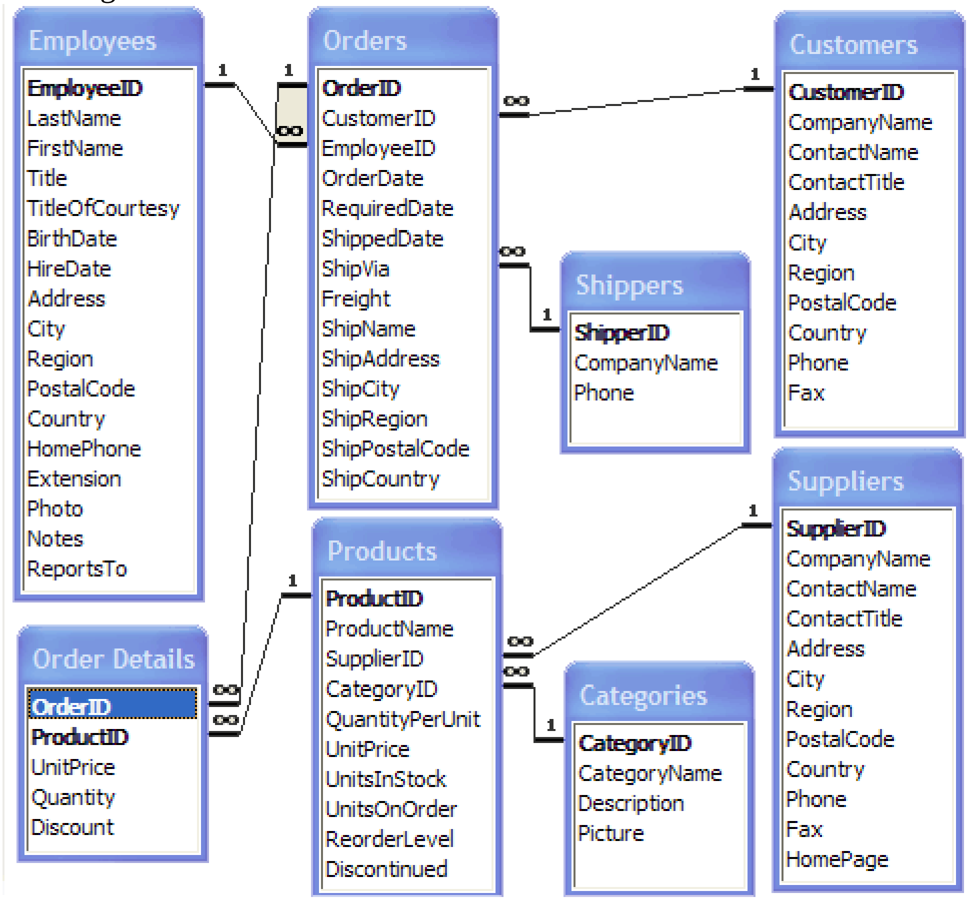
Introduction to the Northwind Database

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

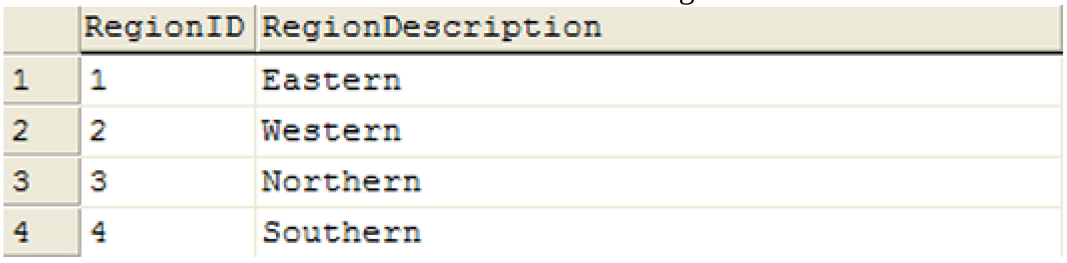
```sql
SELECT table.*
FROM table;

    -- OR

SELECT *
FROM table;
```

Code Sample:

SimpleSelects/Demos/SelectAll.sql

```sql
--Retrieve all columns in the Region table
SELECT *
FROM Region;
```

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.

Like this SQL tutorial? Try our self-paced online SQL course, which includes videos and exercises in addition to the content in this SQL tutorial. Not sure if you want to pay for that? Register for a free demo of the course.

SELECTing Specific Columns

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

Syntax

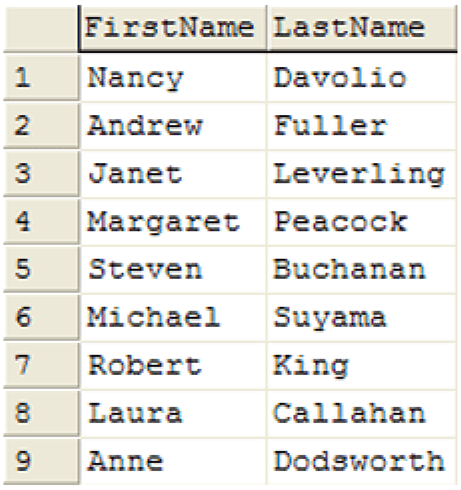
```sql
SELECT table_name.column_name, table_name.column_name
FROM table;

    -- OR

SELECT column, column
FROM table;
```

Code Sample:

SimpleSelects/Demos/SelectCols.sql

```
/*
Select the FirstName and LastName columns from the Employees table.
*/
SELECT FirstName, LastName
FROM Employees;
```

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

Like this SQL tutorial? Try our self-paced online SQL course, which includes videos and exercises in addition to the content in this SQL tutorial. Not sure if you want to pay for that? Register for a free demo of the course.

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.

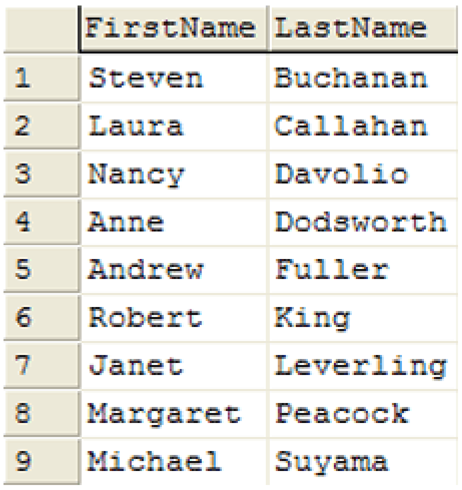
Code Sample:

SimpleSelects/Demos/OrderBy1.sql

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

```sql
SELECT FirstName, LastName
FROM Employees
ORDER 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.

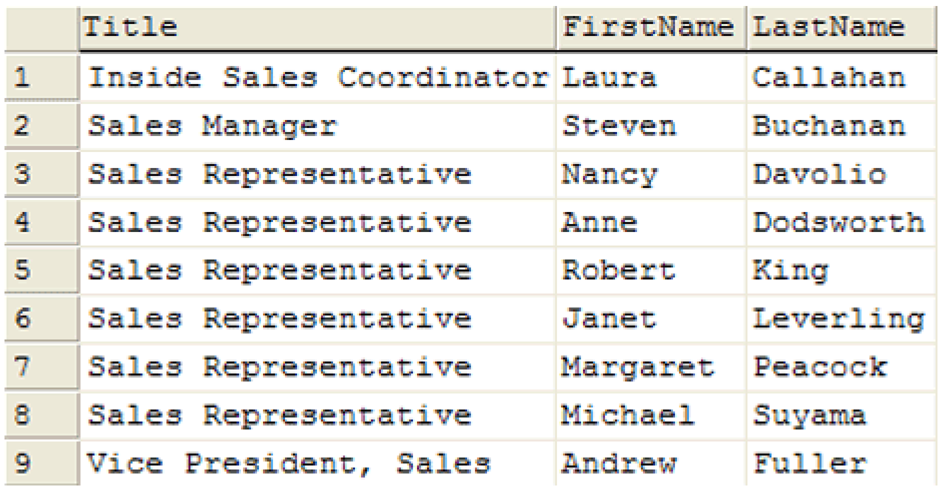
Syntax

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

Code Sample:

SimpleSelects/Demos/OrderBy2.sql

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

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



	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

Sorting By Column Position

It is also possible to sort tables by the position of a column in the `SELECT` list. To do so, specify the column numbers in the `ORDER BY` clause.

Syntax

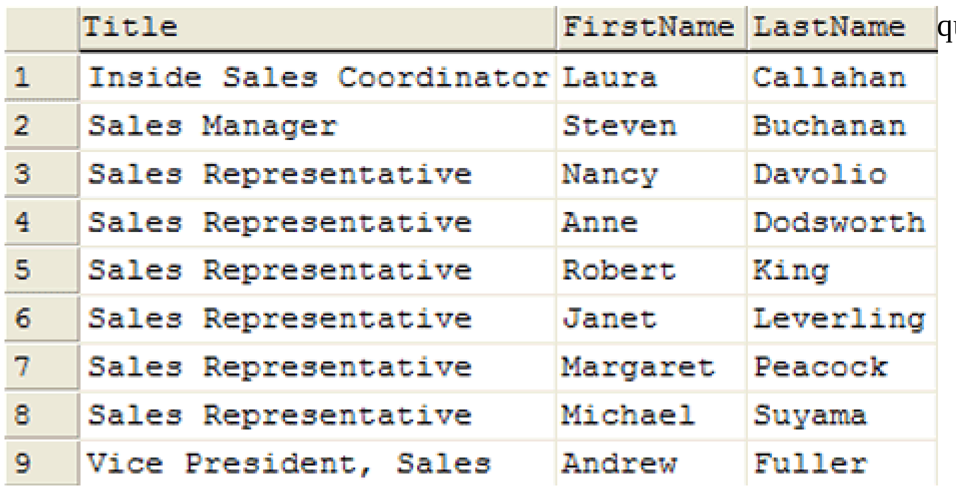
```
SELECT column, column
FROM table
ORDER BY column_position, column_position;
```

Code Sample:

SimpleSelects/Demos/OrderBy3.sql

```
/*
Select the Title, FirstName and LastName columns from the Employees
table.
Sort first by Title (position 1) and then by LastName (position 3).
*/

