

Webucator's Free SQL Tutorial

Lesson: Simple SELECTs

Welcome to our free SQL tutorial. This tutorial is based on Webucator's [Introduction to SQL Training course](https://www.webucator.com/database-training/course/introduction-sql-training-using-standard-sql.cfm) (<https://www.webucator.com/database-training/course/introduction-sql-training-using-standard-sql.cfm>).

The **SELECT** statement is used to retrieve data from tables. **SELECT** statements can be used to perform simple tasks such as retrieving records from a single table or complicated tasks such as retrieving data from multiple tables with record grouping and sorting. In this lesson, we will look at several of the more basic ways to retrieve data from a single table.

Lesson Goals

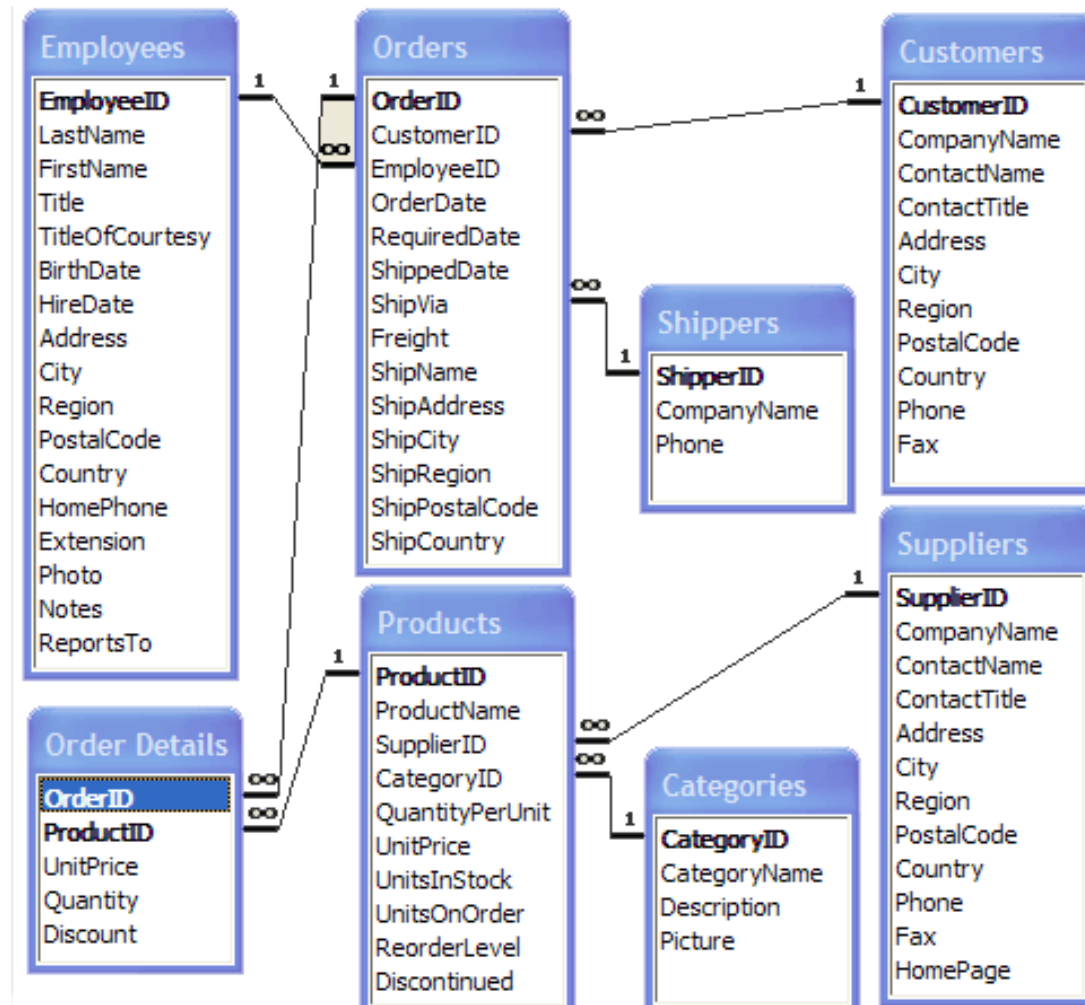
- Learn about the database we'll be using in class.
- Learn to comment your SQL code.
- Learn to understand SQL syntax.
- Learn to select all rows from a table.
- Learn to sort record sets.
- Learn to filter records.

