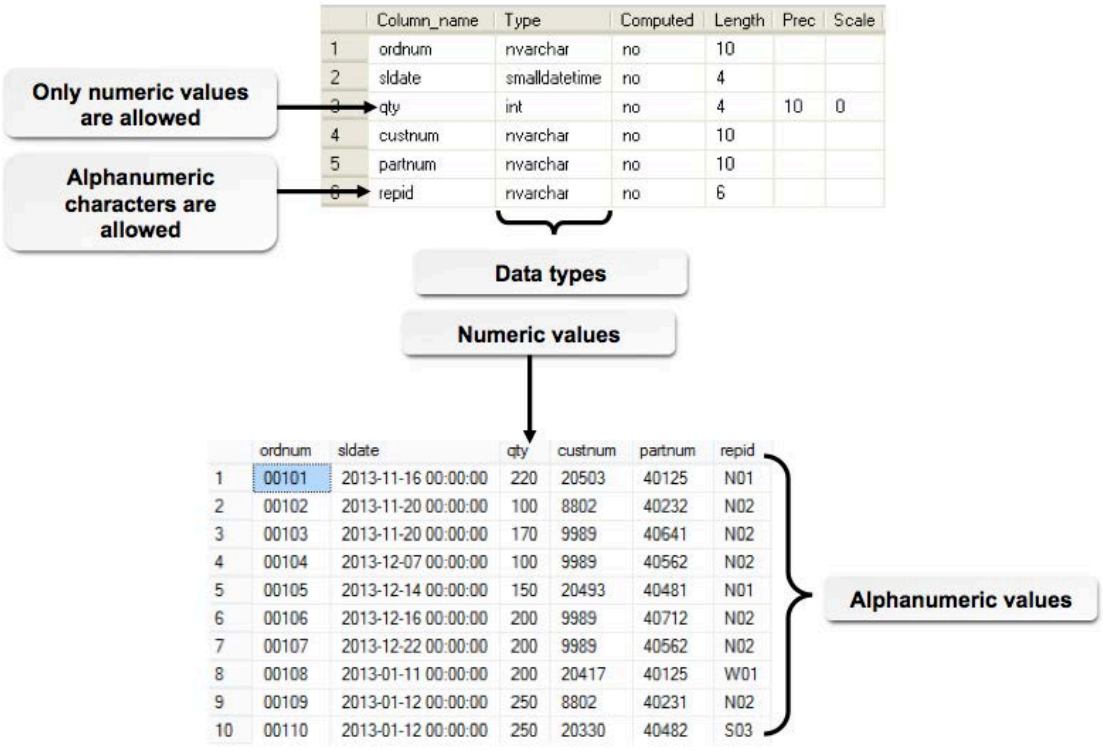
Modify and Execute a Saved Query

After you have saved a query, you can edit it as needed. Simply open the saved query, make any necessary changes, and then execute the query.

If the data you need to retrieve changes, you can easily open saved query script files in SSMS and modify them to reflect these changes. 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

When a database administrator creates a table, she must specify the data type of each column. The *data type* is the classification of data into groups based on their characteristics. The characteristics can include how many characters the value contains, whether it is a number or text, or whether it includes a decimal. The data type determines what calculations you can perform with that data. All values entered into a SQL database can be classified into one of the data types available in Microsoft SQL Server 2012. The data type of the value entered into a column must match the data type the database administrator assigned to that column.



	Column_name	Type	Computed	Length	Prec	Scale
1	ordnum	nvarchar	no	10		
2	sldate	smalldatetime	no	4		
3	qty	int	no	4	10	0
4	custnum	nvarchar	no	10		
5	partnum	nvarchar	no	10		
6	repid	nvarchar	no	6		

	ordnum	sldate	qty	custnum	partnum	repid
1	00101	2013-11-16 00:00:00	220	20503	40125	N01
2	00102	2013-11-20 00:00:00	100	8802	40232	N02
3	00103	2013-11-20 00:00:00	170	9989	40641	N02
4	00104	2013-12-07 00:00:00	100	9989	40562	N02
5	00105	2013-12-14 00:00:00	150	20493	40481	N01
6	00106	2013-12-16 00:00:00	200	9989	40712	N02
7	00107	2013-12-22 00:00:00	200	9989	40562	N02
8	00108	2013-01-11 00:00:00	200	20417	40125	W01
9	00109	2013-01-12 00:00:00	250	8802	40231	N02
10	00110	2013-01-12 00:00:00	250	20330	40482	S03

Figure 1-12: A table structure displaying various column types.

Note: To further explore data types, you can access the LearnTO **Identify SQL Data Types** presentation from the **LearnTO** tile on the LogicalCHOICE Course screen.

Data Types Available in SQL Server

The SQL Server data types are listed in the following table.

Data Type	*Used to Store*
bigint	The integer (whole numbers) data range from –2^63 (–9,223,372,036,854,775,808) through 2^63 – 1 (9,223,372,036,854,775,807) with a storage size of 8 bytes.
int	The integer (whole numbers) data range from –2^31 (–2,147,483,648) through 2^31 – 1 (2,147,483,647) with a storage size of 4 bytes.
smallint	The integer data range from –2^15 (–32,768) through 2^15 – 1 (32,767) with a storage size of 2 bytes.
tinyint	The integer data range from 0 through 255 with a storage size of 1 byte.
bit	The integer data type that can take a value of 1, 0, or NULL.
decimal	Fixed precision and scale numeric data from –10^38 + 1 through 10^38 – 1.
numeric	Functionally equivalent to a decimal.
money	The monetary data value range from –2^63 (–922,337,203,685,477.5808) through 2^63 – 1 (+922,337,203,685,477.5807), with accuracy to a ten-thousandth of a monetary unit with a storage size of 8 bytes.
smallmoney	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.
float	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.
real	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.
datetime	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.
smalldatetime	Date and time data from January 1, 1900, through June 6, 2079, with an accuracy of one minute.
char	Fixed-length non-Unicode character data with a maximum storage size of 8,000 bytes.
varchar	Variable-length non-Unicode data with a maximum storage size of 8,000 bytes.
varchar(max)	Variable-length non-Unicode data with a maximum storage size of 2^31 – 1 bytes.
nchar	Fixed-length Unicode data with a maximum length of 4,000 characters.
nvarchar	Variable-length Unicode data with a maximum length of 4,000 characters. sysname is a system-supplied user-defined data type that is functionally equivalent to nvarchar (128) and is used to reference database object names.
nvarchar(max)	Variable-length Unicode data with a maximum length of 2^31 – 1 bytes.
binary	Fixed-length binary data with a maximum length of 8,000 bytes.
varbinary	Variable-length binary data with a maximum length of 8,000 bytes.
varbinary(max)	Variable-length binary data with a maximum length of 2^31 – 1 bytes.
cursor	A reference to a cursor.

Data Type	Used to Store
sql_variant	A data type that stores values of various SQL Server–supported data types, except for text, ntext, timestamp, and sql_variant.
table	A special data type used to store a result set for later processing.
timestamp	A database-wide unique number that gets updated every time a row gets updated.
uniqueidentifier	A globally unique identifier (GUID).
xml	A built-in data type that stores the XML documents and fragments in a SQL Server database. The stored representation of xml data type cannot exceed 2 GB.

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 a 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 2012 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:

```
sp_help Titles
```

Note: To further explore stored procedures, you can access the LearnTO **Use System Stored Procedures** presentation from the **LearnTO** tile on the LogicalCHOICE 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 a SQL statement. You indicate a single line comment by preceding it with two hyphens. You can specify multiple line comments by enclosing the lines within the `/*` and `*/` characters.

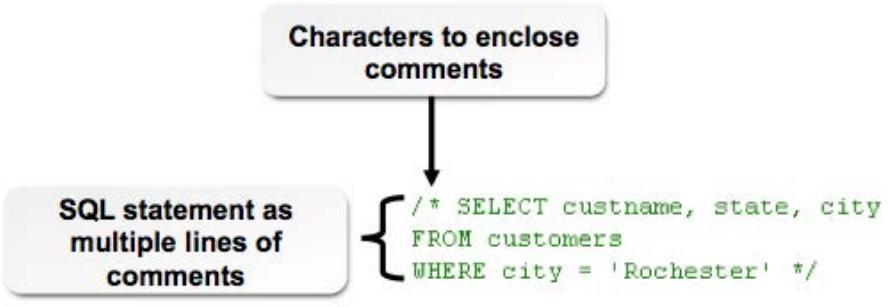


Figure 1-13: A SELECT statement enclosed within comments.

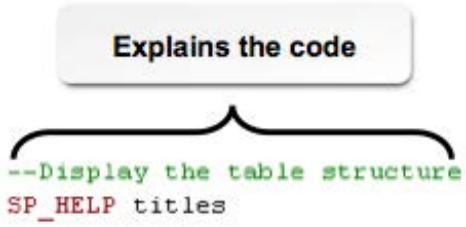


Figure 1-14: A comment statement explaining code.

Figure 1-15: A non-executable comment statement.

	Access the Checklist tile on your LogicalCHOICE course screen for reference information and job aids on How to Work with Saved Queries

ACTIVITY 1-4
Modifying a Saved Query

Scenario
You retrieved all the information from the Titles table to provide a list of titles to the newly hired sales representative. After retrieving the information, you realize that there is no need to provide the sales representative with the information about development cost and the publication date. You have also decided to add a comment to the file to document the `SELECT` statement's purpose. For information on tables and column names in the FullerAckerman database, refer to the table structure in Appendix A.

---

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 **FullerAckerman**.
   c) In the **Editor** pane, type ***sp_help titles***
   d) Select **Execute** to run the query.

Results | Messages

	Name	Owner	Type	Created_datetime
1	Titles	dbo	user table	2013-10-17 13:40:32.

	Column_name	Type	Computed	Length
1	partnum	nvarchar	no	10
2	bktitle	nvarchar	no	80
3	devcost	money	no	8
4	slprice	money	no	8
5	pubdate	smalld...	no	4

2. **Based on the scenario, which columns should you include in your `SELECT` statement?**

3. Close the **Editor** pane without saving the sp_help query.
   a) In the **Editor** pane, select **Close**.
   b) When you're prompted to save the query, select **No**.

4. Open the **My First SQL.sql** file.
   a) Select **File→Open→File**.
   b) In the **Open File** dialog box, navigate to the **C:\094005Data\Executing a Simple Query** folder.
   c) Select the **My First SQL** file and select **Open** to open the query file.

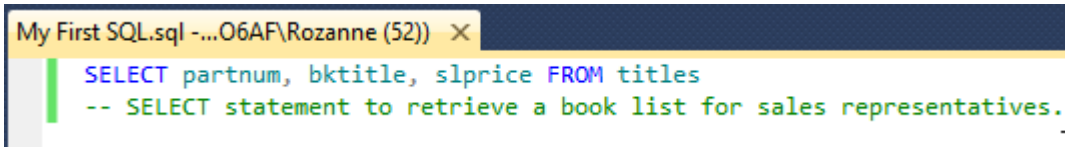
5. Modify the `SELECT` statement to display only the necessary columns and verify the query works.
   a) Edit the `SELECT` statement to read:

```
SELECT partnum, bktitle, slprice
FROM titles
```

b) From the **Available Databases** drop-down list, select **FullerAckerman**.
c) Select the **Execute** button to run the query and verify that it works. You should see 92 rows.

	partnum	bktitle	slprice
1	39843	Clear Cupboards	49.95
2	39905	Developing Mobile Apps	45.00
3	40121	Boating Safety	36.50
4	40122	Sailing	29.15
5	40123	The Sport of Windsurfing	38.50
6	40124	The Sport of Hang Gliding	49.68
7	40125	The Complete Football Reference	49.99
8	40231	How to Play Piano (Beginner)	25.00

6. Add a comment to the query.
   a) On the next line after the `SELECT` statement, type **--** to begin the comment line.
   b) After the --, type ***SELECT statement to retrieve a book list for sales representatives.***



   Notice that the **Editor** pane displays the comment in green.

7. Save the modified query.
   a) Select **File→Save My First SQL.sql As**.
   b) If necessary, in the **Save File As** dialog box, navigate to the **C:\094005Data\Executing a Simple Query** folder.
   c) In the **File name** text box, type ***My Modified Query*** and then select **Save** to save the modified query.
   d) Close the **Query Editor** window.

ACTIVITY 1-5
Executing a Saved Query

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:\094005Data\Executing a Simple Query** folder.
   c) Select the **My Modified Query** file and select **Open** to open the query file.
   d) On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, select **FullerAckerman**.

2. Execute the query.
   a) Select **Execute** to run the query.
   b) In the **Results** pane, observe that the part number along with the book title and sale price is 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?

Which SQL Server language component would you use the most in your current job? Why?

Note: Check your LogicalCHOICE Course screen for opportunities to interact with your classmates, peers, and the larger LogicalCHOICE 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.

2 Performing a Conditional Search

Lesson Time: 1 hour, 45 minutes

Lesson Objectives

In this lesson, you will include a search condition in a simple query. 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.

Lesson Introduction

At this point, you have connected to a SQL 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.

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 SQL Server's performance by including conditional search criteria in your `SELECT` queries.

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

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 FullerAckerman 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.

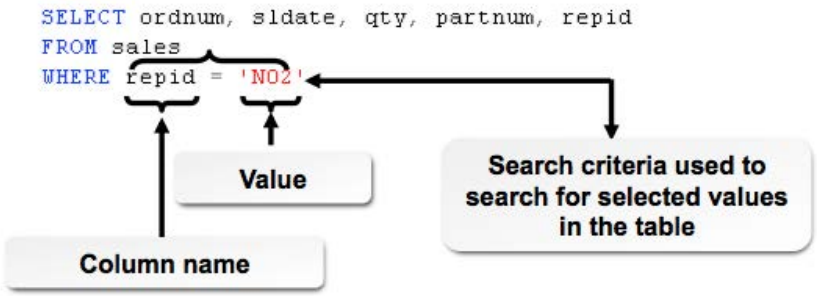


Figure 2-1: A SELECT statement displaying a single search condition.

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 every row in 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.

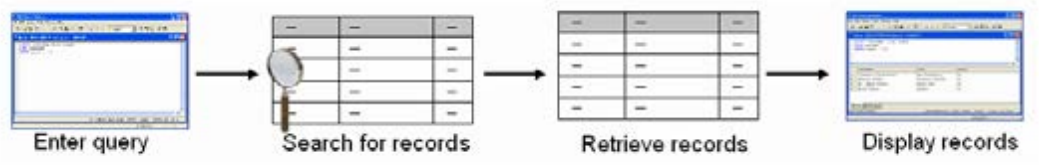


Figure 2-2: Steps involved in a conditional search.

The WHERE Clause

The *WHERE clause* is a clause you use to specify a search condition in a 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.

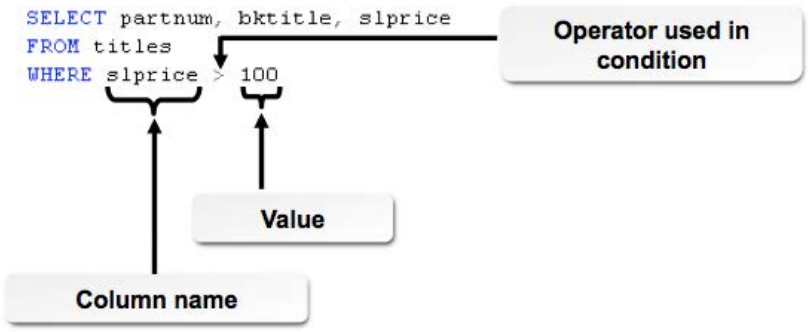


Figure 2-3: A SELECT statement containing a single condition.

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.

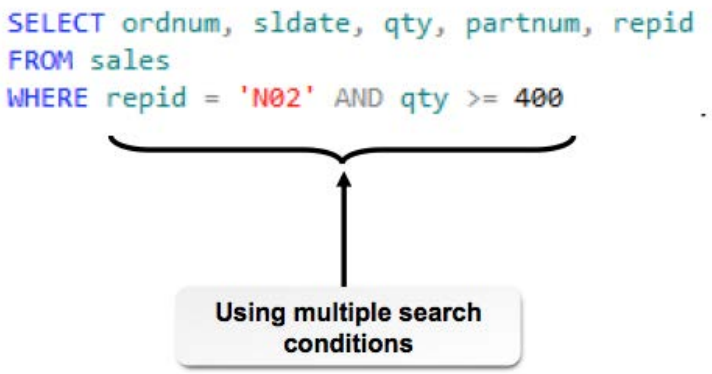


Figure 2-4: A SELECT statement displaying multiple conditions.

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.

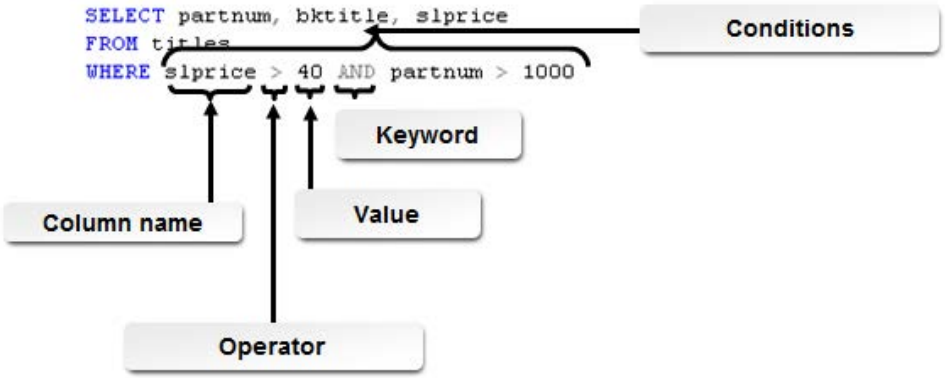


Figure 2-5: A SELECT statement displaying multiple conditional operators.

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.

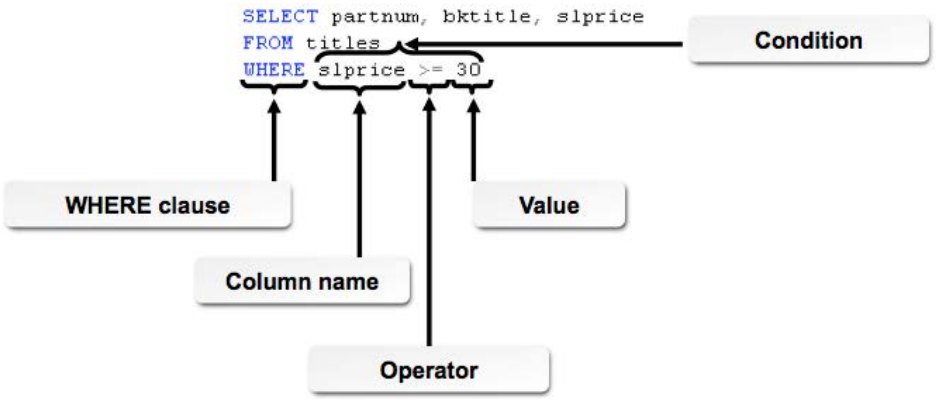


Figure 2-6: A SELECT statement displaying operators used in a WHERE clause.

Operators Used in SQL

There are eight key categories of operators used in Microsoft® SQL Server® 2012.

Operator	*Description*
Arithmetic operators	Perform mathematical operations on two expressions of numeric data.
Assignment operators	Establish the relationship between a column heading and the expression that defines values for the column.
Compound operators	Perform mathematical operations on two numeric expressions by combining one operator with another.
Bitwise operators	Perform bit manipulations between two expressions of the integer data type.
Comparison operators	Test whether two expressions are the same, greater than, or less than.
Logical operators	Test for the truth of a condition. Return a Boolean data type with a value of TRUE or FALSE.
String concatenation operators	Allow string concatenation.
Unary operators	Perform an operation on only one expression of any of the data types of the numeric data type category.

Comparison Operators

Comparison operators are symbols you use in a `WHERE` clause to compare two expressions or values. 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. You cannot use comparison operators with columns that use the text, ntext, or image data types. In SQL Server, you use comparison operators in conditions as part of a `WHERE` clause.

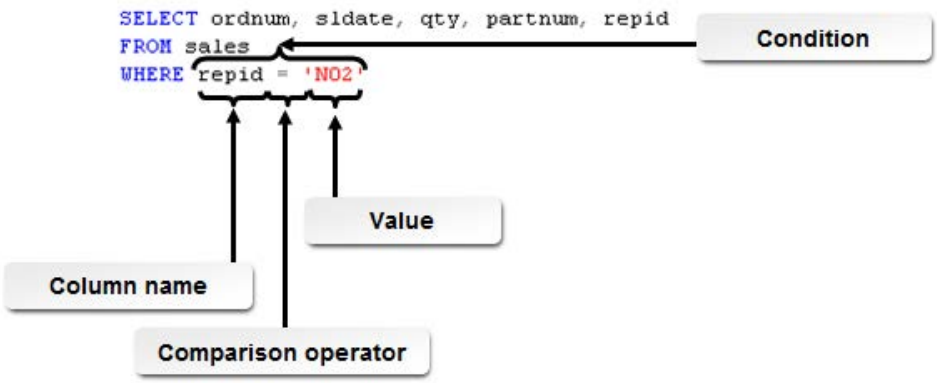


Figure 2-7: A SELECT statement displaying the comparison operator.

Comparison Operators and Their Descriptions

There are nine comparison operators used in SQL Server 2012.

Comparison Operator	*Description*
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to
!=	Not equal to (not SQL-92 standard)
!<	Not less than (not SQL-92 standard)
!>	Not greater than (not SQL-92 standard)

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 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.

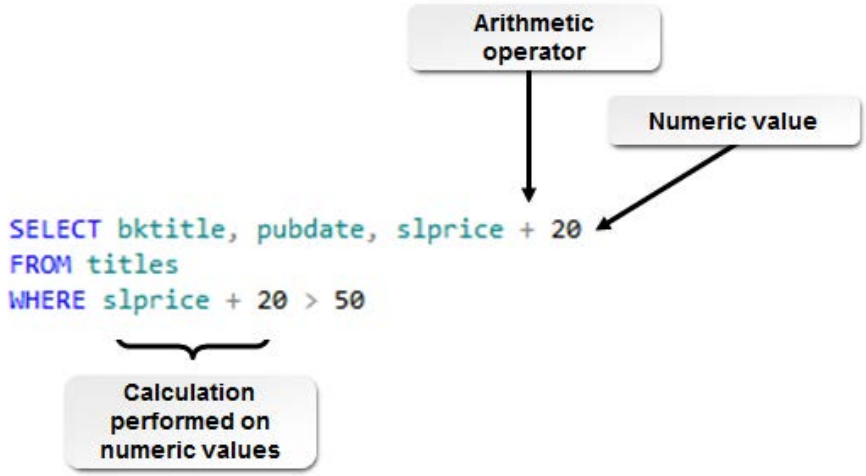


Figure 2-8: A SELECT statement displaying an arithmetic operator.

Arithmetic Operators Used in SQL

There are five arithmetic operators used in SQL.

Arithmetic Operator	*Description*
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Returns the integer remainder of a division

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 displays 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. If you want the alias to contain a space, you must enclose it in single 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.

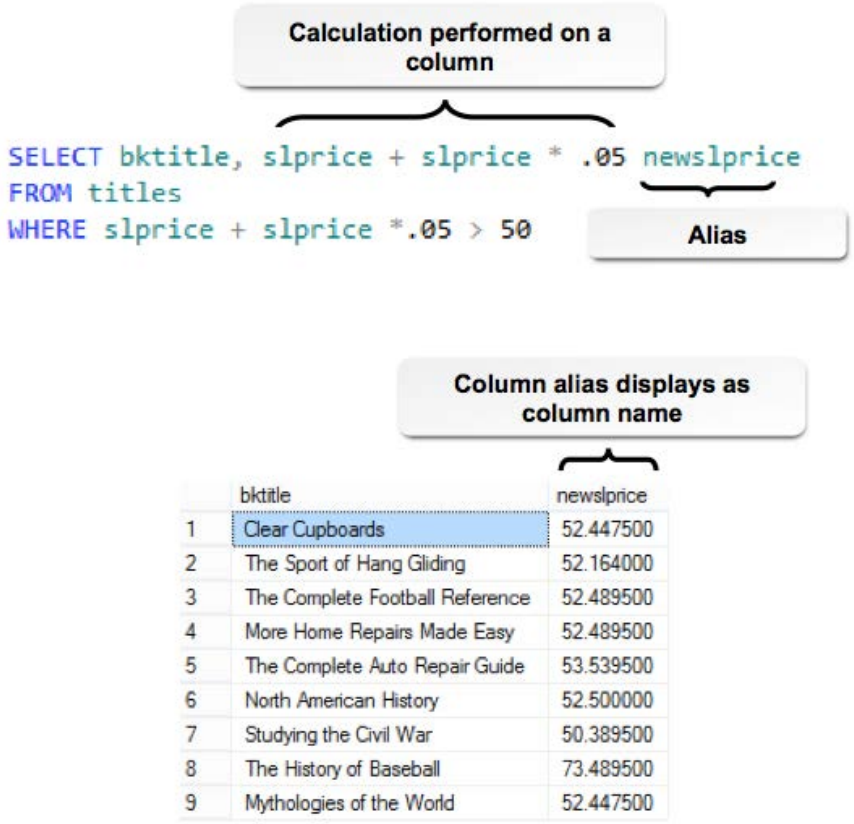


Figure 2-9: A table displaying the column alias.

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. When you use more than one logical operator in a SQL statement, SQL Server executes all operators in an order that is previously determined. You can use parentheses to change the order of evaluation.

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

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.

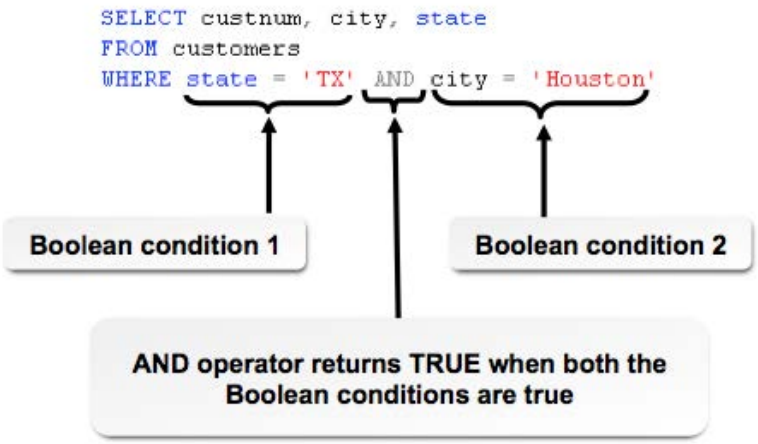


Figure 2-10: Logical operators used in SQL querying.

In the following figure, you see a `WHERE` clause consisting of three search conditions. For this query, SQL Server will return all rows in which either the customer lives in New York and has a customer number greater than 15000, and all customers who live in Massachusetts.