SELECT Title, FirstName, LastName
FROM Employees
ORDER BY 1,3;
```

query:



	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.

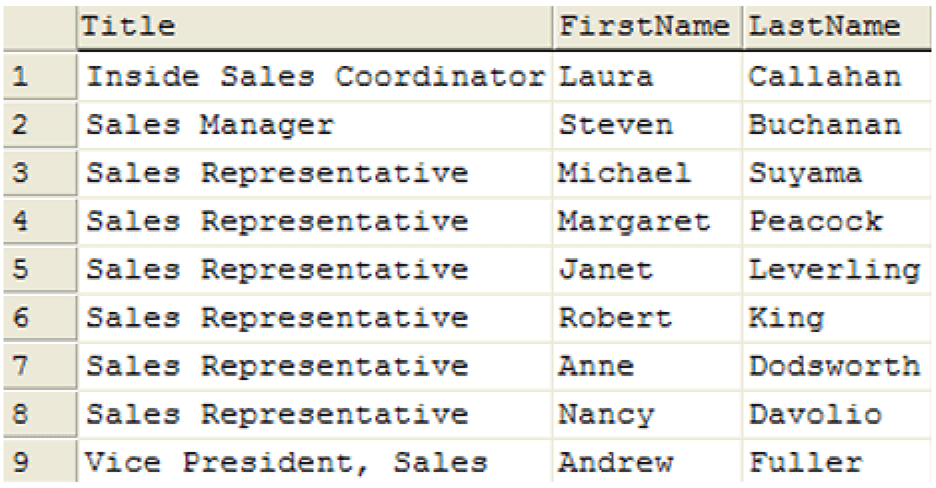
Syntax

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

Code Sample:

SimpleSelects/Demos/OrderBy4.sql

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

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



	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

Like this SQL tutorial? Try our self-paced online SQL course, which includes videos and exercises in addition to the content in this SQL tutorial. Not sure if you want to pay for that? Register for a free demo of the course.

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.

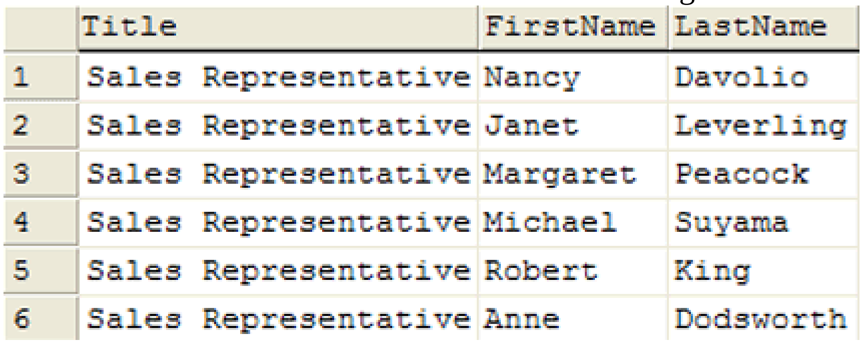
Checking for Equality

Code Sample:

SimpleSelects/Demos/Where-Equal.sql

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

SELECT Title, FirstName, LastName
FROM Employees
WHERE Title = 'Sales Representative';
```

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

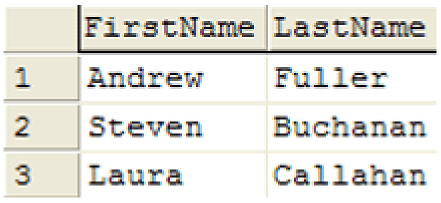
Checking for Inequality

Code Sample:

SimpleSelects/Demos/Where-NotEqual.sql

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

SELECT FirstName, LastName
FROM Employees
WHERE Title <> 'Sales Representative';
```

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

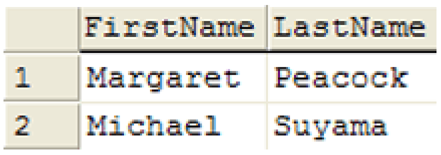
The less than (<) and greater than (>) signs are used to compare numbers, dates, and strings.

Code Sample:

SimpleSelects/Demos/Where-GreaterThanOrEqual.sql

```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.
*/

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

The above SELECT statement will return the following results:



	FirstName	LastName
1	Margaret	Peacock
2	Michael	Suyama

Checking for NULL

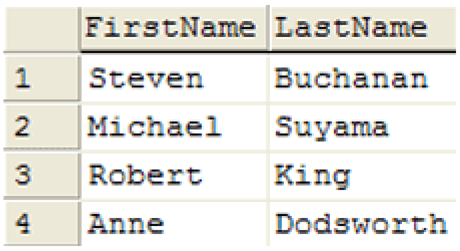
When a field in a row has no value, it is said to be `NULL`. This is not the same as having an empty string. Rather, it means that the field contains no value at all. When checking to see if a field is `NULL`, you cannot use the equals sign (=); rather, use the IS `NULL` expression.

Code Sample:

SimpleSelects/Demos/Where-Null.sql

```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:



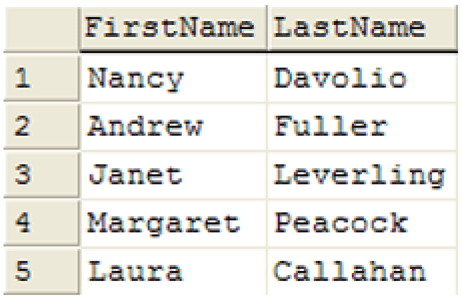
	FirstName	LastName
1	Steven	Buchanan
2	Michael	Suyama
3	Robert	King
4	Anne	Dodsworth

Code Sample:

SimpleSelects/Demos/Where-NotNull.sql

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

SELECT FirstName, LastName
FROM Employees
WHERE Region IS NOT NULL;
```

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

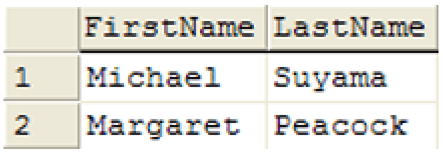
When using `WHERE` and `ORDER BY` together, the `WHERE` clause must come before the `ORDER BY` clause.

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

Like this SQL tutorial? Try our self-paced online SQL course, which includes videos and exercises in addition to the content in this SQL tutorial. Not sure if you want to pay for that? Register for a free demo of the course.

The WHERE Clause and Operator Words

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

SQL Word Operators

Operator	Description
`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

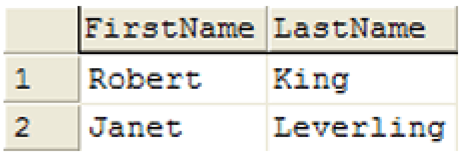
The `BETWEEN` operator is used to check if field values are within a specified inclusive range.

Code Sample:

SimpleSelects/Demos/Where-Between.sql

```sql

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

SELECT FirstName, LastName
FROM Employees
WHERE LastName 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.

The IN Operator

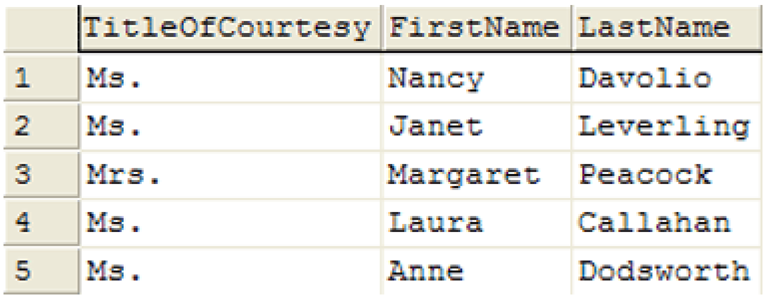
The `IN` operator is used to check if field values are included in a specified comma-delimited list.

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.".
*/

SELECT TitleOfCourtesy, FirstName, LastName
FROM Employees
WHERE TitleOfCourtesy IN ('Ms.','Mrs.');

-- 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.

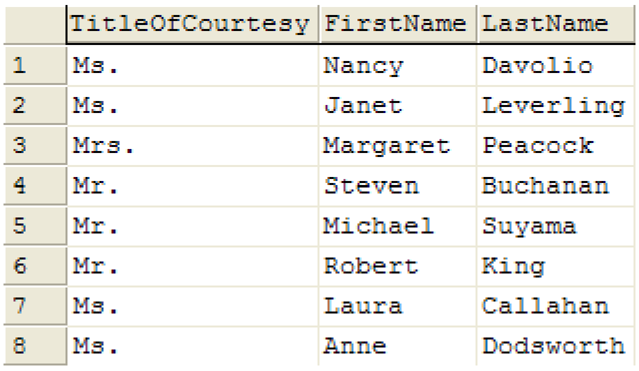
Code Sample:

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".
*/

```

```sql
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	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 (_)

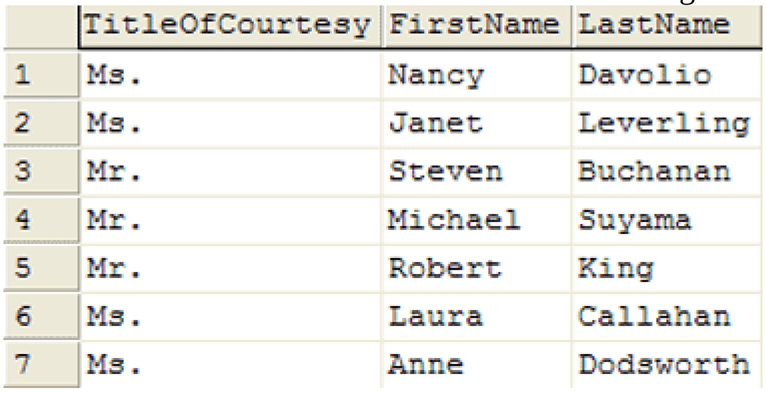
The underscore (_) is used to match any single character.

Code Sample:

SimpleSelects/Demos/Where-Like2.sql

```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

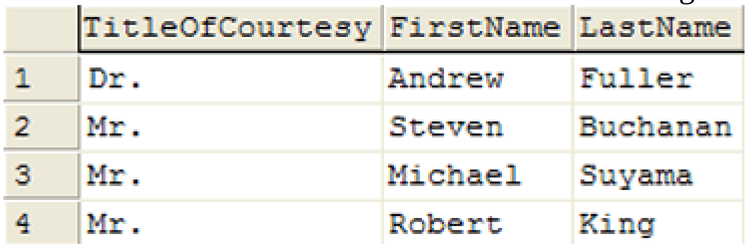
The `NOT` operator is used to negate an operation.

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

Like this SQL tutorial? Try our [self-paced online SQL courses](), which includes videos and exercises in addition to the content in this SQL tutorial. Not sure if you want to pay for that? Register for a [free demo of the course]().

Checking Multiple Conditions

AND

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

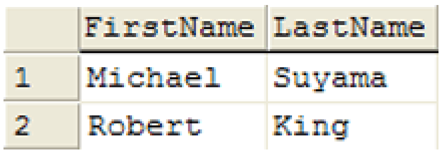
Code Sample:

SimpleSelects/Demos/Where-And.sql

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

SELECT FirstName, LastName
FROM Employees
WHERE Title = 'Sales Representative'
```

```
    AND TitleOfCourtesy = 'Mr.';
```

The above SELECT statement will return the following results:



	FirstName	LastName
1	Michael	Suyama
2	Robert	King

OR

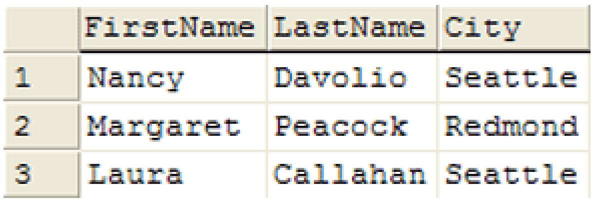
`OR` can be used in a `WHERE` clause to find records that match at least one of several conditions.

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.
*/

SELECT FirstName, LastName, City
FROM Employees
WHERE City = 'Seattle' OR City = '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.

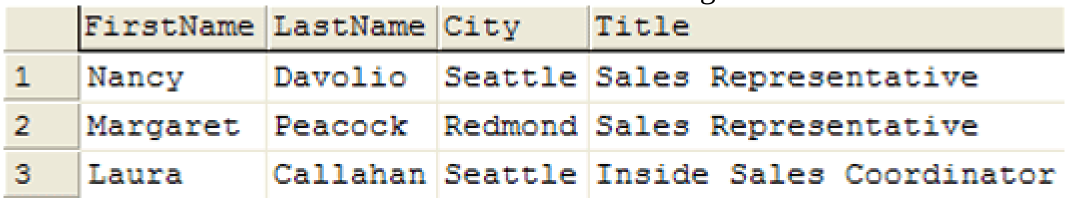
Code Sample:

SimpleSelects/Demos/Where-AndOrPrecedence.sql

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

SELECT FirstName, LastName, City, Title
FROM Employees
```

```
WHERE City = 'Seattle' OR City = 'Redmond'
    AND Title = 'Sales Representative';
```

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.