Introduction to the Northwind Database

The Northwind database is a sample database used by Microsoft to demonstrate the features of some of its products, including SQL Server and Microsoft Access. The database contains the sales data for Northwind Traders, a fictitious specialty foods export-import company.

Although the code taught in this class is not specific to Microsoft products, we use the Northwind database for many of our examples because many people are already familiar with it and because there are many resources for related learning that make use of the same database.

The diagram below shows the table structure of the Northwind database.



The Northwind database has additional tables, but we will only be using the ones shown above. In this lesson, we will explore some of these tables.

Some Basics

Comments

The standard SQL comment is two hyphens (--). However, some databases use other forms of comments as shown in the table below.

SQL Comments

	--	#	/* */
Example	-- Comment	# Comment	/* Comment */
ANSI	YES	NO	NO
SQL Server	YES	NO	YES
Oracle	YES	NO	YES
MySQL	YES	YES	YES

The code sample below shows some sample comments.

Code Sample:

SimpleSelects/Demos/Comments.sql

```
-- Single-line comment
/*
    Multi-line comment used in:
        -SQL Server
        -Oracle
        -MySQL
*/
```

Whitespace and Semi-colons

Whitespace is ignored in SQL statements. Multiple statements are separated with semi-colons. The two statements in the sample below are equally valid.

Code Sample:

SimpleSelects/Demos/WhiteSpace.sql

```
SELECT * FROM Employees;

SELECT *
FROM Employees;
```

Case Sensitivity

SQL is not case sensitive. It is common practice to write reserved words in all capital letters. User-defined names, such as table names and column names may or may not be case sensitive depending on the operating system used.

SELECTing All Columns in All Rows

The following syntax is used to retrieve all columns in all rows of a table.

Syntax

```
SELECT table.*  
FROM table;
```

-- OR

```
SELECT *  
FROM table;
```

Code Sample:

SimpleSelects/Demos/SelectAll.sql

```
--Retrieve all columns in the Region table
```

The above **SELECT** statement will return the following results:

	RegionID	RegionDescription
1	1	Eastern
2	2	Western
3	3	Northern
4	4	Southern

As you can see, the **Region** table has only two columns, **RegionID** and **RegionDescription**, and four rows.

Exploring the Tables

Duration: 10 to 20 minutes.

In this exercise, you will explore all the data in the Northwind database by selecting all the rows of each of the tables.

1. Using your SQL editor, execute the necessary SQL queries to select all columns of all rows from the tables below.

2. The number of records that should be returned is indicated in parentheses next to the table name.

1. Categories (8)
2. Customers (91)
3. Employees (9)
4. Orders (830)
5. Products (77)
6. Shippers (3)
7. Suppliers (29)

Solution:

SimpleSelects/Solutions/SelectAll.sql

```
SELECT * FROM Categories;  
SELECT * FROM Customers;  
SELECT * FROM Employees;  
SELECT * FROM Orders;  
SELECT * FROM Products;  
SELECT * FROM Shippers;  
SELECT * FROM Suppliers;
```

SELECTing Specific Columns

The following syntax is used to retrieve specific columns in all rows of a table.

Syntax

```
SELECT table_name.column_name, table_name.column_name  
FROM table;  
  
-- OR  
  
SELECT column, column  
FROM table;
```

Code Sample:

SimpleSelects/Demos/SelectCols.sql

```
/*
Select the FirstName and LastName columns from the Employees table.
*/
```

The above **SELECT** statement will return the following results:

	FirstName	LastName
1	Nancy	Davolio
2	Andrew	Fuller
3	Janet	Leverling
4	Margaret	Peacock
5	Steven	Buchanan
6	Michael	Suyama
7	Robert	King
8	Laura	Callahan
9	Anne	Dodsworth

SELECTing Specific Columns

Duration: 5 to 15 minutes.

