Introduction to SQL

(SQL101 version 1.4.5)

Copyright Information

© Copyright 2016 Webucator. All rights reserved.

The Author

Nat Dunn

Nat Dunn founded Webucator in 2003 to combine his passion for web development with his business expertise and to help companies benefit from both. Nat began programming games in Basic on a TRS-80 at age 14. He has been developing websites and providing web development training since 1998.

Accompanying Class Files

This manual comes with accompanying class files, which your instructor or sales representative will point out to you. Most code samples and exercise and solution files found in the manual can also be found in the class files at the locations indicated at the top of the code listings.

Due to space limitations, the code listings sometimes have line wrapping, where no line wrapping occurs in the actual code sample. This is indicated in the manual using three greater than signs: >>> at the beginning of each wrapped line.

In other cases, the space limitations are such that we have inserted a forced line break in the middle of a word. When this occurs, we append the following symbol at the end of the line before the actual break: »»

Table of Contents

© Copyright 2016 Webucator. All rights reserved.

1. Relational Database Basics

In this lesson, you will learn...

1. About the history of SQL and relational databases.
2. How relational databases are structured.
3. About some of the most popular relational databases.
4. About the major SQL statements.

SQL stands for Structured Query Language and is pronounced either *ess-que-el* or *sequel*. It is the language used by relational database management systems (RDBMS) to access and manipulate data and to create, structure and destroy databases and database objects.

1.1 Brief History of SQL

In 1970, Dr. E.F. Codd published "A Relational Model of Data for Large Shared Data Banks," an article that outlined a model for storing and manipulating data using tables. Shortly after Codd's article was published, IBM began working on creating a relational database. Between 1979 and 1982, Oracle (then Relational Software, Inc.), Relational Technology, Inc. (later acquired by Computer Associates), and IBM all put out commercial relational databases, and by 1986 they all were using SQL as the data query language.

In 1986, the American National Standards Institute (ANSI) standardized SQL. This standard was updated in 1989, in 1992 (called SQL2), in 1999 (called SQL3), in 2003 (called SQL 2003), in 2006 (called SQL 2006), in 2008 (called SQL 2008), and in 2011 (called SQL 2011). Standard SQL is sometimes called ANSI SQL. All major relational databases support this standard but each has its own proprietary extensions. Unless otherwise noted, the SQL taught in this course is the standard ANSI SQL.

1.2 Relational Databases

A relational database at its simplest is a set of tables used for storing data. Each table has a unique name and may relate to one or more other tables in the database through common values.

Tables

A table in a database is a collection of rows and columns. Tables are also known as entities or relations.

Rows

A row contains data pertaining to a single item or record in a table. Rows are also known as records, instances, or tuples.

Columns

A column contains data representing a specific characteristic of the records in the table. Columns are also known as fields, properties, or attributes.

Relationships

A relationship is a link between two tables (i.e, relations). Relationships make it possible to find data in one table that pertains to a specific record in another table.

Datatypes

Each of a table's columns has a defined datatype that specifies the type of data that can exist in that column. For example, the `FirstName` column might be defined as `varchar(20)`, indicating that it can contain a string of up to 20 characters. Unfortunately, datatypes vary widely between databases.

Primary Keys

Most tables have a column or group of columns that can be used to identify records. For example, an `Employees` table might have a column called `EmployeeID` that is unique for every row. This makes it easy to keep track of a record over time and to associate a record with records in other tables.

Foreign Keys

Foreign key columns are columns that link to primary key columns in other tables, thereby creating a relationship. For example, the `Customers` table might have a foreign key column called `SalesRep` that links to `EmployeeID`, the primary key in the `Employees` table.

Relational Database Management System

A Relational Database Management System (RDBMS), commonly (but incorrectly) called a database, is software for creating, manipulating, and administering a database. For simplicity, we will often refer to RDBMSs as databases.

© Copyright 2016 Webucator. All rights reserved.

1.3 Popular Databases

Commercial Databases

Oracle

Oracle is the most popular relational database. It runs on both Unix and Windows. It used to be many times more expensive than SQL Server and DB2, but it has come down a lot in price. Oracle now offers a scaled down version that is free for personal or business use: Oracle XE (Extended Edition).

SQL Server

SQL Server is Microsoft's database and, not surprisingly, only runs on Windows. It has only a slightly higher market share than Oracle on Windows machines. Many people find it easier to use than Oracle.

DB2

IBM's DB2 was one of the earliest players in the database market. It is still very commonly used on mainframes and runs on both Windows and Unix.

Popular Open Source Databases

MySQL

Because of its small size, its speediness, and its very good documentation, MySQL has quickly become the most popular open source database. MySQL is available on both Windows and Unix, but it lacks some key features such as support for stored procedures.

PostgreSQL

Until recently, PostgreSQL was the most popular open source database until that spot was taken over by MySQL. PostgreSQL now calls itself "the world's most advanced Open Source database software." It is certainly a featureful and robust database management system and a good choice for people who want some of the advanced features that MySQL doesn't yet have. Originally only available on Unix, it became available on Windows in 2005.

1.4 SQL Statements

Database Manipulation Language (DML)

DML statements are used to work with data in an existing database. The most common DML statements are:

- `SELECT`
- `INSERT`
- `UPDATE`
- `DELETE`

Database Definition Language (DDL)

DDL statements are used to structure objects in a database. The most common DDL statements are:

- `CREATE`
- `ALTER`
- `DROP`

Database Control Language (DCL)

DCL statements are used for database administration. The most common DCL statements are:

- `GRANT`
- `DENY (SQL Server Only)`
- `REVOKE`

1.5 Conclusion

We have covered a little bit of the history of SQL, how databases work, and the common SQL statements. Now we will get into learning how to work with SQL.

© Copyright 2016 Webucator. All rights reserved.

2. Simple SELECTs

In this lesson, you will learn...

1. About the database we'll be using in class.
2. To comment your SQL code.
3. To understand SQL syntax.
4. To select all rows from a table.
5. To sort record sets.
6. To filter records.

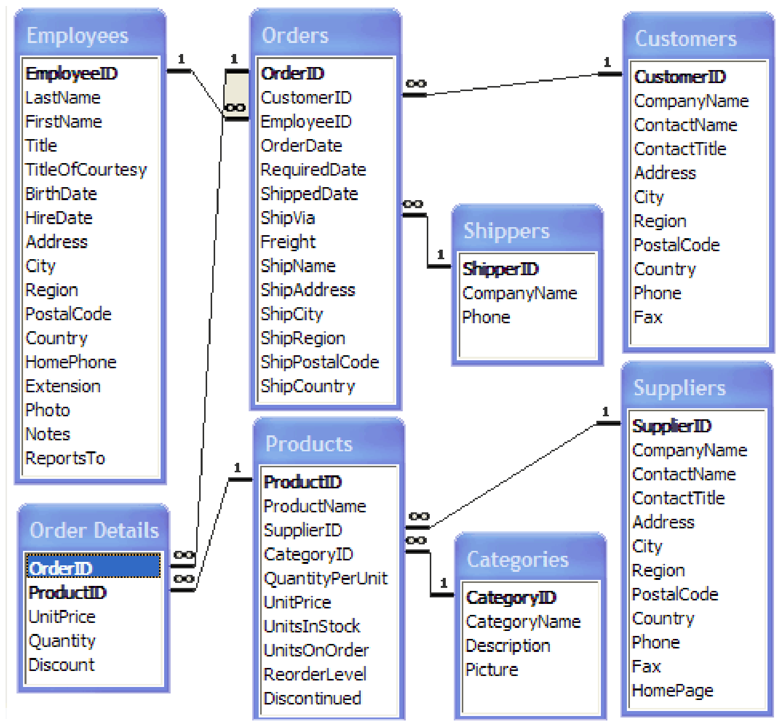
The `SELECT` statement is used to retrieve data from tables. `SELECT` statements can be used to perform simple tasks such as retrieving records from a single table or complicated tasks such as retrieving data from multiple tables with record grouping and sorting. In this lesson, we will look at several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

2.1 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.

2.2 Some Basics

Comments

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

© Copyright 2016 Webucator. All rights reserved.

SQL Comments

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

The code sample below shows some sample comments. Note that MYSQL requires a space after the -- but the other databases do not.

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.

2.3 SELECTing All Columns in All Rows

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

```
Syntax
SELECT table.*
FROM table;

 -- OR

SELECT *
FROM table;
```

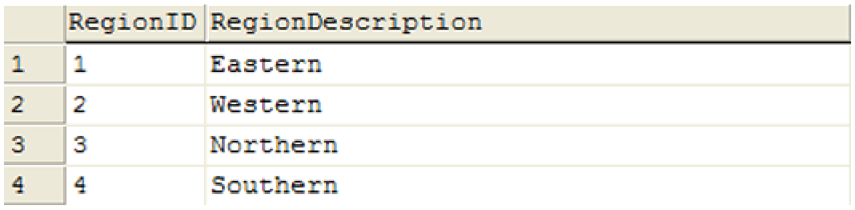
Code Sample

SimpleSelects/Demos/SelectAll.sql

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

Code Explanation

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.

© Copyright 2016 Webucator. All rights reserved.

Exercise 1 Exploring the Tables

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.
   A. Categories (8)
   B. Customers (91)
   C. Employees (9)
   D. Orders (830)
   E. Products (77)
   F. Shippers (3)
   G. Suppliers (29)

Exercise 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;
```

© Copyright 2016 Webucator. All rights reserved.

2.4 SELECTing Specific Columns

While using SELECT * seems like a handy way to display all of the columns in a table, its use is considered a poor practice in a production environment because of potential performance problems. 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;
```

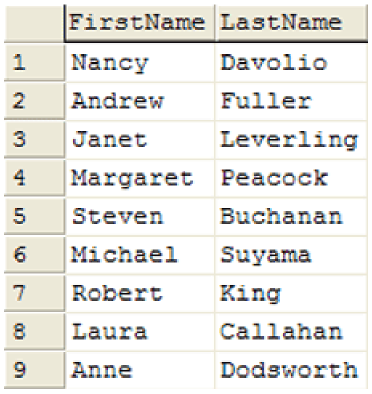
Code Sample

SimpleSelects/Demos/SelectCols.sql

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

Code Explanation

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

Exercise 2 SELECTing Specific Columns

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.