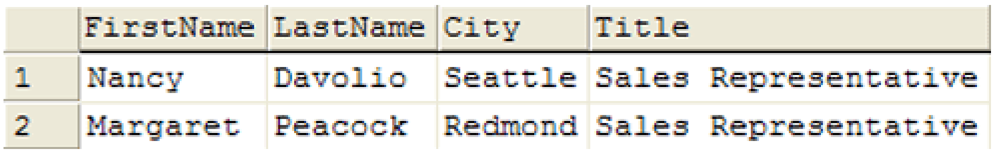
This can be fixed by putting the OR portion of the clause in parentheses.

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.
*/

SELECT FirstName, LastName, City, Title
FROM Employees
WHERE (City = 'Seattle' OR City = 'Redmond')
    AND Title = 'Sales Representative';
```

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.

Advanced SELECTs

In this lesson you will learn to write advanced select statements using SQL functions and grouping.

Lesson Goals

- To use `SELECT` statements to retrieve calculated values.
- To work with aggregate functions and grouping.

- To work with SQL's data manipulation functions.

Lesson Activities

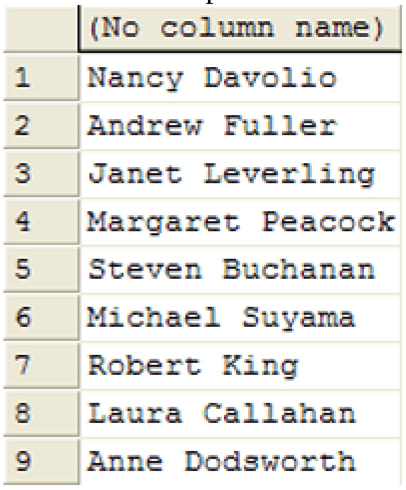
1. Calculated Fields
2. Aggregate Functions and Grouping
3. Built-in Data Manipulation Functions

Calculated Fields

Calculated fields are fields that do not exist in a table, but are created in the `SELECT` statement. For example, you might want to create `FullName` from `FirstName` and `LastName`.

Concatenation

Concatenation is a fancy word for stringing together different words or characters. SQL Server, Oracle and MySQL each has its own way of handling concatenation. All three of the code samples below will return the following results:



	(No column name)
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

In SQL Server, the plus sign (+) is used as the concatenation operator.

Code Sample:

AdvancedSelects/Demos/Concatenate-SqlServer.sql

```
-- Select the full name of all employees. SQL SERVER.

SELECT FirstName + ' ' + LastName
FROM Employees;
```

In Oracle, the double pipe (||) is used as the concatenation operator.

Code Sample:

AdvancedSelects/Demos/Concatenate-Oracle.sql

```
-- Select the full name of all employees. Oracle.

SELECT FirstName || ' ' || LastName
FROM Employees;
```

MySQL does this in yet another way. There is no concatenation operator. Instead, MySQL uses the CONCAT() function .

Code Sample:

AdvancedSelects/Demos/Concatenate-MySQL.sql

```
-- Select the full name of all employees. MySQL.
SELECT CONCAT(FirstName, ' ', LastName)
FROM Employees;
```

Note that concatenation only works with strings. To concatenate other data types, you must first convert them to strings.

Mathematical Calculations

Mathematical calculations in SQL are similar to those in other languages.

Mathematical Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division

Mathematical Operators

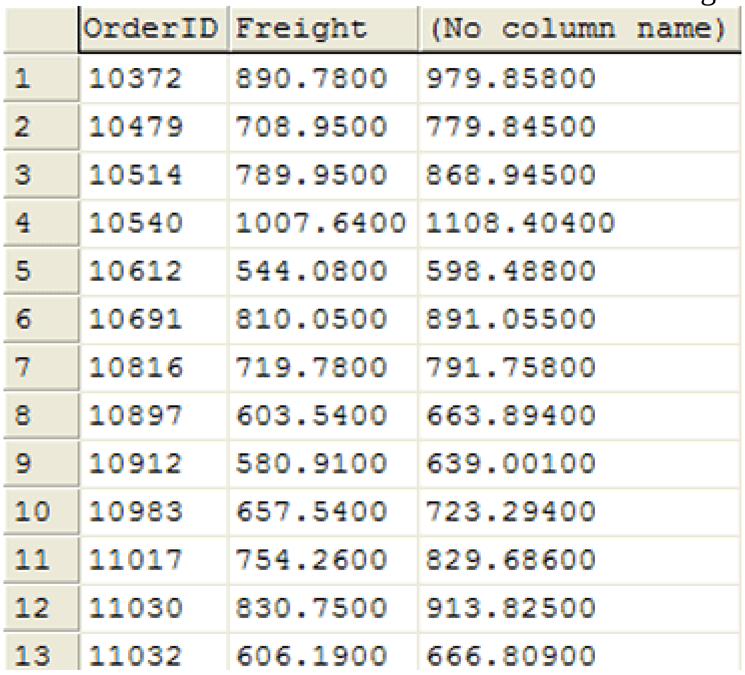
Operator	**Description**
%	Modulus

Code Sample:

AdvancedSelects/Demos/MathCalc.sql

```
/*
If the cost of freight is greater than or equal to $500.00,
it will now be taxed by 10%. Create a report that shows the
order id, freight cost, freight cost with this tax for all
orders of $500 or more.
*/

SELECT OrderID, Freight, Freight * 1.1
FROM Orders
WHERE Freight >= 500;
```

The above SELECT statement will return the following results:



	OrderID	Freight	(No column name)
1	10372	890.7800	979.85800
2	10479	708.9500	779.84500
3	10514	789.9500	868.94500
4	10540	1007.6400	1108.40400
5	10612	544.0800	598.48800
6	10691	810.0500	891.05500
7	10816	719.7800	791.75800
8	10897	603.5400	663.89400
9	10912	580.9100	639.00100
10	10983	657.5400	723.29400
11	11017	754.2600	829.68600
12	11030	830.7500	913.82500
13	11032	606.1900	666.80900

Aliases

You will notice in the examples above that the calculated columns have the header "`(No column name)`". The keyword `AS` is used to provide a named header for the column.

Code Sample:

AdvancedSelects/Demos/Alias.sql

```
SELECT OrderID, Freight, Freight * 1.1 AS FreightTotal
FROM Orders
WHERE Freight >= 500;
```

As you can see, the third column now has the title "`FreightTotal`".

Like this SQL tutorial? Try our self-paced online SQL courses, which includes videos and exercises in addition to the content in this SQL tutorial. Not sure if you want to pay for that? Register for a free demo of the course.

Aggregate Functions and Grouping

Aggregate Functions

Aggregate functions are used to calculate results using field values from multiple records. There are five common aggregate functions.

Common Aggregate Functions

Aggregate Function	Description
`COUNT()`	Returns the number of rows containing non-NULL values in the specified field.
`SUM()`	Returns the sum of the non-NULL values in the specified field.
`AVG()`	Returns the average of the non-NULL values in the specified field.
`MAX()`	Returns the maximum of the non-NULL values in the specified field.
`MIN()`	Returns the minimum of the non-NULL values in the specified field.

Code Sample:

AdvancedSelects/Demos/Aggregate-Count.sql

```sql
-- Find the Number of Employees

SELECT COUNT(*) AS NumEmployees
FROM Employees;
```

Returns 9.

Code Sample:

AdvancedSelects/Demos/Aggregate-Sum.sql

```sql
-- Find the Total Number of Units Ordered of Product ID 3

/******************************
SQL Server
******************************/
SELECT SUM(Quantity) AS TotalUnits
FROM "Order Details"
WHERE ProductID=3;

/******************************
Oracle and MySQL
******************************/
SELECT SUM(Quantity) AS TotalUnits
FROM Order_Details
WHERE ProductID=3;
```

Returns 328.

Code Sample:

AdvancedSelects/Demos/Aggregate-Avg.sql

```
-- Find the Average Unit Price of Products

SELECT AVG(UnitPrice) AS AveragePrice
FROM Products;
```

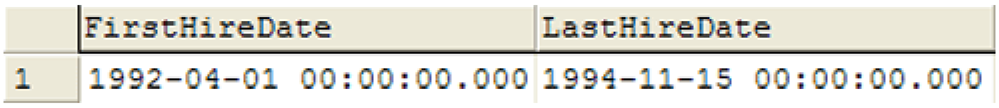
Returns 28.8663.

Code Sample:

AdvancedSelects/Demos/Aggregate-MinMax.sql

```
-- Find the Earliest and Latest Dates of Hire

SELECT MIN(HireDate) AS FirstHireDate,
    MAX(HireDate) AS LastHireDate
FROM Employees;
```

The above SELECT statement will return April 1, 1992 and November 15, 1994 as the `FirstHireDate` and `LastHireDate`, respectively. The date format will vary from database to database.



```
  FirstHireDate            LastHireDate
1 1992-04-01 00:00:00.000  1994-11-15 00:00:00.000
```

Grouping Data

GROUP BY

With the `GROUP BY` clause, aggregate functions can be applied to groups of records based on column values. For example, the following code will return the number of employees in each city.

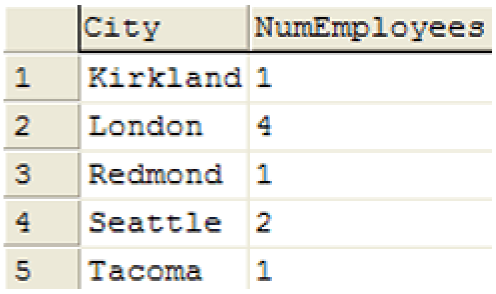
Code Sample:

AdvancedSelects/Demos/Aggregate-GroupBy.sql

```
--Retrieve the number of employees in each city

```

```
SELECT City, COUNT(EmployeeID) AS NumEmployees
FROM Employees
GROUP BY City;
```

The above SELECT statement will return the following results:



	City	NumEmployees
1	Kirkland	1
2	London	4
3	Redmond	1
4	Seattle	2
5	Tacoma	1

HAVING

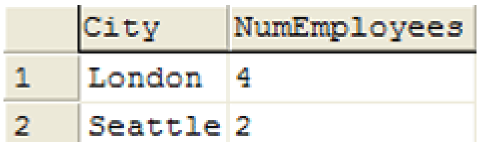
The HAVING clause is used to filter grouped data. For example, the following code specifies that we only want information on cities that have more than one employee.

Code Sample:

AdvancedSelects/Demos/Aggregate-Having.sql

```
/*
    Retrieve the number of employees in each city
    in which there are at least 2 employees.
*/

SELECT City, COUNT(EmployeeID) AS NumEmployees
FROM Employees
GROUP BY City
HAVING COUNT(EmployeeID) > 1;
```

The above SELECT statement will return the following results:



	City	NumEmployees
1	London	4
2	Seattle	2

Order of Clauses

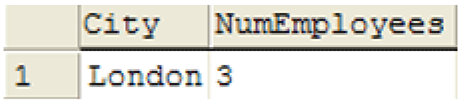
1. `SELECT`
2. `FROM`
3. `WHERE`
4. `GROUP BY`
5. `HAVING`
6. `ORDER BY`

Code Sample:

AdvancedSelects/Demos/Aggregate-OrderOfClauses.sql

```
/*
    Find the number of sales representatives in each city that contains
    at least 2 sales representatives. Order by the number of employees.
*/