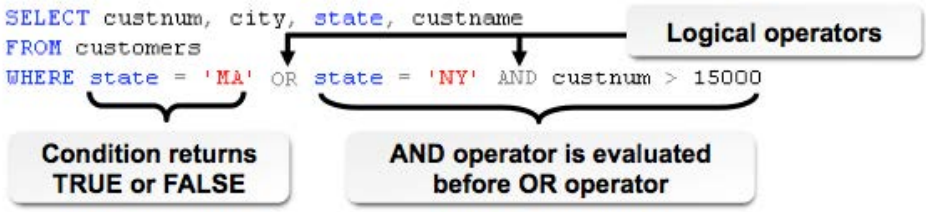


Figure 2-11: Multiple logical operators used in SQL querying.

In the following figure, you see another example in which the `WHERE` clause contains multiple search conditions, but the syntax uses parentheses to indicate the order in which SQL Server should process the search conditions. In this example, SQL Server will return all rows in which customers live in either Massachusetts or New York and have a customer number greater than 15000.

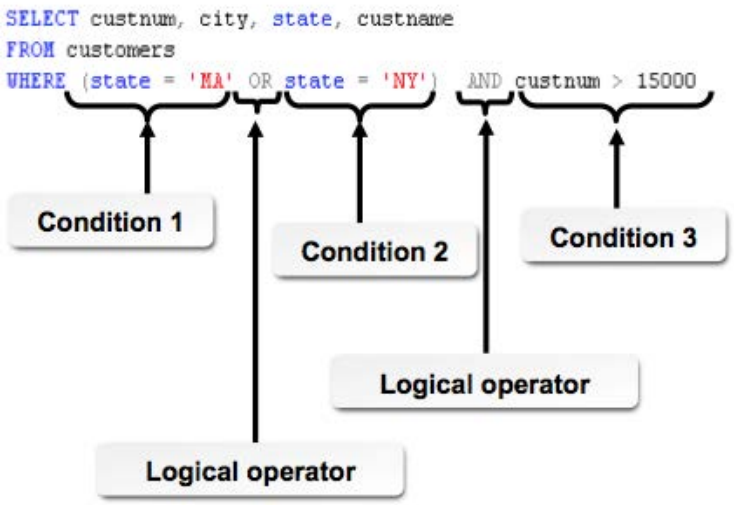


Figure 2-12: Logical operators with parentheses.

A List of Logical Operators Used in SQL

There are 10 logical operators used in SQL Server 2012.

Logical Operator	*Description*
AND	TRUE if both Boolean expressions are TRUE.
OR	TRUE if either Boolean expression is TRUE.
NOT	Reverses the value of any other Boolean operator.
BETWEEN	TRUE if the operand is within a range.
IN	TRUE if the operand is equal to one of a list of expressions.
LIKE	TRUE if the operand matches a pattern.
ALL	TRUE if all of a set of comparisons are TRUE.
ANY	TRUE if any one of a set of comparisons is TRUE.
EXISTS	TRUE if a subquery contains any rows.
SOME	TRUE if some of a set of comparisons are TRUE.

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.

In the following figure, you see 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.

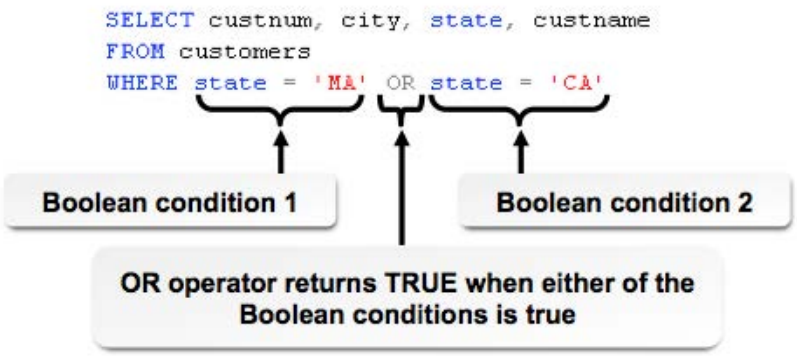


Figure 2-13: Using AND and OR operators in SQL querying.

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.

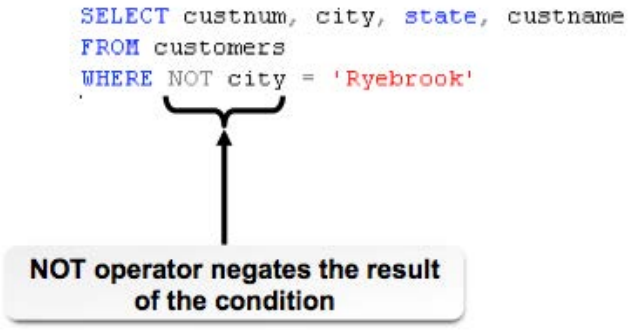


Figure 2-14: Using NOT operators in SQL.

In the following figure, the WHERE clause consists of two conditions separated by the 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.

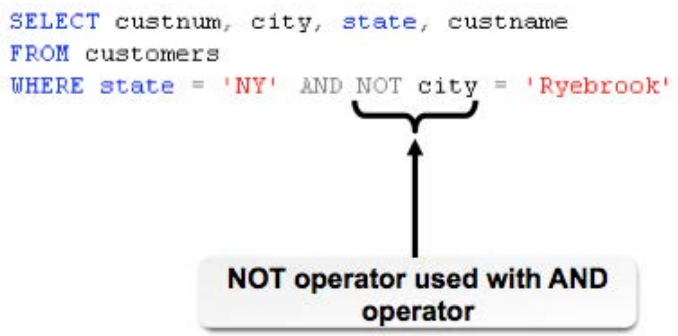


Figure 2-15: Using AND and NOT operators in SQL.

Syntax of Commonly Used Logical Operators

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

Access the Checklist tile on your LogicalCHOICE 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

Before You Begin

- On the **Standard** toolbar, select **New Query** to open a **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** database is selected.

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

---

1. List books for which sale price is greater than or equal to $50.
   a) In the **Editor** pane, type:

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

   b) On the **SQL Editor** toolbar, select **Execute** to execute the query.
   c) In the **Results** pane, observe the book titles for which the sales price is greater than or equal to $50.

2. List the details of the book that has “Sailing” as the book title.
   a) Edit your query to read as follows:

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

   b) On the **SQL Editor** toolbar, select **Execute** to execute the query.
   c) In the **Results** pane, observe that the book with “Sailing” as the title is displayed.
   d) Close the **Query Editor** window without saving the query.

ACTIVITY 2-2
Comparing Column Values

Before You Begin

- On the **Standard** toolbar, select **New Query** to open the **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** database is selected.

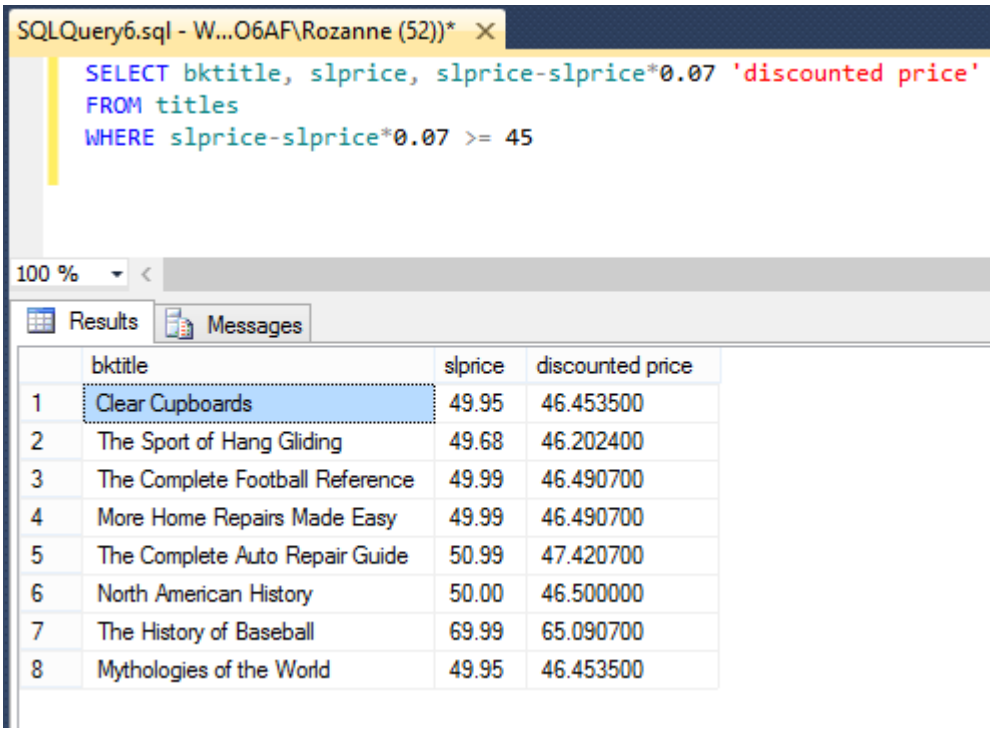
Scenario

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

---

1. Display the titles of books that have a sale price of $45 or above.
   a) In the **Editor** pane, enter this query:
   ```
   SELECT bktitle, slprice
   FROM Titles
   WHERE slprice >= 45
   ```
   b) On the **SQL Editor** toolbar, select **Execute** to execute the query.
   c) In the **Results** pane, observe that a list of books with a sales price greater than or equal to $45 is displayed.

2. Modify the query to display books for which the sale price is $45 and above after the discount. Also, display the sale price of the book after the discount along with the original price of the book.
   a) Edit the query to read as follows:
   ```
   SELECT bktitle, slprice, slprice - slprice*0.07 'discounted price'
   FROM Titles
   WHERE slprice-slprice*0.07 >= 45
   ```
   b) On the **SQL Editor** toolbar, select **Execute** to execute the query.
   c) In the **Results** pane, observe that the list of books for which the sale price is greater than or equal to $45 after the discount, along with their original sale price, is displayed.



	bktitle	slprice	discounted price
1	Clear Cupboards	49.95	46.453500
2	The Sport of Hang Gliding	49.68	46.202400
3	The Complete Football Reference	49.99	46.490700
4	More Home Repairs Made Easy	49.99	46.490700
5	The Complete Auto Repair Guide	50.99	47.420700
6	North American History	50.00	46.500000
7	The History of Baseball	69.99	65.090700
8	Mythologies of the World	49.95	46.453500

d) Close the **Query Editor** window without saving the query.

ACTIVITY 2-3
Searching for Rows Using Multiple Conditions

Before You Begin

- On the **Standard** toolbar, select **New Query** to open a **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** 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. For information on tables and column names in the FullerAckerman database, refer to the table structure in Appendix A.

---

1. List all customers who are from the state of New York.
   a) In the **Editor** pane, enter this query:
   ```
   SELECT city, state, custname
   FROM customers
   WHERE state = 'NY'
   ```
   b) On the **SQL Editor** toolbar, select **Execute** to execute the query.
   c) 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) Change the query to read:
   ```
   SELECT city, state, custname
   FROM customers
   WHERE state = 'NY' OR state = 'MA'
   ```
   b) On the **SQL Editor** toolbar, select **Execute** to 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.

3. **Must you use parentheses when you specify more than two conditions in the `WHERE` clause of a `SELECT` statement?**
   ○ No. You do not need to use parentheses.
   ○ You use parentheses when the operators used in conditions do not follow the operator processing hierarchy.
   ○ Yes. You must always use parentheses.
   ○ You use parentheses to enhance readability.

4. 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) On the **SQL Editor** toolbar, select **Execute** to execute the query.

c) In the **Results** pane, observe that a list of four customers with representative ID S01, who are either from the state of Massachusetts or New York, is displayed.

	city	state	custname	repid
1	Rochester	NY	CK Music!	S01
2	Cambridge	MA	Frank's Music Shop	S01
3	Buffalo	NY	Prince's Pets	S01
4	Ryebrook	NY	Toujours Tours	S01

d) Close the **Query Editor** window without saving the query.

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 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

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 a 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.

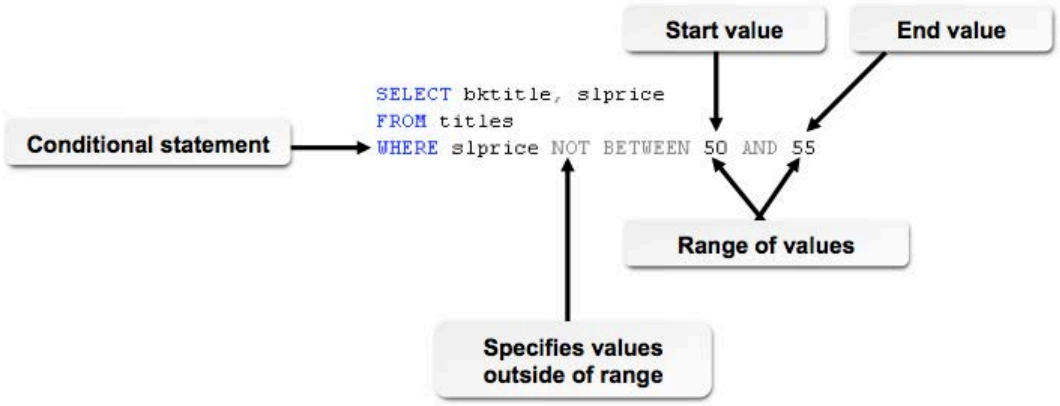


Figure 2-16: Using the BETWEEN.. AND operator in SQL.

Syntax of the BETWEEN...AND Operator

The syntax of the `BETWEEN...AND` operator is:

```
expression1 [ NOT ] BETWEEN expression2 AND expression3
```

Note: 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.

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.

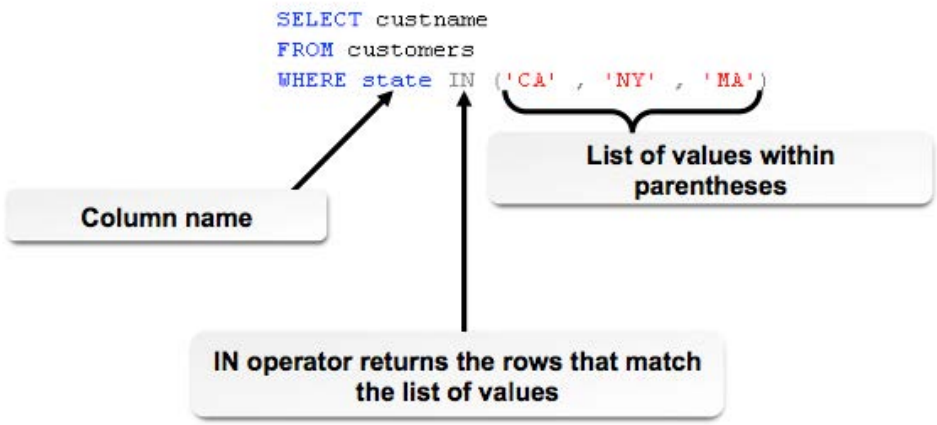


Figure 2-17: Use of the IN operator in SQL.

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.

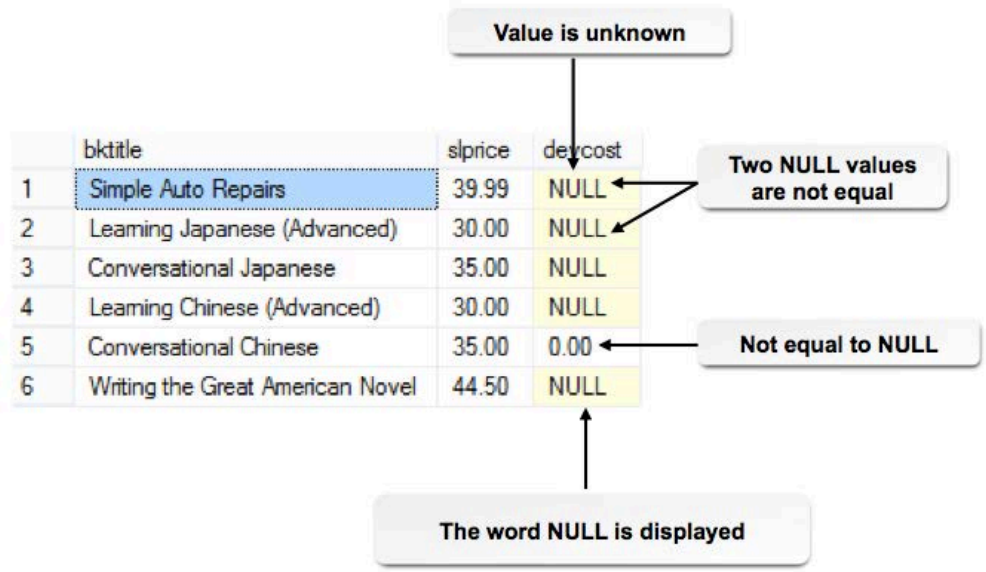


Figure 2-18: Using a NULL value in a table.

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.

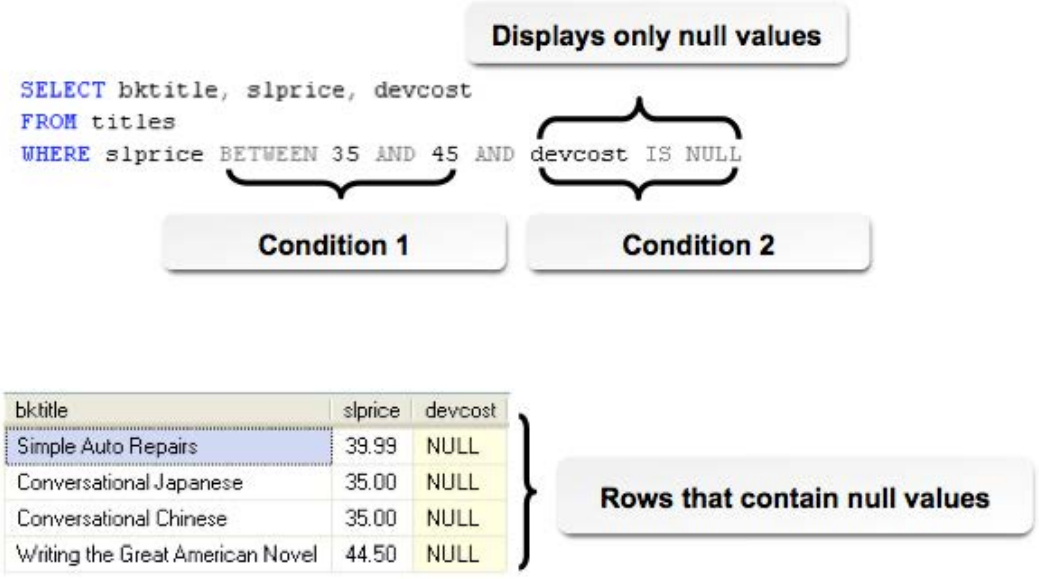


Figure 2-19: Use of the IS NULL clause.

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 sales price between $35 and $45 and have a defined development cost.

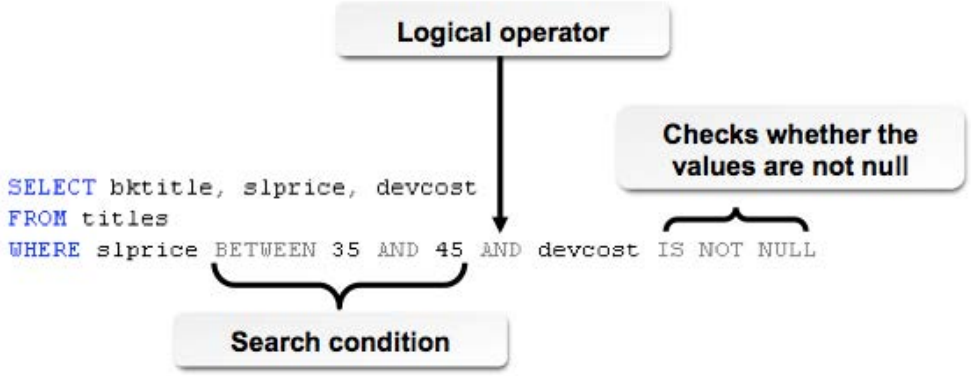


Figure 2-20: Use of the IS NOT NULL clause.

Syntax for the IS NULL Clause

The syntax for the IS NULL clause is:

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

	Access the Checklist tile on your LogicalCHOICE 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

- On the **Standard** toolbar, select **New Query** to open the **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** 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) In the **Editor** pane, enter this query:

```
SELECT bktitle, slprice
FROM Titles
WHERE slprice BETWEEN 35 AND 70
```

   b) On the **SQL Editor** toolbar, select **Execute** to 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.

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

```
SELECT bktitle, slprice, devcost
FROM Titles
WHERE devcost IS NULL
```

   b) On the **SQL Editor** toolbar, select **Execute** to execute the query.
   c) In the **Results** pane, observe that a list of five book titles with a development cost that is not defined are displayed.

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) On the **SQL Editor** toolbar, select **Execute** to execute the query.
   c) In the **Results** pane, observe that a list of 87 titles that have a numerical development cost is displayed.
   d) Close the **Query Editor** window without saving the query.

TOPIC C

Search Data Based on 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.

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.

Wildcard Characters

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".

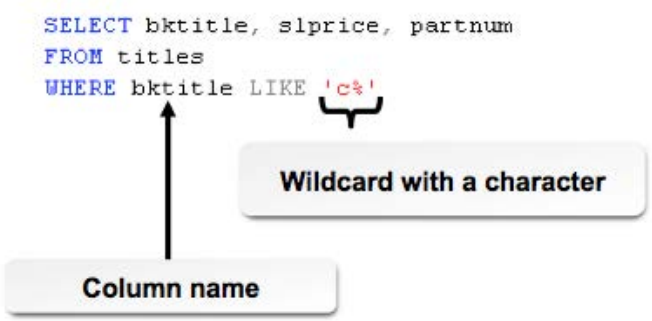


Figure 2-21: Wildcard in SQL querying.

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

Wildcard	*Meaning*
%	Any string of zero or more characters.
_	Any single character.
[ ]	Any single character within the specified range.
[^]	Any single character not within the specified range.

In this next 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."

Figure 2-22: Wildcard used in the Titles 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.

In the following figure, you see a `SELECT` statement that searches for all customers that live in a city in which the second character in the city name is the letter "O". The underscore character (_) is the wildcard for a single character, and the percent character (%) is the wildcard for any number of characters.

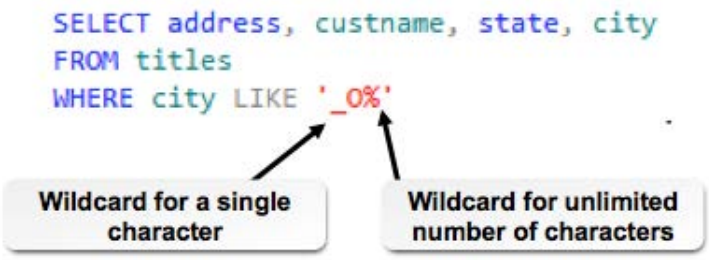


Figure 2-23: Pattern matching in SQL.

Syntax of the LIKE Operator

The syntax of the `LIKE` operator is:

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

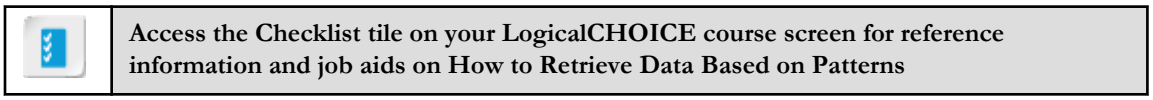
The pattern should be enclosed within single quotes.

Operator Precedence

When you use multiple operators in a complex `WHERE` clause, operator precedence determines the sequence in which SQL Server performs operations. SQL Server evaluates a higher-level operator before it evaluates a lower-level operator. If you don't specify the order of execution for the operators by using parentheses, SQL Server might not generate the correct output.

Operators have precedence levels, which determine the order in which SQL Server executes them. Using parentheses enables you to change the order in which SQL Server executes operators.

Operator Level	*Operator Precedence in Each Level*
1	+ (Positive), - (Negative), ~ (Bitwise NOT)
2	* (Multiply), / (Division), % (Modulo)
3	+ (Add), + (Concatenate), - (Subtract)
4	=, >, <, >=, <=, <>, !=, !>, !< (Comparison operators)
5	^ (Bitwise Exclusive OR), & (Bitwise AND), | (Bitwise OR)
6	NOT
7	AND
8	ALL, ANY, BETWEEN, IN, LIKE, OR, SOME
9	= (Assignment)



Access the Checklist tile on your LogicalCHOICE course screen for reference information and job aids on How to Retrieve Data Based on Patterns

ACTIVITY 2-5
Retrieving Data Based on Patterns

Before You Begin

- On the **Standard** toolbar, select **New Query** to open the **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** 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 characters 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. For information on tables and column names in the FullerAckerman database, refer to the table structure in Appendix A.

---

1. Display the books that have the characters “art” in the title.
   a) Enter this query:
   ```
   SELECT bktitle, partnum, slprice
   FROM Titles
   WHERE bktitle LIKE '%art%'
   ```
   b) On the **SQL Editor** toolbar, select **Execute** to execute the query.
   c) In the **Results** pane, observe that SQL Server displays a list of five books with the characters “art” in their titles. Notice that SQL Server returns the books "Starting a Small Garden" and "Starting a Greenhouse" in addition to the books that have the word "art" in them.

2. Display the details of books for which the titles begin with A, M, or C.
   a) Edit the query to read as follows:
   ```
   SELECT bktitle, partnum, slprice
   FROM Titles
   WHERE bktitle LIKE '[AMC]%'
   ```
   b) On the **SQL Editor** toolbar, select **Execute** to 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 query to use the following `WHERE` clause:
   ```
   SELECT bktitle, partnum, slprice
   FROM Titles
   WHERE bktitle LIKE '[A-G]%'
   ```
   b) Select **Execute** to execute the query.
   c) In the **Results** pane, observe that 21 books have titles beginning with the characters from A to G.
   d) In the **Results** pane, scroll down to view all rows.
   e) Close the **Query Editor** window without saving the query.

4. List the customer details from the Customers table for those who have a four-digit customer number.
   a) On the **Standard** toolbar, select **New Query** to open the **Query Editor** window.
   b) From the **Available Databases** drop-down list, select **FullerAckerman**.
   c) Enter this query:

```
SELECT custnum, custname, city
FROM Customers
WHERE custnum LIKE '____'
```

   d) Execute the query.
   e) 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) Edit the query to read as follows:

```
SELECT custnum, custname, city
FROM Customers
WHERE custnum LIKE '___[19]'
```

   Note that the search condition consists of three underscore characters and [19] to search for 1 or 9 at the end of the customer number.
   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.
   d) Close the **Query Editor** window without saving the query.

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?

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

Note: Check your LogicalCHOICE Course screen for opportunities to interact with your classmates, peers, and the larger LogicalCHOICE 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.

3 Working with Functions

Lesson Time: 1 hour

Lesson Objectives

In this lesson, you will use various functions to perform calculations on data. You will:

- Perform date calculations.
- Calculate data using aggregate functions.
- Manipulate string values in a query.

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.

Suppose you have a database that contains employee information and you need to identify employees who have worked for the company for over five years now, along with their average salary. This information isn't readily available in the table. However, you can obtain this information by using functions to perform calculations on the data that is available in the table.