© Copyright 2016 Webucator. All rights reserved.

Exercise 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;
```

© Copyright 2016 Webucator. All rights reserved.

2.5 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.

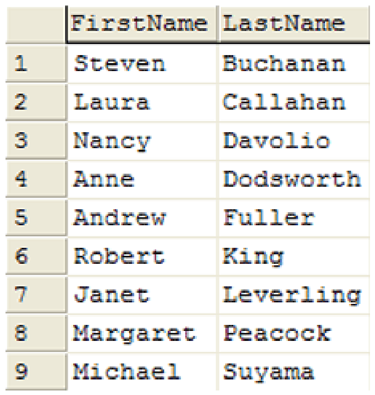
Code Sample

SimpleSelects/Demos/OrderBy1.sql

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

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

Code Explanation

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

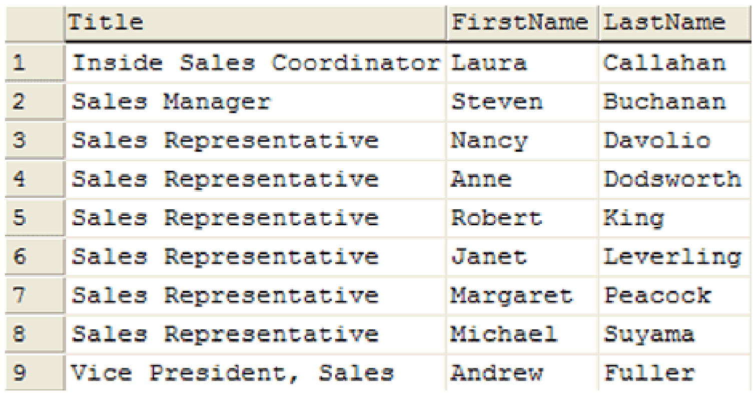
SimpleSelects/Demos/OrderBy2.sql

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

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

© Copyright 2016 Webucator. All rights reserved.

Code Explanation

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_position DESC, column_position ASC;
```

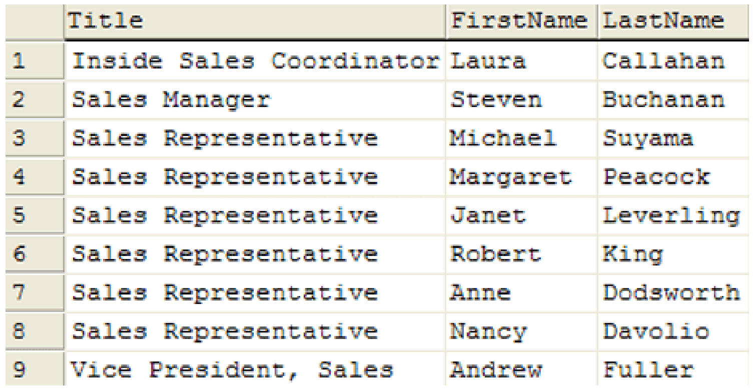
Code Sample

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;
```

Code Explanation

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

© Copyright 2016 Webucator. All rights reserved.

Exercise 3 Sorting Results

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. Note the way your database displays dates. This display is quite often different from database to database.
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.

Exercise Solution

SimpleSelects/Solutions/Sorting.sql

```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;
```

© Copyright 2016 Webucator. All rights reserved.

2.6 The WHERE Clause and Logical 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. Some databases, like Oracle, treat all character data as case sensitive. Other databases only treat character data as case sensitive in a handful of functions. It's always easier to pretend all data is case sensitive in those databases so you don't have to remember when it is and when it isn't. Examples are shown below.

Checking for Equality

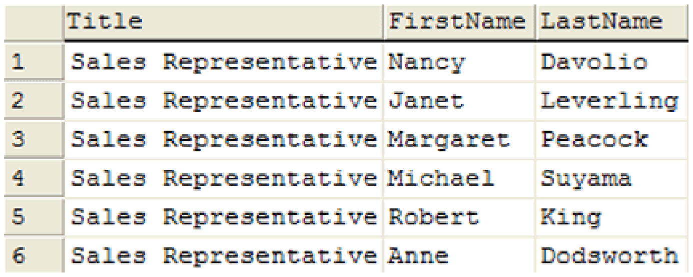
Code Sample

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';
```

Code Explanation

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

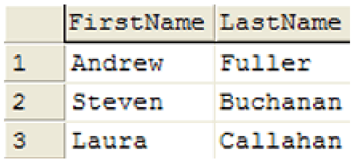
Code Sample

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';
```

Code Explanation

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



	FirstName	LastName
1	Andrew	Fuller
2	Steven	Buchanan
3	Laura	Callahan

© Copyright 2016 Webucator. All rights reserved.

Exercise 4 Using the WHERE Clause to Check for Equality or Inequality

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'`*).*
   - Microsoft users may use either of the above formats.
4. Create a report showing the first name, last name, and country of all employees not in the United States.

Exercise 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';
```

© Copyright 2016 Webucator. All rights reserved.

Checking for Greater or Less Than

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

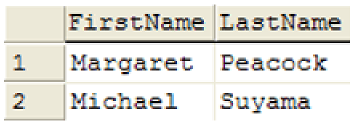
Code Sample

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.
*/

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

Code Explanation

The above SELECT statement will return the following results:



	FirstName	LastName
1	Margaret	Peacock
2	Michael	Suyama

Exercise 5 Using the WHERE Clause to Check for Greater or Less Than

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.

© Copyright 2016 Webucator. All rights reserved.

Exercise Solution

SimpleSelects/Solutions/GreaterThanLessThan.sql

```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;
```

© Copyright 2016 Webucator. All rights reserved.

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 or a 0. 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.

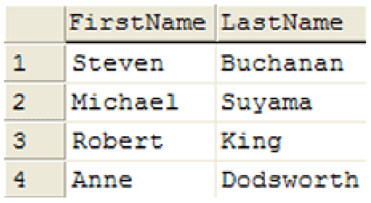
Code Sample

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;
```

Code Explanation

The above SELECT statement will return the following results:



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

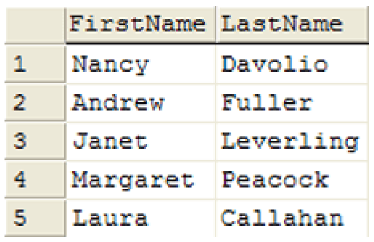
Code Sample

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;
```

Code Explanation

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

© Copyright 2016 Webucator. All rights reserved.

Exercise 6 Checking for NULL

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.

Exercise Solution

SimpleSelects/Solutions/Null.sql

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

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

© Copyright 2016 Webucator. All rights reserved.

WHERE and ORDER BY

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

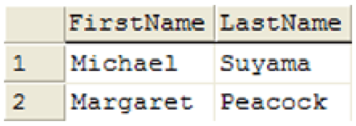
Code Sample

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;
```

Code Explanation

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



	FirstName	LastName
1	Michael	Suyama
2	Margaret	Peacock

Exercise 7 Using WHERE and ORDER BY Together

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.

© Copyright 2016 Webucator. All rights reserved.

Exercise Solution

SimpleSelects/Solutions/Where-OrderBy.sql

```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;
```

© Copyright 2016 Webucator. All rights reserved.

2.7 Checking Multiple Conditions with Boolean Operators

AND

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

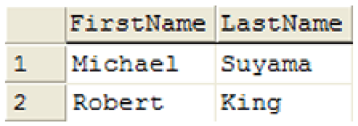
Code Sample

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.';
```

Code Explanation

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.

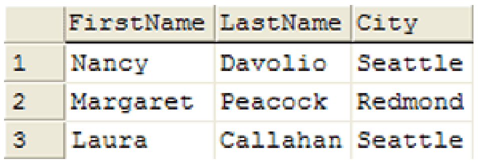
<u>**Code Sample**</u>

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';
```

Code Explanation

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.

<u>**Code Sample**</u>

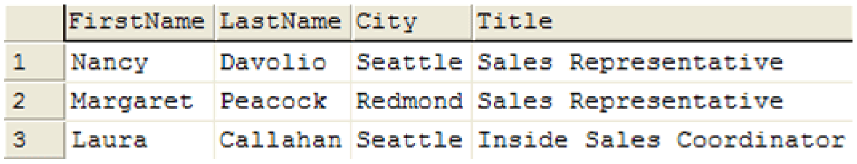
SimpleSelects/Demos/Where-AndOrPrecedence.sql

```
/*
 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';
```

© Copyright 2016 Webucator. All rights reserved.

Code Explanation

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.

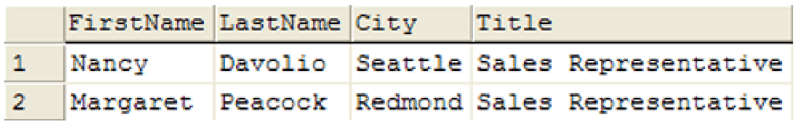
Code Sample

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';
```

Code Explanation

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.

Exercise 8 Writing SELECTs with Multiple Conditions

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.

© Copyright 2016 Webucator. All rights reserved.