SELECT City, COUNT(EmployeeID) AS NumEmployees
FROM Employees
WHERE Title = 'Sales Representative'
GROUP BY City
HAVING COUNT(EmployeeID) > 1
ORDER BY NumEmployees;
```

The above SELECT statement will return the following results:



	City	NumEmployees
1	London	3

Grouping Rules

- Every non-aggregate column that appears in the `SELECT` clause must also appear in the `GROUP BY` clause.
- You may not use aliases in the `HAVING` clause.
- You may use aliases in the `ORDER BY` clause.
- You may only use calculated fields in the `HAVING` clause.
- You may use calculated field aliases or actual fields in the `ORDER BY` clause.

Selecting Distinct Records

The `DISTINCT` keyword is used to select distinct combinations of column values from a table. For example, the following example shows how you would find all the distinct cities in which Northwind has employees.

Code Sample:

AdvancedSelects/Demos/Distinct.sql

```
/*
Find all the distinct cities in which Northwind has employees.
*/

SELECT DISTINCT City
FROM Employees
ORDER BY City
```

`DISTINCT` is often used with aggregate functions. The following example shows how `DISTINCT` can be used to find out in how many different cities Northwind has employees.

Code Sample:

AdvancedSelects/Demos/Distinct-Count.sql

```
/*
Find out in how many different cities Northwind has employees.
*/

SELECT COUNT(DISTINCT City) AS NumCities
FROM Employees
```

Like this SQL tutorial? Try our self-paced online SQL courses, which includes videos and exercises in addition to the content in this SQL tutorial. Not sure if you want to pay for that? Register for a free demo of the course.

Built-in Data Manipulation Functions

In this section, we will discuss some of the more common built-in data manipulation functions. Unfortunately, the functions differ greatly between databases, so you should be sure to check your database documentation when using these functions.

The tables below show some of the more common math, string, and date functions.

Common Math Functions

Common Math Functions

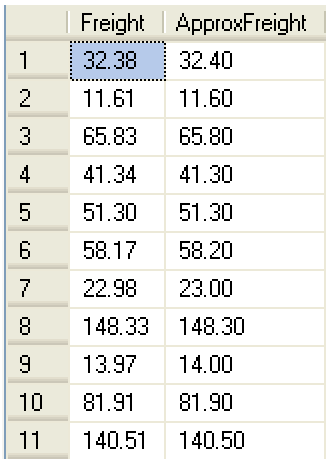
Description	SQL Server	Oracle	MySQL
Absolute value	`ABS`	`ABS`	`ABS`
Smallest integer >= value	`CEILING`	`CEIL`	`CEILING`
Round down to nearest integer	`FLOOR`	`FLOOR`	`FLOOR`
Power	`POWER`	`POWER`	`POWER`
Round	`ROUND`	`ROUND`	`ROUND`
Square root	`SQRT`	`SQRT`	`SQRT`
Formatting numbers to two decimal places	`CAST(num AS decimal(8,2))`	`CAST(num AS decimal(8,2))`	`FORMAT(num,2)` or `CAST(num AS decimal(8,2))`

Code Sample:

AdvancedSelects/Demos/Functions-Math1.sql

```
/*
 Select freight as is and
 freight rounded to the first decimal (e.g, 1.150 becomes 1.200)
 from the Orders tables
 */

SELECT Freight, ROUND(Freight,1) AS ApproxFreight
FROM Orders;
```

The above `SELECT` statement will return the following results (not all rows shown):



	Freight	ApproxFreight
1	32.38	32.40
2	11.61	11.60
3	65.83	65.80
4	41.34	41.30
5	51.30	51.30
6	58.17	58.20
7	22.98	23.00
8	148.33	148.30
9	13.97	14.00
10	81.91	81.90
11	140.51	140.50

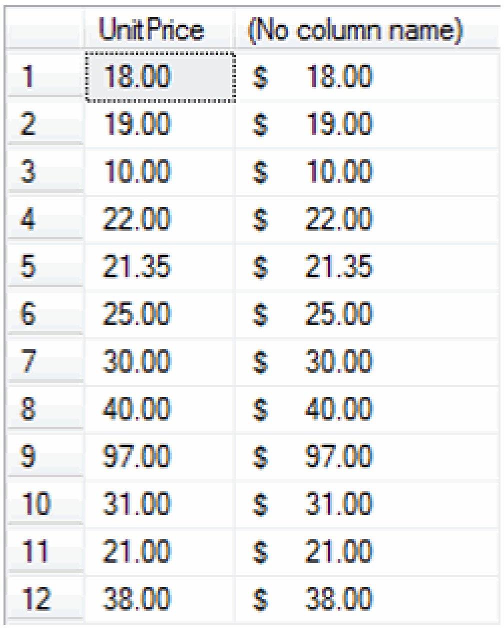
Code Sample:

AdvancedSelects/Demos/Functions-Math2.sql

```
/*
Select the unit price as is and
unit price as a CHAR(10)
from the Products tables
*/
SELECT UnitPrice, CAST(UnitPrice AS CHAR(10))
FROM Products;

/******************************
ADD CONCATENATION
******************************/
/******************************
SQL Server
******************************/
SELECT UnitPrice, '$' + CAST(UnitPrice AS CHAR(10))
FROM Products;

```

```
/*****************************
Oracle
*****************************/
SELECT UnitPrice, '$' || CAST(UnitPrice AS CHAR(10))
FROM Products;

/*****************************
MySQL
*****************************/
SELECT UnitPrice, CONCAT('$',CAST(UnitPrice AS CHAR(10)))
FROM Products;
```

The above SELECT statement will return the following results (not all rows shown):



	UnitPrice	(No column name)
1	18.00	$ 18.00
2	19.00	$ 19.00
3	10.00	$ 10.00
4	22.00	$ 22.00
5	21.35	$ 21.35
6	25.00	$ 25.00
7	30.00	$ 30.00
8	40.00	$ 40.00
9	97.00	$ 97.00
10	31.00	$ 31.00
11	21.00	$ 21.00
12	38.00	$ 38.00

Note that the CHAR(10) creates space for 10 characters and if the unit price required more than 10 characters you would need to increase it accordingly.

Common String Functions

Common String Functions

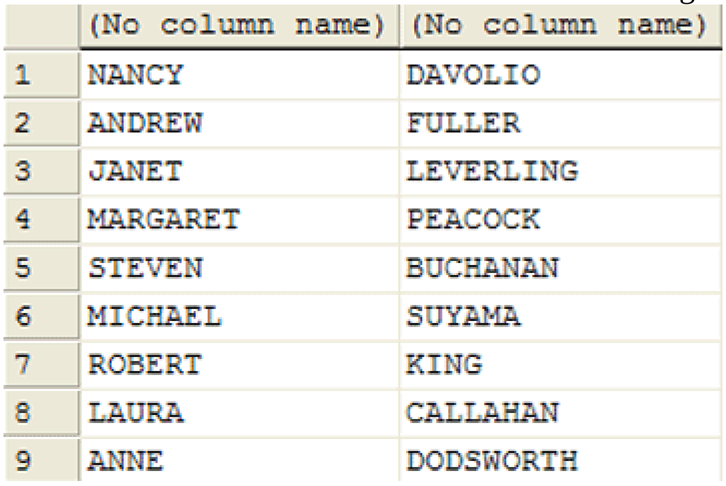
Description	SQL Server	Oracle	MySQL
Convert characters to lowercase	LOWER	LOWER	LOWER
Convert characters to uppercase	UPPER	UPPER	UPPER
Remove trailing blank spaces	RTRIM	RTRIM	RTRIM
Remove leading blank spaces	LTRIM	LTRIM	LTRIM
Substring	SUBSTRING	SUBSTR	SUBSTRING

Code Sample:

AdvancedSelects/Demos/Functions-String1.sql

```
/*
Select first and last name from employees in all uppercase letters
*/
SELECT UPPER(FirstName), UPPER(LastName)
FROM Employees;
```

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



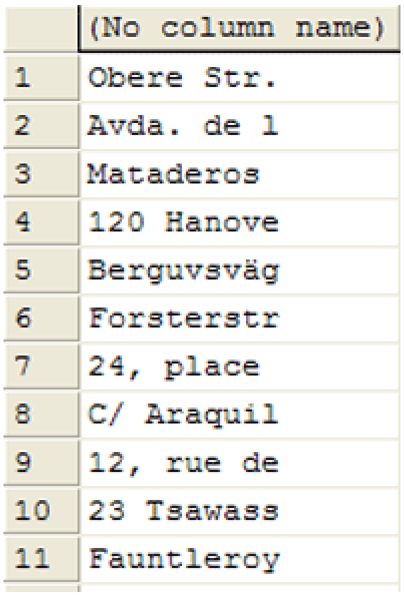
	(No column name)	(No column name)
1	NANCY	DAVOLIO
2	ANDREW	FULLER
3	JANET	LEVERLING
4	MARGARET	PEACOCK
5	STEVEN	BUCHANAN
6	MICHAEL	SUYAMA
7	ROBERT	KING
8	LAURA	CALLAHAN
9	ANNE	DODSWORTH

Code Sample:

AdvancedSelects/Demos/Functions-String2.sql

```
-- Select the first 10 characters of each customer's address

/*****************************
SQL Server and MySQL
*****************************/
SELECT SUBSTRING(Address,1,10)
FROM Customers;

/*****************************
Oracle
*****************************/
SELECT SUBSTR(Address,1,10)
FROM Customers;
```

The above `SELECT` statement will return the following results (not all rows shown):



	(No column name)
1	Obere Str.
2	Avda. de 1
3	Mataderos
4	120 Hanove
5	Berguvsväg
6	Forsterstr
7	24, place
8	C/ Araquil
9	12, rue de
10	23 Tsawass
11	Fauntleroy

Common Date Functions

Common Date Functions

Description	SQL Server	Oracle	MySQL

Common Date Functions

Description	SQL Server	Oracle	MySQL
Date addition	`DATEADD`	`(use +)`	`DATE_ADD`
Date subtraction	`DATEDIFF`	`(use -)`	`DATEDIFF`
Convert date to string	`DATENAME`	`TO_CHAR`	`DATE_FORMAT`
Convert date to number	`DATEPART`	`TO_NUMBER(TO_CHAR)`	`EXTRACT`
Get current date and time	`GETDATE`	`SYSDATE`	`NOW`

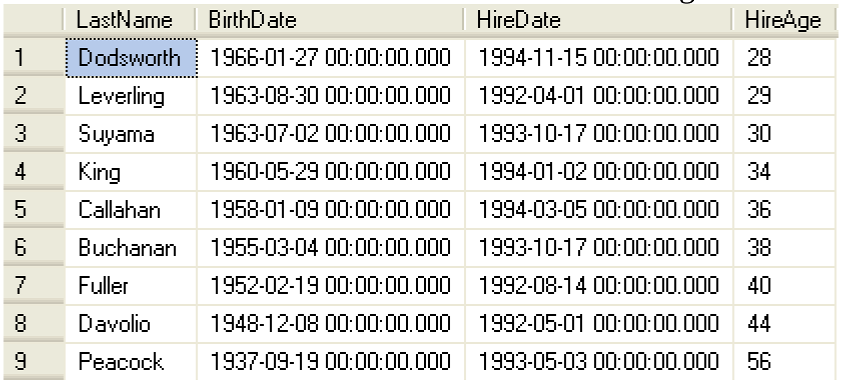
Code Sample:

AdvancedSelects/Demos/Functions-Date1.sql

```
-- Find the hiring age of each employee

/******************************
SQL Server
******************************/
SELECT LastName, BirthDate, HireDate,
DATEDIFF(year,BirthDate,HireDate) AS HireAge
FROM Employees
ORDER BY HireAge;