TOPIC A

Perform Date Calculations

The data present in a table can be of any number of data types. For instance, Microsoft® SQL Server® 2012 stores date information in a table using the data types datetime and smalldatetime. But when SQL Server retrieves this information, you might want to view it in a specific date and time format other than its default format. In this topic, you will use date functions to display information and perform calculations on date information.

Suppose you need to calculate the length of time employees have worked for a company, but this information is not stored in the table. Instead, the table has just the date each employee was hired. You can determine how long an employee has worked for the company by using date functions to perform the necessary calculations before displaying the output.

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 a column name or a value. 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.

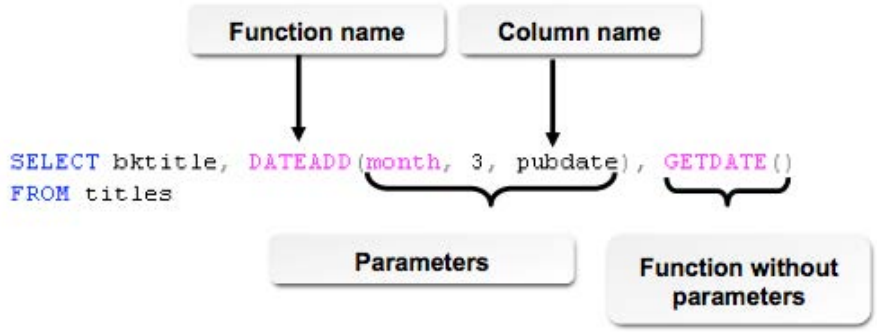


Figure 3-1: A SELECT statement displaying functions.

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 a 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 expressions. The following table describes the categories of functions SQL Server includes.

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

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, '12-31-2013')` function displays the difference in years between the book's publication date (pubdate) and the date 12-31-2013.

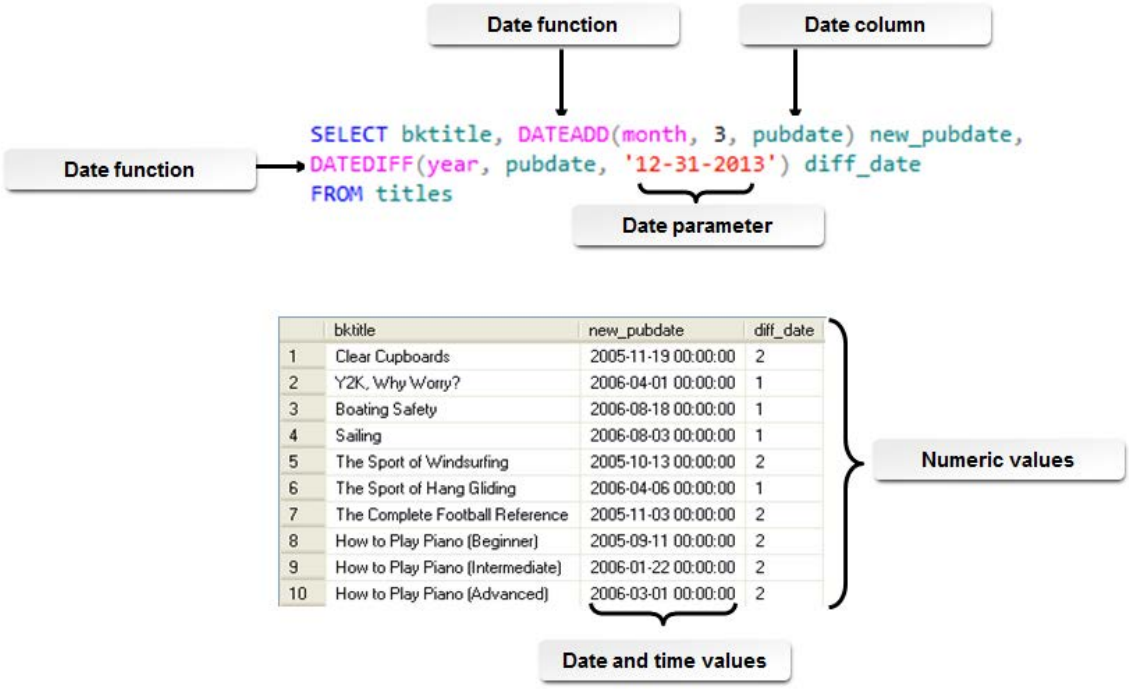


Figure 3-2: A SELECT statement displaying date functions.

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.

Function	*Description*
`DATEADD(datepart, number, date)`	Returns a new datetime value based on adding an interval to the specified date.
`DATEDIFF(datepart, startdate, enddate)`	Returns the number of date and time boundaries crossed between two specified dates.
`DATENAME(datepart, date)`	Returns a character string representing the datepart of the specified date. The datepart can be month, date, or year.
`DATEPART(datepart, date)`	Returns an integer representing the specified datepart of the specified date.
`DAY(date)`	Returns an integer representing the day datepart of the specified date.
`GETDATE( )`	Returns the current system date and time of the SQL Server in the format specified for datetime values.
`GETUTCDATE( )`	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 the SQL Server is running.
`MONTH(date)`	Returns an integer that represents the month part of a specified date.
`YEAR(date)`	Returns an integer that represents the year part of a specified date.

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.

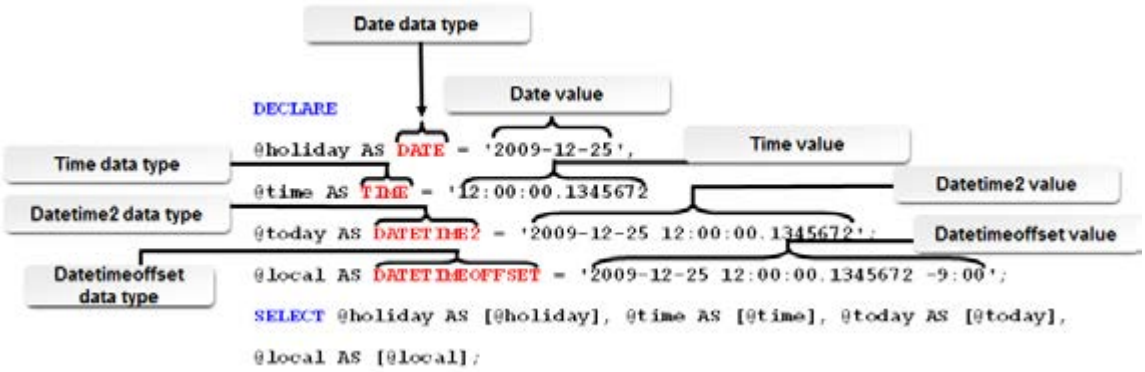


Figure 3-3: Data types representing date and time.

Data Type	*Description*
`DATE`	This data type enables you to store dates in the format 'YYYY-MM-DD.' Columns that use the DATE data type use three storage bytes. Supported values are 0001-01-01 through 9999-12-31. The accuracy of this data type is exactly one day.
`TIME`	The TIME data type enables you to store values in the 'hh:mm:ss.nnnnnnn' format. TIME supports the data range of 00:00:00.0000000 through 23:59:59.9999999. The accuracy of this data type is 100 nanoseconds.
`DATETIME`	The DATETIME data type enables you to store dates in the format 'YYYY-MM-DD hh:mm:ss.nnnnnnn'. DATETIME uses eight bytes to store its information. Supported year values are from 1753 through 9999. The DATETIME data type has the accuracy of one thousandth of a second.
`DATETIME2`	The DATETIME2 data type also represents the date value 'YYYY-MM-DD hh:mm:ss.nnnnnnn'. 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.nnnnnnn [+| -] 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.

The DATEPART() Function

The *DATEPART()* 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()`.

Datepart	*Abbreviations*
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
weekday	dw
hour	hh
minute	mi, n
second	ss, s
millisecond	ms

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.

Access the Checklist tile on your LogicalCHOICE course screen for reference information and job aids on How to Perform Date Calculations

ACTIVITY 3-1
Performing Date Calculations

Before You Begin

- On the **Standard** toolbar, select **New Query** to open a **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** database is selected.

Scenario

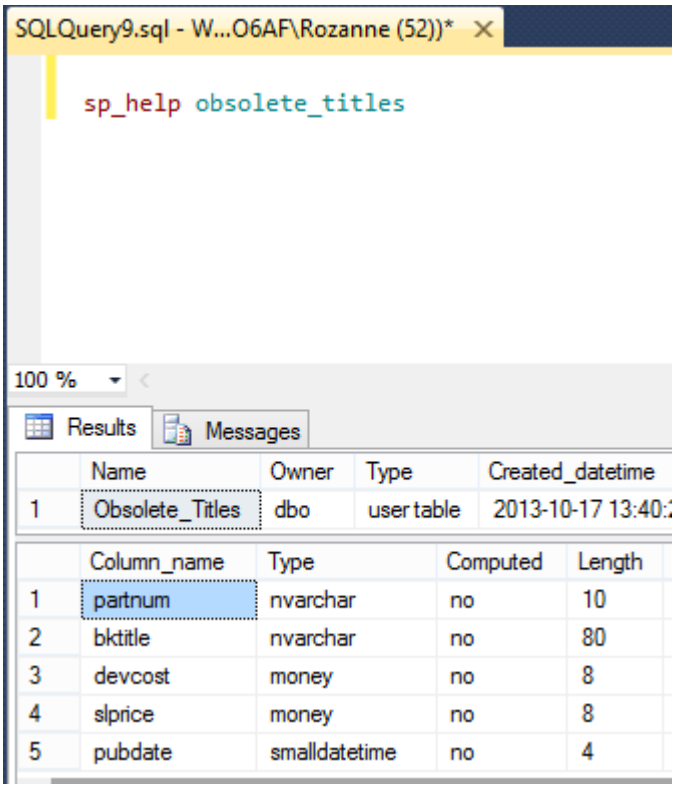
Fuller & Ackerman Publishing is celebrating its 25th year in the marketplace. Management has decided to release a "golden oldies" collection of books published in the 1990s. The details of the old books are available in the "Obsolete_titles" table. The information required is a list of books that were released in the 1990s, the exact year they were released, and the age of each book. For information on tables and column names in the FullerAckerman database, refer to the table structure in Appendix A.

---

1. 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.



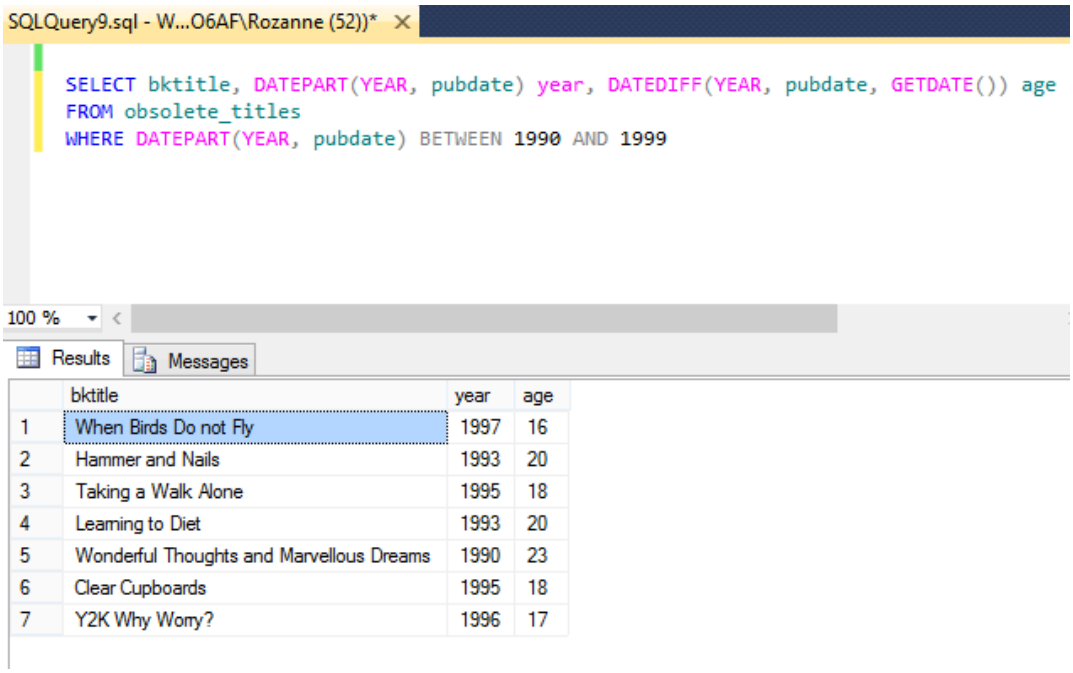
   c) In the **Results** pane, observe that the pubdate column has `smalldatetime` as its data type.
   d) In the **Editor** pane, delete the query `SP_HELP obsolete_titles`.

2. **True or False? You can use the `DATEPART` function to extract the year from the published date of the book.**
   - ☐ True
   - ☐ False

3. 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:

```
SELECT bktitle, DATEPART(YEAR, pubdate) year, DATEDIFF(YEAR, pubdate,
GETDATE()) age
FROM obsolete_titles
WHERE DATEPART(YEAR, pubdate) BETWEEN 1990 AND 1999
```

   b) Execute the query.
   c) In the **Results** pane, observe that books published in the 1990s are displayed.



	bktitle	year	age
1	When Birds Do not Fly	1997	16
2	Hammer and Nails	1993	20
3	Taking a Walk Alone	1995	18
4	Learning to Diet	1993	20
5	Wonderful Thoughts and Marvellous Dreams	1990	23
6	Clear Cupboards	1995	18
7	Y2K Why Worry?	1996	17

   d) Close the **Query Editor** window without saving the query.

TOPIC B

Calculate Data Using Aggregate Functions

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

Databases can contain a large collection of unprocessed information. Sometimes, just extracting the information might not be adequate for your needs. What might be more useful is to extract the data from the database, perform a calculation on it, and return a result of that calculation. For example, if you have an employee table and need to know for a survey the average age of employees working in the organization, instead of manually going through the information in the table and using a calculator, you can use SQL functions to perform the calculations for you.

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.

In the following figure, you see a `SELECT` statement that uses the `COUNT()` and `AVG()` aggregate functions. The `COUNT()` function enables SQL Server to count the values in a column. The `AVG()` function calculates the average of all the values in the column.

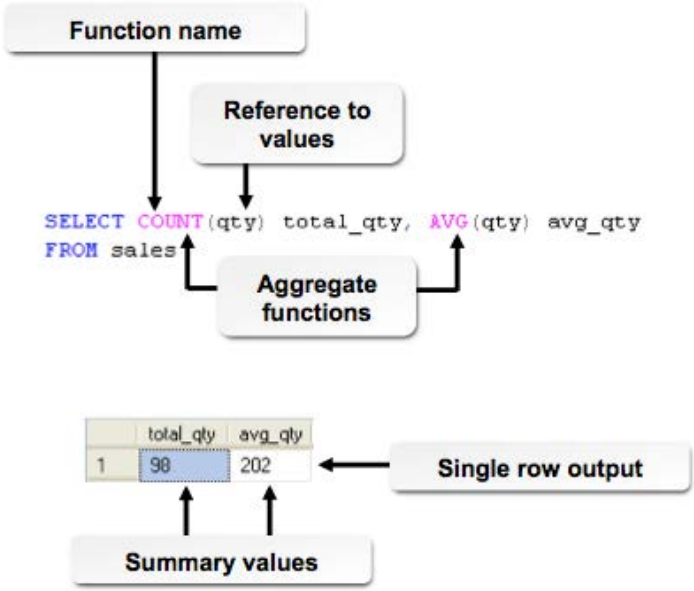


Figure 3-4: A SELECT statement using aggregate functions.

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

Aggregate Function	Description
`AVG(expression)`	Returns the average of values in a column. The column can contain only numeric data.
`COUNT(expression), 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 *). `COUNT(expression)` ignores NULL values, but `COUNT(*)` includes them in the count.
`MAX(expression)`	Returns the highest value in a column (last value alphabetically for text data types). Ignores NULL values.
`MIN(expression)`	Returns the lowest value in a column (first value alphabetically for text data types). Ignores NULL values.
`SUM(expression)`	Returns the total of values in a column. The column can contain only numeric data.

Keywords

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. Because a keyword has a predefined meaning in SQL, if used outside the predetermined context, you must enclose it within double quotes.

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

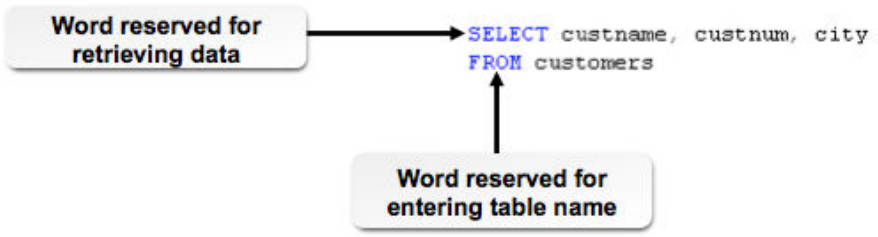


Figure 3-5: A SELECT statement displaying keywords.

It's possible for you to use SQL keywords outside of their specified context. In this next figure, the `SELECT` statement is using the SQL keyword "from" as an alias for the custnum column in the output. Because this query uses the "from" keyword outside its predefined context (the `FROM` clause), you must enclose it in double quotes.

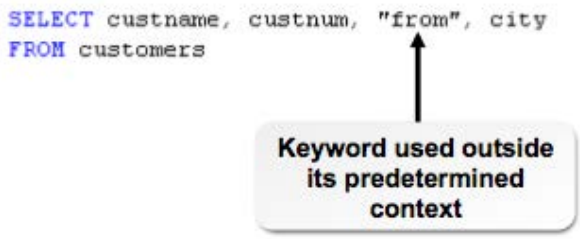


Figure 3-6: A SELECT statement displaying keywords outside the context.

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 that you can use in a `SELECT` statement to retrieve unique rows from a table. 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.

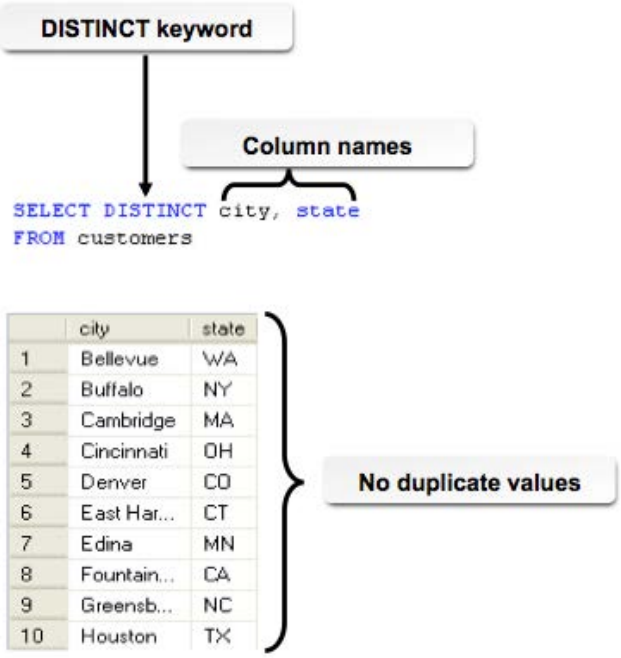


Figure 3-7: A SELECT statement displaying the DISTINCT keyword.

Access the Checklist tile on your LogicalCHOICE 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

- On the **Standard** toolbar, select **New Query** to open a new **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** 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 2013 for this analysis.

In order to perform their analysis, management would like you to retrieve the following information:

- The titles released in 2013.
- A count of the titles released, the total development cost for all books, and the average development cost for a title.
- A list of books for which the development cost is not defined.

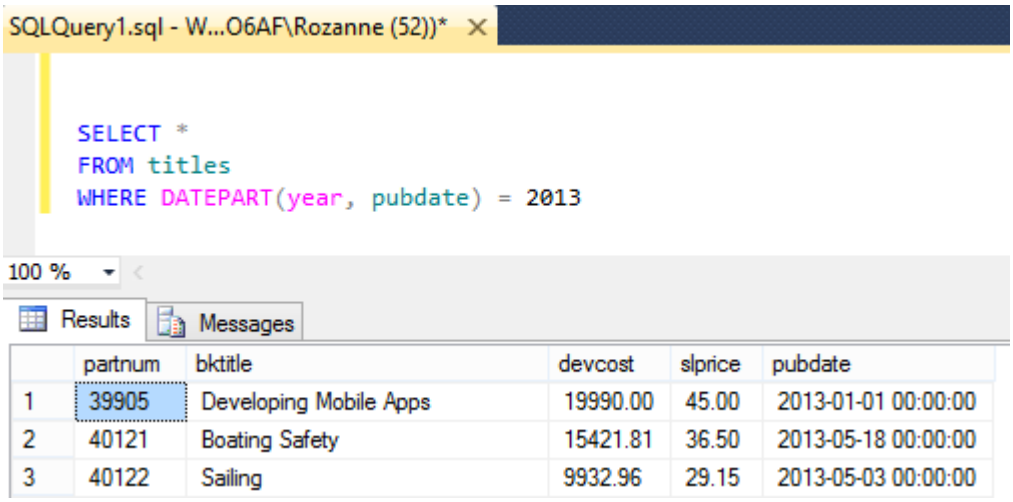
For information on tables and column names in the FullerAckerman database, refer to the table structure in Appendix A.

---

1. List all titles that were developed in 2013.
   a) In the **Query Editor** window, enter the following query:

```
SELECT *
FROM Titles
WHERE DATEPART(year, pubdate) = 2013
```

   b) Execute the query.
   c) In the **Results** pane, observe that SQL Server displays all books that were developed in 2013.

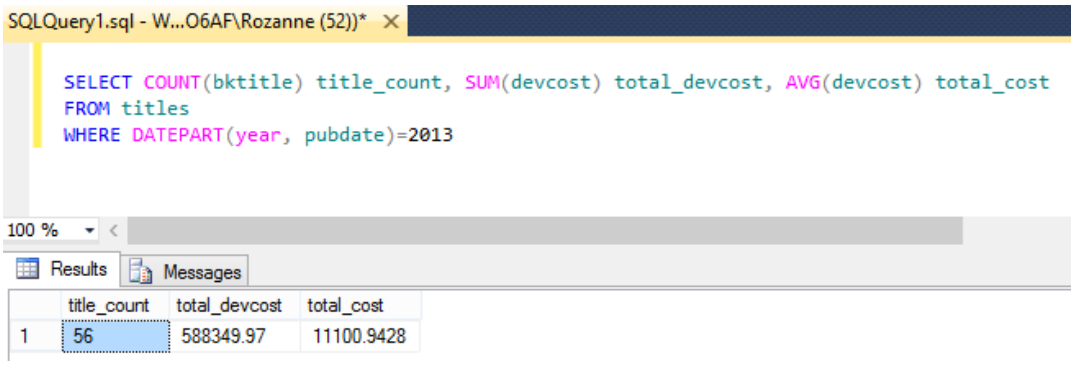


	partnum	bktitle	devcost	slprice	pubdate
1	39905	Developing Mobile Apps	19990.00	45.00	2013-01-01 00:00:00
2	40121	Boating Safety	15421.81	36.50	2013-05-18 00:00:00
3	40122	Sailing	9932.96	29.15	2013-05-03 00:00:00

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 2013.
   a) Edit the existing query to read:

```
SELECT COUNT(bktitle) title_count, SUM(devcost) total_devcost,
AVG(devcost) average_cost
FROM titles
WHERE DATEPART(year, pubdate)=2013
```

   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.



3. List titles that were published in 2013 that have NULL values for the development cost.
   a) Edit the existing query to read:

```
SELECT bktitle, pubdate, devcost
FROM titles
WHERE DATEPART(year, pubdate)=2013 AND devcost IS NULL
```

   b) Execute the query.
   c) In the **Results** pane, observe that three titles are displayed.
   d) Close the **Query Editor** window without saving the query.

TOPIC C

Manipulate String Values

At this point, you have used date and aggregate functions. SQL includes functions that you can use with other data types. These functions allow you to manipulate column values and extract information that is required. In this topic, you will use string functions to effectively extract information from alphanumeric columns.

You display your output and find that values in two of the columns contain a lot of blank spaces. You're very sure you didn't put them there. How can you get rid of them? This is one situation that calls for the use of string values and functions. With string values, you can use functions to display the characters in the table based on your preferences.

Strings

A *string* is a collection of characters that you cannot use in an arithmetic calculation. SQL Server uses the char, varchar, and text data types to store strings. 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.

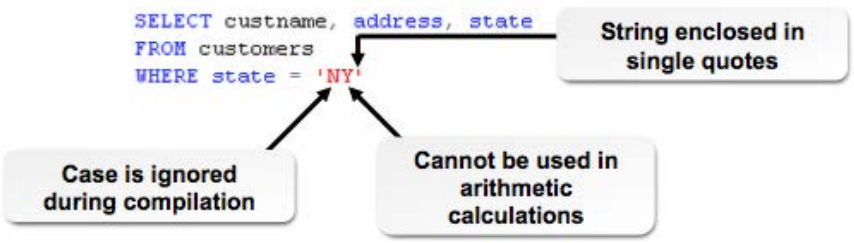


Figure 3-8: A SELECT statement in which the WHERE clause uses a string.

String Functions

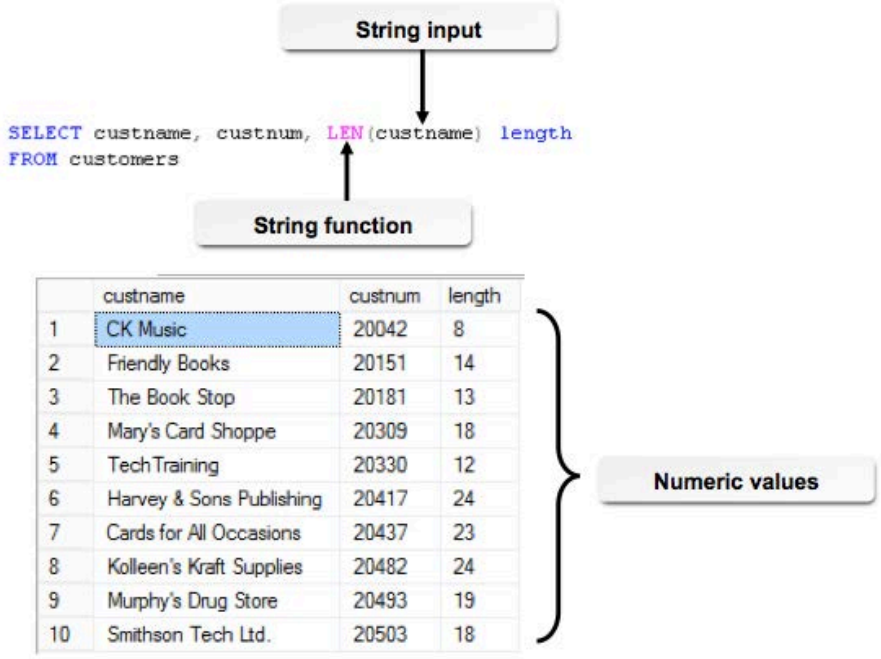
String functions are functions that perform an operation on a string input value and return a string or a numeric value. The following table describes key string functions in SQL Server.