In this exercise, you will practice selecting specific columns from tables in the Northwind database.

1. Select **CategoryName** and **Description** from the **Categories** table.
2. Select **ContactName**, **CompanyName**, **ContactTitle** and **Phone** from the **Customers** table.
3. Select **EmployeeID**, **Title**, **FirstName**, **LastName**, and **Region** from the **Employees** table.
4. Select **RegionID** and **RegionDescription** from the **Region** table.
5. Select **CompanyName**, **Fax**, **Phone** and **HomePage** from the **Suppliers** table.

Solution:

SimpleSelects/Solutions/SelectCols.sql

```
SELECT CategoryName, Description
FROM Categories;
```

```
SELECT ContactName, CompanyName, ContactTitle, Phone
FROM Customers;
```

```
SELECT EmployeeID, Title, FirstName, LastName, Region
FROM Employees;
```

```
SELECT RegionID, RegionDescription
FROM Region;
```

```
SELECT CompanyName, Fax, Phone, HomePage
FROM Suppliers;
```

Sorting Records

The **ORDER BY** clause of the **SELECT** statement is used to sort records.

Sorting By a Single Column

To sort by a single column, simply name that column in the **ORDER BY** clause.

Syntax


```
SELECT column, column
FROM table
ORDER BY column;
```

Note that columns in the **ORDER BY** clause do not have to appear in the **SELECT** clause.

Code Sample:

SimpleSelects/Demos/OrderBy1.sql

```
/*          Select the FirstName and LastName columns from the Employees table.
           Sort by LastName.
*/
```



The above **SELECT** statement will return the following results:

	FirstName	LastName
1	Steven	Buchanan
2	Laura	Callahan
3	Nancy	Davolio
4	Anne	Dodsworth
5	Andrew	Fuller
6	Robert	King
7	Janet	Leverling
8	Margaret	Peacock
9	Michael	Suyama

Sorting By Multiple Columns

To sort by multiple columns, comma-delimit the column names in the ORDER BY clause.


Syntax

```
SELECT column, column  
FROM table  
ORDER BY column, column;
```

Code Sample:

SimpleSelects/Demos/OrderBy2.sql

```
/*  
Select the Title, FirstName and LastName columns from the Employees table.  
Sort first by Title and then by LastName.  
*/
```



The above **SELECT** statement will return the following results:

	Title	FirstName	LastName
1	Inside Sales Coordinator	Laura	Callahan
2	Sales Manager	Steven	Buchanan
3	Sales Representative	Nancy	Davolio
4	Sales Representative	Anne	Dodsworth
5	Sales Representative	Robert	King
6	Sales Representative	Janet	Leverling
7	Sales Representative	Margaret	Peacock
8	Sales Representative	Michael	Suyama
9	Vice President, Sales	Andrew	Fuller

Ascending and Descending Sorts

By default, when an **ORDER BY** clause is used, records are sorted in ascending order. This can be explicitly specified with the **ASC** keyword. To sort records in descending order, use the **DESC** keyword.

Syntax

```
SELECT column, column
FROM table
ORDER BY column DESC, column ASC;
```

Code Sample:

SimpleSelects/Demos/OrderBy4.sql

```
/*
    Select the Title, FirstName and LastName columns from the Employees
    Sort first by Title in ascending order and then by LastName
    in descending order.
*/
```



The above **SELECT** statement will return the following results:

	Title	FirstName	LastName
1	Inside Sales Coordinator	Laura	Callahan
2	Sales Manager	Steven	Buchanan
3	Sales Representative	Michael	Suyama
4	Sales Representative	Margaret	Peacock
5	Sales Representative	Janet	Leverling
6	Sales Representative	Robert	King
7	Sales Representative	Anne	Dodsworth
8	Sales Representative	Nancy	Davolio
9	Vice President, Sales	Andrew	Fuller

Sorting Results

Duration: 5 to 15 minutes.