/******************************
Oracle
******************************/
SELECT LastName, BirthDate, HireDate, FLOOR((HireDate -
BirthDate)/365.25) AS HireAge
FROM Employees
ORDER BY HireAge;

```

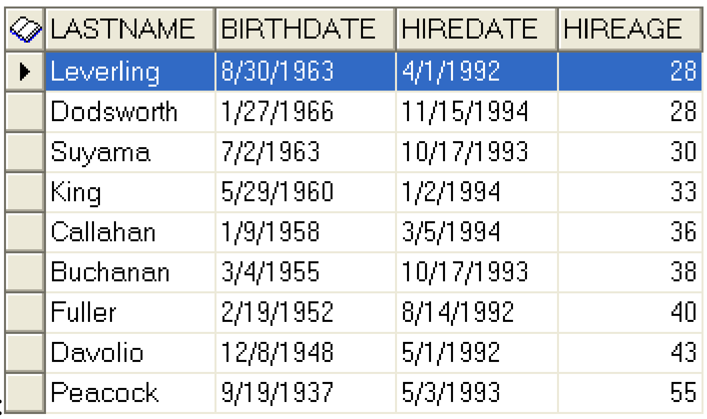
```
/******************************
MySQL
******************************/
-- Find the hiring age of each employee
-- in versions of MySQL prior to 4.1.1
SELECT LastName, BirthDate, HireDate, YEAR(HireDate)-YEAR(BirthDate) AS HireAge
FROM Employees;

-- In MySQL 4.1.1 and later, DATEDIFF() returns the number of days between
-- two dates. You can then divide and floor to get age.
SELECT LastName, BirthDate, HireDate, FLOOR(DATEDIFF(HireDate,BirthDate)/365) AS HireAge
FROM Employees
ORDER BY HireAge;
```

The above `SELECT` statement will return the following results in SQL Server:



	LastName	BirthDate	HireDate	HireAge
1	Dodsworth	1966-01-27 00:00:00.000	1994-11-15 00:00:00.000	28
2	Leverling	1963-08-30 00:00:00.000	1992-04-01 00:00:00.000	29
3	Suyama	1963-07-02 00:00:00.000	1993-10-17 00:00:00.000	30
4	King	1960-05-29 00:00:00.000	1994-01-02 00:00:00.000	34
5	Callahan	1958-01-09 00:00:00.000	1994-03-05 00:00:00.000	36
6	Buchanan	1955-03-04 00:00:00.000	1993-10-17 00:00:00.000	38
7	Fuller	1952-02-19 00:00:00.000	1992-08-14 00:00:00.000	40
8	Davolio	1948-12-08 00:00:00.000	1992-05-01 00:00:00.000	44
9	Peacock	1937-09-19 00:00:00.000	1993-05-03 00:00:00.000	56

And like this in Oracle:



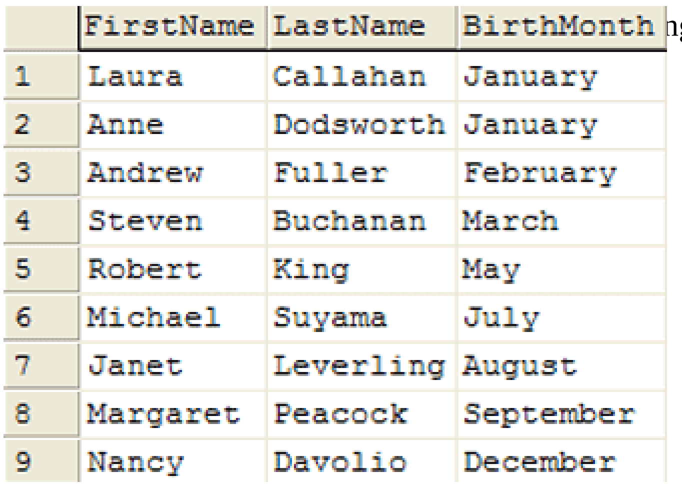
LASTNAME	BIRTHDATE	HIREDATE	HIREAGE
Leverling	8/30/1963	4/1/1992	28
Dodsworth	1/27/1966	11/15/1994	28
Suyama	7/2/1963	10/17/1993	30
King	5/29/1960	1/2/1994	33
Callahan	1/9/1958	3/5/1994	36
Buchanan	3/4/1955	10/17/1993	38
Fuller	2/19/1952	8/14/1992	40
Davolio	12/8/1948	5/1/1992	43
Peacock	9/19/1937	5/3/1993	55

Note for SQL Server users: SQL Server is subtracting the year the employee was born from the year (s)he was hired. This does not give us an accurate age. We'll fix this in an upcoming exercise.

Code Sample:

AdvancedSelects/Demos/Functions-Date2.sql

```

-- Find the Birth month for every employee

/******************************
SQL Server
******************************/
SELECT FirstName, LastName, DATENAME(month,BirthDate) AS BirthMonth
FROM Employees
ORDER BY DATEPART(month,BirthDate);

/******************************
Oracle
******************************/
SELECT FirstName, LastName, TO_CHAR(BirthDate,'MONTH') AS BirthMonth
FROM Employees
ORDER BY TO_NUMBER(TO_CHAR(BirthDate,'MM'));

/******************************
MySQL
******************************/
SELECT FirstName, LastName, DATE_FORMAT(BirthDate, '%M') AS BirthMonth
FROM Employees
ORDER BY EXTRACT(MONTH FROM BirthDate);

```



	FirstName	LastName	BirthMonth
1	Laura	Callahan	January
2	Anne	Dodsworth	January
3	Andrew	Fuller	February
4	Steven	Buchanan	March
5	Robert	King	May
6	Michael	Suyama	July
7	Janet	Leverling	August
8	Margaret	Peacock	September
9	Nancy	Davolio	December

ng results:

Subqueries, Joins and Unions

Often the data you need will be stored in multiple tables. In this lesson, you'll learn to create reports from two or more tables based on data in one of those tables or even in a separate table altogether.

Lesson Goals

- To write queries with subqueries.
- To select columns from multiple tables with joins.
- To select records from multiple tables with unions.

Lesson Activities

1. Subqueries
2. Joins
3. Outer Joins
4. Unions

Subqueries

Subqueries are queries embedded in queries. They are used to retrieve data from one table based on data in another table. They generally are used when tables have some kind of relationship. For example, in the Northwind database, the `Orders` table has a `CustomerID` field, which references a customer in the `Customers` table. Retrieving the `CustomerID` for a specific order is pretty straightforward.

Code Sample:

SubqueriesJoinsUnions/Demos/Subquery-SelectCustomerID.sql

```
1/*
```

```
Find the CustomerID of the company that placed order 10290.
*/

SELECT CustomerID
FROM Orders
WHERE OrderID = 10290;
```

This will return COMMI, which is very likely meaningless to the people reading the report. The next query uses a subquery to return a meaningful result.

Code Sample:

SubqueriesJoinsUnions/Demos/Subquery-SelectCompanyName.sql

```
-- Find the name of the company that placed order 10290.

SELECT CompanyName
FROM Customers
WHERE CustomerID = (SELECT CustomerID
            FROM Orders
            WHERE OrderID = 10290);
```

The above code returns `ComÃ©rcio Mineiro`, which is a lot more useful than `COMMI`.

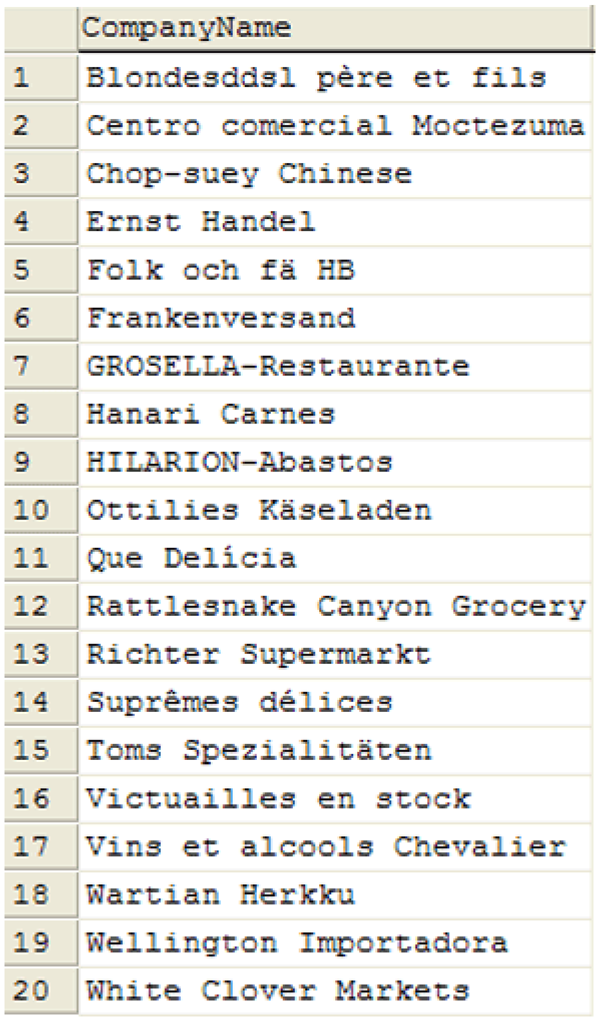
The subquery can contain any valid `SELECT` statement, but it must return a single column with the expected number of results. For example, if the subquery returns only one result, then the main query can check for equality, inequality, greater than, less than, etc. On the other hand, if the subquery returns more than one record, the main query must check to see if a field value is (or is `NOT`) `IN` the set of values returned.

Code Sample:

SubqueriesJoinsUnions/Demos/Subquery-IN.sql

```
-- Find the Companies that placed orders in 1997

/*****************************
Both of the queries below will work in SQL Server

Oracle
*****************************/
SELECT CompanyName
FROM Customers
WHERE CustomerID IN (SELECT CustomerID
            FROM Orders
            WHERE OrderDate BETWEEN '1-Jan-1997' AND '31-Dec-1997');

/*****************************
MySQL
*****************************/
SELECT CompanyName
FROM Customers
WHERE CustomerID IN (SELECT CustomerID
            FROM Orders
            WHERE OrderDate  BETWEEN '1997-01-01' AND '1997-12-31');
```

The above SELECT statement will return the following results:



	CompanyName
1	Blondesddsl père et fils
2	Centro comercial Moctezuma
3	Chop-suey Chinese
4	Ernst Handel
5	Folk och fä HB
6	Frankenversand
7	GROSELLA-Restaurante
8	Hanari Carnes
9	HILARION-Abastos
10	Ottilies Käseladen
11	Que Delícia
12	Rattlesnake Canyon Grocery
13	Richter Supermarkt
14	Suprêmes délices
15	Toms Spezialitäten
16	Victuailles en stock
17	Vins et alcools Chevalier
18	Wartian Herkku
19	Wellington Importadora
20	White Clover Markets

Like this SQL tutorial? Try our self-paced online SQL courses, which includes videos and exercises in addition to the content in this SQL tutorial. Not sure if you want to pay for that? Register for a free demo of the course.

Joins

How can we find out

- Which products are provided by which suppliers?
- Which customers placed which orders?
- Which customers are buying which products?

Such reports require data from multiple tables. Enter joins.

Syntax