Exercise 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');
```

© Copyright 2016 Webucator. All rights reserved.

2.8 The WHERE Clause and Logical Operator Keywords

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.

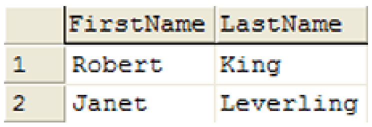
Code Sample

SimpleSelects/Demos/Where-Between.sql

```
/*
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';
```

Code Explanation

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.

<u>Code Sample</u>

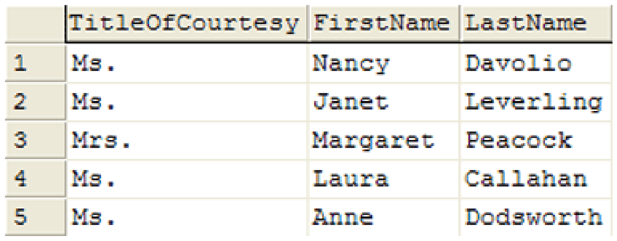
SimpleSelects/Demos/Where-In.sql

```
/*
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.';
```

© Copyright 2016 Webucator. All rights reserved.

Code Explanation

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.

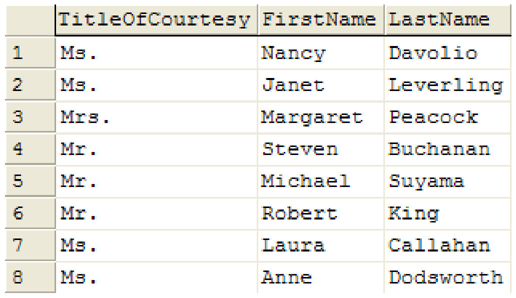
Code Sample

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".
      >>>
*/

SELECT TitleOfCourtesy, FirstName, LastName
FROM Employees
WHERE TitleOfCourtesy LIKE 'M%';
```

Code Explanation

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

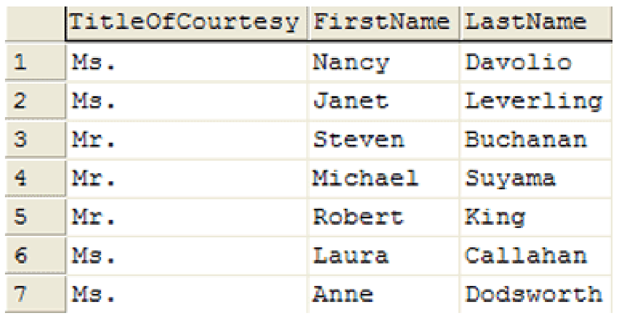
SimpleSelects/Demos/Where-Like2.sql

```
/*
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_.';
```

© Copyright 2016 Webucator. All rights reserved.

Code Explanation

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.

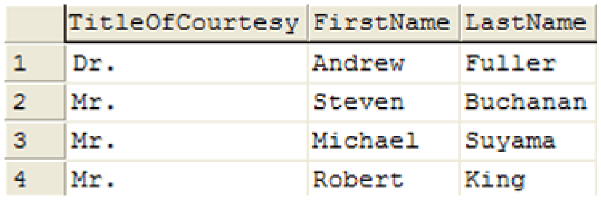
Code Sample

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.');
```

Code Explanation

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

Exercise 9 More SELECTs with WHERE

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. *Hint*: 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, please 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".

© Copyright 2016 Webucator. All rights reserved.

Exercise Solution

SimpleSelects/Solutions/WordOperators.sql

```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 23:59:59';

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

/* The result of the following "HINT" query must be obtained first in
   >>> order to write the solution query for #2 */
select supplierid
from suppliers
where companyname in ('Exotic Liquids',
    'Grandma Kelly"s Homestead',
    'Tokyo Traders');


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%';
```

© Copyright 2016 Webucator. All rights reserved.

2.9 Conclusion

In this lesson, you have learned a lot about creating reports with `SELECT`. However, this is just the tip of the iceberg. `SELECT` statements can get a lot more powerful and, of course, a lot more complicated.

© Copyright 2016 Webucator. All rights reserved.

3. Advanced SELECTs

In this lesson, you will learn...

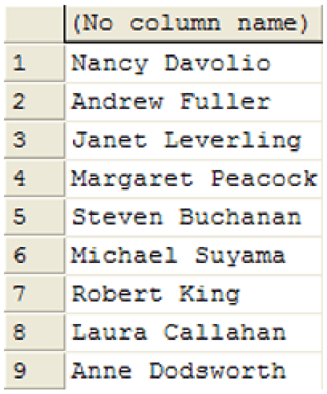
1. To use `SELECT` statements to retrieve calculated values.
2. To work with aggregate functions and grouping.
3. To work with SQL's data manipulation functions.

3.1 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

```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, which is also the ANSI/ISO concatenation operator.

Code Sample

AdvancedSelects/Demos/Concatenate-Oracle.sql

```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 [1].

Code Sample

AdvancedSelects/Demos/Concatenate-MySQL.sql

```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. [2]

Mathematical Calculations

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

1. We'll look at functions more in Built-in Data Manipulation Functions (see page 67).
2. Conversion is covered briefly in Built-in Data Manipulation Functions (see page 67).

© Copyright 2016 Webucator. All rights reserved.

Mathematical Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (Oracle does not support the modulus operator. Instead use the MOD Function. My SQL supports both.)

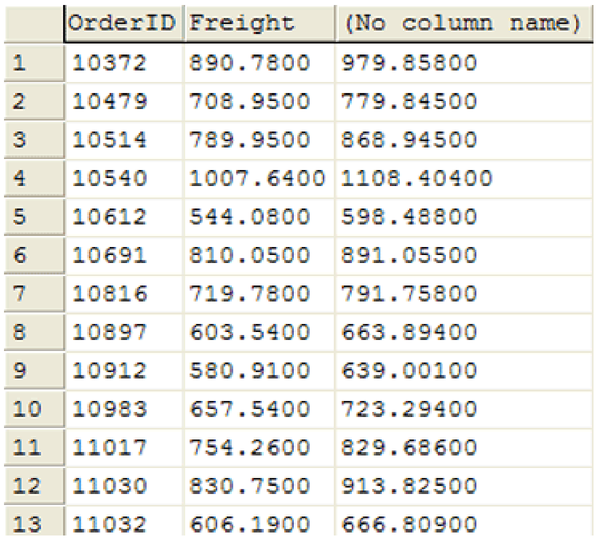
Code Sample

AdvancedSelects/Demos/MathCalc.sql

```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;
```

Code Explanation

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 for Microsoft SQL Server that the calculated columns have the header "`(No column name)`". MySQL and Oracle display the expression as the column header. Regardless, the keyword AS is used to provide a named header for the column.

Note: you cannot use aliases in a WHERE clause!

Code Sample

AdvancedSelects/Demos/Alias.sql

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

Code Explanation

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

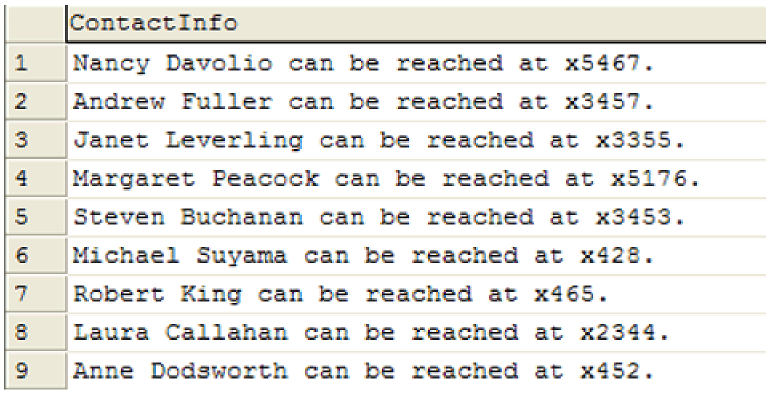
© Copyright 2016 Webucator. All rights reserved.

Exercise 10 Calculating Fields

10 to 20 minutes

In this exercise, you will practice writing `SELECT` statements with calculated fields.

1. Create a report that shows the unit price, quantity, discount, and the calculated total price using these three fields. Using the appropriate columns from the appropriate table, you'll find the total price of each item by using this formula: unitprice * quantity * (1 - discount)
   - Note for SQL Server users only: You will be using the `Order Details` table. Because this table name has a space in it, you will need to put it in double quotes in the `FROM` clause (e.g, `FROM "Order Details"`).
2. Write a `SELECT` statement that outputs the following. It's okay if your result set's rows are in a different order.



	ContactInfo
1	Nancy Davolio can be reached at x5467.
2	Andrew Fuller can be reached at x3457.
3	Janet Leverling can be reached at x3355.
4	Margaret Peacock can be reached at x5176.
5	Steven Buchanan can be reached at x3453.
6	Michael Suyama can be reached at x428.
7	Robert King can be reached at x465.
8	Laura Callahan can be reached at x2344.
9	Anne Dodsworth can be reached at x452.

Exercise Solution

AdvancedSelects/Solutions/Calculations.sql

```
/*******************************
SQL Server Solutions
*******************************/
SELECT UnitPrice, Quantity, Discount, UnitPrice * Quantity * (1-Discount)
   >>>  AS TotalPrice
FROM "Order Details";

SELECT FirstName + ' ' + LastName + ' can be reached at x' + Extension
   >>>  + '.' AS ContactInfo
FROM Employees;

/*******************************
Oracle Solutions
*******************************/
SELECT UnitPrice, Quantity, Discount, UnitPrice * Quantity * (1-Discount)
   >>>  AS TotalPrice
FROM Order_Details;

SELECT FirstName || ' ' || LastName || ' can be reached at x' || Exten »»
   >>> sion || '.' AS ContactInfo
FROM Employees;

/*******************************
MySQL Solutions
*******************************/
SELECT UnitPrice, Quantity, Discount, UnitPrice * Quantity * (1-Discount)
   >>>  AS TotalPrice
FROM Order_Details;

SELECT CONCAT(FirstName, ' ', LastName, ' can be reached at x', Exten »»
   >>> sion, '.') AS ContactInfo
FROM Employees;
```

© Copyright 2016 Webucator. All rights reserved.

3.2 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.
COUNT(*)	Returns the number of rows in the intermediate result set.
`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

```
-- Find the Number of Employees

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

Code Explanation

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;
```

Code Explanation

Returns 328.

Code Sample

AdvancedSelects/Demos/Aggregate-Avg.sql

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

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

Code Explanation

Returns 28.8663.