In this exercise, you will practice sorting results in **SELECT** statements.

1. Select **CategoryName** and **Description** from the **Categories** table sorted by **CategoryName** .
2. Select **ContactName** , **CompanyName** , **ContactTitle** , and **Phone** from the **Customers** table sorted by Phone.
3. Create a report showing employees' first and last names and hire dates sorted from newest to oldest employee.
4. Create a report showing Northwind's orders sorted by **Freight** from most expensive to cheapest. Show **OrderID** , **OrderDate** , **ShippedDate** , **CustomerID** , and **Freight** .
5. Select **CompanyName** , **Fax** , **Phone** , **HomePage** and **Country** from the **Suppliers** table sorted by **Country** in descending order and then by **CompanyName** in ascending order.
6. Create a list of employees showing title, first name, and last name. Sort by **Title** in ascending order and then by **LastName** in descending order.

Solution:

SimpleSelects/Solutions/Sorting.sql

```
SELECT CategoryName, Description
FROM Categories
ORDER BY CategoryName;
```

```
SELECT ContactName, CompanyName, ContactTitle, Phone
FROM Customers
ORDER BY Phone;
```

```
SELECT FirstName, LastName, HireDate
FROM Employees
ORDER BY HireDate DESC;
```

```
SELECT OrderID, OrderDate, ShippedDate, CustomerID, Freight
FROM Orders
ORDER BY Freight DESC;
```

```
SELECT CompanyName, Fax, Phone, HomePage, Country
FROM Suppliers
ORDER BY Country DESC, CompanyName;
```

```
SELECT Title, FirstName, LastName
FROM Employees
ORDER BY Title ASC, LastName DESC;
```

The WHERE Clause and Operator Symbols

The **WHERE** clause is used to retrieve specific rows from tables. The **WHERE** clause can contain one or more conditions that specify which rows should be returned.

Syntax

```
SELECT column, column
FROM table
WHERE conditions;
```

The following table shows the symbolic operators used in **WHERE** conditions.

SQL Symbol Operators

Operator	Description
----------	-------------

=	Equals
<>	Not Equal
>	Greater Than

- < Less Than
- >= Greater Than or Equal To
- <= Less Than or Equal To

Note that non-numeric values (e.g, dates and strings) in the **WHERE** clause must be enclosed in single quotes. Examples are shown below.

Checking for Equality

Code Sample:

SimpleSelects/Demos/Where-Equal.sql

```
/*
Create a report showing the title and the first and last name
of all sales representatives.
*/
```



The above **SELECT** statement will return the following results:

	Title	FirstName	LastName
1	Sales Representative	Nancy	Davolio
2	Sales Representative	Janet	Leverling
3	Sales Representative	Margaret	Peacock
4	Sales Representative	Michael	Suyama
5	Sales Representative	Robert	King
6	Sales Representative	Anne	Dodsworth

Checking for Inequality

Code Sample:

SimpleSelects/Demos/Where-NotEqual.sql

```
/*  
Create a report showing the first and last name of all employees  
excluding sales representatives.  
*/
```

The above **SELECT** statement will return the following results:

	FirstName	LastName
1	Andrew	Fuller
2	Steven	Buchanan
3	Laura	Callahan

Checking for Greater or Less Than

The less than (<) and greater than (>) signs are used to compare numbers, dates, and strings.

Code Sample:

SimpleSelects/Demos/Where-GreaterThanOrEqual.sql

```
/*  
Create a report showing the first and last name of all employees whose  
last names start with a letter in the last half of the alphabet.  
*/
```

The above **SELECT** statement will return the following results:

	FirstName	LastName
1	Margaret	Peacock
2	Michael	Suyama

Checking for NULL

When a field in a row has no value, it is said to be **NULL** . This is not the same as having an empty string. Rather, it means that the field contains no value at all. When checking to see if a field is **NULL** , you cannot use the equals sign (=); rather, use the IS **NULL** expression.