String Function	*Description*
`ASCII (character_expression)`	Returns the ASCII code value of the leftmost character of a character expression.
`CHAR (integer_expression)`	Converts an integer ASCII code to a character.
`LEFT (character_expression, integer_expression)`	Returns the part of a character string starting at a specified number of characters from the left.
`RIGHT (character_expression, integer_expression)`	Returns the part of a character string starting at a specified

String Function	Description
	number of characters from the right.
`LEN (string_expression)`	Returns the number of characters, rather than the number of bytes, of the given string expression, excluding any trailing blanks.
`STR (float_expression [, length [, decimal ] ] )`	Returns character data converted from numeric data.
`LOWER (character_expression)`	Returns a character expression after converting uppercase character data to lowercase.
`UPPER (character_expression)`	Returns a character expression with lowercase character data converted to uppercase.
`LTRIM (character_expression)`	Returns a character expression after removing leading blanks.
`RTRIM (character_expression)`	Returns a character string after truncating all trailing blanks.
`REPLACE ('string_expression1', 'string_expression2', 'string_expression3')`	Replaces all occurrences of the second given string expression in the first string expression with a third expression.
`REVERSE (character_expression)`	Returns the reverse of a character expression.
`REPLICATE (character_expression, integer_expression)`	Repeats a character expression for a specified number of times.
`SPACE (integer_expression)`	Returns a string of repeated spaces.
`STUFF (character_expression, start, length, character_expression)`	Deletes a specified length of characters and inserts another set of characters at a specified starting point.
`SUBSTRING (expression, start, length)`	Returns part of a character, binary, text, or image expression.
`UNICODE ('ncharacter_expression')`	Returns the integer value, as defined by the Unicode standard, for the first character of the input expression.
`NCHAR (integer_expression)`	Returns the Unicode character associated with the given integer code, as defined by the Unicode standard.
`SOUNDEX (character_expression)`	Returns a four-character (SOUNDEX) code to evaluate the similarity of two strings.

String Function	Description
`DIFFERENCE (character_expression, character_expression)`	Returns the difference between the SOUNDEX values of two character expressions as an integer.
`QUOTENAME ('character_string' [, 'quote_character' ])`	Returns a Unicode string with the delimiters added to make the input string a valid Microsoft SQL Server delimited identifier.
`PATINDEX ('%pattern%', expression)`	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.
`CHARINDEX (expression1, expression2 [, start_location ])`	Returns the starting position of the specified expression in a character string.

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.



	custname	custnum	length
1	CK Music	20042	8
2	Friendly Books	20151	14
3	The Book Stop	20181	13
4	Mary's Card Shoppe	20309	18
5	TechTraining	20330	12
6	Harvey & Sons Publishing	20417	24
7	Cards for All Occasions	20437	23
8	Kolleen's Kraft Supplies	20482	24
9	Murphy's Drug Store	20493	19
10	Smithson Tech Ltd.	20503	18

Figure 3-9: A SELECT statement displaying a string function.

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.

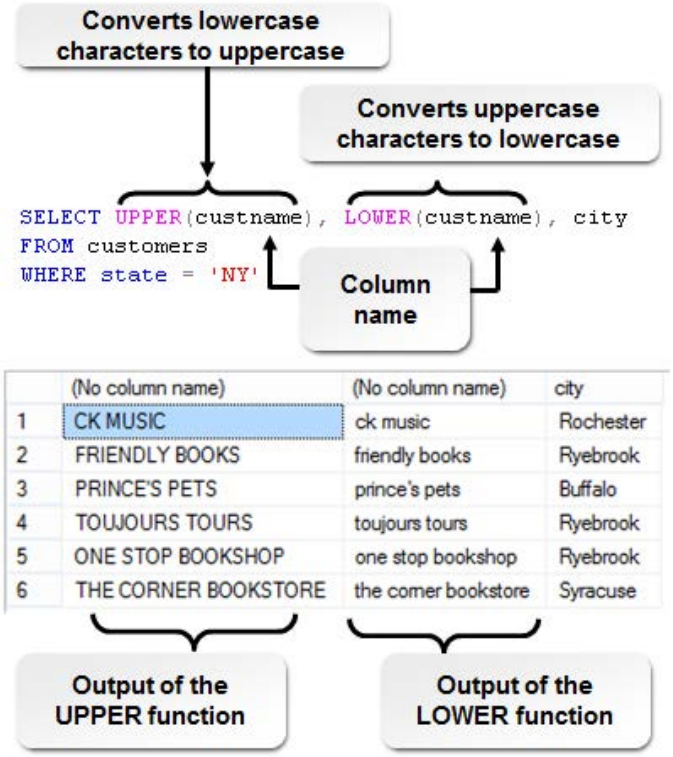


Figure 3-10: A SELECT statement consisting of case conversion functions.

Note: In the **Editor** pane, the case of characters can be converted by selecting the words to be converted and pressing the key combinations **Ctrl+Shift+U** to convert the text to uppercase and **Ctrl+Shift+L** to convert the text to lowercase.

Column Leading and Trailing Spaces

Leading and trailing spaces are spaces that are present in a column when the data stored in a column is less than the maximum number of characters that the column can contain. Spaces inserted before the value are called leading spaces, while those inserted at the end of the value are called trailing spaces.

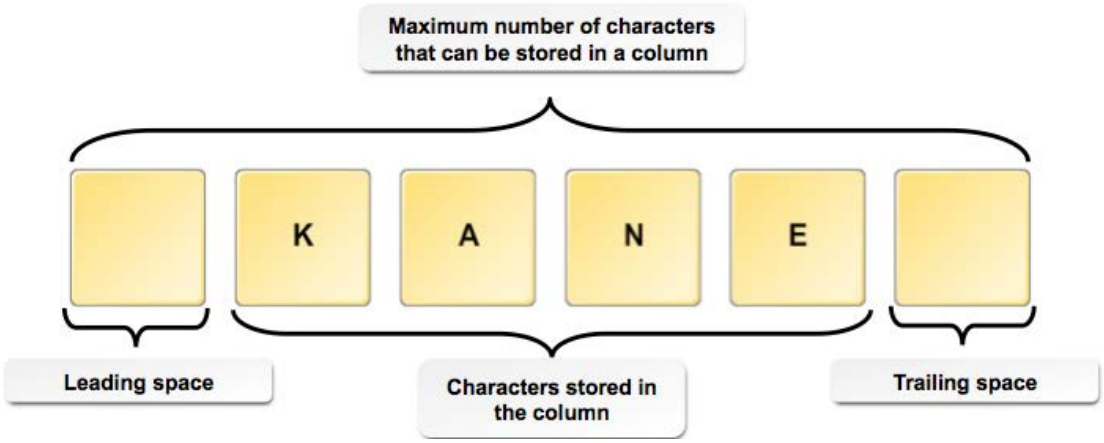


Figure 3-11: Leading and trailing spaces.

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 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.

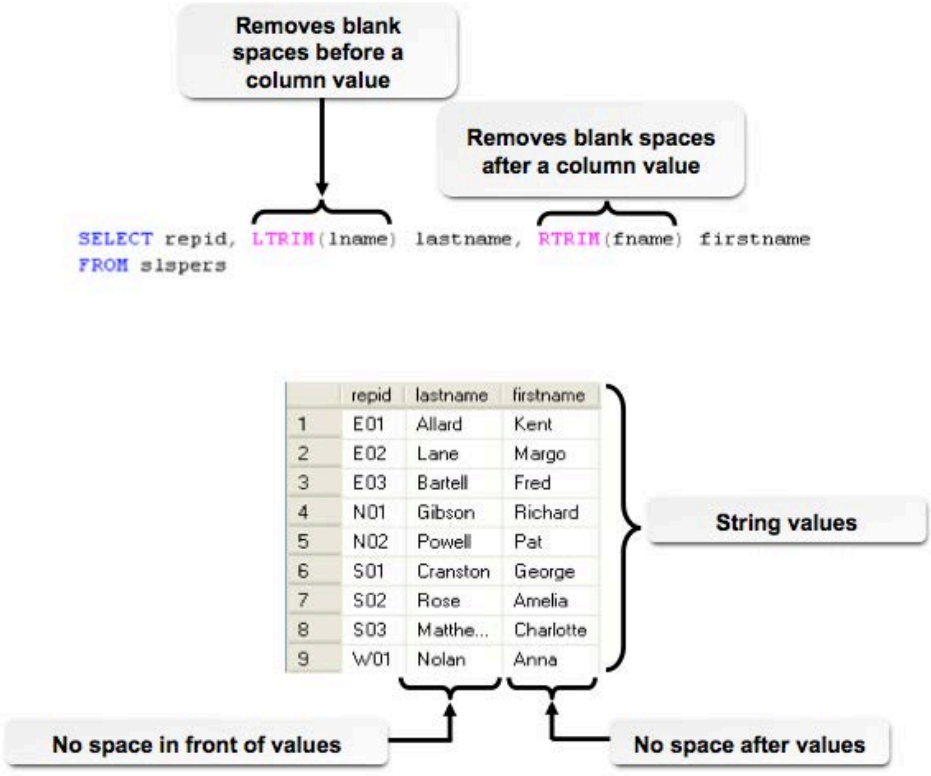


Figure 3-12: A SELECT statement displaying trim functions.

Character Extraction

Character extraction is the process of extracting specific characters from a string value. The extracted string is called a substring. You can extract characters from the beginning, end, or anywhere in the string.

The SUBSTRING Function

SUBSTRING is the function you use to extract characters from 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".

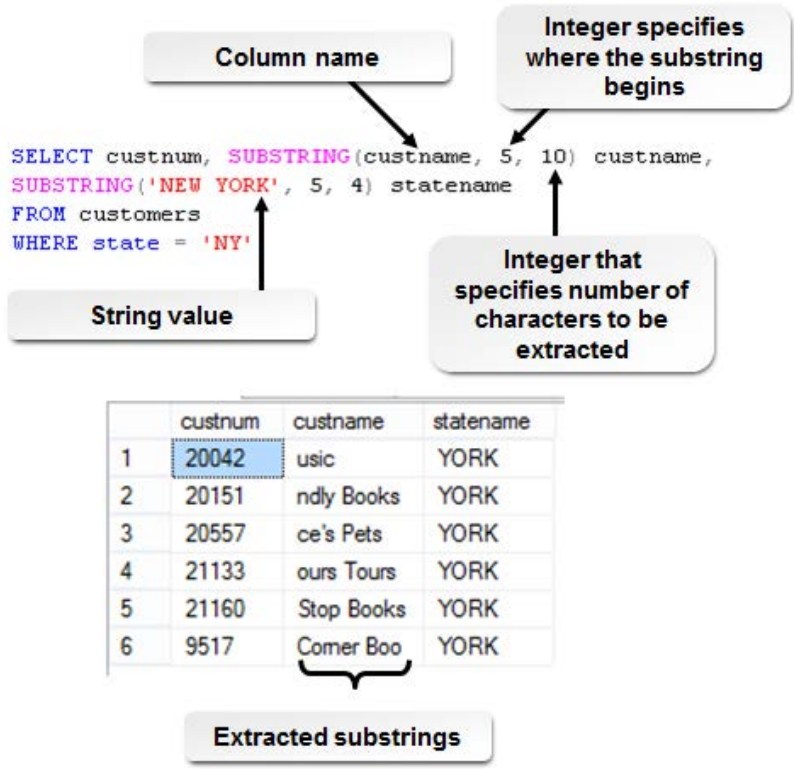


Figure 3-13: A SELECT statement displaying the SUBSTRING function.

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. If there are leading or trailing spaces in either of the expressions, then the other expression which does not contain leading or trailing spaces is appended following the spaces. 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 lastname column has trailing spaces, which result in a gap between the last name and the first name.

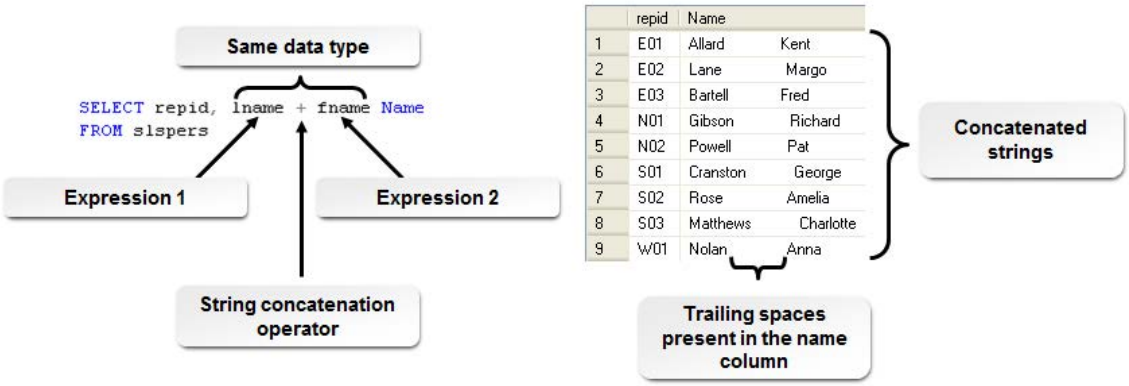


Figure 3-14: A SELECT statement displaying the concatenation operation.

You can resolve the problem of the extra spaces between the last and first names by using the `RTRIM()` function on the lname column, as follows:

```
SELECT repid, RTRIM(lname)+fname Name
FROM slspers
```

You might also want to use string concatenation to insert specific characters between two expressions. For example, if you want to concatenate the values in the last name and first name columns for a table but separate them by a comma and a space, you would use this syntax:

```
SELECT RTRIM(lname)+', '+fname Name
FROM slspers
```

SQL Server simply handles the characters within the single quotes as another expression to concatenate to the lname and fname columns. The following figure shows this query and its result set.

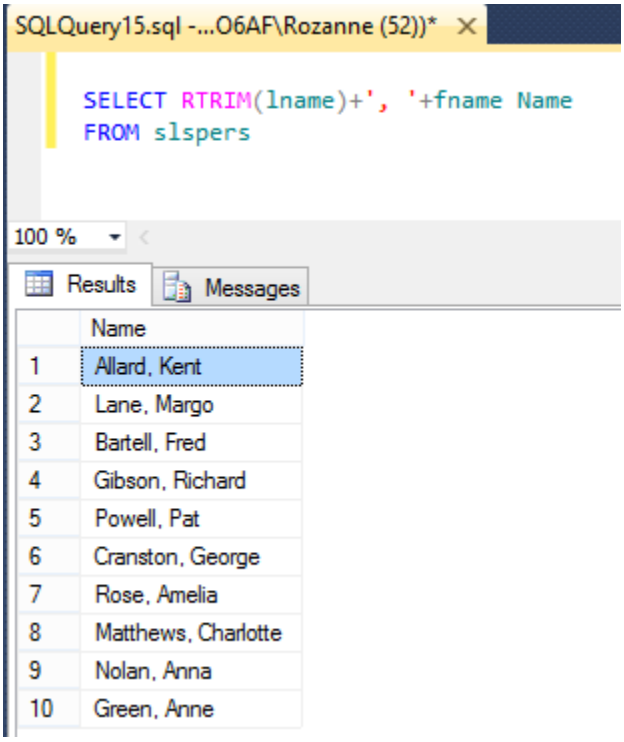


Figure 3-15: Concatenating column values and static values.

	Access the Checklist tile on your LogicalCHOICE course screen for reference information and job aids on How to Manipulate String Values

ACTIVITY 3-3
Manipulating String Values

Before You Begin

- On the **Standard** toolbar, select **New Query** to open a new **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** database is selected.

Scenario

Management has asked you to provide the following information:

- After reviewing the performance of sales representatives, management decided to arrange a weekend trip to recognize their contribution. For booking tickets and fixing accommodations, management has asked you to provide a list of the names of representatives along with their representative IDs.
- To show their appreciation for customers and to encourage them to purchase more titles, management decides to send a "thank you" note to the customers along with some gifts. You must prepare a list of customers' names and addresses.

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

---

1. List the repid, first name, and last name of each representative from the "Slspers" table. Display each sales representative's name as their first and last names and separated by a space.
   a) In the **Editor** pane, enter this query to display the representative ID along with the representative name:
   ```
   SELECT repid, RTRIM(fname)+' '+lname representative_name
   FROM slspers
   ```
   b) Execute the query.
   c) In the **Results** pane, observe that 10 rows are displayed.

2. 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 the following query:
   ```
   SELECT custname, RTRIM(address)+','+RTRIM(city)+
   ','+RTRIM(state)+','+RTRIM(zipcode) customer_address
   FROM customers
   ```
   c) Execute the query.
   d) In the **Results** pane, observe that the list of the names of customers and their addresses is displayed. Notice that the address information is displayed in a single column instead of each component in its own column.
   e) Close the **Query Editor** window without saving the query.

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?

Which function would you use most often? Why?

Note: Check your LogicalCHOICE Course screen for opportunities to interact with your classmates, peers, and the larger LogicalCHOICE 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.

4 Organizing Data

Lesson Time: 1 hour, 30 minutes

Lesson Objectives

In this lesson, you will organize the data obtained from a query before it is displayed on-screen. 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.

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.

In a table that has thousands of rows, only a few hundred rows might satisfy a given query. If you need to identify the highest or lowest value from a list, you need to browse through the list manually. But if this list is sorted in either ascending or descending order, you can easily identify the values you're looking for.

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.

In a table that contains a large collection of information, finding the required information involves using a search condition. For example, in a banking table, if you list the transactions for one day, a few thousand rows might be listed. From this list, if you need to identify the transactions for a single customer, sorting the list by customer name would enable you to view a list of all transactions by each customer in order.

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.

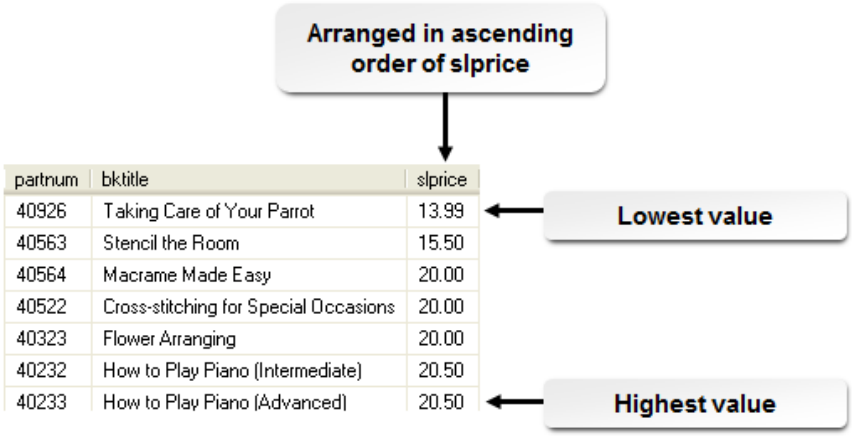


Figure 4-1: A table displaying the sort in ascending order by sale price.

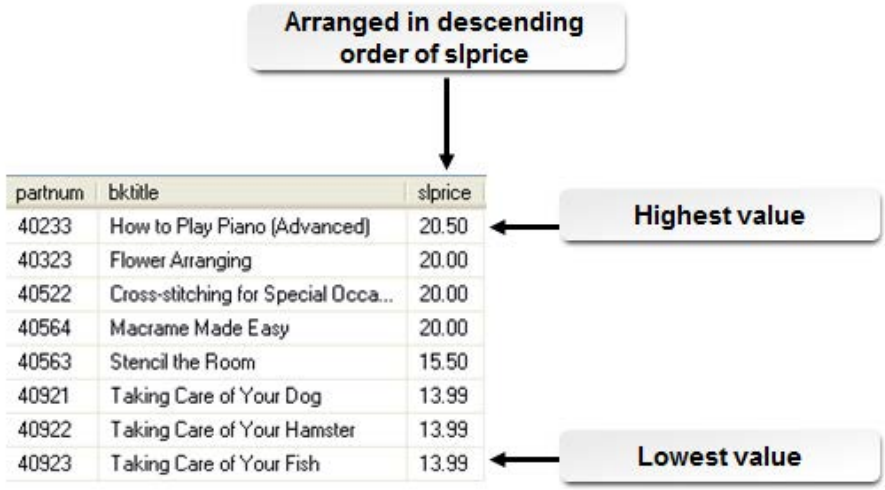


Figure 4-2: A table displaying the sort in descending order by sale price.

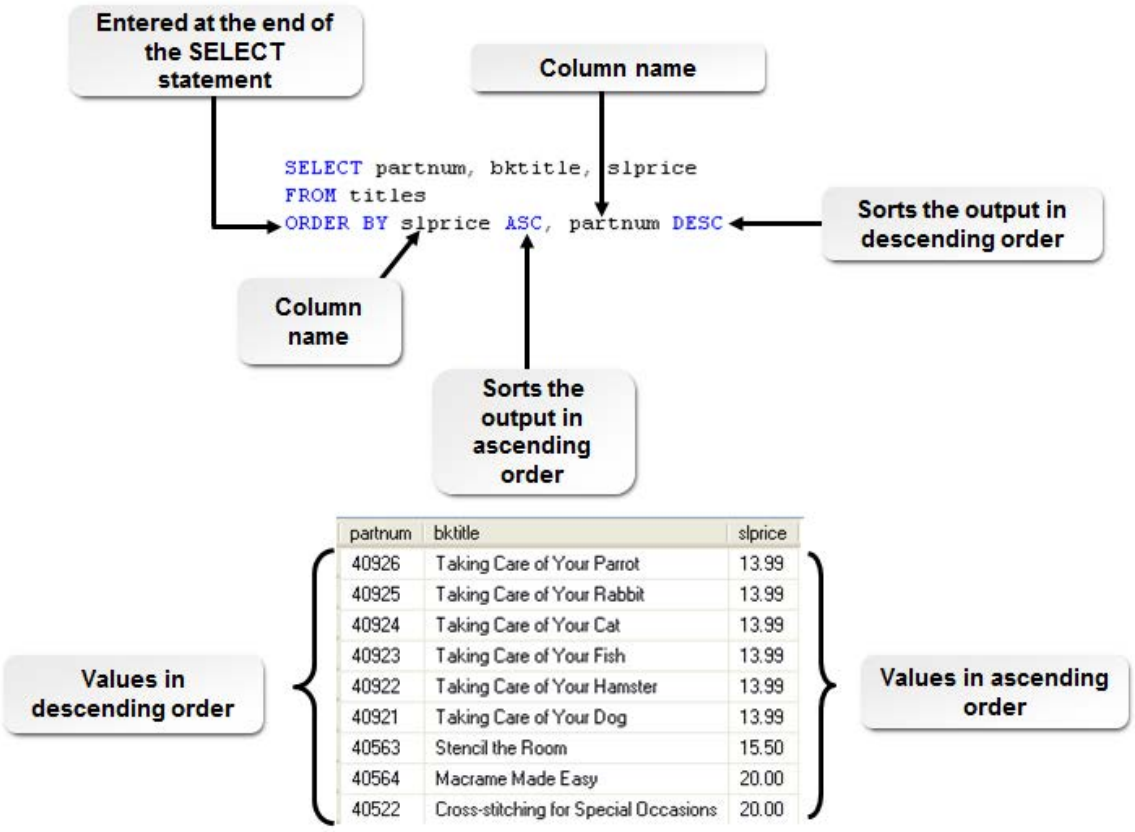
The ORDER BY Clause

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

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.



partnum	bktitle	slprice
40926	Taking Care of Your Parrot	13.99
40925	Taking Care of Your Rabbit	13.99
40924	Taking Care of Your Cat	13.99
40923	Taking Care of Your Fish	13.99
40922	Taking Care of Your Hamster	13.99
40921	Taking Care of Your Dog	13.99
40563	Stencil the Room	15.50
40564	Macrame Made Easy	20.00
40522	Cross-stitching for Special Occasions	20.00

Figure 4-3: A SELECT statement displaying the ORDER BY clause along with the resultant table.

Access the Checklist tile on your LogicalCHOICE course screen for reference information and job aids on How to Sort Data

ACTIVITY 4-1
Sorting Data

Before You Begin

- On the **Standard** toolbar, select **New Query** to open the **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** database is selected.

Scenario

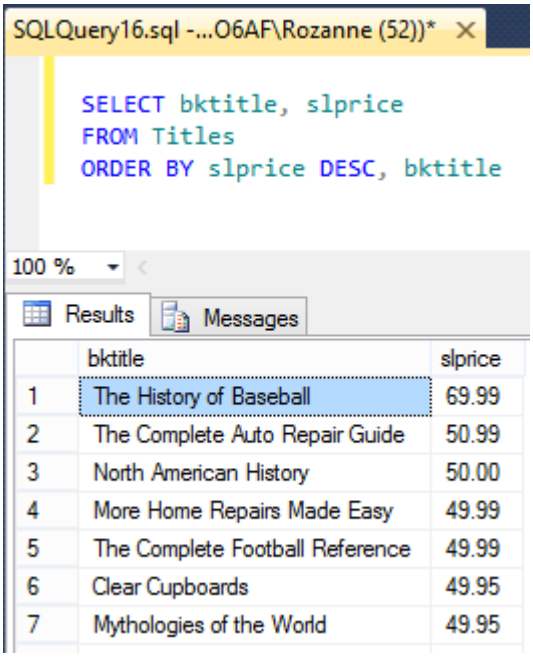
The management of Fuller & Ackerman Publishing has set up a team to analyze their profit and loss. They require information about book titles and their prices, listing books in descending order of price. They also want the number of books that need to be sold to break even. For information on tables and column names in the FullerAckerman database, refer to the table structure in Appendix A.

---

1. 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 titles.
   a) Enter the following query:

```
SELECT bktitle, slprice
FROM Titles
ORDER BY slprice DESC, bktitle
```

   b) Execute the query.
   c) In the **Results** pane, observe that 87 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.