Code Sample

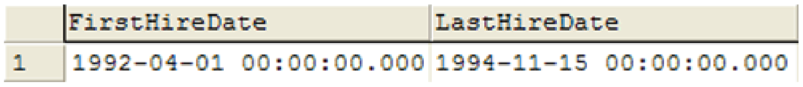
AdvancedSelects/Demos/Aggregate-MinMax.sql

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

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

© Copyright 2016 Webucator. All rights reserved.

Code Explanation

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

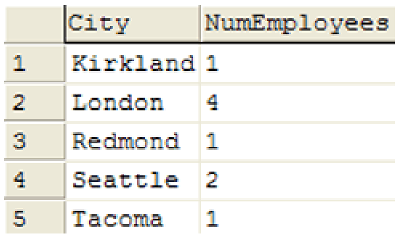
With the `GROUP BY` clause, aggregate functions can be applied to groups of records based on column values. Whenever the SELECT clause of your SELECT statement includes column values and aggregate functions, you must include the GROUP BY clause. The GROUP BY clause must list all of the column values used in the SELECT clause. For example, the following code will return the number of employees in each city (a column value).

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

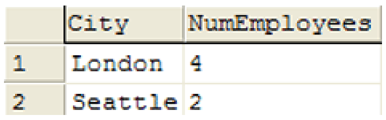
The HAVING clause is used to filter grouped data. Therefore, whenever you want to filter data based on an aggregate function, you do so in the HAVING clause. For example, the following code specifies that we only want information on cities that have more than one employee.

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`

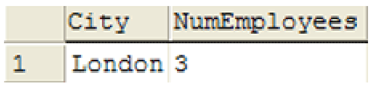
© Copyright 2016 Webucator. All rights reserved.

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.[3]
- You may use aliases in the `ORDER BY` clause.
- When you want to filter out rows from your result set based on aggregate functions, you must use the `HAVING` clause.
- You should declare column alilases for any calculated fields in the SELECT clause.
- You may not use aggregate functions in the `WHERE` clause.

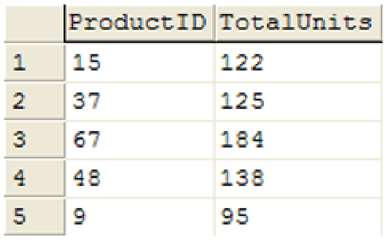
3. MySQL allows usage of aliases in the `HAVING` clause, but you may want to avoid this to keep your code as cross-database compatible as possible.

Exercise 11 Working with Aggregate Functions

10 to 20 minutes

In this exercise, you will practice working with aggregate functions. For all of these questions, it's okay if your result set's rows are in a different order.

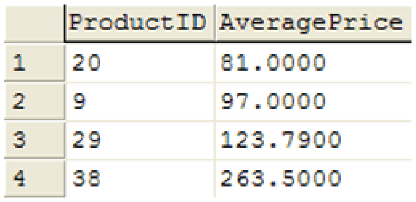
1. Create a report that returns the following from the `Order_Details` table.



	ProductID	TotalUnits
1	15	122
2	37	125
3	67	184
4	48	138
5	9	95

   The report should only return rows for which `TotalUnits` is less than 200.

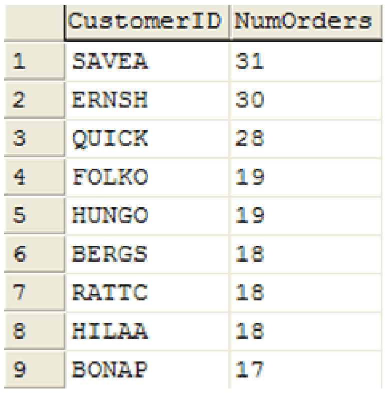
2. Create a report that returns the following from the `Products` table.



	ProductID	AveragePrice
1	20	81.0000
2	9	97.0000
3	29	123.7900
4	38	263.5000

   The report should only return rows for which the average unit price of a product is greater than 70.

3. Create a report that returns the following from the `Orders` table.



	CustomerID	NumOrders
1	SAVEA	31
2	ERNSH	30
3	QUICK	28
4	FOLKO	19
5	HUNGO	19
6	BERGS	18
7	RATTC	18
8	HILAA	18
9	BONAP	17

   `NumOrders` represents the number of orders placed by a certain customer. Only return rows where `NumOrders` is greater than 15.

Query number 2 above has something strange about it. It is, in fact, a ridiculous query. Why? Try to get the exact same results without using an aggregate function.

© Copyright 2016 Webucator. All rights reserved.

Exercise Solution

AdvancedSelects/Solutions/Grouping.sql

```sql
SELECT ProductID, SUM(Quantity) AS TotalUnits
FROM Order_Details /* SQL Server users should use "Order Details" */
GROUP BY ProductID
HAVING SUM(Quantity) < 200;

SELECT ProductID, AVG(UnitPrice) AS AveragePrice
FROM Products
GROUP BY ProductID
HAVING AVG(UnitPrice) > 70
ORDER BY AveragePrice;

SELECT CustomerID, COUNT(OrderID) AS NumOrders
FROM Orders
GROUP BY CustomerID
HAVING COUNT(OrderID) > 15
ORDER BY NumOrders DESC;

/*CHALLENGE ANSWER:*/
SELECT ProductID, UnitPrice
FROM Products
WHERE UnitPrice > 70
ORDER BY UnitPrice;
```

© Copyright 2016 Webucator. All rights reserved.

Selecting Distinct Records

The `DISTINCT` keyword is used to select distinct combinations of column values from the intermediate result set. 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
```

3.3 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

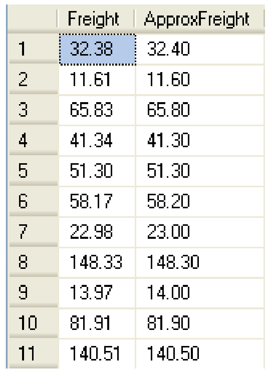
AdvancedSelects/Demos/Functions-Math1.sql

```
/*
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;
```

© Copyright 2016 Webucator. All rights reserved.

Code Explanation

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

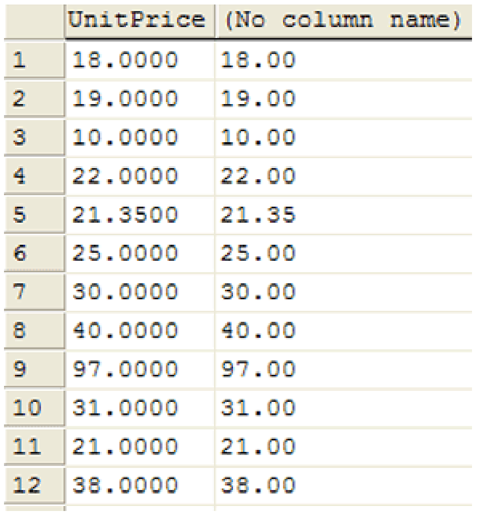
AdvancedSelects/Demos/Functions-Math2.sql

```
/*
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;
```

© Copyright 2016 Webucator. All rights reserved.

Code Explanation

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



```
   UnitPrice  (No column name)
1  18.0000    18.00
2  19.0000    19.00
3  10.0000    10.00
4  22.0000    22.00
5  21.3500    21.35
6  25.0000    25.00
7  30.0000    30.00
8  40.0000    40.00
9  97.0000    97.00
10 31.0000    31.00
11 21.0000    21.00
12 38.0000    38.00
```

Note that you would round to a whole number by passing 0 as the second parameter: `ROUND(field,0)`; and to the tens place by passing -1: `ROUND(field,-1)`.

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
Returns part of a string	SUBSTRING	SUBSTR	SUBSTRING

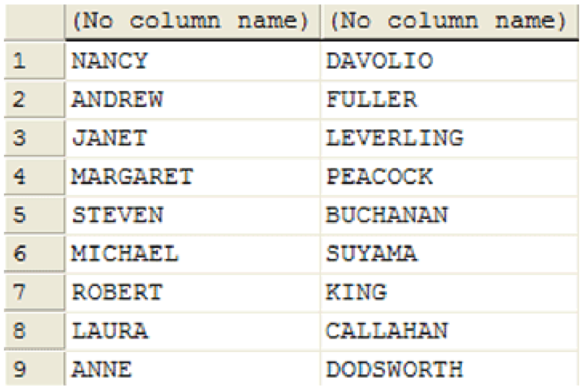
Code Sample

AdvancedSelects/Demos/Functions-String1.sql

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

Code Explanation

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

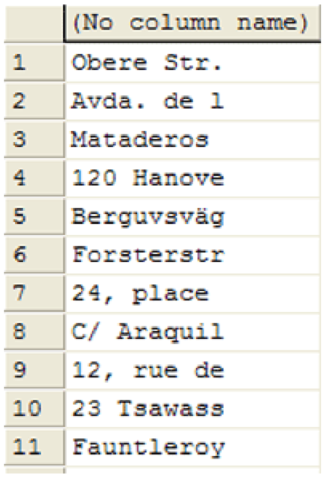
AdvancedSelects/Demos/Functions-String2.sql

```
-- Select the first 10 characters, starting at position 1 of the string,
   >>>   of each customer's address

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

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

© Copyright 2016 Webucator. All rights reserved.

Code Explanation

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



	(No column name)
1	Obere Str.
2	Avda. de l
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
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()
Get current date		CURRENT_DATE	CURRENT_DATE
Get date as number of days since the year 0			TO_DAYS

Code Sample

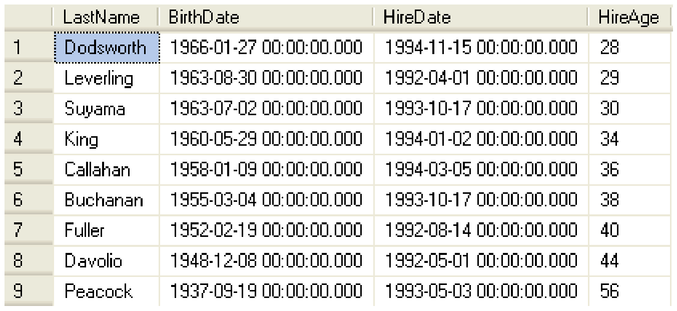
AdvancedSelects/Demos/Functions-Date1.sql

```
-- 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 - Birth »»
   >>> Date)/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 be »»
   >>> tween
-- two dates. You can then divide and floor to get age.
SELECT LastName, BirthDate, HireDate, FLOOR(DATEDIFF(HireDate,Birth »»
   >>> Date)/365) AS HireAge
FROM Employees
ORDER BY HireAge;
```

© Copyright 2016 Webucator. All rights reserved.