Code Sample:

SimpleSelects/Demos/Where-Null.sql

```
/*  
Create a report showing the first and last names of  
all employees whose region is unspecified.  
*/  
  
SELECT FirstName, LastName  
FROM Employees  
WHERE Region IS NULL;
```

The above SELECT statement will return the following results:

	FirstName	LastName
1	Steven	Buchanan
2	Michael	Suyama
3	Robert	King
4	Anne	Dodsworth

Code Sample:

SimpleSelects/Demos/Where-NotNull.sql

```
/*  
Create a report showing the first and last names of all  
employees who have a region specified.  
*/
```



The above SELECT statement will return the following results:

	FirstName	LastName
1	Nancy	Davolio
2	Andrew	Fuller
3	Janet	Leverling
4	Margaret	Peacock
5	Laura	Callahan

WHERE and ORDER BY

When using **WHERE** and **ORDER BY** together, the **WHERE** clause must come before the **ORDER BY** clause.

Code Sample:

SimpleSelects/Demos/Where-OrderBy.sql

```
/*
Create a report showing the first and last name of all employees whose
last names start with a letter in the last half of the alphabet.
Sort by LastName in descending order.
*/

SELECT FirstName, LastName
FROM Employees
WHERE LastName >= 'N'
ORDER BY LastName DESC;
```

The above **SELECT** statement will return the following results:

	FirstName	LastName
1	Michael	Suyama
2	Margaret	Peacock

Using the WHERE clause to check for equality or inequality

Duration: 5 to 15 minutes.

In this exercise, you will practice using the **WHERE** clause to check for equality and inequality.

1. Create a report showing all the company names and contact names of Northwind's customers in Buenos Aires.
2. Create a report showing the product name, unit price and quantity per unit of all products that are out of stock.
3. Create a report showing the order date, shipped date, customer id, and freight of all orders placed on May 19, 1997.
 - *Oracle users will have to use following date format: 'dd-mmm-yyyy' (e.g, '19-May-1997').*
 - *MySQL users will have to use following date format: 'yyyy-mm-dd' (e.g, '1997-05-19').*
4. Create a report showing the first name, last name, and country of all employees not in the United States.

Solution:

SimpleSelects/Solutions/EqualityAndInequality.sql

```
SELECT CompanyName, ContactName
FROM Customers
WHERE City = 'Buenos Aires';
```

```
SELECT ProductName, UnitPrice, QuantityPerUnit
FROM Products
WHERE UnitsInStock=0;
```

/*****

For the third problem, both of the solutions below will work in SQL Server

Oracle Solution

*****/

```
SELECT OrderDate, ShippedDate, CustomerID, Freight
FROM Orders
WHERE OrderDate = '19-May-1997';
```

/*****

MySQL Solution

*****/

```
SELECT OrderDate, ShippedDate, CustomerID, Freight
FROM Orders
WHERE OrderDate = '1997-05-19';
```

```
SELECT FirstName, LastName, Country
FROM Employees
WHERE Country <> 'USA';
```

Using the WHERE clause to check for greater or less than

Duration: 5 to 15 minutes.

In this exercise, you will practice using the **WHERE** clause to check for values greater than or less than a specified value.

1. Create a report that shows the employee id, order id, customer id, required date, and shipped date of all orders that were shipped later than they were required.
2. Create a report that shows the city, company name, and contact name of all customers who are in cities that begin with "A" or "B."
3. Create a report that shows all orders that have a freight cost of more than \$500.00.

4. Create a report that shows the product name, units in stock, units on order, and reorder level of all products that are up for reorder.

Solution:

SimpleSelects/Solutions/GreaterThanLessThan.sql

```
SELECT EmployeeID, OrderID, CustomerID, RequiredDate, ShippedDate
FROM Orders
WHERE ShippedDate > RequiredDate;
```

```
SELECT City, CompanyName, ContactName
FROM Customers
WHERE City < 'C';
```

```
SELECT OrderID, OrderDate, Freight
FROM Orders
WHERE Freight > 500;
```

```
SELECT ProductName, UnitsInStock, UnitsOnOrder, ReorderLevel
FROM Products
WHERE UnitsInStock <= ReorderLevel;
```

Checking for NULL

Duration: 5 to 15 minutes.