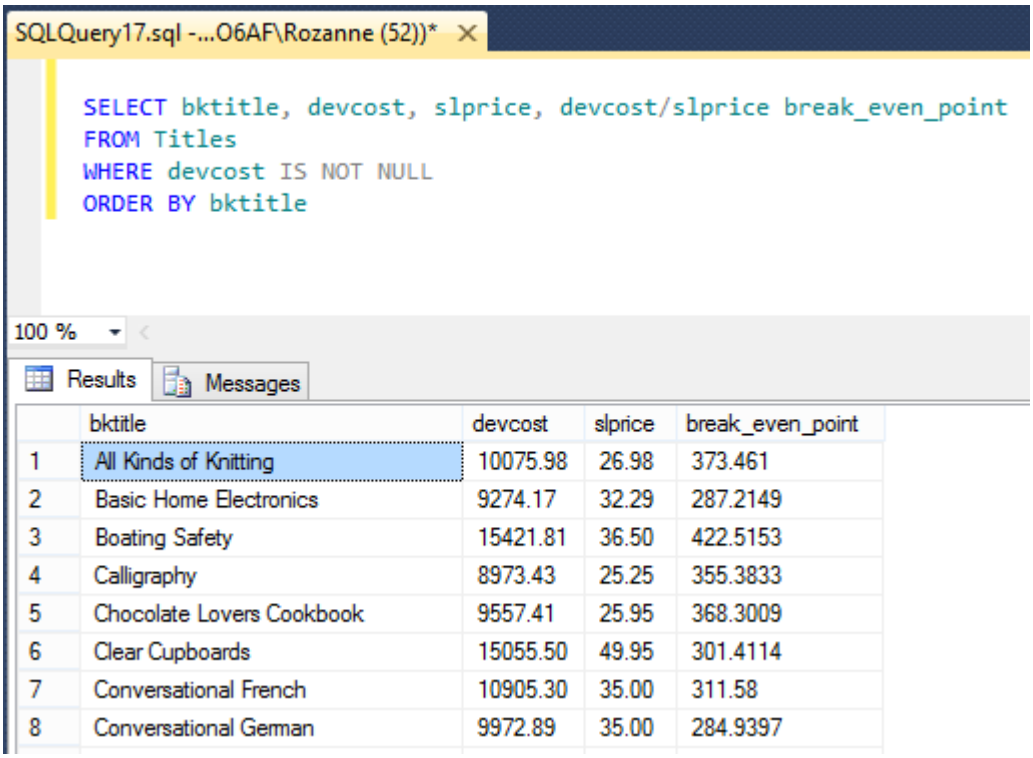
	bktitle	slprice
1	The History of Baseball	69.99
2	The Complete Auto Repair Guide	50.99
3	North American History	50.00
4	More Home Repairs Made Easy	49.99
5	The Complete Football Reference	49.99
6	Clear Cupboards	49.95
7	Mythologies of the World	49.95

   d) Close the **Query Editor** window without saving the query.

2. Display the information about the books and the number of copies that must be sold to determine the break even point for books that have the development cost listed. (The break even point represents the number of books Fuller & Ackerman Publishing must sell before they recoup their development cost for a book.) Sort the list in alphabetical order by title.
   a) On the **Standard** toolbar, select **New Query** to open the **Query Editor** window.
   b) On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, select **FullerAckerman**.
   c) Enter the following query:

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

   d) Execute the query.
   e) In the **Results** pane, observe that the book titles along with their development cost, sale price, and the number of copies that must be sold to break even are displayed.



	bktitle	devcost	slprice	break_even_point
1	All Kinds of Knitting	10075.98	26.98	373.461
2	Basic Home Electronics	9274.17	32.29	287.2149
3	Boating Safety	15421.81	36.50	422.5153
4	Calligraphy	8973.43	25.25	355.3833
5	Chocolate Lovers Cookbook	9557.41	25.95	368.3009
6	Clear Cupboards	15055.50	49.95	301.4114
7	Conversational French	10905.30	35.00	311.58
8	Conversational German	9972.89	35.00	284.9397

   f) Close the **Query Editor** window without saving the query.

TOPIC B

Rank Data

You sorted data. But what if you want to rank data based on predefined criteria? In this topic, you will rank data.

For a survey on how many people have a master's degree in an organization, you might want to identify and group the information available in the table based on each employee's area of specialization. After grouping, you might want to rank the people in each group based on their job grade. You can obtain these results by ranking grouped data.

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 `ROW_NUMBER()` function assigns a unique row number to each row in the output.
- 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).

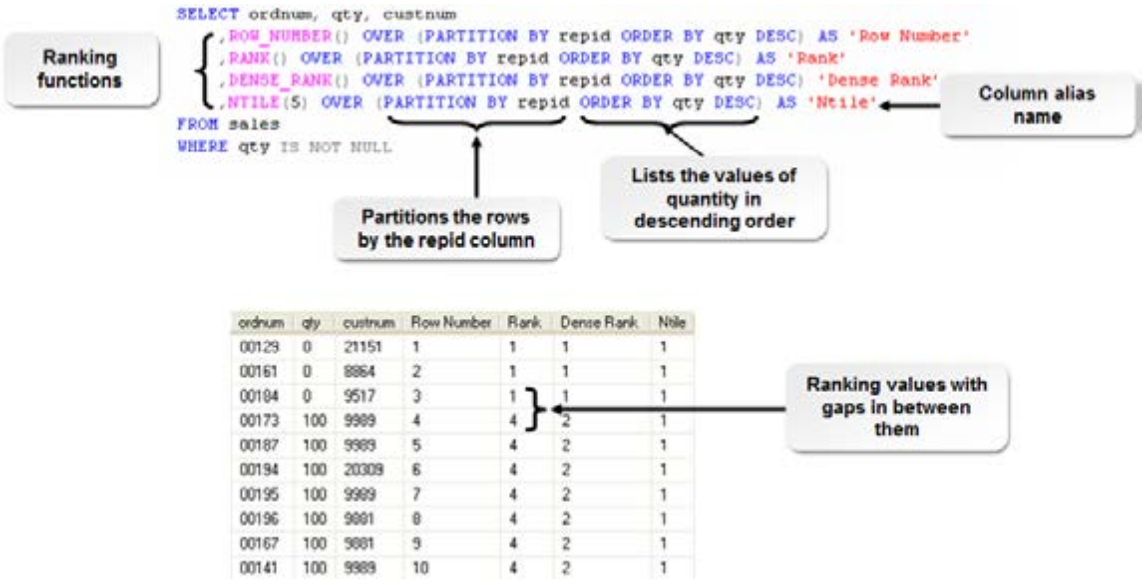


Figure 4-4: A SELECT statement displaying ranking functions.

Syntax of a Ranking Function

The syntax of a ranking function is:

```
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.

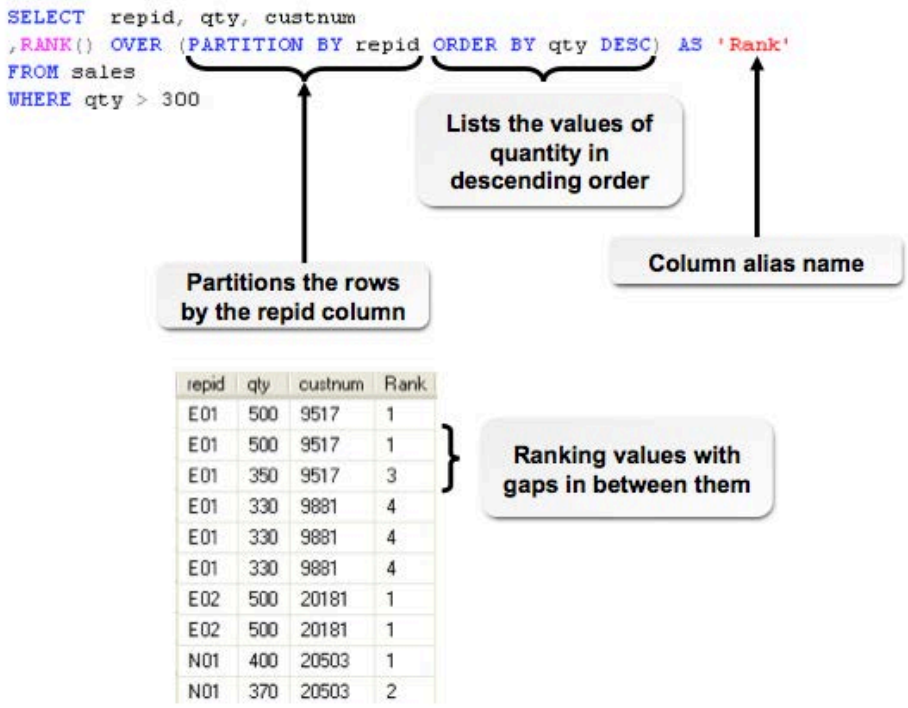


Figure 4-5: A SELECT statement displaying the RANK function.

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.

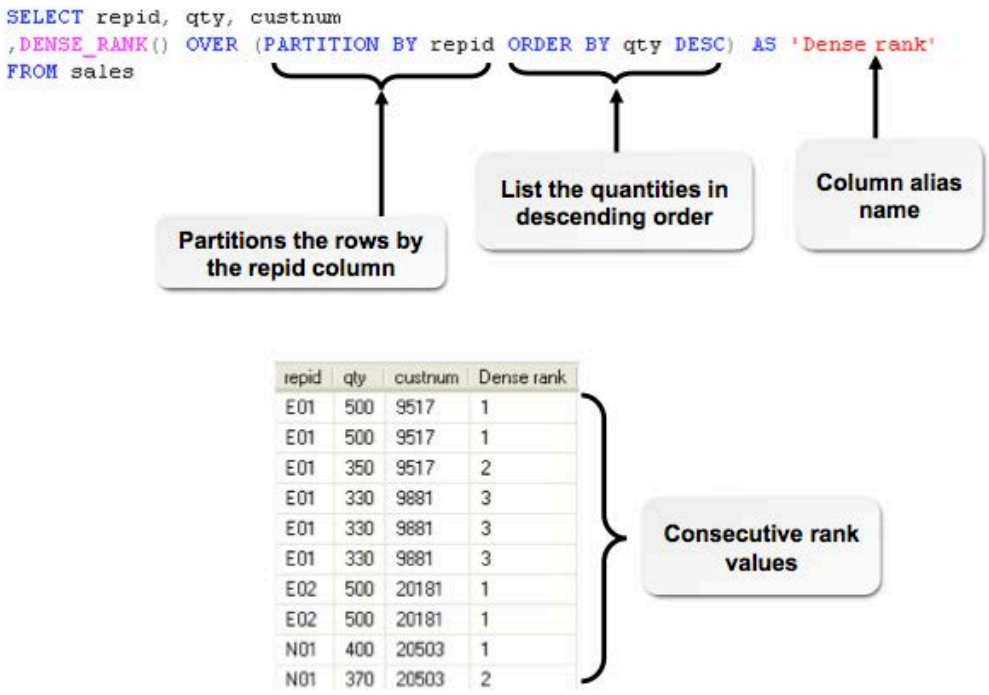


Figure 4-6: A SELECT query displaying the DENSE_RANK function.

Syntax of the DENSE_RANK Function

The syntax of the `DENSE_RANK` function is:

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

The ROW_NUMBER Function

The *ROW_NUMBER* function is a ranking function that uses an `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.

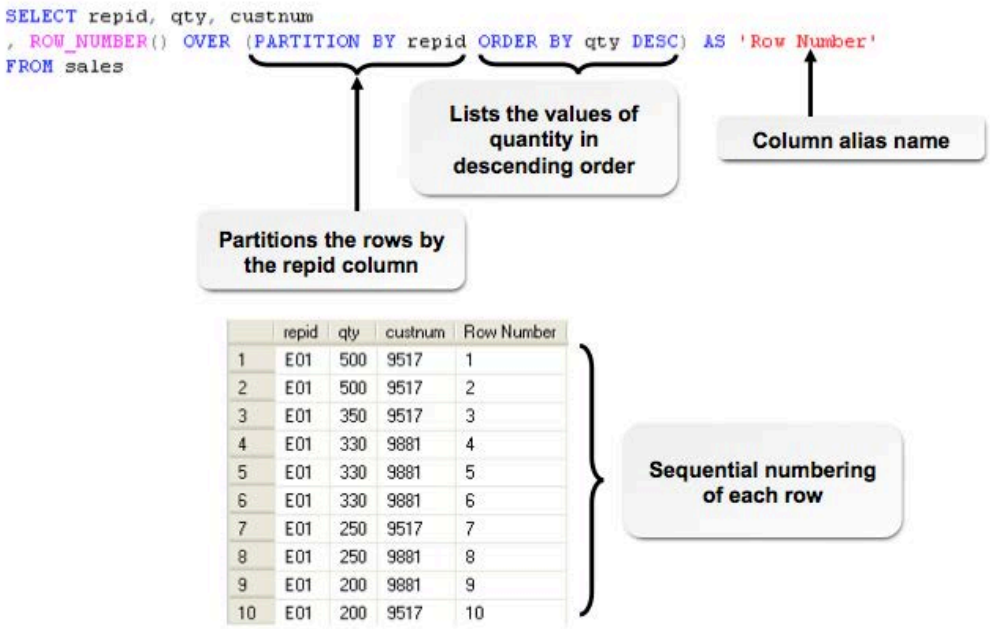


Figure 4-7: A SELECT statement displaying the ROW_NUMBER function.

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.

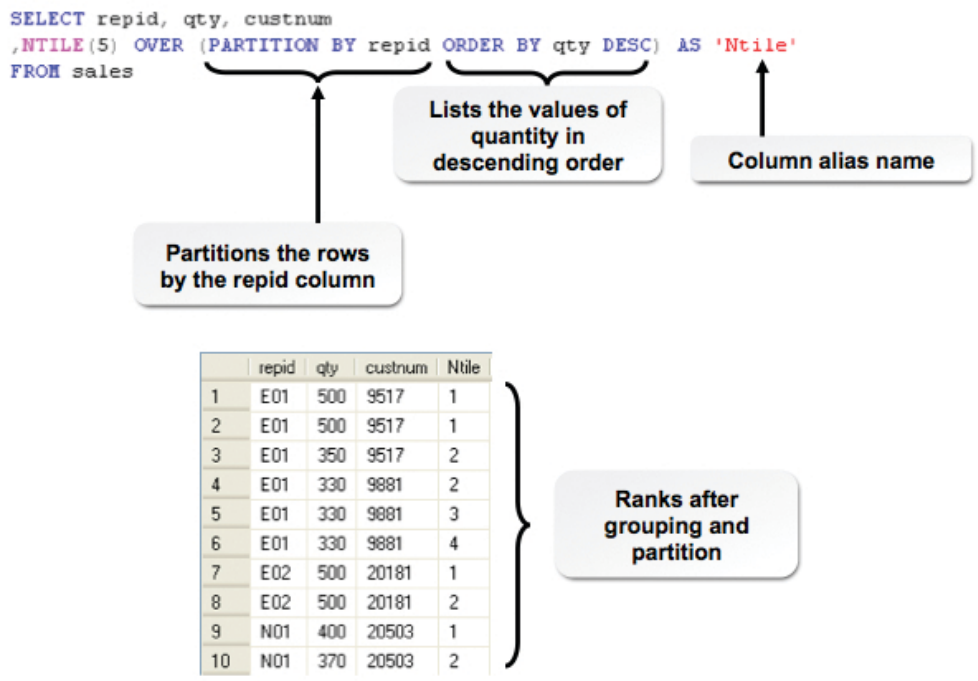


Figure 4-8: A SELECT statement displaying the NTILE function.

Syntax of the NTILE function

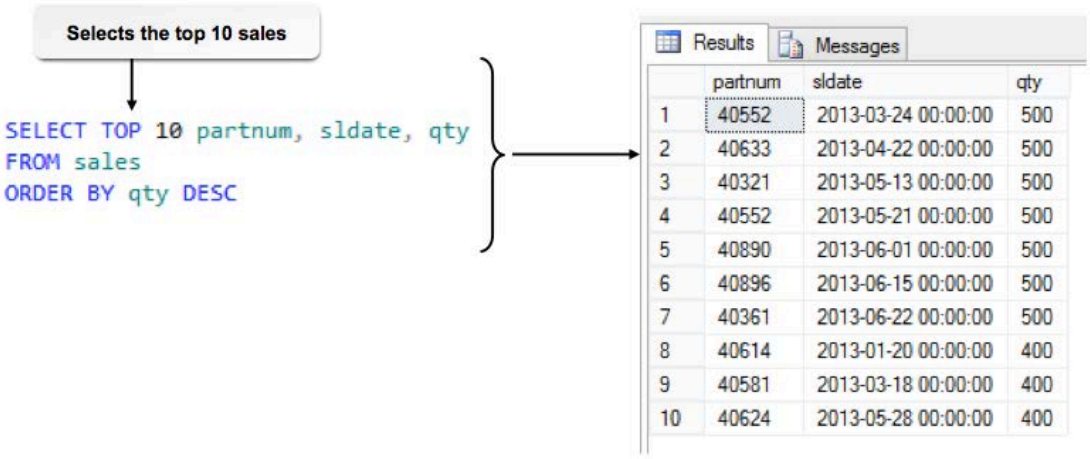
The syntax of the `NTILE` function is:

```
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 ten 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.



	partnum	sldate	qty
1	40552	2013-03-24 00:00:00	500
2	40633	2013-04-22 00:00:00	500
3	40321	2013-05-13 00:00:00	500
4	40552	2013-05-21 00:00:00	500
5	40890	2013-06-01 00:00:00	500
6	40896	2013-06-15 00:00:00	500
7	40361	2013-06-22 00:00:00	500
8	40614	2013-01-20 00:00:00	400
9	40581	2013-03-18 00:00:00	400
10	40624	2013-05-28 00:00:00	400

Figure 4-9: A SELECT statement using the TOP n keyword.

	Access the Checklist tile on your LogicalCHOICE course screen for reference information and job aids on How to Rank Data

ACTIVITY 4-2
Ranking Data

Before You Begin

- On the **Standard** toolbar, select **New Query** to open the **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** 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 2013. Even though they have a report containing the total sales quantities for 2013, 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 2013. For information on tables and column names in the FullerAckerman database, refer to the table structure in Appendix A.

---

1. Which ranking function should you use to partition rows into a specified number of groups?
   - ○ ROW_NUMBER ()
   - ○ DENSE_RANK ()
   - ○ NTILE ()
   - ○ RANK ()

2. Use the Sales table to list the representative IDs along with the quantity of books sold by each representative and the customer number for each sale. Rank the representative IDs based on the quantity of books sold by each representative.
   a) In the **Editor** pane, enter the first line of the query to retrieve the representative IDs, the quantity of books sold, and the customer number:
   ```
   SELECT repid, qty, custnum
   ```
   b) Enter the second line of the query. This line ranks values in the output after partitioning by representative ID and listing the sales quantity in descending order.
   ```
   SELECT repid, qty, custnum
      , RANK() OVER(PARTITION BY repid ORDER BY qty DESC) AS 'Rank'
   ```
   c) Enter the third line of the query. In this line, you want to obtain the density ranking for each row based on the representative ID and quantity sold. SQL Server will assign consecutive ranking values for each row in the result set by representative ID and quantity sold.
   ```
   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'
   ```

3. Divide the rows into five groups and rank the representative IDs based on the quantity of books sold by each representative, and display the row number for each row.
   a) Edit the query to add the `NTILE` function. Use the `NTILE` function to specify that you want five groups of rows, and rank each row after partitioning the result set on the representative ID and sales quantity.
   ```
   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'
   , NTILE(5) OVER(PARTITION BY repid ORDER BY qty DESC) AS 'Ntile'
```

b) Edit the query to generate a row number for each row after partitioning the result set based on the representative ID and sales quantity.

```
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'
   , NTILE(5) OVER(PARTITION BY repid ORDER BY qty DESC) AS 'Ntile'
   , ROW_NUMBER() OVER(PARTITION BY repid ORDER BY qty DESC) AS 'Row
Number'
```

4. Rank all representatives based on their sales quantity listed in the Sales table for 2013.

a) Edit the query to add the `FROM` clause.

```
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'
   , NTILE(5) OVER(PARTITION BY repid ORDER BY qty DESC) AS 'Ntile'
   , ROW_NUMBER() OVER(PARTITION BY repid ORDER BY qty DESC) AS 'Row
Number'
FROM sales
```

b) Add a `WHERE` clause to the query to select only those sales in 2013.

```
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'
   , NTILE(5) OVER(PARTITION BY repid ORDER BY qty DESC) AS 'Ntile'
   , ROW_NUMBER() OVER(PARTITION BY repid ORDER BY qty DESC) AS 'Row
Number'
FROM sales
WHERE DATEPART(year, sldate) = 2013
```

c) Execute the query.

d) In the **Results** pane, observe that different ranks are given for the representative IDs based on the quantity of sales they made during 2013. You should see a total of 94 rows.

e) Close the **Query Editor** window without saving the query.

TOPIC C

Group Data

You ranked the output of a query. In addition to ranking the output, you can also group data based on categories. In this topic, you will group data that belongs to the same category.

People use organizers to keep track of their daily activities. If you need information about the activities performed on a particular date, then all you need to do is turn to the page for the particular date and obtain the list of information. Similarly, when the information in a database is large, accessing the information becomes easier when you can categorize it; this helps in obtaining the required information rather than manually searching for it.

Groups

A *group* is a collection of two or more records combined into one unit based on the values in one or more columns. The records present in each group are listed together 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. The columns you specify in the `SELECT` statement must 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.

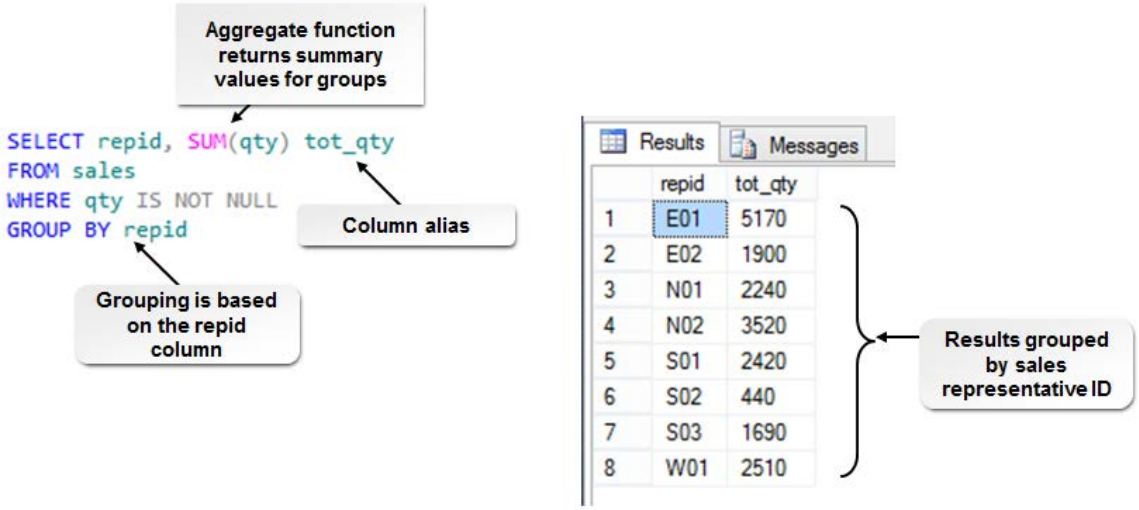


Figure 4-11: A SELECT statement using the GROUP BY clause.

If you do not include an `ORDER BY` clause is in the query, SQL Server returns the groups for the `GROUP BY` clause in any 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.

Specification	*Description*
Multilevel groups	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.
Aggregate functions	If you use aggregate functions in the `SELECT` clause, SQL Server calculates the summary values after the groups are formed.
Non-aggregate lists	When a non-aggregate list of values is entered in the `SELECT` clause, the entire list of values should be included in the `GROUP BY` list.
NULL values	If the column on which the group is formed contains NULL values, they are all placed in a single group.
Column aliases	The column alias used in the `SELECT` clause cannot be used to specify a grouping column.

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 which 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) total_sales
FROM sales
WHERE DATEPART(year,sldate) = 2008 AND
DATEPART(month,sldate) BETWEEN 1 AND 6
GROUP BY GROUPING SETS((custnum,repid),(repid))
```

Column values in grouping sets

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

Note: Sometimes, when a GROUP BY clause is used along with the GROUPING SETS feature, it will generate a result that is equivalent to multiple simple Transact-SQL GROUP BY statements. However, writing Transact-SQL statements is easier than using the GROUPING SETS subclause as it avoids the need to write multiple queries. The GROUPING SETS subclause enables you to list all grouping sets that you require. Another added advantage with these grouping set subclauses is that the SQL Server optimizes data access and the calculation of aggregates.

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 pre-defined 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.

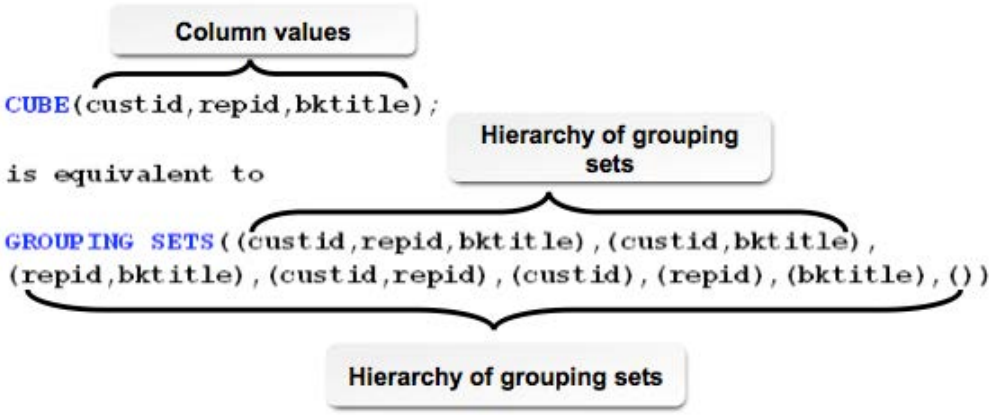


Figure 4-13: The CUBE and GROUPING SETS subclauses.

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.

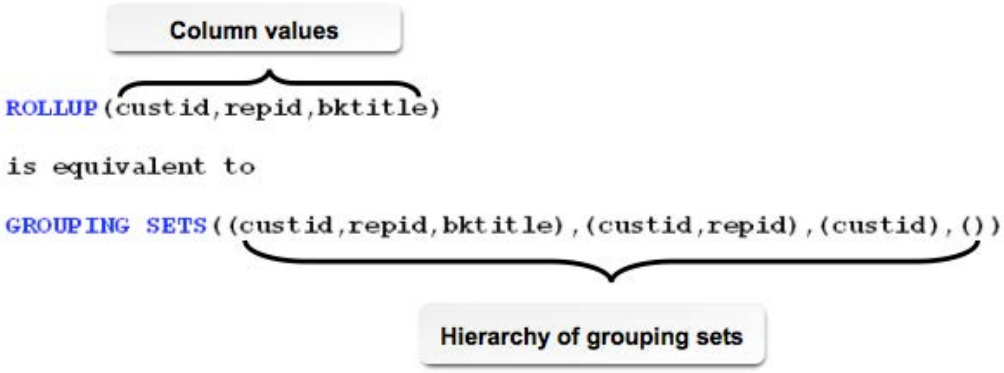


Figure 4-14: The ROLLUP and GROUPING SETS subclauses.

Example of the CUBE Subclause

The given example code illustrates the `CUBE` subclause. Consider the columns from the FullerAckerman database. `CUBE(custid,repid,tsales);` is equivalent to the following:

```
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 FullerAckerman database. `ROLLUP(custid,repid,tsales` is equivalent to the following:

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

	Access the Checklist tile on your LogicalCHOICE course screen for reference information and job aids on How to Group Data

ACTIVITY 4-3
Grouping Data

Before You Begin

- On the **Standard** toolbar, select **New Query** to open the **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** database is selected.

Scenario

You are given the task of collecting information about the total sales made by each representative for each customer over a period of six months. This information will be used to evaluate the representative's performance and also to set target sales goals for the next six months. For information on tables and column names in the FullerAckerman database, refer to the table structure in Appendix A.