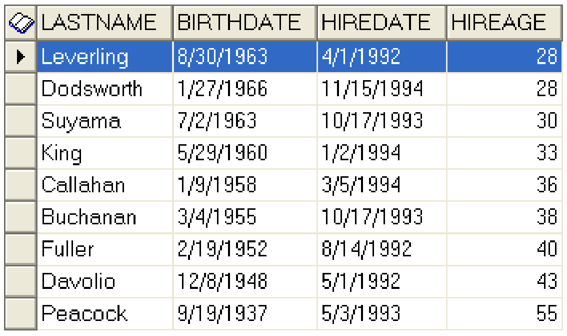
Code Explanation

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. The same note applies to MySQL users who are using prior to version 4.1.1.

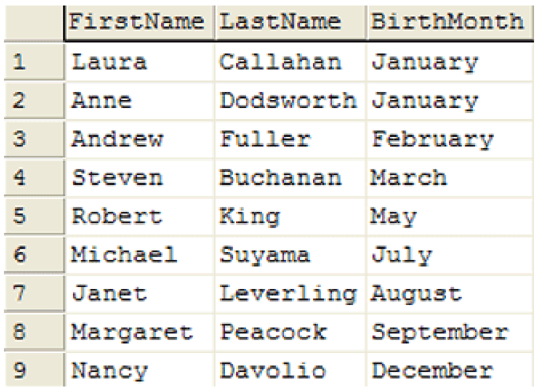
Code Sample

AdvancedSelects/Demos/Functions-Date2.sql

```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);
```

Code Explanation

The above SELECT statement will return the following results:



	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

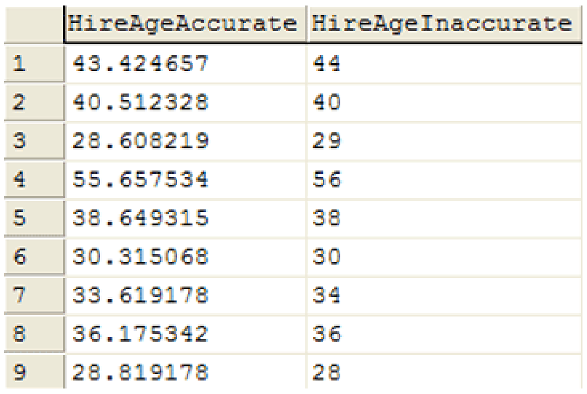
© Copyright 2016 Webucator. All rights reserved.

Exercise 12 Data Manipulation Functions

10 to 20 minutes

In this exercise, you will practice using data manipulation functions.

1. Create a report that shows the units in stock, unit price, the total price value of all units in stock, the total price value of all units in stock rounded down, and the total price value of all units in stock rounded up. Sort by the total price value descending.
2. SQL SERVER AND MYSQL (if using PRIOR to 4.1.1) USERS ONLY: In an earlier demo (see page 74), you saw a report that returned the age of each employee when hired. That report was not entirely accurate as it didn't account for the month and day the employee was born. Fix that report, showing both the original (inaccurate) hire age and the actual hire age. The result will look like this.



	HireAgeAccurate	HireAgeInaccurate
1	43.424657	44
2	40.512328	40
3	28.608219	29
4	55.657534	56
5	38.649315	38
6	30.315068	30
7	33.619178	34
8	36.175342	36
9	28.819178	28

3. Create a report that shows the first and last names and birth month (as a string) for each employee born in the current month. Note: if the current month is April, June, October, or November, you should get no results.
4. Create a report that shows the contact title in all lowercase letters of each customer contact.

Exercise Solution

AdvancedSelects/Solutions/Functions.sql

```sql
/*******************************
SQL Server
*******************************/
SELECT UnitsInStock, UnitPrice,
 UnitsInStock * UnitPrice AS TotalPrice,
 FLOOR(UnitsInStock * UnitPrice) AS TotalPriceDown,
 CEILING(UnitsInStock * UnitPrice) AS TotalPriceUp
FROM Products
ORDER BY TotalPrice DESC;

SELECT DATEDIFF(day,BirthDate,HireDate)/365.25 AS HireAgeAccurate,
 DATEDIFF(year,BirthDate,HireDate) AS HireAgeInaccurate
FROM Employees;

SELECT FirstName, LastName, DATENAME(month,BirthDate) AS BirthMonth
FROM Employees
WHERE DATEPART(month,BirthDate) = DATEPART(month,GETDATE());

SELECT LOWER(ContactTitle) AS Title
FROM Customers;

/*******************************
Oracle
*******************************/
SELECT UnitsInStock, UnitPrice,
 UnitsInStock * UnitPrice AS TotalPrice,
FLOOR(UnitsInStock * UnitPrice) AS TotalPriceDown,
CEIL(UnitsInStock * UnitPrice) AS TotalPriceUp
FROM Products
ORDER BY TotalPrice DESC;

SELECT  FLOOR((HireDate - BirthDate)/365.25) AS HireAgeInAccurate,
 (HireDate - BirthDate)/365.25 AS HireAgeAccurate
FROM Employees;


SELECT FirstName, LastName, TO_CHAR(BirthDate,'MONTH') AS BirthMonth
FROM Employees
WHERE TO_CHAR(BirthDate,'MM') = TO_CHAR(SYSDATE,'MM');
```

© Copyright 2016 Webucator. All rights reserved.

```

SELECT LOWER(ContactTitle) AS Title
FROM Customers;

/******************************
MySQL
******************************/
SELECT UnitsInStock, UnitPrice,
 UnitsInStock * UnitPrice AS TotalPrice,
 FLOOR(UnitsInStock * UnitPrice) AS TotalPriceDown,
 CEILING(UnitsInStock * UnitPrice) AS TotalPriceUp
FROM Products
ORDER BY TotalPrice DESC;

SELECT (TO_DAYS(HireDate)-TO_DAYS(BirthDate))/365.25 AS HireAgeAccurate,
    >>>
 YEAR(HireDate)-YEAR(BirthDate) AS HireAgeInaccurate
FROM Employees;

SELECT FirstName, LastName, DATE_FORMAT(BirthDate, '%M') AS BirthMonth
    >>>
FROM Employees
WHERE EXTRACT(MONTH FROM BirthDate) = EXTRACT(MONTH FROM NOW());

SELECT LOWER(ContactTitle) AS Title
FROM Customers;
```

3.4 Conclusion

In this lesson, you have continued to use `SELECT` to create reports from data stored in a single table. In the next lesson, you will learn to create reports from data in multiple tables.

© Copyright 2016 Webucator. All rights reserved.

4. Subqueries, Joins and Unions

In this lesson, you will learn...

1. To write queries with subqueries.
2. To select columns from multiple tables with joins.
3. To select records from multiple result sets with unions.

4.1 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

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

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

Code Explanation

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);
```

Code Explanation

The above code returns `Comércio 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

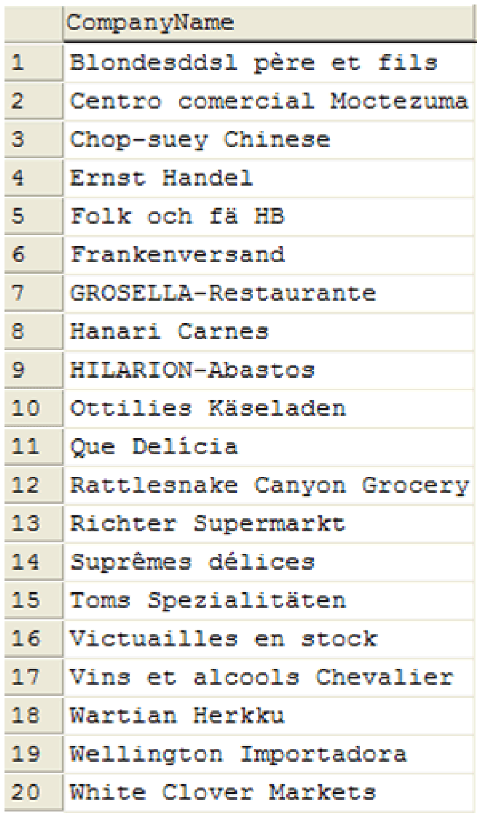
SubqueriesJoinsUnions/Demos/Subquery-IN.sql

```
-- 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');
```

© Copyright 2016 Webucator. All rights reserved.

Code Explanation

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

Exercise 13 Subqueries

20 to 30 minutes

In this exercise, you will practice writing subqueries.

1. 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.
   - You will need to escape the apostrophe in "Grandma Kelly's Homestead." To do so, place another apostrophe in front of it. For example,

     ```
     SELECT *
     FROM Suppliers
     WHERE CompanyName='Grandma Kelly''s Homestead';
     ```

2. Create a report that shows all products by name that are in the Seafood category.
3. Create a report that shows all companies by name that sell products in `CategoryID` 8.
4. Create a report that shows all companies by name that sell products in the Seafood category.

© Copyright 2016 Webucator. All rights reserved.

Exercise Solution

SubqueriesJoinsUnions/Solutions/Subqueries.sql

```sql
SELECT ProductName, SupplierID
FROM Products
WHERE SupplierID IN (SELECT SupplierID
   FROM Suppliers
   WHERE CompanyName IN
   ('Exotic Liquids', 'Grandma Kelly''s Homestead', 'Tokyo Traders'));