In this exercise, you will practice selecting records with fields that have **NULL** values.

1. Create a report that shows the company name, contact name and fax number of all customers that have a fax number.
2. Create a report that shows the first and last name of all employees who do not report to anybody.

Solution:

SimpleSelects/Solutions/Null.sql

```
SELECT CompanyName, ContactName, Fax
FROM Customers
WHERE Fax IS NOT NULL;
```

```
SELECT FirstName, LastName
FROM Employees
WHERE ReportsTo IS NULL;
```

Using WHERE and ORDER BY Together

Duration: 5 to 15 minutes.

In this exercise, you will practice writing **SELECT** statements that use both **WHERE** and **ORDER BY**.

1. Create a report that shows the company name, contact name and fax number of all customers that have a fax number. Sort by company name.
2. Create a report that shows the city, company name, and contact name of all customers who are in cities that begin with "A" or "B." Sort by contact name in descending order.

Solution:

SimpleSelects/Solutions/Where-OrderBy.sql

```
SELECT CompanyName, ContactName, Fax
FROM Customers
WHERE Fax IS NOT NULL
ORDER BY CompanyName;
```

```
SELECT City, CompanyName, ContactName
FROM Customers
WHERE City < 'C'
ORDER BY ContactName DESC;
```

The WHERE Clause and Operator Words

The following table shows the word operators used in WHERE conditions.

SQL Word Operators

OperatorDescription

BETWEEN	Returns values in an inclusive range
IN	Returns values in a specified subset
LIKE	Returns values that match a simple pattern
NOT	Negates an operation

The BETWEEN Operator

The **BETWEEN** operator is used to check if field values are within a specified inclusive range.

Code Sample:

SimpleSelects/Demos/Where-Between.sql

```
/*  
Create a report showing the first and last name of all employees  
whose last names start with a letter between "J" and "M".  
*/
```

-- The above SELECT statement is the same as the one below.

```
SELECT FirstName, LastName  
FROM Employees  
WHERE LastName >= 'J' AND LastName <= 'M';
```

The above SELECT statements will both return the following results:

	FirstName	LastName
1	Robert	King
2	Janet	Leverling

Note that a person with the last name "M" would be included in this report, but a person's whose last name starts with "M" would not.

The IN Operator

The **IN** operator is used to check if field values are included in a specified comma-delimited list.

Code Sample:

SimpleSelects/Demos/Where-In.sql

```
/*
Create a report showing the title of courtesy and the first and
last name of all employees whose title of courtesy is "Mrs." or "Ms.".
*/
```

-- The above SELECT statement is the same as the one below

```
SELECT TitleOfCourtesy, FirstName, LastName
FROM Employees
WHERE TitleOfCourtesy = 'Ms.' OR TitleOfCourtesy = 'Mrs.';
```

The above SELECT statements will both return the following results:

	TitleOfCourtesy	FirstName	LastName
1	Ms.	Nancy	Davolio
2	Ms.	Janet	Leverling
3	Mrs.	Margaret	Peacock
4	Ms.	Laura	Callahan
5	Ms.	Anne	Dodsworth

The LIKE Operator

The **LIKE** operator is used to check if field values match a specified pattern.

The Percent Sign (%)

The percent sign (**%**) is used to match any zero or more characters.

Code Sample:

SimpleSelects/Demos/Where-Like1.sql

```
/*
Create a report showing the title of courtesy and the first
and last name of all employees whose title of courtesy begins with "M".
*/
```

The above **SELECT** statement will return the following results:

	TitleOfCourtesy	FirstName	LastName
1	Ms.	Nancy	Davolio
2	Ms.	Janet	Leverling
3	Mrs.	Margaret	Peacock
4	Mr.	Steven	Buchanan
5	Mr.	Michael	Suyama
6	Mr.	Robert	King
7	Ms.	Laura	Callahan
8	Ms.	Anne	Dodsworth

The Underscore (_)

The underscore () is used to match any single character.