---

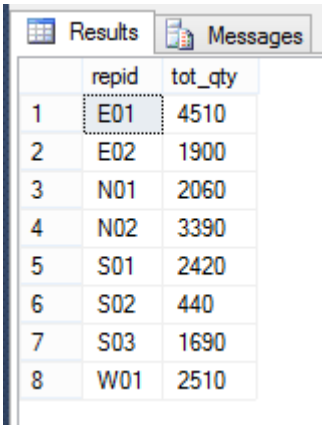
1. **Which columns need to be listed in the `SELECT` statement in order to list the total quantity of books sold by the representative to a customer?**
   - ○ AVG(qty), custnum, repid
   - ○ custnum, repid
   - ○ SUM(qty) total_qty, custnum, repid
   - ○ SUM(qty)/COUNT(qty), custnum, repid

2. **True or False? The columns used in the `SELECT` clause that do not have the aggregate function must also be used in the `GROUP BY` clause.**
   - ☐ True
   - ☐ False

3. Display the total sales made by each sales representative during 2013. Also, eliminate rows that have the NULL value as the quantity. Sort the rows in order by sales representative IDs.
   a) Enter the `SELECT` statement followed by the `FROM` clause to retrieve the customer number, representative ID, and sum of the quantity from the Sales table.
   ```
   SELECT repid, SUM(qty) tot_qty
   FROM sales
   ```
   b) Enter the `WHERE` clause to retrieve only the records that fall within 2013 where the quantity is not NULL.
   ```
   SELECT repid, SUM(qty) tot_qty
   FROM sales
   WHERE DATEPART(year, sldate) = 2013 AND qty IS NOT NULL
   ```
   c) Add the `GROUP BY` clause to group the rows by representative ID.
   ```
   SELECT repid, SUM(qty) tot_qty
   FROM sales
   WHERE DATEPART(year, sldate) = 2013 AND qty IS NOT NULL
   GROUP BY repid
   ```
   d) Add an `ORDER BY` clause to sort the results by the sales representatives' IDs.
   ```
   SELECT repid, SUM(qty) tot_qty
   FROM sales
   WHERE DATEPART(year, sldate) = 2013 AND qty IS NOT NULL
   ```

```
GROUP BY repid
ORDER BY repid
```

4. View the query results.
   a) Execute the query.
   b) In the **Results** pane, observe that the total sales by each representative for 2013 are displayed. There are a total of eight rows, one for each sales representative.


Results Messages

	repid	tot_qty
1	E01	4510
2	E02	1900
3	N01	2060
4	N02	3390
5	S01	2420
6	S02	440
7	S03	1690
8	W01	2510

   c) Close the **Query Editor** window without saving the query.

TOPIC D

Filter Grouped Data

You grouped data. When data is grouped, 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.

Suppose you have a table that contains the sales details of representatives in an organization. If you want to list only the representatives who have made sales of $2000 or more for a particular month, you need to group the records based on the representatives and calculate their total sales. Then, you can identify the representatives whose total sales are greater than $2000.

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. When the `GROUP BY` clause is not used, the `HAVING` clause behaves like a `WHERE` 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 600 or more.

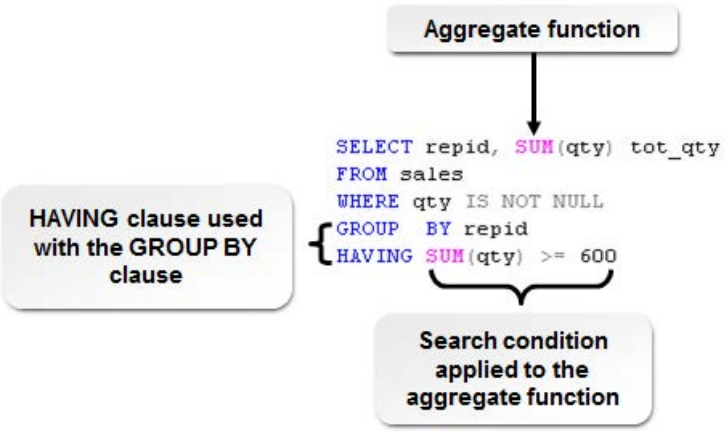


Figure 4-15: A SELECT statement displaying the HAVING clause with the GROUP BY clause.

Both the `HAVING` clause and the `WHERE` clause enable you to filter the results SQL Server displays in the output of a query. These clauses differ in what they can filter. The `WHERE` clause enables you to filter 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 LogicalCHOICE course screen for reference information and job aids on How to Filter Grouped Data

ACTIVITY 4-4
Filtering Grouped Data

Before You Begin

- On the **Standard** toolbar, select **New Query** to open the **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** database is selected.

Scenario

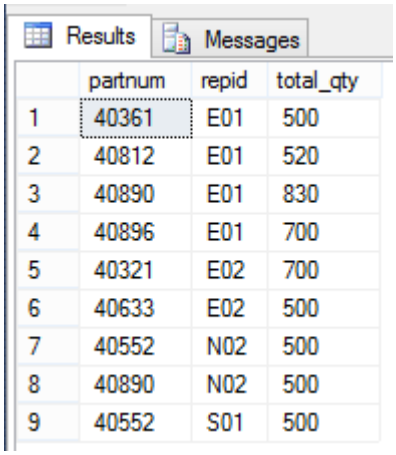
Fuller & Ackerman Publishing wants to increase by 10 percent the production of books that have sold 500 copies or more. The publishing department requires the part numbers of these books so that they can increase the production of the specified books. As an incentive, management decided to provide a bonus to representatives who have sold 500 copies. The financial department requires the IDs of these representatives to credit the bonus into their salary account. For information on tables and column names in the FullerAckerman database, refer to the table structure in Appendix A.

---

1. **True or False? To retrieve books that have sold more than 500 copies, you need to calculate the sum of the sales quantity of books that have the same part number and that sold greater than five hundred.**
   - ☐ True
   - ☐ False

2. Identify the book part number and representative IDs for titles that sold 500 copies or more. Eliminate rows that have NULL values as the quantity.
   a) Enter the `SELECT` statement followed by the `FROM` clause to display the part number, representative ID, and sum of quantity from the Sales table. Use the alias name total_qty for the sum of the quantity column.
   ```
   SELECT partnum, repid, SUM(qty) total_qty
   FROM sales
   ```
   b) Enter the `WHERE` clause to eliminate rows that have NULL values in the qty column.
   ```
   SELECT partnum, repid, SUM(qty) total_qty
   FROM sales
   WHERE qty IS NOT NULL
   ```
   c) Enter the `GROUP BY` clause to group data based on the part number of the book and the representative ID.
   ```
   SELECT partnum, repid, SUM(qty) total_qty
   FROM sales
   WHERE qty IS NOT NULL
   GROUP BY partnum, repid
   ```
   d) Enter the `HAVING` clause with the aggregate condition, the sum of quantity is greater than or equal to 500.
   ```
   SELECT partnum, repid, SUM(qty) total_qty
   FROM sales
   WHERE qty IS NOT NULL
   GROUP BY partnum, repid
   HAVING SUM(qty)>=500
   ```
   e) Execute the query.

f) In the **Results** pane, observe that the part numbers of books and the representative IDs for titles that have sold 500 or more copies are displayed. You should see a total of nine rows.


Results Messages

	partnum	repid	total_qty
1	40361	E01	500
2	40812	E01	520
3	40890	E01	830
4	40896	E01	700
5	40321	E02	700
6	40633	E02	500
7	40552	N02	500
8	40890	N02	500
9	40552	S01	500

g) Close the **Query Editor** window without saving the query.

TOPIC E

Summarize Grouped Data

You filtered grouped data using an aggregate condition. 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.

For a survey on how many people have a master's degree in an organization, you need to identify and group the information available in a table based on their area of specialization. After grouping, you obtained the count for each group. You then realized that a total number of people who have a master's degree is also required. This grand total can be calculated by summarizing 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.

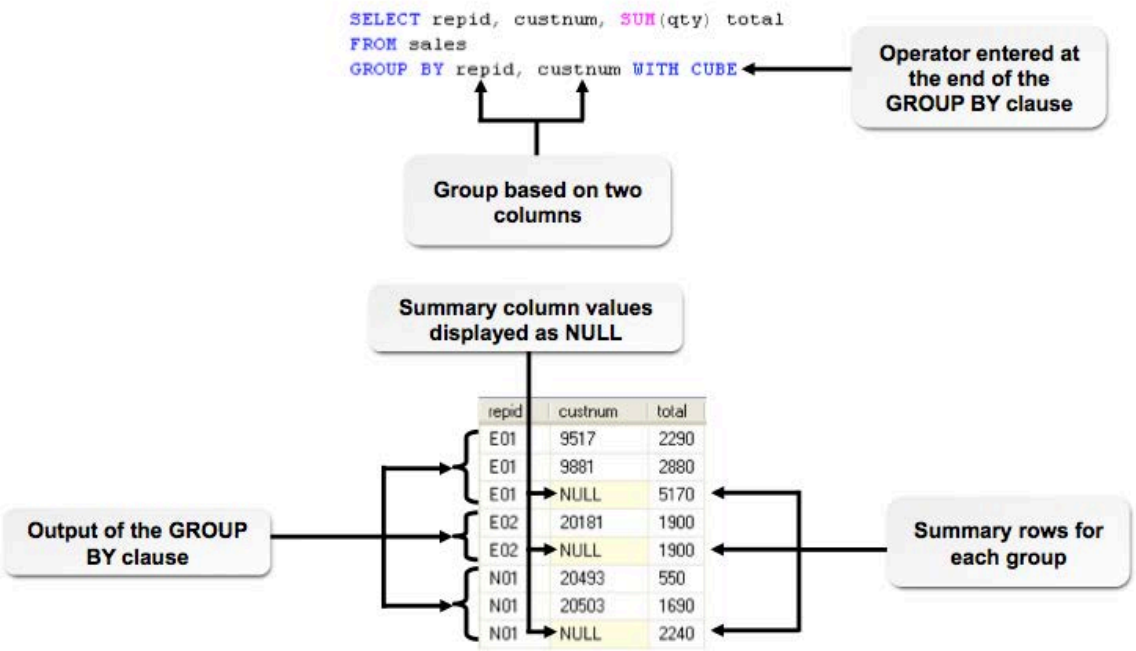


Figure 4-16: A SELECT statement displaying the CUBE operator.

When you use the *ROLLUP operator*, groups are summarized in the 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.

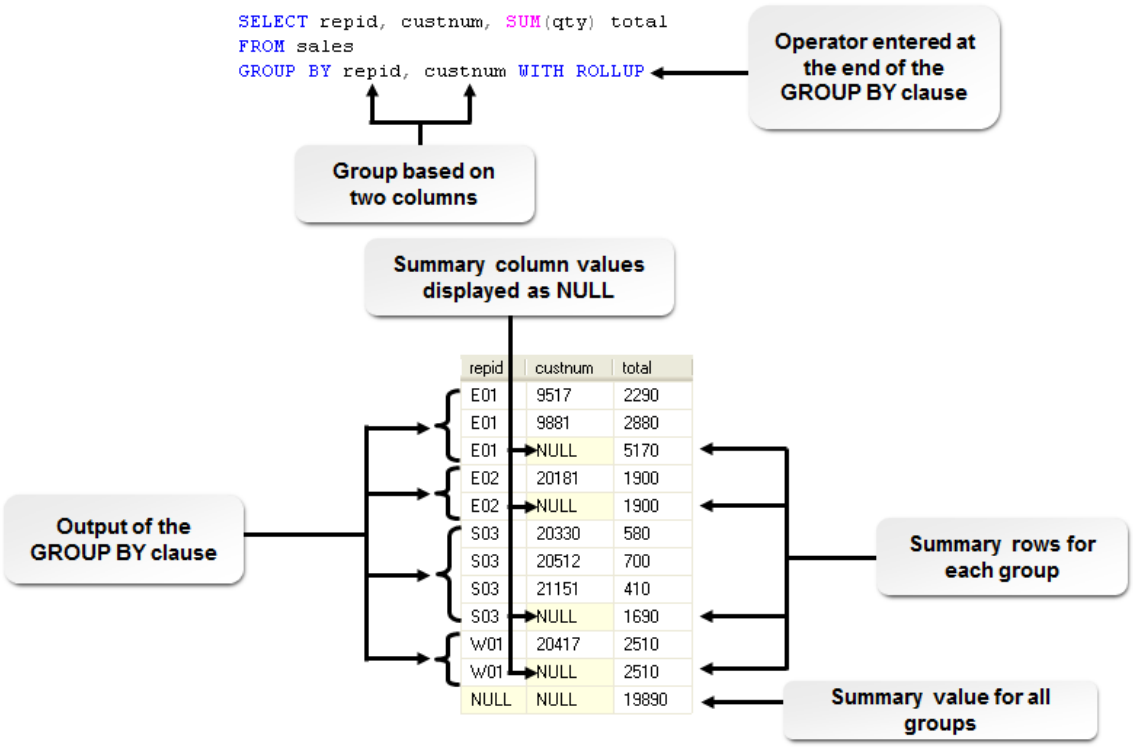


Figure 4-17: A SELECT statement displaying the ROLLUP operator.

	Access the Checklist tile on your LogicalCHOICE course screen for reference information and job aids on How to Summarize Grouped Data

ACTIVITY 4-5
Summarizing Grouped Data

Before You Begin

- On the **Standard** toolbar, select **New Query** to open the **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** database is selected.

Scenario

To evaluate the performance of sales representatives in a publishing company, data about the total sales made by each representative is required. The query output should list the representative IDs along with the sum of the sale quantity made by each representative, and the grand total of sales made by all representatives. For information on tables and column names in the FullerAckerman database, refer to the table structure in Appendix A.

---

1. **Which operator enables SQL Server to summarize groups in hierarchical order, from the lowest level in the group to the highest?**
   - ○ GROUP WITH
   - ○ ROLLUP
   - ○ WITH CUBE
   - ○ WITH ROLLUP

2. Display the representative IDs and total sales made by each representative.
   a) Enter the `SELECT` statement to retrieve the representative IDs, customer numbers, and the sum of the quantity sold from the Sales table. Use the alias name tot_sales for the sum of the quantity column.

```
SELECT repid, custnum, SUM(qty) tot_sales
FROM sales
```

   b) Enter the `GROUP BY` clause to return the grand total of sales made by all representatives.

```
SELECT repid, custnum, SUM(qty) tot_sales
FROM sales
GROUP BY repid, custnum WITH ROLLUP
```

   c) Execute the query.
   d) In the **Results** pane, observe that the representative IDs, the customers to whom they have sold books, and their total sales to each customer are displayed. In addition, SQL Server includes a row that totals the sales for each representative ID. Also observe that the grand total of sales made by all representatives is displayed in the last row.

	repid	custnum	tot_sales
1	E01	9517	2290
2	E01	9881	2880
3	E01	NULL	5170
4	E02	20181	1900
5	E02	NULL	1900
6	N01	20493	550
7	N01	20503	1690
8	N01	NULL	2240
9	N02	8802	760
10	N02	8864	250

e) Close the **Query Editor** window without saving the query.

TOPIC F

Use PIVOT and UNPIVOT Operators

You summarized data using the CUBE and ROLLUP operators. After you have summarized data, you may wish to rotate column values into multiple columns so as to 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.

Imagine that you would like to calculate the number of monitors and CPUs sold by each of the sales representatives in a company. You decide to separate the number of monitors and CPUs into columns listed against the names of representatives involved. The `PIVOT` and `UNPIVOT` relational operators enable you to accomplish this task.

The PIVOT and UNPIVOT Operators

PIVOT and *UNPIVOT* are relational operators that are 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 any of the columns and display the resultant data in a pivoted table. The `UNPIVOT` operator performs just the opposite of what the `PIVOT` operator does by rotating multiple columns into the values of a single column.

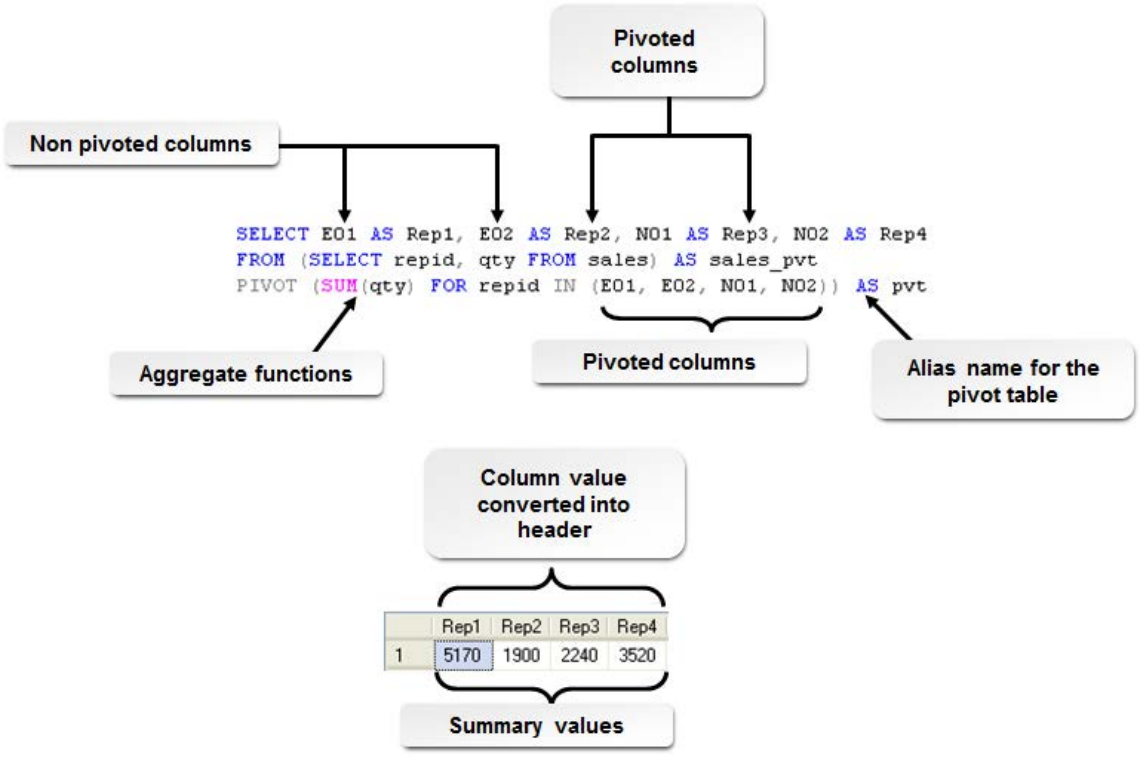


Figure 4-18: A SELECT statement using the PIVOT operator.

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.

Syntax of the PIVOT Operator

The syntax of a `PIVOT` operator is:

```
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 he pivot table>
<optional ORDER BY clause>
```

Access the Checklist tile on your LogicalCHOICE course screen for reference information and job aids on How to Use the PIVOT and UNPIVOT Operators

ACTIVITY 4-6
Using the PIVOT and UNPIVOT Operators

Data Files
sales_report.sql

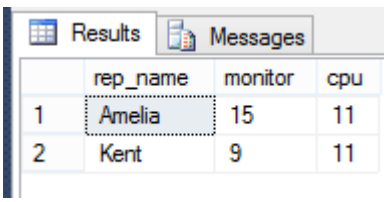
Before You Begin
- In the **C:\094005Data\Organizing Data** folder, open the **sales_report.sql** file and execute the query.
- Close the **Query Editor** window.
- On the **Standard** toolbar, select **New Query** to open the **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** database is selected.

Scenario
A marketing firm maintains a database containing information about sales representatives and the total quantity of computer peripherals sold by each of the representatives. The accounts manager wants a list of each of the peripherals sold by various representatives. You have decided to use relational operators to generate cross-tabulation reports to determine the number of peripherals sold by each representative. For information on tables and column names in the FullerAckerman database, refer to the table structure in Appendix A.

---

1. Create a new table named Sales_pvt that uses the column values from Sales_Report as column headers and summarizes the sales quantity from the Sales_Report table.
   a) Enter the following query:
   ```
   SELECT pvt.* INTO sales_pvt
   FROM Sales_Report
   PIVOT (SUM(quantity) FOR product IN (Monitor,CPU)) AS pvt
   ```
   b) Execute the query.
   c) In the **Results** pane, on the **Messages** tab, observe that the message **(2 row(s) affected)** is displayed.

2. View the content of the Sales_pvt table.
   a) Edit the query to read as follows:
   ```
   SELECT *
   FROM Sales_pvt
   ```
   b) Execute the query.
   c) In the **Results** pane, observe that the column values **Monitor** and **CPU** have been rotated into column headers along with corresponding aggregate values of the quantity for each representative.

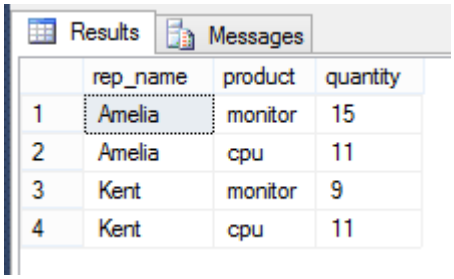


	rep_name	monitor	cpu
1	Amelia	15	11
2	Kent	9	11

3. Create a new table that changes the column headers into column values from the Sales_pvt table.
   a) Edit the query to read:

```
SELECT unpvt.rep_name, unpvt.product, unpvt.quantity
FROM Sales_pvt
UNPIVOT (quantity FOR product IN (Monitor, CPU)) AS unpvt
```

   b) Execute the `UNPIVOT` query.
   c) In the **Results** pane, observe that the column headers are rotated back to column values.



	rep_name	product	quantity
1	Amelia	monitor	15
2	Amelia	cpu	11
3	Kent	monitor	9
4	Kent	cpu	11

   d) Close the **Query Editor** window without saving the query.

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?

When might you group the data in a table?

Note: Check your LogicalCHOICE Course screen for opportunities to interact with your classmates, peers, and the larger LogicalCHOICE 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.

5 Retrieving Data from Multiple Tables

Lesson Time: 1 hour, 30 minutes

Lesson Objectives

In this lesson, you will retrieve data from multiple tables. You will:

- Combine results obtained from two queries to get a single output.
- Compare results of two queries to get distinct values.
- Retrieve data by joining tables.

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.

For various reasons, database designers might split data into multiple tables within a database. To obtain specific information from multiple tables, you might need to combine the information present in these tables. Sometimes it is not possible to retrieve information from a single table, using simple conditions. You might have to combine a table with itself in order to retrieve information from it.

TOPIC A

Combine the Results of Two Queries

Up to this point in the course, you have retrieved data from simple queries from one table at a time. Sometimes, however, you might want to query two different tables and generate the results in a single, unified result set. In this topic, you will combine the results of two queries into a single result set.

Consider a banking database in which the debit and credit transactions are stored in separate tables. If you want to view a list of a customer's debit and credit transactions, you would need to enter two queries. If the output of the queries is similar, you can combine the outputs and list them as a single output that displays all of a customer's transactions.

Note: To further explore database design, you can access the LearnTO **Normalize a Database** presentation from the LearnTO tile on the LogicalCHOICE Course screen.

The UNION Operator

The *UNION operator* is an operator that you can use 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.

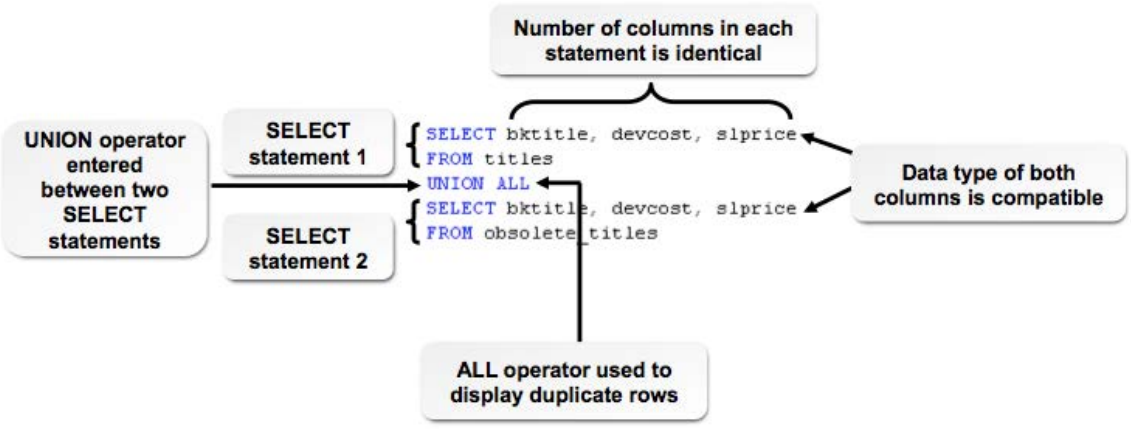


Figure 5-1: Two SELECT statements using the UNION ALL operator.

Syntax of the UNION Operator

The syntax of the `UNION` operator is:

```
SELECT column1, column2, ...
FROM table1
UNION [ALL]
SELECT column1, column2, ...
FROM table2
[UNION [ALL]]
```

Access the Checklist tile on your LogicalCHOICE 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

Before You Begin

- On the **Standard** toolbar, select **New Query** to open a **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** database is selected.