```
SELECT table1.column, table2.column
FROM table1 JOIN table2
    ON (table1.column=table2.column)
WHERE conditions
```

Creating a report that returns the employee id and order id from the `Orders` table is not difficult.

Code Sample:

SubqueriesJoinsUnions/Demos/Joins-NoJoin.sql

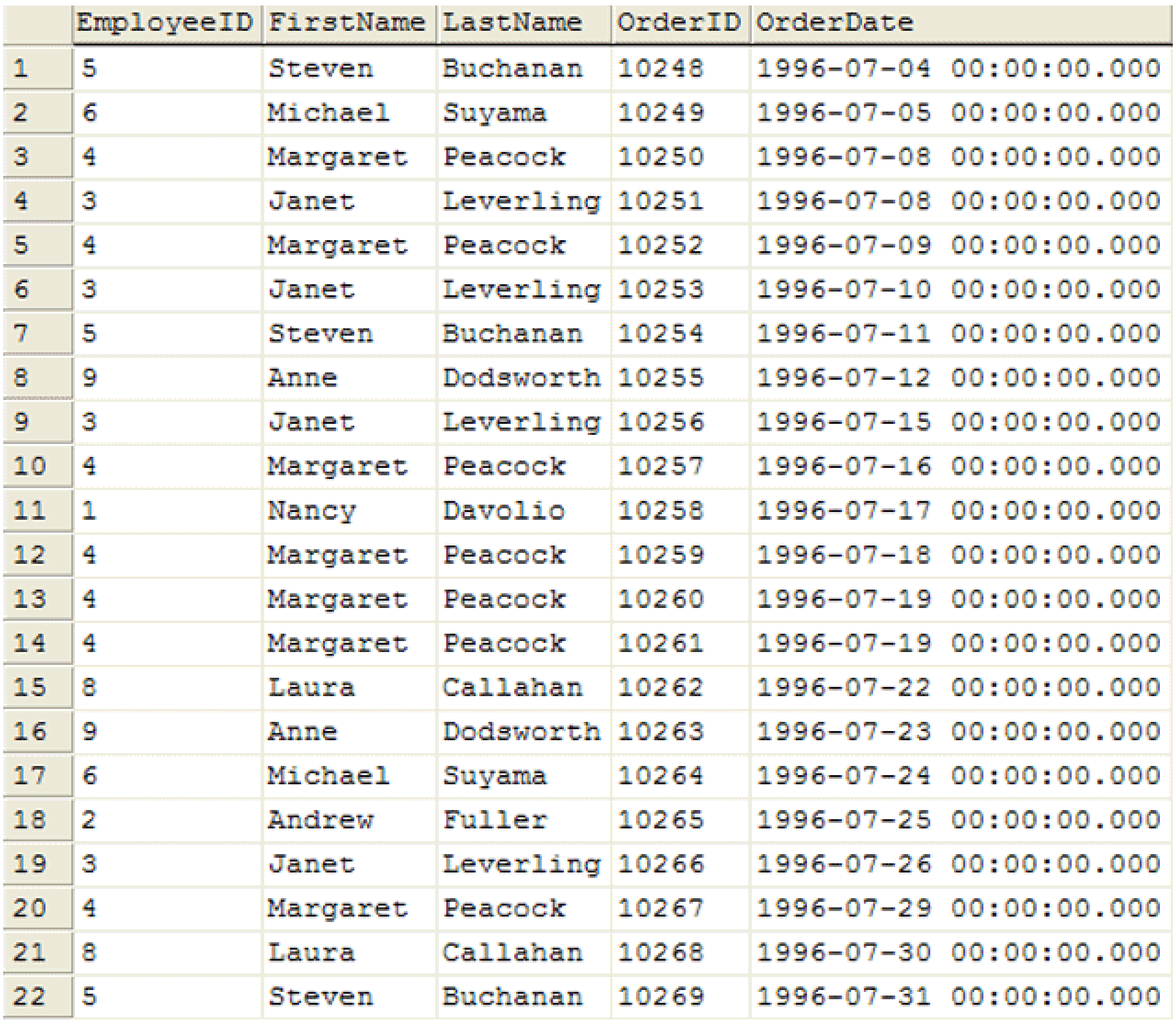
```
-- Find the EmployeeID and OrderID for all orders

SELECT EmployeeID, OrderID
FROM Orders;
```

But this is not very useful as we cannot tell who the employee is that got this order. The next sample shows how we can use a join to make the report more useful.

Code Sample:

SubqueriesJoinsUnions/Demos/Joins-EmployeeOrders.sql

```
-- Create a report showing employee orders.

SELECT Employees.EmployeeID, Employees.FirstName,
    Employees.LastName, Orders.OrderID, Orders.OrderDate
FROM Employees JOIN Orders ON
    (Employees.EmployeeID = Orders.EmployeeID)
ORDER BY Orders.OrderDate;
```



	EmployeeID	FirstName	LastName	OrderID	OrderDate
1	5	Steven	Buchanan	10248	1996-07-04 00:00:00.000
2	6	Michael	Suyama	10249	1996-07-05 00:00:00.000
3	4	Margaret	Peacock	10250	1996-07-08 00:00:00.000
4	3	Janet	Leverling	10251	1996-07-08 00:00:00.000
5	4	Margaret	Peacock	10252	1996-07-09 00:00:00.000
6	3	Janet	Leverling	10253	1996-07-10 00:00:00.000
7	5	Steven	Buchanan	10254	1996-07-11 00:00:00.000
8	9	Anne	Dodsworth	10255	1996-07-12 00:00:00.000
9	3	Janet	Leverling	10256	1996-07-15 00:00:00.000
10	4	Margaret	Peacock	10257	1996-07-16 00:00:00.000
11	1	Nancy	Davolio	10258	1996-07-17 00:00:00.000
12	4	Margaret	Peacock	10259	1996-07-18 00:00:00.000
13	4	Margaret	Peacock	10260	1996-07-19 00:00:00.000
14	4	Margaret	Peacock	10261	1996-07-19 00:00:00.000
15	8	Laura	Callahan	10262	1996-07-22 00:00:00.000
16	9	Anne	Dodsworth	10263	1996-07-23 00:00:00.000
17	6	Michael	Suyama	10264	1996-07-24 00:00:00.000
18	2	Andrew	Fuller	10265	1996-07-25 00:00:00.000
19	3	Janet	Leverling	10266	1996-07-26 00:00:00.000
20	4	Margaret	Peacock	10267	1996-07-29 00:00:00.000
21	8	Laura	Callahan	10268	1996-07-30 00:00:00.000
22	5	Steven	Buchanan	10269	1996-07-31 00:00:00.000

Table names are used as prefixes of the column names to identify the table in which to find the column. Although this is only required when the column name exists in both tables, it is always a good idea to include the prefixes as it makes the code more efficient and easier to read.

Table Aliases

Using full table names as prefixes can make SQL queries unnecessarily wordy. Table aliases can make the code a little more concise. The example below, which is identical in functionality to the query above, illustrates the use of table aliases.

Code Sample:

SubqueriesJoinsUnions/Demos/Joins-Aliases.sql

```
-- Create a report showing employee orders using Aliases.

SELECT e.EmployeeID, e.FirstName, e.LastName,
    o.OrderID, o.OrderDate
```

```
FROM Employees e JOIN Orders o ON
    (e.EmployeeID = o.EmployeeID)
ORDER BY o.OrderDate;

```

Multi-table Joins

Multi-table joins can get very complex and may also take a long time to process, but the syntax is relatively straightforward.

Syntax

```
SELECT table1.column, table2.column, table3.column
FROM table1
    JOIN table2 ON (table1.column=table2.column)
    JOIN table3 ON (table2.column=table3.column)
WHERE conditions
```

Note that, to join with a table, that table must be in the `FROM` clause or must already be joined with the table in the `FROM` clause. Consider the following.

```
SELECT table1.column, table2.column, table3.column
FROM table1
    JOIN table3 ON (table2.column=table3.column)
    JOIN table2 ON (table1.column=table2.column)
WHERE conditions
```

The above code would break because it attempts to join `table3` with `table2` before `table2` has been joined with `table1`.

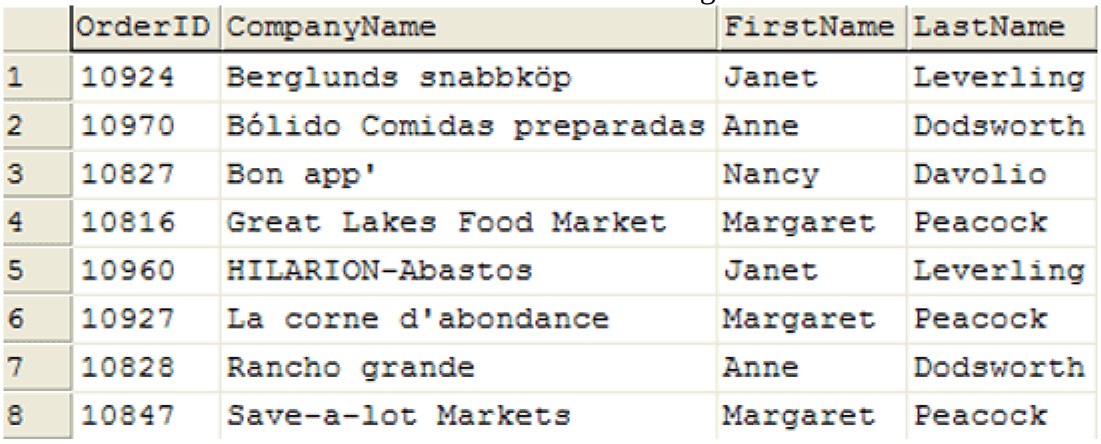
Code Sample:

SubqueriesJoinsUnions/Demos/Joins-MultiTable.sql

```
/*
Create a report showing the Order ID, the name of the company that placed the order,
and the first and last name of the associated employee.
Only show orders placed after January 1, 1998 that shipped after they were required.
Sort by Company Name.
*/

/******************************
Both of the queries below will work in SQL Server

Oracle
******************************/
SELECT o.OrderID, c.CompanyName, e.FirstName, e.LastName
FROM Orders o
    JOIN Employees e ON (e.EmployeeID = o.EmployeeID)
    JOIN Customers c ON (c.CustomerID = o.CustomerID)
WHERE o.ShippedDate > o.RequiredDate AND o.OrderDate > '1-Jan-1998'
```

```
ORDER BY c.CompanyName;

/******************************
MySQL
******************************/
SELECT o.OrderID, c.CompanyName, e.FirstName, e.LastName
FROM Orders o
    JOIN Employees e ON (e.EmployeeID = o.EmployeeID)
    JOIN Customers c ON (c.CustomerID = o.CustomerID)
WHERE o.ShippedDate > o.RequiredDate AND o.OrderDate > '1998-01-01'
ORDER BY c.CompanyName;
```

The above SELECT statement will return the following results:



	OrderID	CompanyName	FirstName	LastName
1	10924	Berglunds snabbköp	Janet	Leverling
2	10970	Bólido Comidas preparadas	Anne	Dodsworth
3	10827	Bon app'	Nancy	Davolio
4	10816	Great Lakes Food Market	Margaret	Peacock
5	10960	HILARION-Abastos	Janet	Leverling
6	10927	La corne d'abondance	Margaret	Peacock
7	10828	Rancho grande	Anne	Dodsworth
8	10847	Save-a-lot Markets	Margaret	Peacock

Like this SQL tutorial? Try our self-paced online SQL courses, which includes videos and exercises in addition to the content in this SQL tutorial. Not sure if you want to pay for that? Register for a free demo of the course.

Outer Joins

So far, all the joins we have worked with are inner joins, meaning that rows are only returned that have matches in both tables. For example, when doing an inner join between the `Employees` table and the `Orders` table, only employees that have matching orders and orders that have matching employees will be returned.

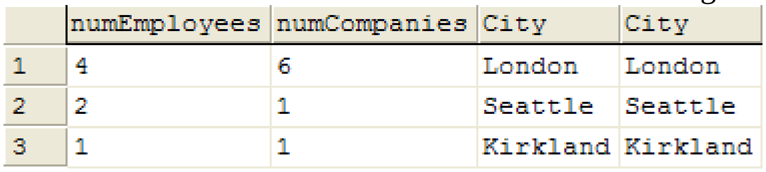
As a point of comparison, let's first look at another inner join.

Code Sample:

SubqueriesJoinsUnions/Demos/OuterJoins-Inner.sql