SELECT ProductName
FROM Products
WHERE CategoryID = (SELECT CategoryID
   FROM Categories
   WHERE CategoryName = 'Seafood');

SELECT CompanyName
FROM Suppliers
WHERE SupplierID IN (SELECT SupplierID
   FROM Products
   WHERE CategoryID = 8);

SELECT CompanyName
FROM Suppliers
WHERE SupplierID IN (SELECT SupplierID
   FROM Products
   WHERE CategoryID = (SELECT CategoryID
      FROM Categories
      WHERE CategoryName = 'Seafood'));
```

© Copyright 2016 Webucator. All rights reserved.

4.2 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.

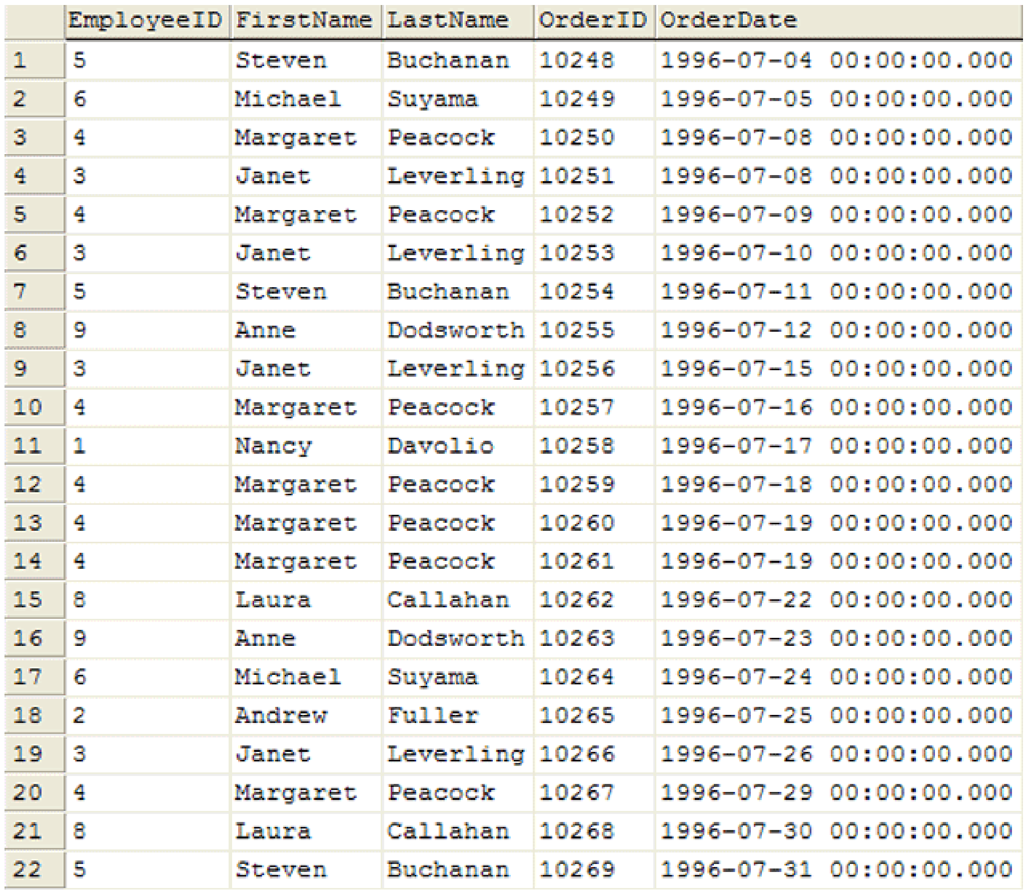
Code Sample

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;
```

Code Explanation

The above SELECT statement will return the following results:



	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. Some people and books call these prefixes "qualifiers".

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.

© Copyright 2016 Webucator. All rights reserved.

Code Sample

SubqueriesJoinsUnions/Demos/Joins-Aliases.sql

```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

```sql
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.

```sql
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

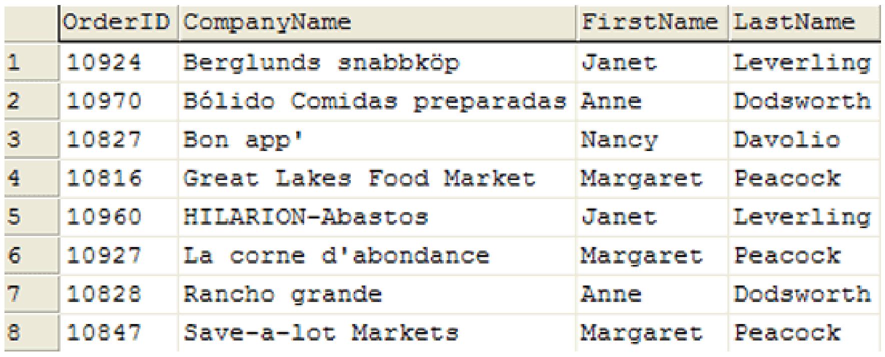
SubqueriesJoinsUnions/Demos/Joins-MultiTable.sql

```
/*
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;
```

© Copyright 2016 Webucator. All rights reserved.

Code Explanation

The above SELECT statement will return the following results:



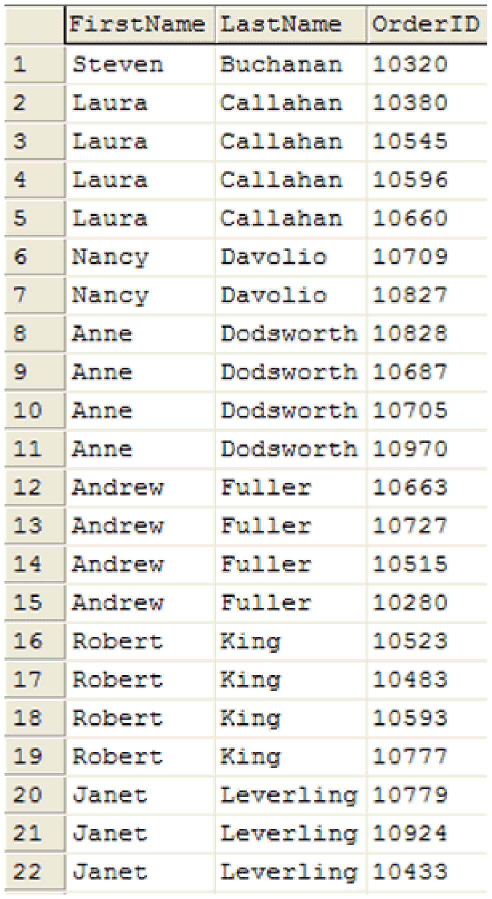
	OrderID	CompanyName	FirstName	LastName
1	10924	Berglunds snabbköp	Janet	Leverling
2	10970	Bólido Comidas preparadas	Anne	Dodsworth
3	10827	Bon app'	Nancy	Davolio
4	10816	Great Lakes Food Market	Margaret	Peacock
5	10960	HILARION-Abastos	Janet	Leverling
6	10927	La corne d'abondance	Margaret	Peacock
7	10828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

Exercise 14 Using Joins

25 to 40 minutes

In this exercise, you will practice using joins.

1. Create a report that shows the order ids and the associated employee names for orders that shipped after the required date. It should return the following. There should be 37 rows returned.

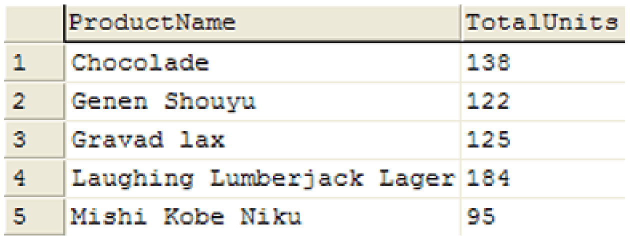


	FirstName	LastName	OrderID
1	Steven	Buchanan	10320
2	Laura	Callahan	10380
3	Laura	Callahan	10545
4	Laura	Callahan	10596
5	Laura	Callahan	10660
6	Nancy	Davolio	10709
7	Nancy	Davolio	10827
8	Anne	Dodsworth	10828
9	Anne	Dodsworth	10687
10	Anne	Dodsworth	10705
11	Anne	Dodsworth	10970
12	Andrew	Fuller	10663
13	Andrew	Fuller	10727
14	Andrew	Fuller	10515
15	Andrew	Fuller	10280
16	Robert	King	10523
17	Robert	King	10483
18	Robert	King	10593
19	Robert	King	10777
20	Janet	Leverling	10779
21	Janet	Leverling	10924
22	Janet	Leverling	10433

2. Create a report that shows the total quantity of products (from the `Order_Details` table) ordered. Only show records for products for which

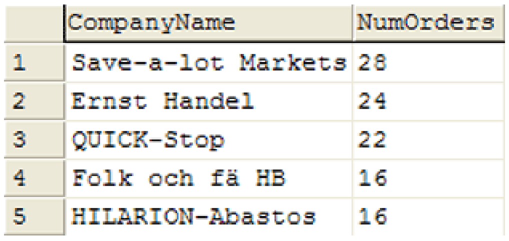
© Copyright 2016 Webucator. All rights reserved.

the quantity ordered is fewer than 200. The report should return the following 5 rows.



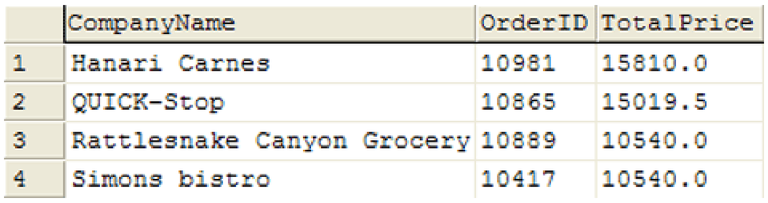
	ProductName	TotalUnits
1	Chocolade	138
2	Genen Shouyu	122
3	Gravad lax	125
4	Laughing Lumberjack Lager	184
5	Mishi Kobe Niku	95

3. Create a report that shows the total number of orders by Customer since December 31, 1996. The report should only return rows for which the NumOrders is greater than 15. The report should return the following 5 rows.



	CompanyName	NumOrders
1	Save-a-lot Markets	28
2	Ernst Handel	24
3	QUICK-Stop	22
4	Folk och fä HB	16
5	HILARION-Abastos	16

4. Create a report that shows the company name, order id, and total price of all products of which Northwind has sold more than $10,000 worth. There is no need for a `GROUP BY` clause in this report.



	CompanyName	OrderID	TotalPrice
1	Hanari Carnes	10981	15810.0
2	QUICK-Stop	10865	15019.5
3	Rattlesnake Canyon Grocery	10889	10540.0
4	Simons bistro	10417	10540.0

Exercise Solution

SubqueriesJoinsUnions/Solutions/Joins.sql