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

---

1. List the book title and the publishing date of all books in the Titles table.
   a) Enter the following query to list the book title and publishing date of all books and add the alias names book_title and publishing_date for the bktitle and pubdate columns, respectively.

```
SELECT bktitle book_title, pubdate publishing_date
FROM titles
```

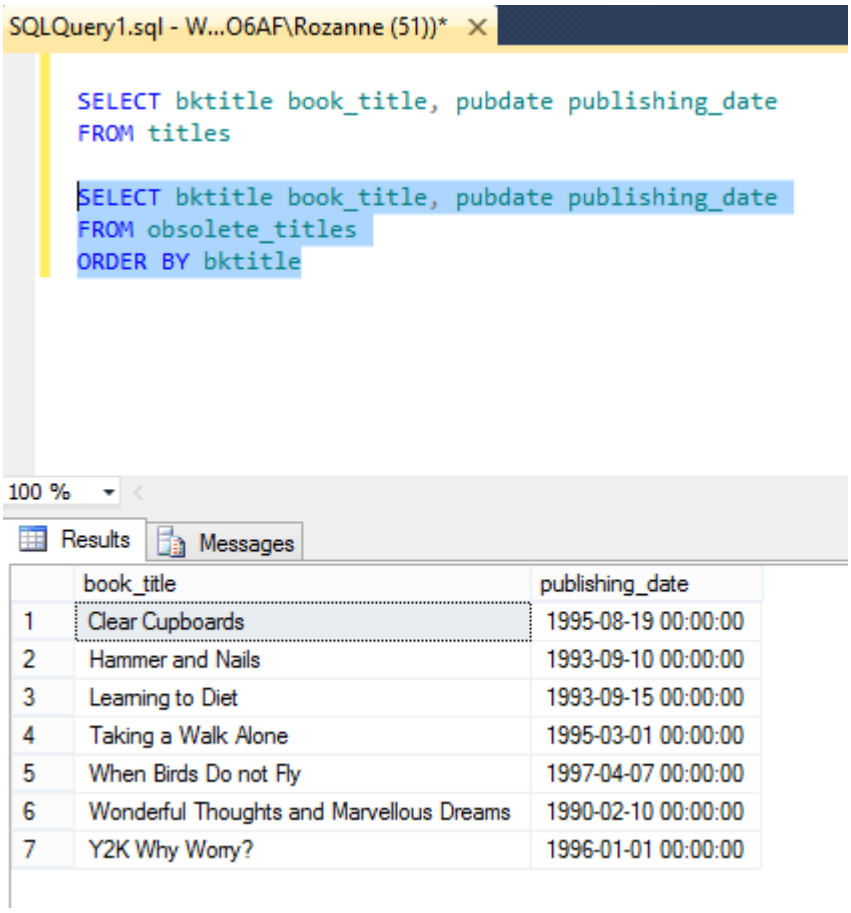
   b) Execute the query. You see a list of 92 books and their publication dates from the Titles table.

	book_title	publishing_date
1	Clear Cupboards	2012-08-19 00:00:00
2	Developing Mobile Apps	2013-01-01 00:00:00
3	Boating Safety	2013-05-18 00:00:00
4	Sailing	2013-05-03 00:00:00
5	The Sport of Windsurfing	2012-07-13 00:00:00
6	The Sport of Hang Gliding	2013-01-06 00:00:00

2. List the book title and the publishing date of all books in the Obsolete_titles table, and sort the book titles by their publishing dates.
   a) Press **Enter** twice to insert a blank line after the previous query.
   b) Enter this query to list the book titles and publication dates of the books in the Obsolete_titles table.

```
SELECT bktitle book_title, pubdate publishing_date
FROM obsolete_titles
ORDER BY bktitle
```

   c) Select this second query with the mouse and then select **Execute** to view a list of the seven books in the Obsolete_titles table.



	book_title	publishing_date
1	Clear Cupboards	1995-08-19 00:00:00
2	Hammer and Nails	1993-09-10 00:00:00
3	Learning to Diet	1993-09-15 00:00:00
4	Taking a Walk Alone	1995-03-01 00:00:00
5	When Birds Do not Fly	1997-04-07 00:00:00
6	Wonderful Thoughts and Marvellous Dreams	1990-02-10 00:00:00
7	Y2K Why Worry?	1996-01-01 00:00:00

3. Combine the results of the two tables to generate a single book list, and execute the query.
   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.

```
SELECT bktitle book_title, pubdate publishing_date
FROM titles
UNION
SELECT bktitle book_title, pubdate publishing_date
FROM obsolete_titles
ORDER BY bktitle
```

   b) Execute the query.
   c) In the **Results** pane, observe that SQL Server displays the combined output of both tables (99 rows).
   d) Close the **Query Editor** window without saving the query.

TOPIC B

Compare the Results of Two Queries

You combined 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.

In some situations, you might want to retrieve only those transactions that appear in one table but not the other. For example, you might want to retrieve a list of all customers who have not yet made purchases this year. Alternatively, you might want to retrieve a list of only those customers who have purchased a specific book. SQL includes commands that enable you to retrieve just such result sets.

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.

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
```

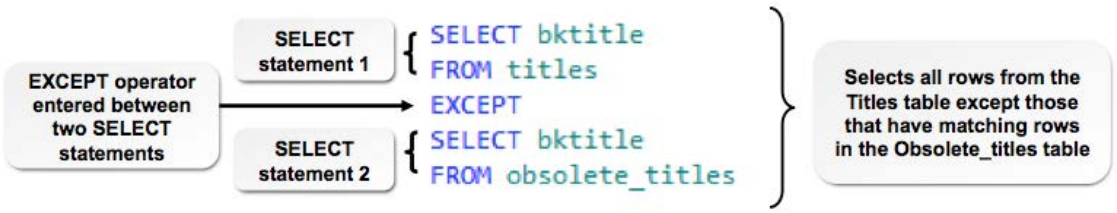


Figure 5-2: Two SELECT statements using the EXCEPT operator.

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
```

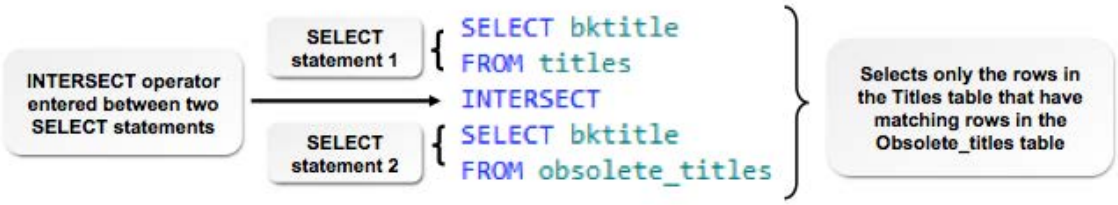


Figure 5-3: Two SELECT statements with the INTERSECT operator.

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 LogicalCHOICE 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

- On the **Standard** toolbar, select **New Query** to open a **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** 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. For information on tables and column names in the FullerAckerman database, refer to the table structure in Appendix A.

1. Retrieve any records in the Titles table that are also in the Obsolete_titles table.
   a) Enter the following query:
   ```
   SELECT bktitle
   FROM titles
   INTERSECT
   SELECT bktitle
   FROM obsolete_titles
   ```
   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) Edit the query to change the keyword `INTERSECT` to `EXCEPT` as follows:
   ```
   SELECT bktitle
   FROM titles
   EXCEPT
   SELECT bktitle
   FROM obsolete_titles
   ```
   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 records.
   d) Close the **Query Editor** window without saving the query.

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.

Imagine an organization in which the sales information is stored in one table and the customer information in another. The sales information table contains a list of sales transactions; the customer table contains a list of customers by name and address. SQL Server links the two tables together based on each customer's ID number. In this scenario, if you wanted a report with a list of customer names and a list of their purchases, you must join the two tables based on the customer ID number.

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, and self joins.

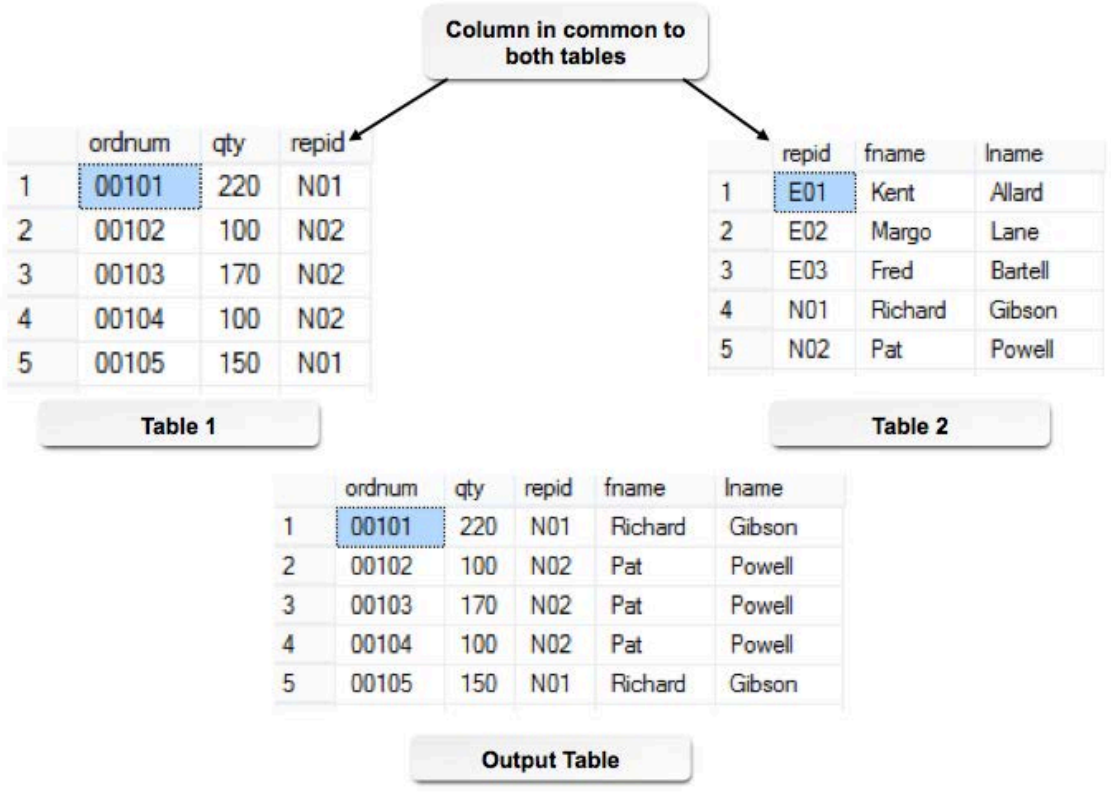


Figure 5-4: An output table generated by joining Table 1 and Table 2.

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

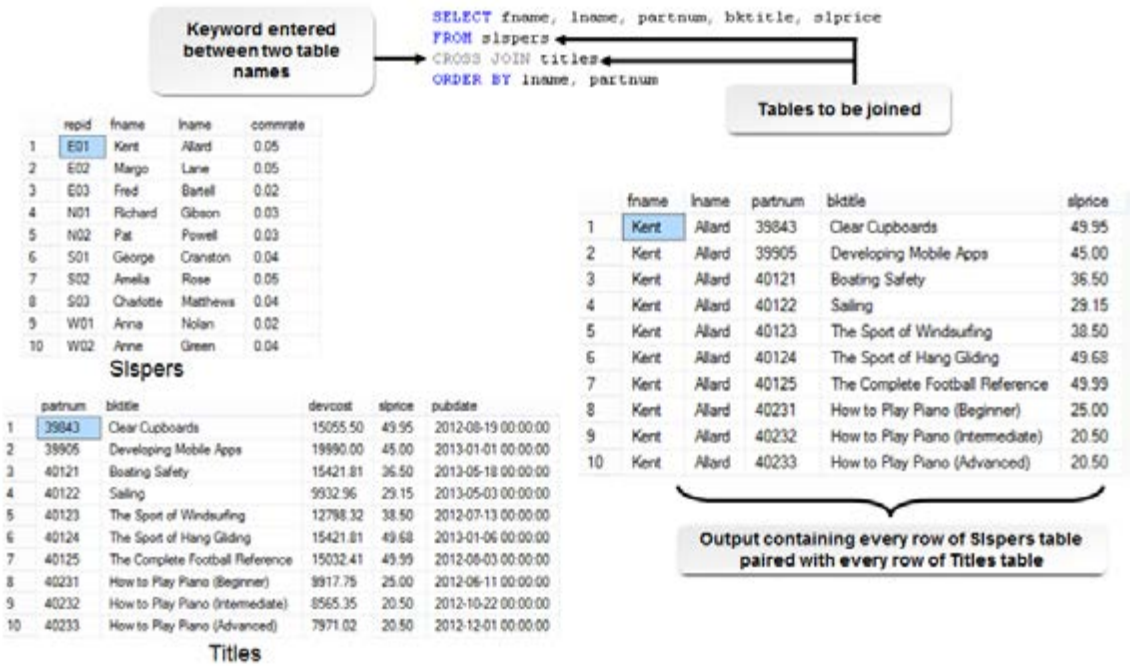


Figure 5-5: A cross join combining the values in the Slspers and Titles tables.

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

Syntax of a Cross Join

The syntax of a cross join is:

```
SELECT colname1, [colname2, ...]
FROM tablename1 CROSS JOIN tablename2
ON tablename1.column = tablename2.column
```

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.

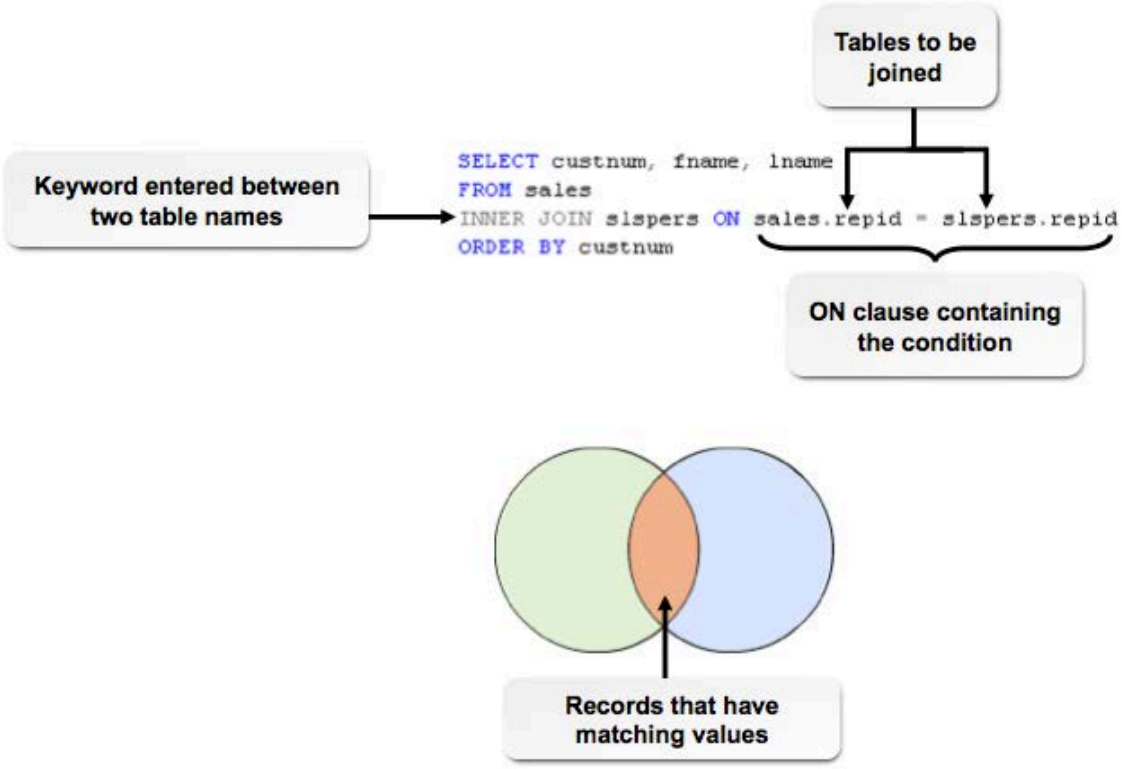


Figure 5-6: The Sales table with an inner join to the Slspers table on the repid column.

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 FullerAckerman 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.

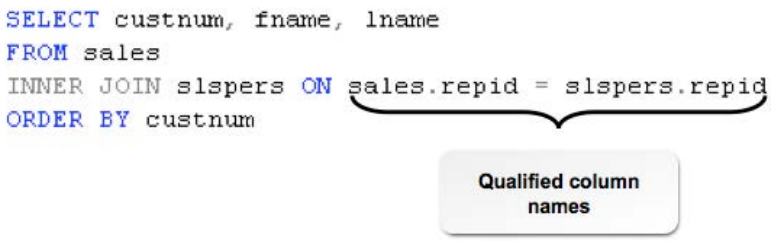


Figure 5-7: Qualified column names.

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.

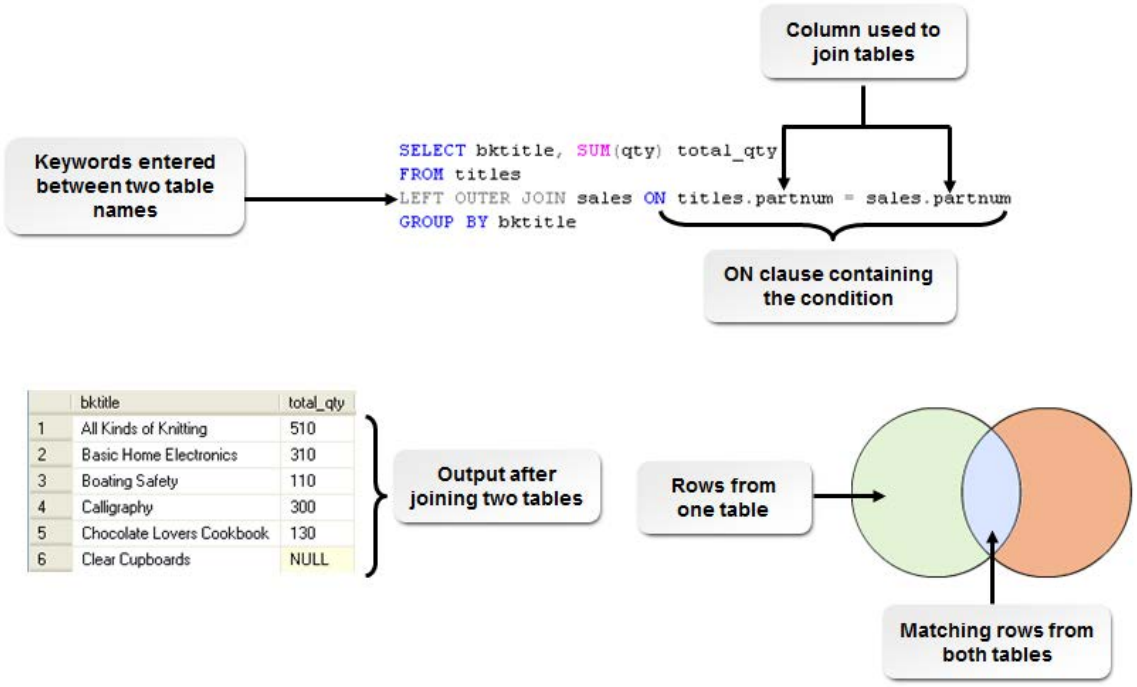


Figure 5-8: The results of a left outer join.

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:

```
SELECT column1, columnn2, ...
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:

```
SELECT column1, columnn2, ...
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:

```
SELECT column1, columnn2, ...
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 a 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 ot to the Obsolete_titles table and the alias of t to the Titles table.

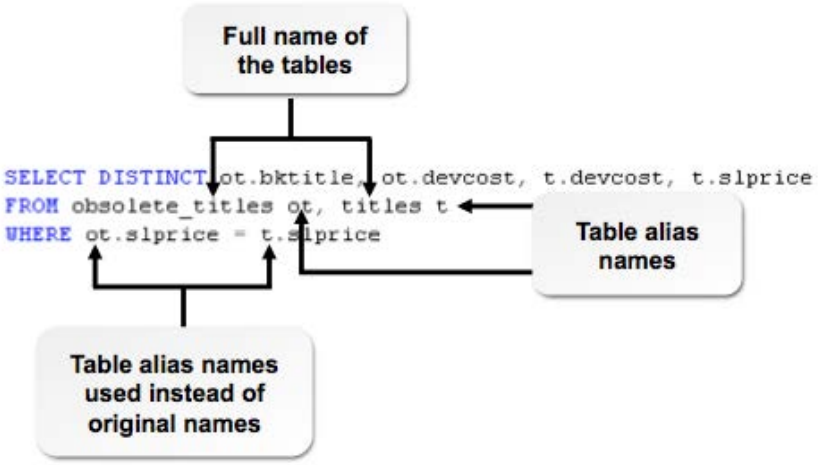


Figure 5-9: A SELECT statement displaying the table alias names.

Note: You can optionally use 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.

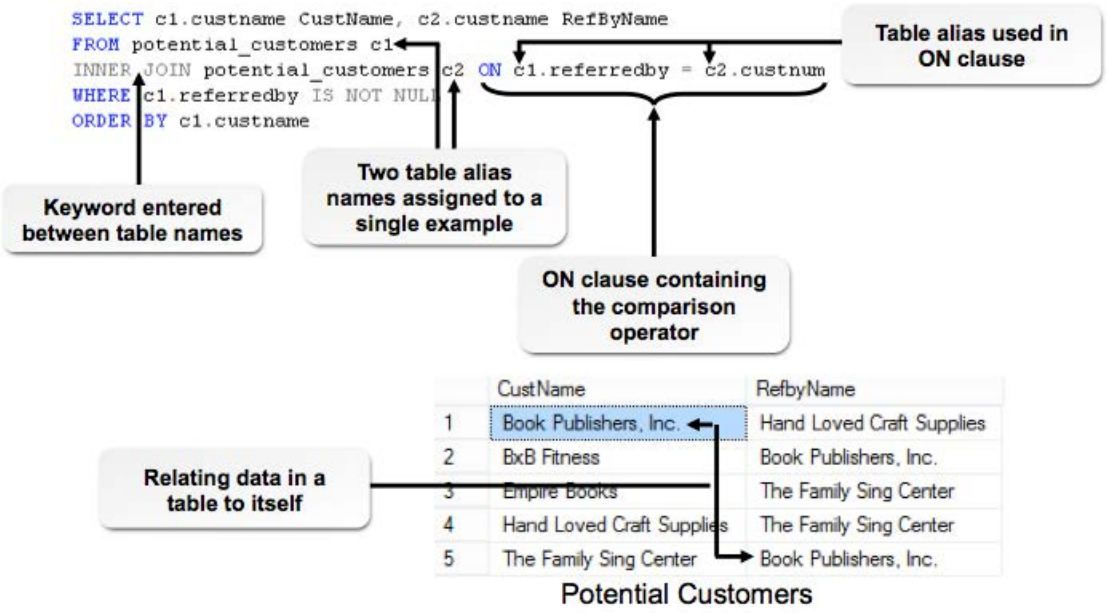


Figure 5-10: A code sample using a self join.

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 FullerAckerman database. Consider the structure of the following tables, as shown in the figure.

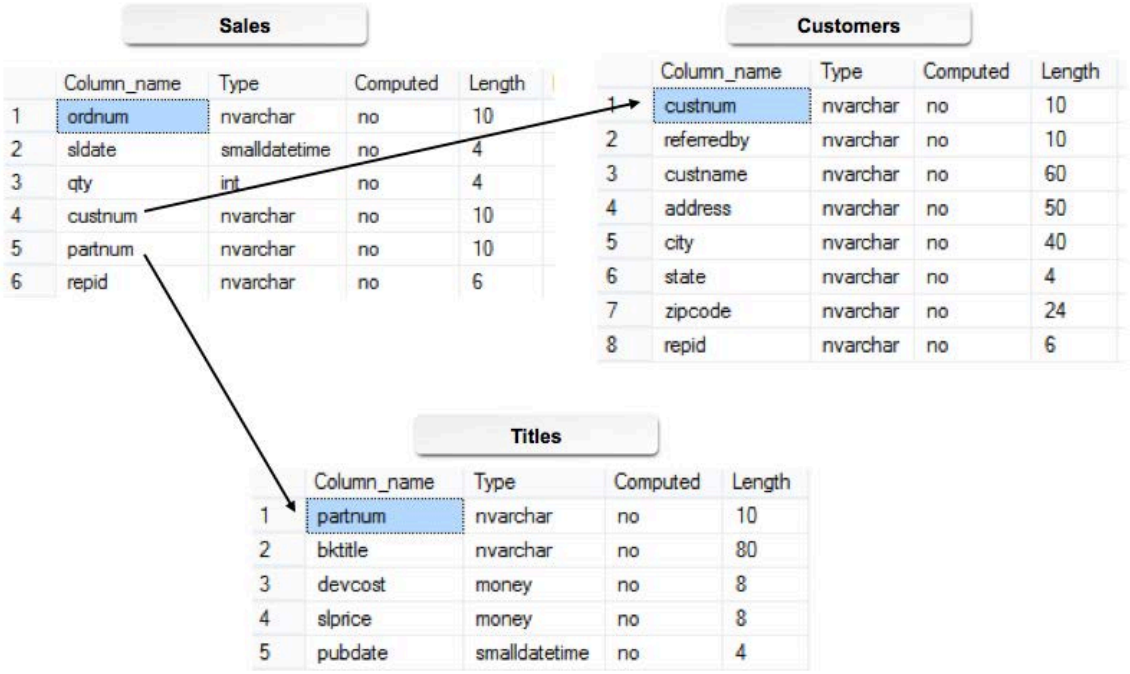


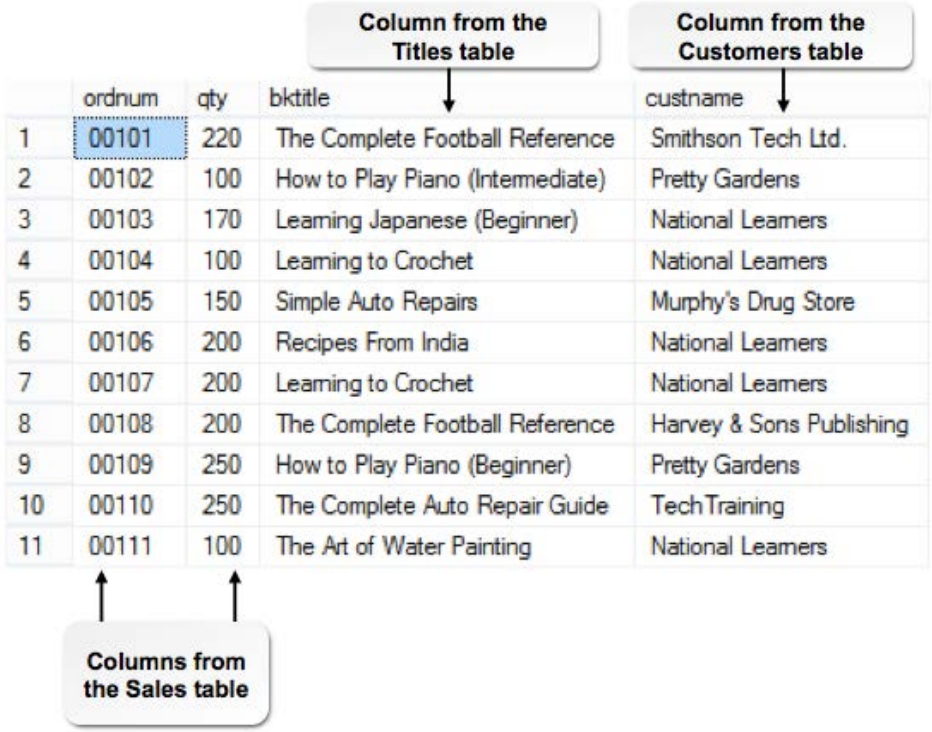
Figure 5-11: The structure of tables in the FullerAckerman database.

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 that 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. Here is the syntax:

```
SELECT ordnum, qty, bktitle, custname
FROM sales INNER JOIN customers
ON sales.custnum = customers.custnum
INNER JOIN titles
ON sales.partnum = titles.partnum
```

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.



	ordnum	qty	bktitle	custname
1	00101	220	The Complete Football Reference	Smithson Tech Ltd.
2	00102	100	How to Play Piano (Intermediate)	Pretty Gardens
3	00103	170	Learning Japanese (Beginner)	National Learners
4	00104	100	Learning to Crochet	National Learners
5	00105	150	Simple Auto Repairs	Murphy's Drug Store
6	00106	200	Recipes From India	National Learners
7	00107	200	Learning to Crochet	National Learners
8	00108	200	The Complete Football Reference	Harvey & Sons Publishing
9	00109	250	How to Play Piano (Beginner)	Pretty Gardens
10	00110	250	The Complete Auto Repair Guide	TechTraining
11	00111	100	The Art of Water Painting	National Learners

Figure 5-12: Results of the multiple table join query.

Note: To further explore joins, you can access the LearnTO **Choose a Join Type** presentation from the LearnTO tile on the LogicalCHOICE Course screen.

Access the Checklist tile on your LogicalCHOICE course screen for reference information and job aids on How to Retrieve Data by Joining Tables

ACTIVITY 5-3
Retrieving Data by Joining Tables

Before You Begin

- On the **Standard** toolbar, select **New Query** to open a **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** database is selected.

Scenario

Management has asked you to provide them with the following information:

- 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, 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.
- A list of all books published by Fuller & Ackerman and their total sales (if any).
- A list of all potential customers and the customers that referred them.

---

1. Enter an inner join query to retrieve a list of all sales along with the sales representatives' names.
   a) Enter the following query to display the order number, sale date, quantity, first name, and last name of the sales representative:

```
SELECT ordnum, sldate, qty, fname, lname
```

   b) Enter the `FROM` clause to specify that you want to retrieve all rows from the Sales table.