```
/*
    Create a report that shows the number of
    employees and customers from each city that has employees in it.
*/

SELECT COUNT(DISTINCT e.EmployeeID) AS numEmployees,
    COUNT(DISTINCT c.CustomerID) AS numCompanies,
    e.City, c.City
FROM Employees e JOIN Customers c ON
    (e.City = c.City)
GROUP BY e.City, c.City
ORDER BY numEmployees DESC;
```

The above SELECT statement will return the following results:



	numEmployees	numCompanies	City	City
1	4	6	London	London
2	2	1	Seattle	Seattle
3	1	1	Kirkland	Kirkland

Left Joins

A `LEFT JOIN` (also called a `LEFT OUTER JOIN`) returns all the records from the first table even if there are no matches in the second table.

Syntax

```
SELECT table1.column, table2.column
FROM table1
    LEFT [OUTER] JOIN table2 ON (table1.column=table2.column)
WHERE conditions
```

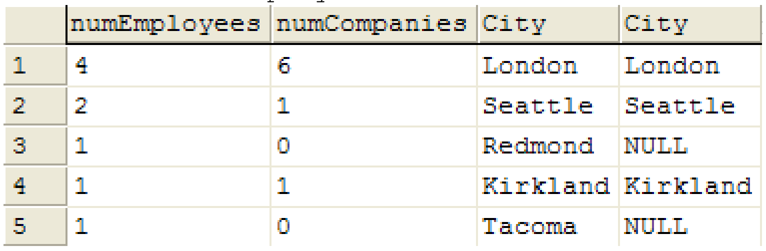
All rows in `table1` will be returned even if they do not have matches in `table2`.

Code Sample:

SubqueriesJoinsUnions/Demos/OuterJoins-Left.sql

```

/*
    Create a report that shows the number of
    employees and customers from each city that has employees in it.
*/

SELECT COUNT(DISTINCT e.EmployeeID) AS numEmployees,
    COUNT(DISTINCT c.CustomerID) AS numCompanies,
    e.City, c.City
FROM Employees e LEFT JOIN Customers c ON
    (e.City = c.City)
GROUP BY e.City, c.City
ORDER BY numEmployees DESC;
```

All records in the `Employees` table will be counted whether or not there are matching



	numEmployees	numCompanies	City	City
1	4	6	London	London
2	2	1	Seattle	Seattle
3	1	0	Redmond	NULL
4	1	1	Kirkland	Kirkland
5	1	0	Tacoma	NULL

Right Joins

A `RIGHT JOIN` (also called a `RIGHT OUTER JOIN`) returns all the records from the second table even if there are no matches in the first table.

Syntax

```
SELECT table1.column, table2.column
FROM table1
 RIGHT [OUTER] JOIN table2 ON (table1.column=table2.column)
WHERE conditions
```

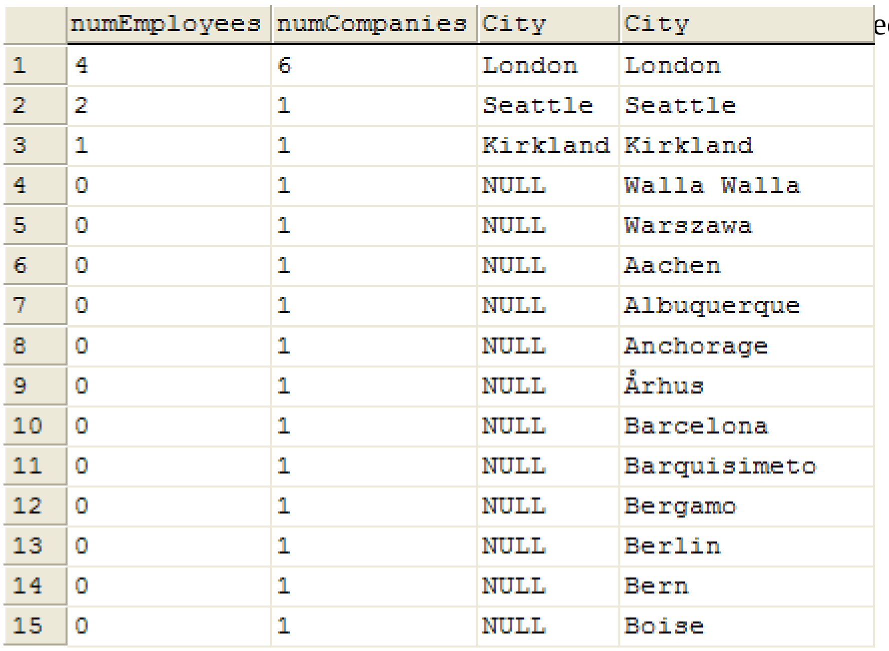
All rows in `table2` will be returned even if they do not have matches in `table1`.

Code Sample:

SubqueriesJoinsUnions/Demos/OuterJoins-Right.sql

```

/*
    Create a report that shows the number of
    employees and customers from each city that has customers in it.
*/

SELECT COUNT(DISTINCT e.EmployeeID) AS numEmployees,
    COUNT(DISTINCT c.CustomerID) AS numCompanies,
    e.City, c.City
FROM Employees e RIGHT JOIN Customers c ON
    (e.City = c.City)
GROUP BY e.City, c.City
ORDER BY numEmployees DESC;

```

All records in the Customers table will be counted whether or not there are matching



	numEmployees	numCompanies	City	City
1	4	6	London	London
2	2	1	Seattle	Seattle
3	1	1	Kirkland	Kirkland
4	0	1	NULL	Walla Walla
5	0	1	NULL	Warszawa
6	0	1	NULL	Aachen
7	0	1	NULL	Albuquerque
8	0	1	NULL	Anchorage
9	0	1	NULL	Århus
10	0	1	NULL	Barcelona
11	0	1	NULL	Barquisimeto
12	0	1	NULL	Bergamo
13	0	1	NULL	Berlin
14	0	1	NULL	Bern
15	0	1	NULL	Boise

ecords shown):

Full Outer Joins

A `FULL JOIN` (also called a `FULL OUTER JOIN`) returns all the records from each table even if there are no matches in the joined table.

Full outer joins are not supported in MySQL 5.x and earlier.

Syntax

```
SELECT table1.column, table2.column
FROM table1
     FULL [OUTER] JOIN table2 ON (table1.column=table2.column)
WHERE conditions
```

All rows in `table1` and `table2` will be returned.

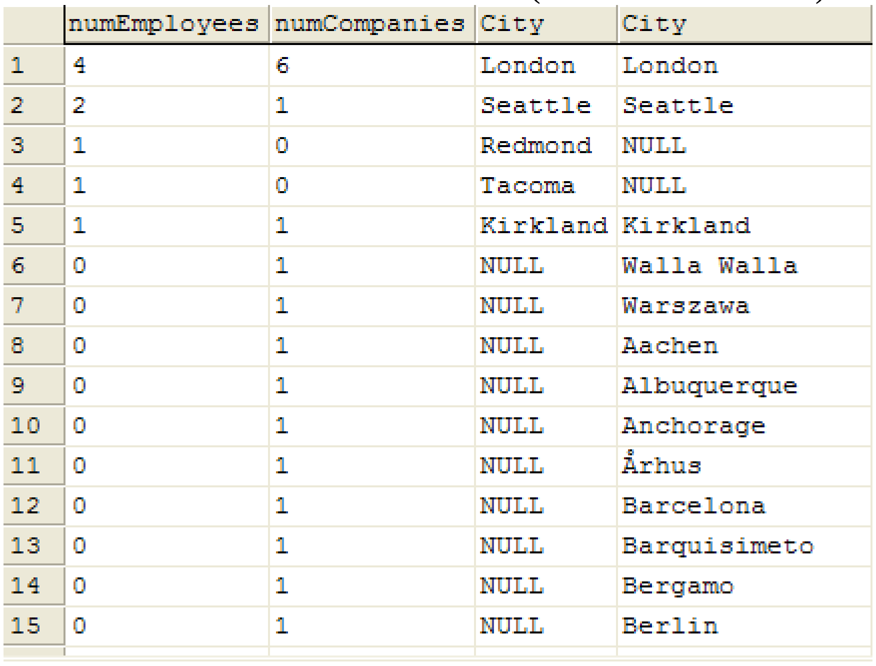
Code Sample:

SubqueriesJoinsUnions/Demos/OuterJoins-Full.sql

```
/*
     Create a report that shows the number of
     employees and customers from each city.

     Note that MySQL 5.x does NOT support full outer joins.
*/

SELECT COUNT(DISTINCT e.EmployeeID) AS numEmployees,
     COUNT(DISTINCT c.CustomerID) AS numCompanies,
     e.City, c.City
FROM Employees e FULL JOIN Customers c ON
```

```
    (e.City = c.City)
GROUP BY e.City, c.City
ORDER BY numEmployees DESC;
```

All records in each table will be counted whether or not there are matching cities in the other table. The results are shown below (not all records shown):



	numEmployees	numCompanies	City	City
1	4	6	London	London
2	2	1	Seattle	Seattle
3	1	0	Redmond	NULL
4	1	0	Tacoma	NULL
5	1	1	Kirkland	Kirkland
6	0	1	NULL	Walla Walla
7	0	1	NULL	Warszawa
8	0	1	NULL	Aachen
9	0	1	NULL	Albuquerque
10	0	1	NULL	Anchorage
11	0	1	NULL	Århus
12	0	1	NULL	Barcelona
13	0	1	NULL	Barquisimeto
14	0	1	NULL	Bergamo
15	0	1	NULL	Berlin

Like this SQL tutorial? Try our self-paced online SQL course, which includes videos and exercises in addition to the content in this SQL tutorial. Not sure if you want to pay for that? Register for a free demo of the course.

Unions

Unions are used to retrieve records from multiple tables or to get multiple record sets from a single table.

Code Sample:

SubqueriesJoinsUnions/Demos/Unions.sql