```
SELECT e.FirstName, e.LastName, o.OrderID
FROM Employees e JOIN Orders o ON
 (e.EmployeeID = o.EmployeeID)
WHERE o.RequiredDate < o.ShippedDate
ORDER BY e.LastName, e.FirstName;

SELECT p.ProductName, SUM(od.Quantity) AS TotalUnits
FROM Order_Details od JOIN Products p ON
 (p.ProductID = od.ProductID)
GROUP BY p.ProductName
HAVING SUM(Quantity) < 200;

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

Oracle Solution
******************************/
SELECT c.CompanyName, COUNT(o.OrderID) AS NumOrders
FROM Customers c JOIN Orders o ON
 (c.CustomerID = o.CustomerID)
WHERE OrderDate > '31-Dec-1996'
GROUP BY c.CompanyName
HAVING COUNT(o.OrderID) > 15
ORDER BY NumOrders DESC;

/******************************
MySQL
******************************/
SELECT c.CompanyName, COUNT(o.OrderID) AS NumOrders
FROM Customers c JOIN Orders o ON
 (c.CustomerID = o.CustomerID)
WHERE OrderDate > '1996-12-31'
GROUP BY c.CompanyName
HAVING COUNT(o.OrderID) > 15
ORDER BY NumOrders DESC;

SELECT c.CompanyName, o.OrderID,
 od.UnitPrice * od.Quantity * (1-od.Discount) AS TotalPrice
```

© Copyright 2016 Webucator. All rights reserved.

```
FROM Order_Details od
 JOIN Orders o ON (o.OrderID = od.OrderID)
 JOIN Customers c ON (c.CustomerID = o.CustomerID)
WHERE od.UnitPrice * od.Quantity * (1-od.Discount) > 10000
ORDER BY TotalPrice DESC;

/*
 SQL Server users will replace Order_Details with "Order Details"
*/
```

4.3 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. List the number of employees and customers from each city that has both employees and customers in it.

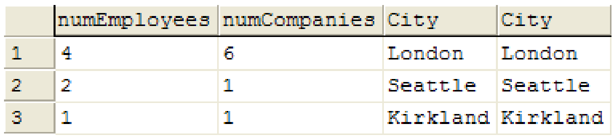
Code Sample

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;
```

Code Explanation

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.

© Copyright 2016 Webucator. All rights reserved.

```
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`.

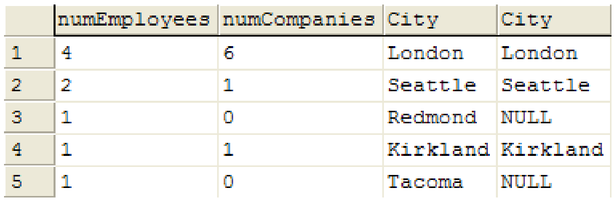
Code Sample

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;
```

Code Explanation

All records in the `Employees` table will be counted whether or not there are matching cities in the `Customers` table. The results are shown below:



	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

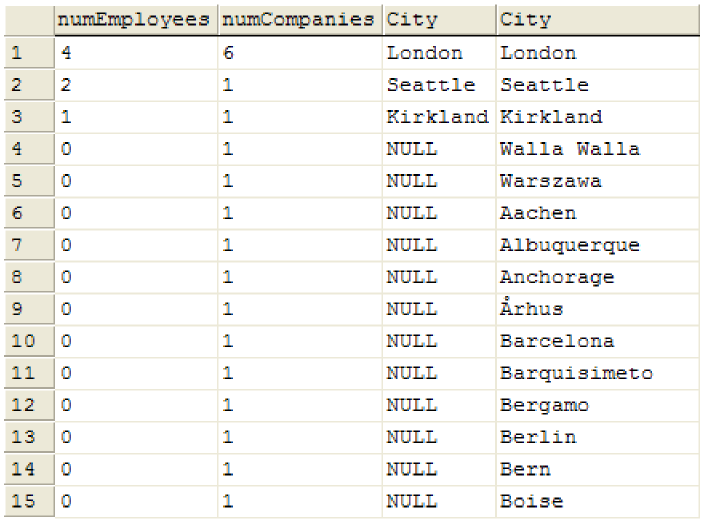
SubqueriesJoinsUnions/Demos/OuterJoins-Right.sql

```
/*
 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;
```

© Copyright 2016 Webucator. All rights reserved.

Code Explanation

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

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

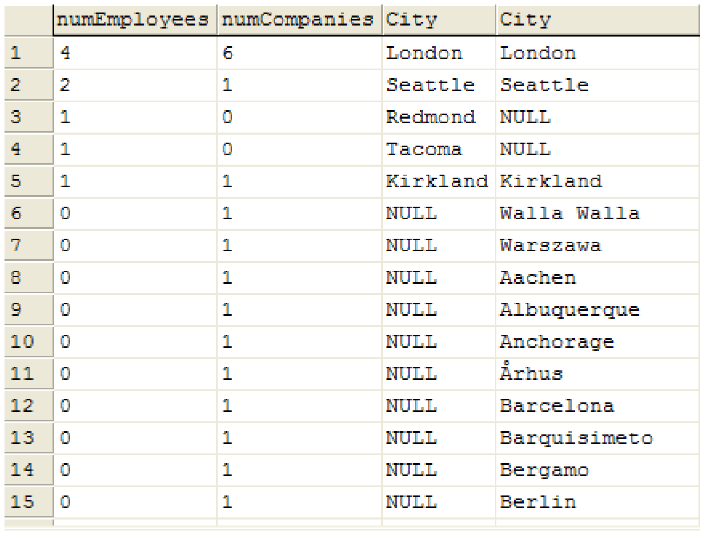
SubqueriesJoinsUnions/Demos/OuterJoins-Full.sql

```
/*
 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;
```

© Copyright 2016 Webucator. All rights reserved.

Code Explanation

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

4.4 Unions

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

In the last section, we noted that MySQL 5.x does not support full outer joins. However, we can simulate a full outer join in MySQL with the help of the `UNION` operator.

Code Sample

SubqueriesJoinsUnions/Demos/MySQLSimulatedFullJoin.sql

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

Code Explanation

This query returns the same result in MySQL as the full outer join code sample does in SQL Server and Oracle.

Code Sample

SubqueriesJoinsUnions/Demos/Unions.sql

```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;
```

Code Explanation

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

© Copyright 2016 Webucator. All rights reserved.

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 if the column is displaying a literal value.

Exercise 15 Working with Unions

10 to 20 minutes

In this exercise, you will practice using `UNION`.

1. Create a report showing the contact name and phone numbers for all employees, customers, and suppliers.

© Copyright 2016 Webucator. All rights reserved.

Exercise Solution

SubqueriesJoinsUnions/Solutions/Unions.sql

```
/*****************************
SQL Server Solution
*****************************/
SELECT FirstName + ' ' + LastName AS Contact, HomePhone As Phone
FROM Employees
 UNION
SELECT ContactName, Phone
FROM Customers
 UNION
SELECT ContactName, Phone
FROM Suppliers
ORDER BY Contact;

/*****************************
Oracle Solution
*****************************/
SELECT FirstName || ' ' || LastName AS Contact, HomePhone As Phone
FROM Employees
 UNION
SELECT ContactName AS Contact, Phone
FROM Customers
 UNION
SELECT ContactName AS Contact, Phone
FROM Suppliers
ORDER BY Contact;

/*****************************
MySQL Solution
*****************************/
SELECT CONCAT(FirstName, ' ', LastName) AS Contact, HomePhone As Phone
   >>>
FROM Employees
 UNION
SELECT ContactName, Phone
FROM Customers
 UNION
SELECT ContactName, Phone
FROM Suppliers
ORDER BY Contact;
```

© Copyright 2016 Webucator. All rights reserved.

4.5 Conclusion

In this lesson, you have learned to create reports using data from multiple tables.

© Copyright 2016 Webucator. All rights reserved.

5. Conditional Processing with CASE

In this lesson, you will learn...