Code Sample:

SimpleSelects/Demos/Where-Like2.sql

```
/*
Create a report showing the title of courtesy and the first and
last name of all employees whose title of courtesy begins with "M" and
is followed by any character and a period (.).
*/

SELECT TitleOfCourtesy, FirstName, LastName
FROM Employees
WHERE TitleOfCourtesy LIKE 'M_.';
```

The above **SELECT** statement will return the following results:

	TitleOfCourtesy	FirstName	LastName
1	Ms.	Nancy	Davolio
2	Ms.	Janet	Leverling
3	Mr.	Steven	Buchanan
4	Mr.	Michael	Suyama
5	Mr.	Robert	King
6	Ms.	Laura	Callahan
7	Ms.	Anne	Dodsworth

Wildcards and Performance

Using wildcards can slow down performance, especially if they are used at the beginning of a pattern. You should use them sparingly.

The NOT Operator

The **NOT** operator is used to negate an operation.

Code Sample:

SimpleSelects/Demos/Where-Not.sql

```
/*
Create a report showing the title of courtesy and the first and last name
of all employees whose title of courtesy is not "Ms." or "Mrs.".
*/

SELECT TitleOfCourtesy, FirstName, LastName
FROM Employees
WHERE NOT TitleOfCourtesy IN ('Ms.', 'Mrs.');
```

The above **SELECT** statement will return the following results:

	TitleOfCourtesy	FirstName	LastName
1	Dr.	Andrew	Fuller
2	Mr.	Steven	Buchanan
3	Mr.	Michael	Suyama
4	Mr.	Robert	King

More SELECTs with WHERE

Duration: 5 to 15 minutes.

In this exercise, you will practice writing **SELECT** statements that use **WHERE** with word operators.

1. Create a report that shows the first and last names and birth date of all employees born in the 1950s.
2. Create a report that shows the product name and supplier id for all products supplied by Exotic Liquids, Grandma Kelly's Homestead, and Tokyo Traders. *Hints:* You will need to first do a separate **SELECT** on the Suppliers table to find the supplier ids of these three companies. You will need to escape the apostrophe in "Grandma Kelly's Homestead" in the first separate **SELECT**. To do so, place another apostrophe in front of it.
3. Create a report that shows the shipping postal code, order id, and order date for all orders with a ship postal code beginning with "02389".
4. Create a report that shows the contact name and title and the company name for all customers whose contact title does not contain the word "Sales".

Solution:

SimpleSelects/Solutions/WordOperators.sql

```
/******
```

For the first problem, both of the solutions below will work in SQL Server

Oracle Solution

```
*****/
```

```
SELECT FirstName, LastName, BirthDate
FROM Employees
WHERE BirthDate BETWEEN '1-Jan-1950' AND '31-Dec-1959';
```

```
/******
```

MySQL Solution

```
*****/
```

```
SELECT FirstName, LastName, BirthDate
FROM Employees
WHERE BirthDate BETWEEN '1950-01-01' AND '1959-12-31 23:59:59';
```

```
SELECT ProductName, SupplierID
FROM Products
WHERE SupplierID IN (1,3,4);
```

```
SELECT ShipPostalCode, OrderID, OrderDate
FROM Orders
WHERE ShipPostalCode LIKE '02389%';
```

```
SELECT ContactName, ContactTitle, CompanyName
FROM Customers
WHERE NOT ContactTitle LIKE '%Sales%';
```

Checking Multiple Conditions


AND

AND can be used in a **WHERE** clause to find records that match more than one condition.

Code Sample:

SimpleSelects/Demos/Where-And.sql

```
/*
Create a report showing the first and last name of all
sales representatives whose title of courtesy is "Mr.".
*/
```



The above SELECT statement will return the following results:

	FirstName	LastName
1	Michael	Suyama
2	Robert	King


OR

OR can be used in a **WHERE** clause to find records that match at least one of several conditions.