```
/*
Get the phone numbers of all shippers, customers, and suppliers
*/

SELECT CompanyName, Phone
FROM Shippers
    UNION
SELECT CompanyName, Phone
FROM Customers
    UNION
SELECT CompanyName, Phone
FROM Suppliers
ORDER BY CompanyName;
```

This query will return the company name and phone number of all shippers, customers and suppliers.

UNION ALL

By default, all duplicates are removed in `UNION`s. To include duplicates, use `UNION ALL` in place of `UNION`.

UNION Rules

- Each query must return the same number of columns.
- The columns must be in the same order.
- Column datatypes must be compatible.
- In Oracle, you can only ORDER BY columns that have the same name in every SELECT clause in the UNION

Conditional Processing with CASE

In this lesson you will learn how to use CASE to add conditional logic to your queries.

Lesson Goals

- To use the CASE function to display different values depending on the values of a column or columns.

Lesson Activities

1. Using CASE

Using CASE

`CASE` functions contain one or more `WHEN` clauses as shown below.

Syntax

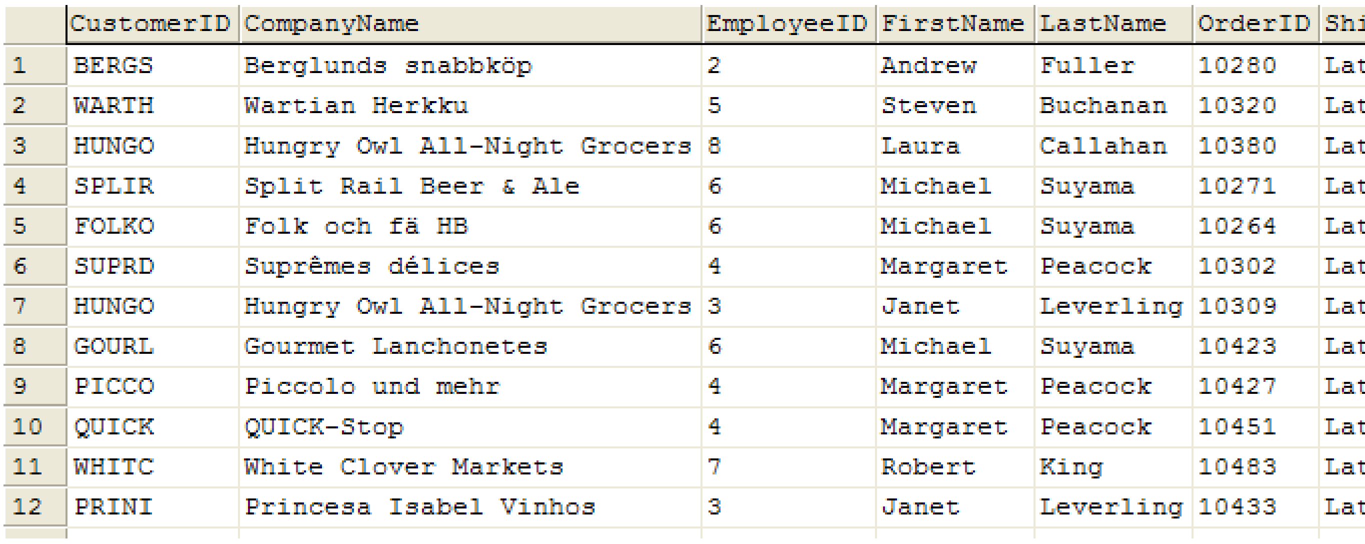
```
--OPTION 1
SELECT CASE column
              WHEN VALUE  THEN RETURN_VALUE
              WHEN VALUE  THEN RETURN_VALUE
              WHEN VALUE  THEN RETURN_VALUE
              WHEN VALUE  THEN RETURN_VALUE
              ELSE RETURN_VALUE
         END
```

```
    AS ColumnName
FROM table

--OPTION 2
SELECT CASE
            WHEN EXPRESSION  THEN RETURN_VALUE
            WHEN EXPRESSION  THEN RETURN_VALUE
            WHEN EXPRESSION  THEN RETURN_VALUE
            WHEN EXPRESSION  THEN RETURN_VALUE
            ELSE RETURN_VALUE
        END
    AS ColumnName
FROM table
```

Code Sample:

Case/Demos/Case.sql

```
/*
Create a report showing the customer ID and company name,
employee id, firstname and lastname, and the order id
and a conditional column called "Shipped" that displays "On Time"
if the order was shipped on time and "Late" if the order was shipped
late.
*/

SELECT c.CustomerID, c.CompanyName, e.EmployeeID, e.FirstName,
e.LastName, OrderID,
    (CASE
        WHEN ShippedDate < RequiredDate
            THEN 'On Time'
            ELSE 'Late'
            END) AS Shipped
FROM Orders o
    JOIN Employees e ON (e.EmployeeID = o.EmployeeID)
    JOIN Customers c ON (c.CustomerID = o.CustomerID)
ORDER BY Shipped;
```

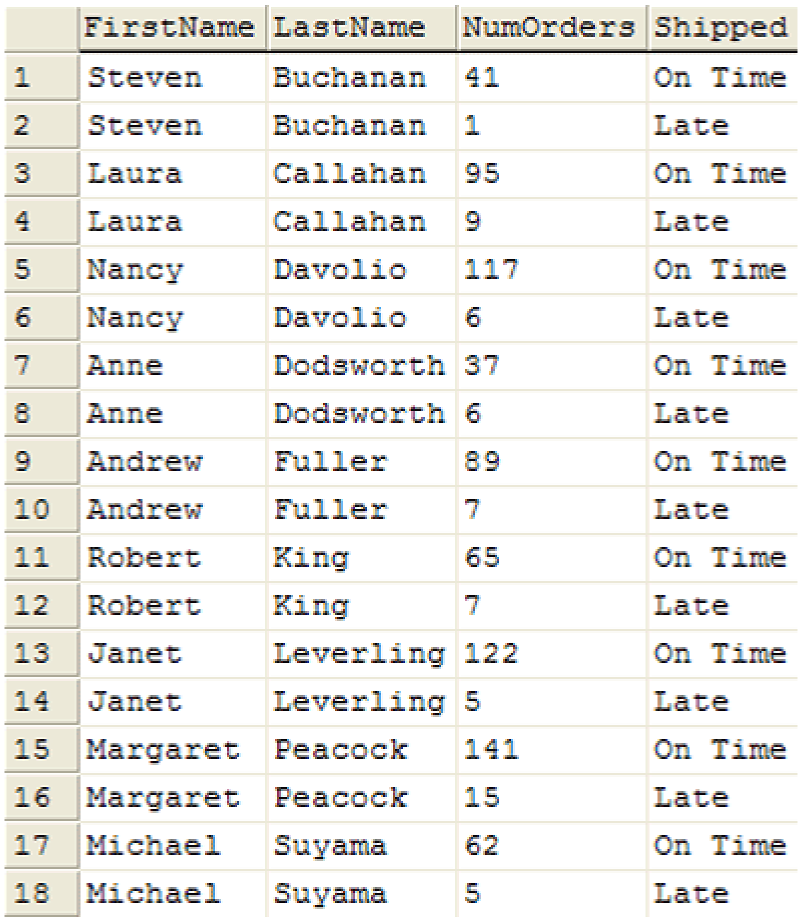


	CustomerID	CompanyName	EmployeeID	FirstName	LastName	OrderID	Shi
1	BERGS	Berglunds snabbköp	2	Andrew	Fuller	10280	Lat
2	WARTH	Wartian Herkku	5	Steven	Buchanan	10320	Lat
3	HUNGO	Hungry Owl All-Night Grocers	8	Laura	Callahan	10380	Lat
4	SPLIR	Split Rail Beer & Ale	6	Michael	Suyama	10271	Lat
5	FOLKO	Folk och fä HB	6	Michael	Suyama	10264	Lat
6	SUPRD	Suprêmes délices	4	Margaret	Peacock	10302	Lat
7	HUNGO	Hungry Owl All-Night Grocers	3	Janet	Leverling	10309	Lat
8	GOURL	Gourmet Lanchonetes	6	Michael	Suyama	10423	Lat
9	PICCO	Piccolo und mehr	4	Margaret	Peacock	10427	Lat
10	QUICK	QUICK-Stop	4	Margaret	Peacock	10451	Lat
11	WHITC	White Clover Markets	7	Robert	King	10483	Lat
12	PRINI	Princesa Isabel Vinhos	3	Janet	Leverling	10433	Lat

Code Sample:

Case/Demos/Case-GroupBy.sql

```


/*
Create a report showing the employee firstname and lastname,
a "NumOrders" column with a count of the orders taken, and a
conditional column called "Shipped" that displays "On Time" if
the order shipped on time and "Late" if the order shipped late.
Group records by employee firstname and lastname and then by the
"Shipped" status. Order by employee lastname, then by firstname,
and then descending by number of orders.
*/

SELECT e.FirstName, e.LastName, COUNT(o.OrderID) As NumOrders,
     (CASE
         WHEN o.ShippedDate < o.RequiredDate
              THEN 'On Time'
              ELSE 'Late'
              END)
         AS Shipped
FROM Orders o
     JOIN Employees e ON (e.EmployeeID = o.EmployeeID)
GROUP BY e.FirstName, e.LastName,
     (CASE
         WHEN o.ShippedDate < o.RequiredDate
              THEN 'On Time'
              ELSE 'Late'
              END)
ORDER BY e.LastName, e.FirstName, NumOrders DESC;


```



	FirstName	LastName	NumOrders	Shipped
1	Steven	Buchanan	41	On Time
2	Steven	Buchanan	1	Late
3	Laura	Callahan	95	On Time
4	Laura	Callahan	9	Late
5	Nancy	Davolio	117	On Time
6	Nancy	Davolio	6	Late
7	Anne	Dodsworth	37	On Time
8	Anne	Dodsworth	6	Late
9	Andrew	Fuller	89	On Time
10	Andrew	Fuller	7	Late
11	Robert	King	65	On Time
12	Robert	King	7	Late
13	Janet	Leverling	122	On Time
14	Janet	Leverling	5	Late
15	Margaret	Peacock	141	On Time
16	Margaret	Peacock	15	Late
17	Michael	Suyama	62	On Time
18	Michael	Suyama	5	Late

Inserting, Updating and Deleting Records

Inserting new records into a table is not difficult. Dangerously, it is even easier to update and delete records.

Lesson Goals

- To insert records into a table.
- To update records in a table.
- To delete records from a table.

Lesson Activities

1. INSERT
2. UPDATE and DELETE

INSERT

To insert a record into a table, you must specify values for all fields that do not have default values and cannot be `NULL`.

Syntax

```
INSERT INTO table
(columns)
VALUES (values);
```

The second line of the above statement can be excluded if all required columns are inserted and the values are listed in the same order as the columns in the table. We recommend you include the second line all the time though as the code will be easier to read and update and less likely to break as the database is modified.

Code Sample:

InsertsUpdatesDeletes/Demos/Insert.sql

```


-- Insert a New Employee

/******************************
Both of the inserts below will work in SQL Server

Oracle
******************************/
INSERT INTO Employees
(LastName, FirstName, Title, TitleOfCourtesy,
    BirthDate, HireDate, Address, City, Region,
    PostalCode, Country, HomePhone, Extension)
VALUES ('Dunn','Nat','Sales Representative','Mr.','19-Feb-1970',
    '15-Jan-2004','4933 Jamesville Rd.','Jamesville','NY',
    '13078','USA','315-555-5555','130');

/******************************
MySQL
******************************/
INSERT INTO Employees
(LastName, FirstName, Title, TitleOfCourtesy,
    BirthDate, HireDate, Address, City, Region,
    PostalCode, Country, HomePhone, Extension)
VALUES ('Dunn','Nat','Sales Representative','Mr.','1970-02-19',
    '2004-01-15','4933 Jamesville Rd.','Jamesville','NY',
    '13078','USA','315-555-5555','130');
```

If the `INSERT` is successful, the output will read something to this effect:

```
(1 row(s) affected)
```

Like this SQL tutorial? Try our self-paced online SQL course, which includes videos and exercises in addition to the content in this SQL tutorial. Not sure if you want to pay for that? Register for a free demo of the course.

UPDATE

The `UPDATE` statement allows you to update one or more fields for any number of records in a table. *You must be very careful not to update more records than you intend to!*

Syntax

```
UPDATE table
SET field = value,
    field = value,
    field = value
WHERE conditions;
```

Code Sample:

InsertsUpdatesDeletes/Demos/Update.sql

```
-- Update an Employee

UPDATE Employees
SET FirstName = 'Nathaniel'
WHERE FirstName = 'Nat';
```

If the `UPDATE` is successful, the output will read something to this effect:

```
(1 row(s) affected)
```

DELETE

The `DELETE` statement allows you to delete one or more records in a table. *Like with UPDATE, you must be very careful not to delete more records than you intend to!*

Syntax

```
DELETE FROM Employees
WHERE conditions;
```

Code Sample:

InsertsUpdatesDeletes/Demos/Delete.sql

```
-- Delete an Employee

DELETE FROM Employees
WHERE FirstName = 'Nathaniel';
```

If the `DELETE` is successful, the output will read something to this effect:

```
(1 row(s) affected)
```