```
SELECT ordnum, sldate, qty, fname, lname
FROM sales
```

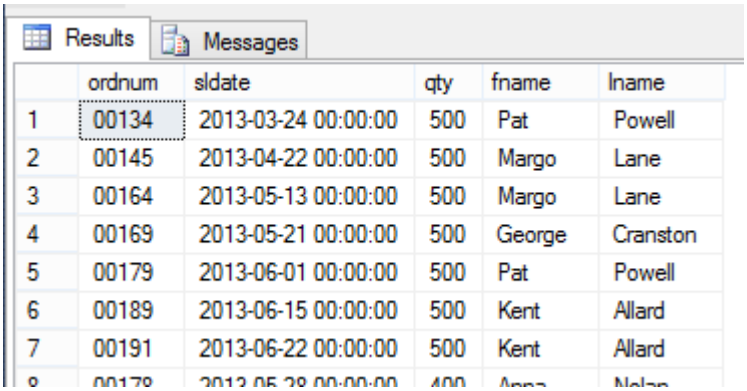
   c) Add the `INNER JOIN` clause to join the Sales table to the Slspers table.

```
SELECT ordnum, sldate, qty, fname, lname
FROM sales
INNER JOIN slspers ON sales.repid = slspers.repid
```

   d) Add the `ORDER BY` clause to sort the output in alphabetical order by sales representative name.

```
SELECT ordnum, sldate, qty, 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 view the records. Observe that the results are in descending order by the quantity sold.



Results | Messages

	ordnum	sldate	qty	fname	lname
1	00134	2013-03-24 00:00:00	500	Pat	Powell
2	00145	2013-04-22 00:00:00	500	Margo	Lane
3	00164	2013-05-13 00:00:00	500	Margo	Lane
4	00169	2013-05-21 00:00:00	500	George	Cranston
5	00179	2013-06-01 00:00:00	500	Pat	Powell
6	00189	2013-06-15 00:00:00	500	Kent	Allard
7	00191	2013-06-22 00:00:00	500	Kent	Allard

2. Enter a left outer join query to retrieve a list of all titles that Fuller & Ackerman publishes and the sales for each title.
   a) Edit the existing query to display the book title and the total quantity sold.

```
SELECT bktitle, SUM(qty) quantity
```

   b) Add the `FROM` clause to join all rows in the Titles table to the Sales table.

```
SELECT bktitle, SUM(qty) quantity
FROM titles LEFT OUTER JOIN sales
```

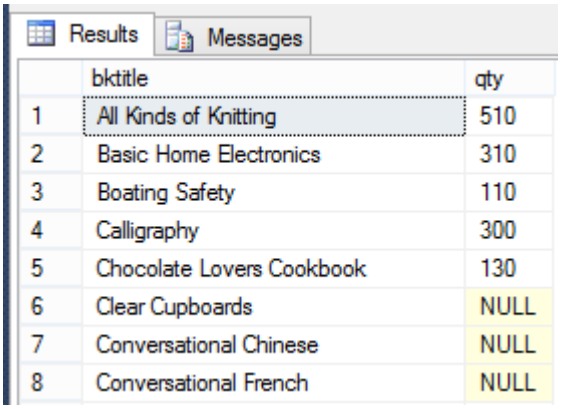
   c) Specify the `ON` clause to join the Titles and Sales tables together on the partnum column.

```
SELECT bktitle, SUM(qty) quantity
FROM titles LEFT OUTER JOIN sales
ON titles.partnum = sales.partnum
```

   d) Enter the `GROUP BY` clause so that SQL Server will calculate the total quantities sold for each book.

```
SELECT bktitle, SUM(qty) quantity
FROM titles LEFT OUTER JOIN sales
ON titles.partnum = sales.partnum
GROUP BY bktitle
```

   e) Execute the query.
   f) In the **Results** pane, scroll down to view the records. Observe that SQL Server displays NULL for the quantity for books that have no sales.



Results | Messages

	bktitle	qty
1	All Kinds of Knitting	510
2	Basic Home Electronics	310
3	Boating Safety	110
4	Calligraphy	300
5	Chocolate Lovers Cookbook	130
6	Clear Cupboards	NULL
7	Conversational Chinese	NULL
8	Conversational French	NULL

3. Enter a self join query to retrieve a list of potential customers and the customers that referred them to Fuller & Ackerman Publishing.
   a) Edit the existing query to display the customer name and the name of the customer that referred them to Fuller & Ackerman. Include table aliases as part of the column names.

```
SELECT p.custname, r.custname referred_by
```

   b) Add the `FROM` clause that specifies how you want to join the Potential_customers table to itself.

```
SELECT p.custname, r.custname referred_by
FROM potential_customers AS p INNER JOIN potential_customers AS r
```

c) Add the `ON` clause to join the table to itself using the custnum and referredby columns.

```
SELECT p.custname, r.custname referred_by
FROM potential_customers AS p INNER JOIN potential_customers AS r
ON p.referredby = r.custnum
```

d) Execute the query.

e) Observe the results. SQL Server lists the customers and the names of the customers that referred them.

	custname	referred_by
1	Empire Books	The Family Sing Center
2	Book Publishers, Inc.	Hand Loved Craft Supplies
3	BxB Fitness	Book Publishers, Inc.
4	The Family Sing Center	Book Publishers, Inc.
5	Hand Loved Craft Supplies	The Family Sing Center

f) Close the **Query Editor** window without saving the query.

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?

When might you join a table to itself? Why?

Note: Check your LogicalCHOICE Course screen for opportunities to interact with your classmates, peers, and the larger LogicalCHOICE 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.

6 Exporting Query Results

Lesson Time: 30 minutes

Lesson Objectives

In this lesson, you will export query results. You will:

- Generate a text file.
- Generate an XML file.

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.

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® 2012 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, you can access the LearnTO **Export and Use a Text Data File** presentation from the LearnTO tile on the LogicalCHOICE Course screen.

Access the Checklist tile on your LogicalCHOICE course screen for reference information and job aids on How to Save the Query Results

ACTIVITY 6-1
Saving the Query Results

Before You Begin

- On the **Standard** toolbar, select **New Query** to open the **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** database is selected.

Scenario

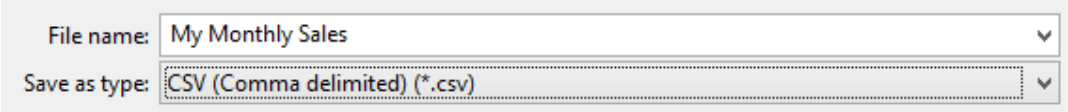
To analyze sales, management of a publishing company requires a list of book titles with their corresponding sales figures sorted by the sales value in descending order. You generate this list every month. As the database keeps changing, this list also varies every month. The sales team would like to be able to analyze the variance in sales on a monthly basis, so they have asked you to provide them with these results so they can use them in Excel. For information on tables and column names in the FullerAckerman database, refer to the table structure in Appendix A.

---

1. List the book titles that have a corresponding entry in the Sales table along with the sum of the sale quantity, sorted by the sum of the sale quantity in descending order.
   a) Enter the `SELECT` statement to display the book title with the alias "Book Title" and the sum of the sales quantity with the alias "Total Sales".
   ```
   SELECT bktitle 'Book Title', SUM(qty) 'Total Sales'
   ```
   b) Enter the `FROM` clause to retrieve data from the Titles table.
   ```
   SELECT bktitle 'Book Title', SUM(qty) 'Total Sales'
   FROM titles
   ```
   c) Enter the `RIGHT OUTER JOIN` keywords followed by the Sales table.
   ```
   SELECT bktitle 'Book Title', SUM(qty) 'Total Sales'
   FROM titles RIGHT OUTER JOIN sales
   ```
   d) Enter the `ON` keyword to match the rows of the two tables based on the part number.
   ```
   SELECT bktitle 'Book Title', SUM(qty) 'Total Sales'
   FROM titles RIGHT OUTER JOIN sales
   ON titles.partnum = sales.partnum
   ```
   e) Enter the `GROUP BY` clause to group the rows based on book titles.
   ```
   SELECT bktitle 'Book Title', SUM(qty) 'Total Sales'
   FROM titles RIGHT OUTER JOIN sales
   ON titles.partnum = sales.partnum
   GROUP BY bktitle
   ```
   f) Specify the `ORDER BY` clause to display the output in descending order of sum of quantity.
   ```
   SELECT bktitle 'Book Title', SUM(qty) 'Total Sales'
   FROM titles RIGHT OUTER JOIN sales
   ON titles.partnum = sales.partnum
   GROUP BY bktitle
   ORDER BY sum(qty) DESC
   ```
   g) Execute the query.
   h) In the **Results** pane, observe that the book titles along with their total sales are displayed.

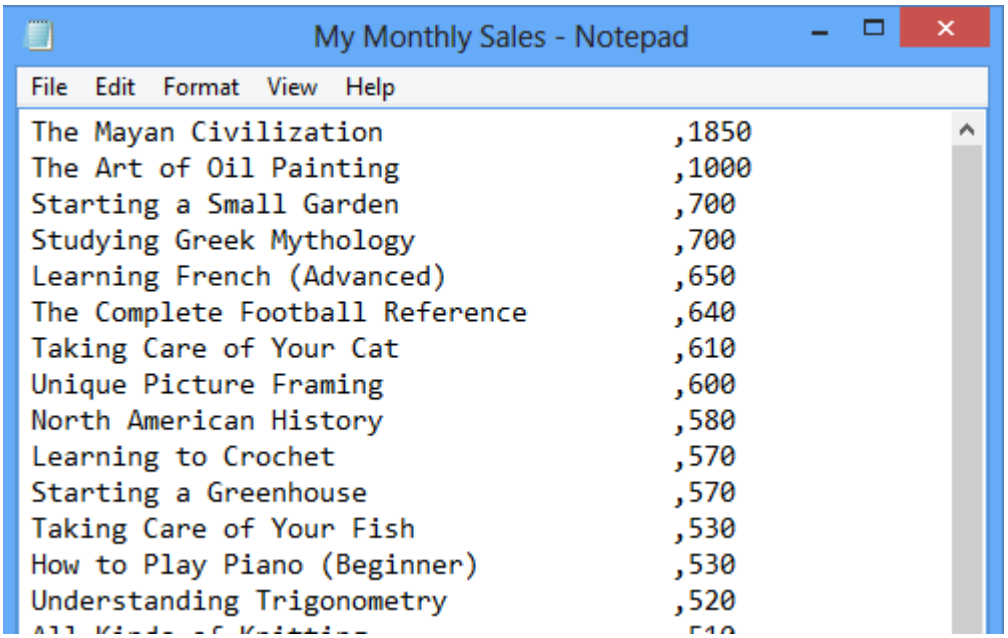
	Book Title	Total Sales
1	The Mayan Civilization	1850
2	The Art of Oil Painting	1000
3	Starting a Small Garden	700
4	Studying Greek Mythology	700
5	Learning French (Advanced)	650
6	The Complete Football Reference	640
7	Taking Care of Your Cat	610
8	Unique Picture Framing	600

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:\094005Data\Exporting Query Results** folder.
   c) In the **File name** text box, type ***My Monthly Sales***
   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) Close the **Query Editor** window without saving the query.

3. Open the **My Monthly Sales.csv** file and view its content.
   a) In **File Explorer**, navigate to the **C:\094005Data\Exporting Query Results** folder.
   b) Double-click **My Monthly Sales.csv** to open the file.
   c) If you are prompted to select an application with which to open the file, select **Notepad**.
   d) Observe the format of the file. The file displays the columns in the result set separated by commas, and the rows in the result set on separate lines.



   e) Close **Notepad** and **File Explorer**.

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 XML.

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.

Mode	*Description*
`RAW`	Generates a single element per row in the rowset with a generic identifier as the element tag.
`AUTO`	Returns query results in a simple XML tree.
`EXPLICIT`	Defines the shape of the resulting XML tree. It requires a specific format for the resulting rowset that is generated.
`PATH`	Generates an element wrapper for each row in a rowset.

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.

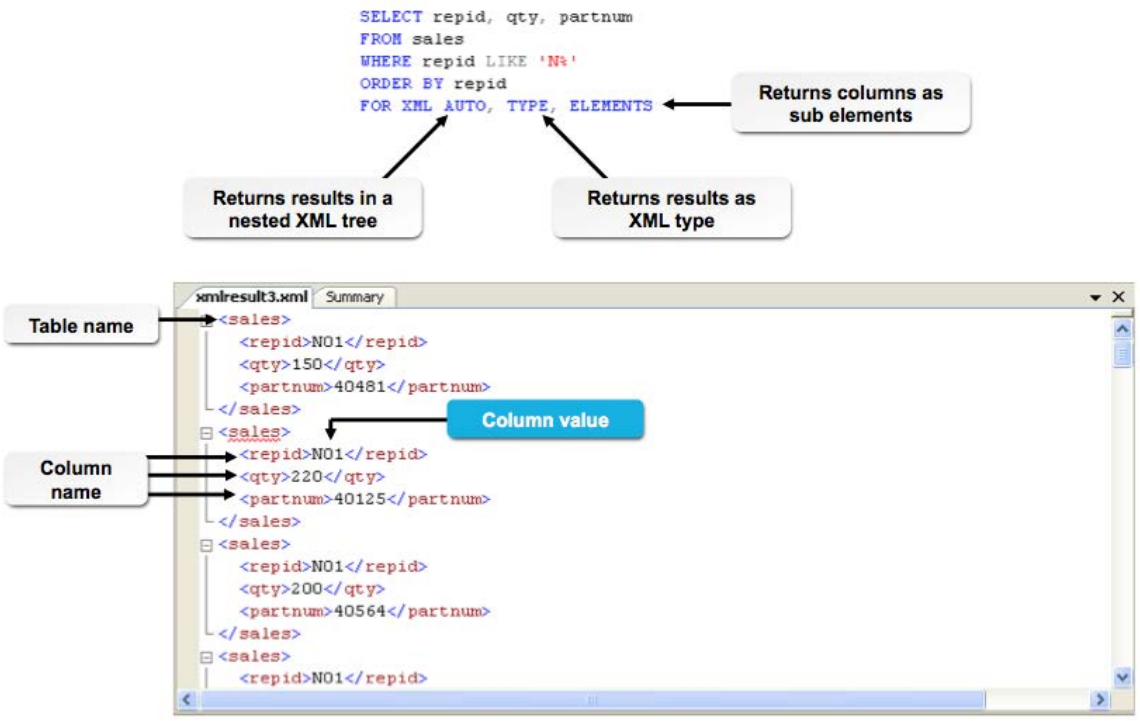


Figure 6-1: A FOR XML clause illustration.

Syntax of the FOR XML Clause

The syntax of the `FOR XML` clause is:

```
SELECT (columnname1,columnname2,....)
FROM table name
WHERE (condition)
ORDER BY (expression)
FOR XML mode
```

	Access the Checklist tile on your LogicalCHOICE 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

- On the **Standard** toolbar, select **New Query** to open the **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** database is selected.

Scenario

The office assistant has asked you for a list of all sales by the sales representatives whose IDs begin with "N" along with the sales quantity and part numbers of the books ordered. She would like the list in order by sales representative ID. She is going to use this information in a web application that requires the XML format for input data. For information on tables and column names in the FullerAckerman database, refer to the table structure in Appendix A.

---

1. Enter the query to list the representative IDs that start with N and their sales quantity along with the part number.
   a) Enter the `SELECT` statement to display the representative ID, sales quantity, and part number columns.
   ```
   SELECT repid, qty, partnum
   ```
   b) Enter the `FROM` clause followed by the Sales table.
   ```
   SELECT repid, qty, partnum
   FROM sales
   ```
   c) Enter the `WHERE` clause to retrieve representative IDs that begin with “N”.
   ```
   SELECT repid, qty, partnum
   FROM sales
   WHERE repid LIKE 'n%'
   ```
   d) Enter the `ORDER BY` clause to order the output based on the representative ID column name.
   ```
   SELECT repid, qty, partnum
   FROM sales
   WHERE repid LIKE 'n%'
   ORDER BY repid
   ```
   e) Type **`FOR XML AUTO, TYPE, ELEMENTS`** to specify the result as an XML type in a nested tree and an element-centric document.
   ```
   SELECT repid, qty, partnum
   FROM sales
   WHERE repid LIKE 'n%'
   ORDER BY repid
   FOR XML AUTO, TYPE, ELEMENTS
   ```
   f) Execute the query.
   g) In the **Results** pane, observe the screen.

2. Save the XML report.
   a) In the **Results** pane, select the hyperlink.
   b) In the **XML Editor** window, observe that all column values are displayed within their respective column names as XML elements.
   c) Select **Save** to save the XML file.

d) In the **Save File As** dialog box, navigate to the **C:\094005Data\Exporting Query Results** folder.
e) In the **File name** text box, type ***My XML Results***
f) In the **Save as type** drop-down list, verify that **XML Files** is selected.
g) Select **Save** to save the XML file.
h) Select **Close** to close the **XML Editor** window.
i) Close the **Query Editor** window without saving the query.

3. Exit SQL Server Management Studio.

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?

How is the ability to generate an XML file beneficial?

Note: Check your LogicalCHOICE Course screen for opportunities to interact with your classmates, peers, and the larger LogicalCHOICE 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 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 LogicalCHOICE Course screen.

A The FullerAckerman Database

The database used in this book, called the FullerAckerman database, is being used in a hypothetical book publishing company called Fuller & Ackerman Publishing. The following tables constitute the FullerAckerman 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

Column Name	*Data Type*	*Length*	*Description*
custnum	nvarchar	5	The customer number for each client. Each customer is assigned a unique customer number.
referredby	nvarchar	5	The customer number of the client who referred this potential customer to Fuller & Ackerman Publishing.
custname	nvarchar	30	The customer's name, or business name.
address	nvarchar	25	The customer's street address.
city	nvarchar	20	The city in which the customer resides.
state	nvarchar	2	The state in which the customer resides.
zipcode	nvarchar	12	The state's zip code.
repid	nvarchar	3	The customer's sales representative's identification number.

The Sales Table

Column Name	*Data Type*	*Length*	*Description*
ordnum	nvarchar	5	The order number for each book sale. Each sales order is assigned a unique order number.

Column Name	Data Type	Length	Description
sldate	smalldatetime	4	The date of sale.
qty	int	4	The number of books ordered.
custnum	nvarchar	5	The customer number for the customer purchasing books.
partnum	nvarchar	5	The part number of the book being ordered.
repid	nvarchar	3	The sales representative responsible for the sale.

The Slspers Table

Column Name	Data Type	Length	Description
repid	nvarchar	3	The identification number for each salesperson. Each sales representative is assigned a unique identification number.
fname	nvarchar	10	The first name of the sales representative.
lname	nvarchar	20	The last name of the sales representative.
commrate	float	8	The sales representative's commission rate.

The Titles Table

Column Name	Data Type	Length	Description
partnum	nvarchar	5	The part number for each book published by Fuller & Ackerman. Each book is assigned a unique part number.
bktitle	nvarchar	40	The title of the book.
devcost	money	8	The development cost of the book.
slprice	money	8	The sale price of the book.
pubdate	smalldatetime	4	The date on which the book was published.

The Obsolete_titles Table

Column Name	Data Type	Length	Description
partnum	nvarchar	5	The part number for each book considered obsolete.
bktitle	nvarchar	40	The title of the obsolete book.
devcost	money	8	The development cost of the obsolete book.
slprice	money	8	The price of the book.
pubdate	smalldatetime	4	The date on which the book was published.

The Potential_customers Table

Column Name	Data Type	Length	Description
custnum	nvarchar	5	A unique number assigned for the potential customer.
referredby	nvarchar	5	The customer number of the client who referred this potential customer to Fuller & Ackerman Publishing.
custname	nvarchar	30	The potential customer's name, or business name.
address	nvarchar	25	The potential customer's street address.
city	nvarchar	20	The city in which the potential customer resides.
state	nvarchar	2	The state in which the potential customer resides.
zipcode	nvarchar	12	The potential customer's zip code.
repid	nvarchar	3	The identification number of the sales representative in the potential customer's area.

Lesson Labs

Lesson labs are provided for certain lessons as additional learning resources for this course. Lesson labs are developed for selected lessons within a course in cases when they seem most instructionally useful as well as technically feasible. In general, labs 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 labs are appropriate for you to perform, and at what point during the class. If you do not perform the labs in class, your instructor can tell you if you can perform them independently as self-study, and if there are any special setup requirements.

Lesson Lab 1-1
Executing a Simple Query

Activity Time: 15 minutes

Before You Begin

- Launch SQL Server Management Studio and connect to the server.
- On the **Standard** toolbar, select **New Query** to open the **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** database is selected.

Scenario

You have been hired at Fuller & Ackerman Publishing. Information about customers, book titles, sales representatives, and sales is stored in separate tables in the FullerAckerman database. Your job is to query this 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. Launch **SQL Server Management Studio** and connect to the server.

2. Select the **FullerAckerman** database.

3. In the **Object Explorer** pane, explore the tables included in the FullerAckerman database and their structures.

4. Write and execute `SELECT` statements to view all columns and all rows in each of the Customers, Titles, Slspers, and Sales tables.

5. Write and execute a query that retrieves these columns from the Customers table: custname, address, city, state, and zipcode.

6. Write and execute a query that retrieves the bktitle, partnum, and slprice columns from the Titles table.

7. Write and execute a query that retrieves the repid, fname, and lname columns from the Slspers table.

8. Write and execute a query to retrieve the ordnum, partnum, and qty columns from the Sales table.

9. Close the **Query Editor** window without saving your changes.

Lesson Lab 2-1
Performing a Conditional Search

Activity Time: 15 minutes

Before You Begin

- On the **Standard** toolbar, select **New Query** to open the **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** database is selected.

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.

1. List the book title, part number, and sale price of books that are priced between $10 and $30. Execute the query.
2. List the sales details of representatives whose IDs start with either E or N. Execute the query.
3. Enter another condition to the query in step 2 to list the sales details if the sale quantity is greater than 400, and execute the query.
4. Close the **Query Editor** window without saving your changes.

Lesson Lab 3-1
Working with Functions

Activity Time: 15 minutes

Before You Begin

- On the **Standard** toolbar, select **New Query** to open the **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** database is selected.

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 by them during the first six months of 2013. 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.

---

1. List all columns from the Sales table that occurred during the first six months of 2013, and execute the query.

2. 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.

3. 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 2013, and execute the query.

4. List the details of customers who have a four-digit customer ID.

5. Close the **Query Editor** window without saving changes.

Lesson Lab 4-1
Organizing Data

Activity Time: 15 minutes

Before You Begin

- On the **Standard** toolbar, select **New Query** to open the **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** database is selected.

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 above, sorted in descending order of total sales quantity.

---

1. List the part number and sum of quantity from the Sales table.
2. Eliminate any NULL values in the quantity, and group the list based on the books.
3. Sort the list based on the sum of quantity in descending order and execute the query.
4. Write a new query to list the representatives who have sold more than 2000 books, eliminating NULL values.
5. Sort the list based on the sum of quantity in descending order and execute the query.
6. Close the **Query Editor** window without saving changes.

Lesson Lab 5-1
Retrieving Data from Multiple Tables

Activity Time: 15 minutes

Before You Begin

- On the **Standard** toolbar, select **New Query** to open a **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** database is selected.

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.

---

1. 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.

2. Write a query that lists sales by order number, customer name, book title, sales representative name, and quantity sold.

3. Close the **Query Editor** window without saving changes.

Lesson Lab 6-1
Exporting Query Results

Activity Time: 15 minutes

Before You Begin

- On the **Standard** toolbar, select **New Query** to open the **Query Editor** window.
- On the **SQL Editor** toolbar, from the **Available Databases** drop-down list, ensure that the **FullerAckerman** database is selected.

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.

---

1. Display book titles along with their respective development cost and sale price.
2. 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***

Solutions

ACTIVITY 1-3: Saving a Query

2. **True or False? A saved query is automatically named after the database table name being queried.**

   ☐ True

   ☑ False

ACTIVITY 1-4: Modifying a Saved Query

2. **Based on the scenario, which columns should you include in your SELECT statement?**

   A: Because the sales representative does not need information about the book's development cost or publication date, you should exclude those columns. The columns you should include in the `SELECT` statement are partnum, bktitle, and slprice.

ACTIVITY 2-3: Searching for Rows Using Multiple Conditions

3. **Must you use parentheses when you specify more than two conditions in the WHERE clause of a SELECT statement?**

   ○ No. You do not need to use parentheses.

   ◉ You use parentheses when the operators used in conditions do not follow the operator processing hierarchy.

   ○ Yes. You must always use parentheses.

   ○ You use parentheses to enhance readability.

ACTIVITY 3-1: Performing Date Calculations

2. **True or False? You can use the DATEPART function to extract the year from the published date of the book.**

- ☑ True
- ☐ False

ACTIVITY 4-2: Ranking Data

1. **Which ranking function should you use to partition rows into a specified number of groups?**
   - ○ ROW_NUMBER ()
   - ○ DENSE_RANK ()
   - ◉ NTILE ()
   - ○ RANK ()

ACTIVITY 4-3: Grouping Data

1. **Which columns need to be listed in the SELECT statement in order to list the total quantity of books sold by the representative to a customer?**
   - ○ AVG(qty), custnum, repid
   - ○ custnum, repid
   - ◉ SUM(qty) total_qty, custnum, repid
   - ○ SUM(qty)/COUNT(qty), custnum, repid

2. **True or False? The columns used in the SELECT clause that do not have the aggregate function must also be used in the GROUP BY clause.**
   - ☑ True
   - ☐ False

ACTIVITY 4-4: Filtering Grouped Data

1. **True or False? To retrieve books that have sold more than 500 copies, you need to calculate the sum of the sales quantity of books that have the same part number and that sold greater than five hundred.**
   - ☑ True
   - ☐ False

ACTIVITY 4-5: Summarizing Grouped Data

1. **Which operator enables SQL Server to summarize groups in hierarchical order, from the lowest level in the group to the highest?**
   - ○ GROUP WITH
   - ◉ ROLLUP
   - ○ WITH CUBE

- WITH ROLLUP

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.

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 value 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 value in a column.

SELECT statement
A 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 a 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.

tiered architecture
A database architecture type that helps in the easy usage and maintenance of a database by providing different views to different levels of users.

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

T

U

W

X

094005S rev 1.0
ISBN-13 978-1-4246-2197-2
ISBN-10 1-4246-2197-6