Code Sample:

SimpleSelects/Demos/Where-Or.sql

```
/*
        Create a report showing the first and last name and the city of all
        employees who are from Seattle or Redmond.
*/
```



The above SELECT statement will return the following results:

	FirstName	LastName	City
1	Nancy	Davolio	Seattle
2	Margaret	Peacock	Redmond
3	Laura	Callahan	Seattle

Order of Evaluation

By default, SQL processes **AND** operators before it processes **OR** operators. To illustrate how this works, take a look at the following example.

Code Sample:

SimpleSelects/Demos/Where-AndOrPrecedence.sql

```
/*  
    Create a report showing the first and last name of all sales  
    representatives who are from Seattle or Redmond.  
*/
```

The above SELECT statement will return the following results:

	FirstName	LastName	City	Title
1	Nancy	Davolio	Seattle	Sales Representative
2	Margaret	Peacock	Redmond	Sales Representative
3	Laura	Callahan	Seattle	Inside Sales Coordinator

Notice that Laura Callahan is returned by the query even though she is not a sales representative. This is because this query is looking for employees from Seattle OR sales representatives from Redmond.

This can be fixed by putting the OR portion of the clause in parentheses.

Code Sample:

SimpleSelects/Demos/Where-AndOrPrecedence2.sql

```
/*  
    Create a report showing the first and last name of all sales  
    representatives who are from Seattle or Redmond.  
*/
```

The parentheses specify that the OR portion of the clause should be evaluated first, so the above SELECT statement will return the same results minus Laura Callahan.

	FirstName	LastName	City	Title
1	Nancy	Davolio	Seattle	Sales Representative
2	Margaret	Peacock	Redmond	Sales Representative

If only to make the code more readable, it's a good idea to use parentheses whenever the order of precedence might appear ambiguous.

Writing SELECTs with Multiple Conditions

Duration: 5 to 15 minutes.

In this exercise, you will practice writing **SELECT** statements that filter records based on multiple conditions.

1. Create a report that shows the first and last names and cities of employees from cities other than Seattle in the state of Washington.
2. Create a report that shows the company name, contact title, city and country of all customers in Mexico or in any city in Spain except Madrid.

Solution:

SimpleSelects/Solutions/MultipleConditions.sql

```
SELECT FirstName, LastName, City
FROM Employees
WHERE Region = 'WA' AND City <> 'Seattle';

SELECT CompanyName, ContactTitle, City, Country
FROM Customers
WHERE Country = 'Mexico' OR (Country = 'Spain'
    AND City <> 'Madrid');
```

Next Lesson: Advanced SELECTs → (<https://www.webucator.com/tutorial/learn-sql/advanced-selects.cfm>)

WEBUCATOR DELIVERS INSTRUCTOR-LED AND SELF-PACED TRAINING

Microsoft Training(<https://www.webucator.com/microsoft-training/index.cfm>)

Database Training(<https://www.webucator.com/database-training/index.cfm>)

Web Development Training
(<https://www.webucator.com/webdev-training/index.cfm>)

Java Training(<https://www.webucator.com/java-training/index.cfm>)

Adobe Training(<https://www.webucator.com/adobe-training/index.cfm>)

Programming Training

(<https://www.webucator.com/programming-training/index.cfm>)

Big Data Training(<https://www.webucator.com/big-data-training/index.cfm>)

Cloud Training(<https://www.webucator.com/cloud-training/index.cfm>)

Business Skills Training

(<https://www.webucator.com/business-skills-training/index.cfm>)

Web Application Server Training

(<https://www.webucator.com/web-application-servers-training/index.cfm>)



(<https://www.facebook.com/webucator>) (<https://www.linkedin.com/company/webucator>) (<https://twitter.com/webucator>) (<mailto:info@webucator.com>)

© 2004-2020 Webucator, Inc. All Rights Reserved.