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

5.1 Using CASE

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

Syntax

```
-- OPTION 1: The Simple or Selected Case
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: The Searched Case
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

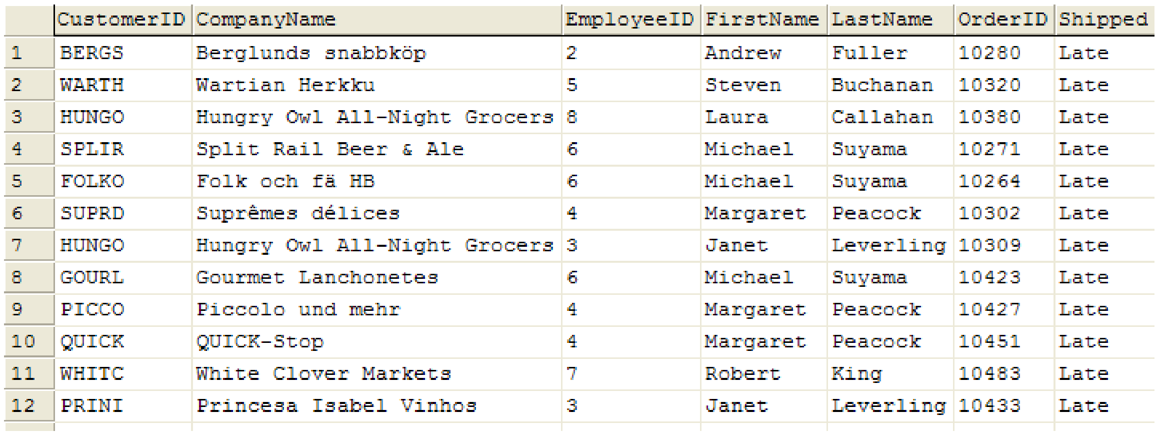
Case/Demos/Case.sql

```
/*
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.Last »»
    >>> Name, 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;
```

© Copyright 2016 Webucator. All rights reserved.

Code Explanation

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



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

Code Sample

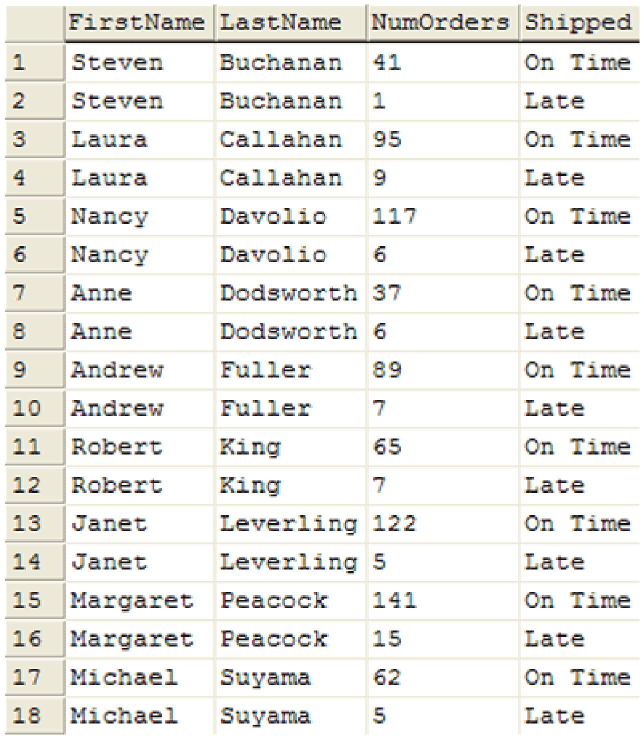
Case/Demos/Case-GroupBy.sql

```
/*
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;
```

© Copyright 2016 Webucator. All rights reserved.

Code Explanation

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



	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

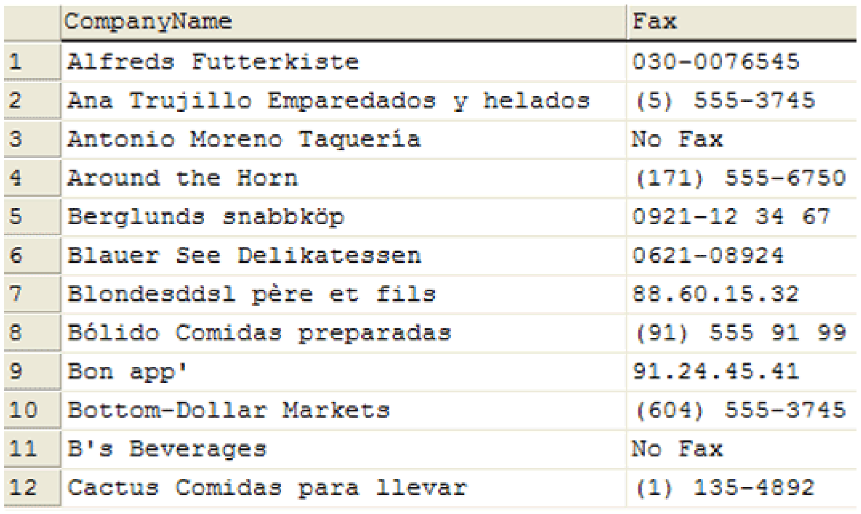
Notice how the `GROUP BY` clause contains the same `CASE` statement that is in the `SELECT` clause. This is required because all non-aggregate columns in the `SELECT` clause must also be in the `GROUP BY` clause and the `GROUP BY` clause cannot contain aliases defined in the `SELECT` clause.

Exercise 16 Working with CASE

10 to 15 minutes

In this exercise you will practice using CASE.

1. Create a report that shows the company names and faxes for all customers. If the customer doesn't have a fax, the report should show "No Fax" in that field as shown below.



	CompanyName	Fax
1	Alfreds Futterkiste	030-0076545
2	Ana Trujillo Emparedados y helados	(5) 555-3745
3	Antonio Moreno Taquería	No Fax
4	Around the Horn	(171) 555-6750
5	Berglunds snabbköp	0921-12 34 67
6	Blauer See Delikatessen	0621-08924
7	Blondesddsl père et fils	88.60.15.32
8	Bólido Comidas preparadas	(91) 555 91 99
9	Bon app'	91.24.45.41
10	Bottom-Dollar Markets	(604) 555-3745
11	B's Beverages	No Fax
12	Cactus Comidas para llevar	(1) 135-4892

© Copyright 2016 Webucator. All rights reserved.

Exercise Solution

Case/Solutions/Case.sql

```
SELECT CompanyName,
 (CASE
  WHEN Fax IS NULL
   THEN 'No Fax'
   ELSE Fax
 END)
 AS Fax
FROM Customers;
```

© Copyright 2016 Webucator. All rights reserved.

5.2 Conclusion

In this lesson, you learned about using `CASE` to output different values in reports based on data contained in table fields.

© Copyright 2016 Webucator. All rights reserved.

A1. Inserting, Updating and Deleting Records

In this lesson, you will learn...

1. To insert records into a table.
2. To update records in a table.
3. To delete records from a table.

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

A1.1 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');
```

Code Explanation

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

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

© Copyright 2016 Webucator. All rights reserved.

Exercise 17 Inserting Records

5 to 15 minutes

In this exercise, you will practice inserting records.

1. Insert yourself into the `Employees` table.
   - Include the following fields: `LastName`, `FirstName`, `Title`, `TitleOfCourtesy`, `BirthDate`, `HireDate`, `City`, `Region`, `PostalCode`, `Country`, `HomePhone`, `ReportsTo`
2. Insert an order for yourself in the `Orders` table.
   - Include the following fields: `CustomerID`, `EmployeeID`, `OrderDate`, `RequiredDate`
3. Insert order details in the `Order_Details` table.
   - Include the following fields: `OrderID`, `ProductID`, `UnitPrice`, `Quantity`, `Discount`

Exercise Solution

InsertsUpdatesDeletes/Solutions/Inserts.sql

```
/******************************
Oracle Solution
******************************/
INSERT INTO Employees
(EmployeeID, LastName, FirstName, Title, TitleOfCourtesy,
 BirthDate, HireDate, City, Region,
 PostalCode, Country, HomePhone, ReportsTo)
VALUES(10, 'Dunn','Nat','Trainer','Mr.',
 '01-Feb-1970','04-Mar-1997','Jamesville','NY',
 '13078','USA','315-555-5555','1');

/******************************

SQL Server & MySQL Solution
******************************/
INSERT INTO Employees
(LastName, FirstName, Title, TitleOfCourtesy,
 BirthDate, HireDate, City, Region,
 PostalCode, Country, HomePhone, ReportsTo)
VALUES('Dunn','Nat','Trainer','Mr.',
 '1970-02-01','1997-03-04','Jamesville','NY',
 '13078','USA','315-555-5555','1');

/******************************

Oracle Solution
******************************/
INSERT INTO Orders
(OrderID, CustomerID, EmployeeID, OrderDate, RequiredDate)
VALUES(11078, 'COMMI',10,'24-May-2004','12-Jul-2005');

/******************************
SQL Server & MySQL Solution
******************************/
INSERT INTO Orders
(CustomerID, EmployeeID, OrderDate, RequiredDate)
VALUES('COMMI',10,'2005-05-24','2005-07-12');

INSERT INTO Order_Details
```

© Copyright 2016 Webucator. All rights reserved.

```
(OrderID, ProductID, UnitPrice, Quantity, Discount)
VALUES(11078, 3, 10, 100, .1);

/*
 SQL Server users will replace Order_Details with "Order Details"

*/
```

A1.2 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';
```

Code Explanation

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

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

A1.3 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;
```

© Copyright 2016 Webucator. All rights reserved.

Code Sample

InsertsUpdatesDeletes/Demos/Delete.sql

```sql
-- Delete an Employee

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

Code Explanation

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

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

Exercise 18 Updating and Deleting Records

5 to 15 minutes

In this exercise, you will practice updating and deleting records.

1. Update your record in the `Employees` table to include some `Notes`.
2. Raise the unit price of all products in the `Products` table by 10% for all products that are out of stock. This should affect 5 rows.
3. Try to delete yourself from the `Employees` table. Could you?
4. If you were not allowed to delete yourself from the `Employees` table, figure out what other records you have to delete so that you can.

© Copyright 2016 Webucator. All rights reserved.

Exercise Solution

InsertsUpdatesDeletes/Solutions/UpdateDelete.sql

```
UPDATE Employees
SET Notes = 'Nat does not really work here.'
WHERE FirstName = 'Nathaniel' AND LastName = 'Dunn';

UPDATE Products
SET UnitPrice = UnitPrice * 1.1
WHERE UnitsInStock = 0;

DELETE FROM "Order Details"
WHERE OrderID = 11078;
-- Note that your OrderID might be different

DELETE FROM Orders
WHERE OrderID = 11078;
-- Note that your OrderID might be different

DELETE FROM Employees
WHERE FirstName = 'Nathaniel' AND LastName = 'Dunn';
```

© Copyright 2016 Webucator. All rights reserved.

A1.4 Conclusion

In this lesson, you have learned how to insert, update, and delete records. Remember to be careful with updates and deletes. If you forget or mistype the `WHERE` clause, you could lose a lot of data.