Webucator's Free SQL Tutorial

Lesson: Subqueries, Joins and Unions

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

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

Lesson Goals

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

Subqueries

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

Code Sample:

SubqueriesJoinsUnions/Demos/Subquery-SelectCustomerID.sql

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

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

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

Code Sample:

SubqueriesJoinsUnions/Demos/Subquery-SelectCompanyName.sql

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

```

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

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

The above SELECT statement will return the following results:

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

Date ranges seem wrong

Hier ist die angezeigte Ergebnismenge nicht korrekt! Das Ergebnis enthält 86 Zeilen, beginnt alphabetisch sortiert mit 'Alfreds Futterkiste' und endet mit 'Wolski Zaiazd'.

Subqueries

Duration: 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.

Solution:

SubqueriesJoinsUnions/Solutions/Subqueries.sql

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;
```

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.

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.

An alias can be called whatever you want. Though typically it's the first letter(s) of the table name, it can be whatever makes sense to you as the Developer. For example, the alias for a table called ***Courses*** can be ***c***, or ***crs***, or ***debbie***, etc.

Code Sample:

SubqueriesJoinsUnions/Demos/Joins-Aliases.sql

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

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

Multi-table Joins

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

Syntax

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

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

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

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

Code Sample:

SubqueriesJoinsUnions/Demos/Joins-MultiTable.sql

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

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

Oracle
*******************************/
```

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

The above SELECT statement will return the following results:

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

Using Joins

Duration: 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 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.

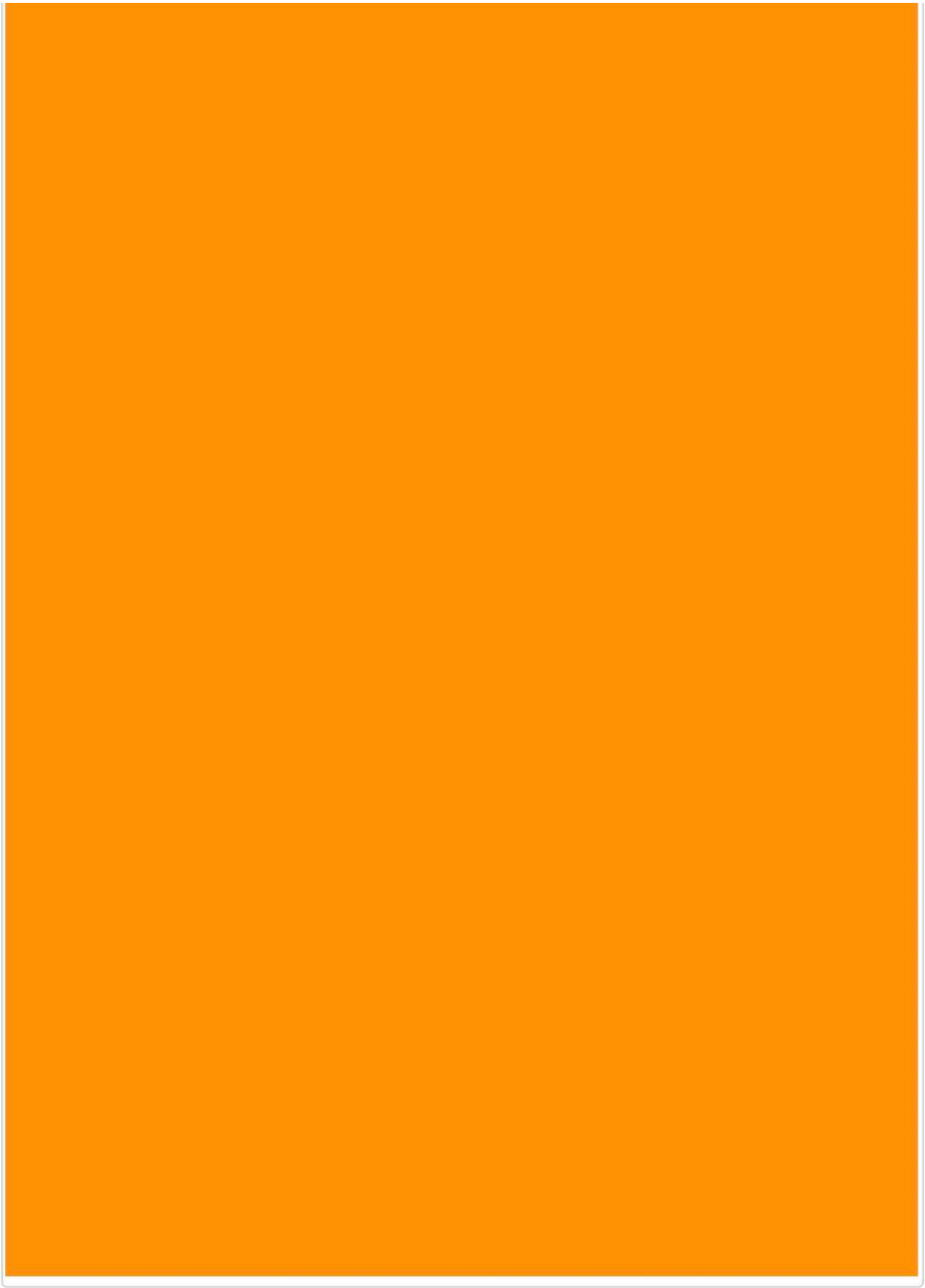
	CompanyName	NumOrders
1	Save-a-lot Markets	28
2	Ernst Handel	24
3	QUICK-Stop	22
4	Folk och fä HB	16
5	HILARION-Abastos	16

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

Solution:

SubqueriesJoinsUnions/Solutions/Joins.sql



Outer Joins

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

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

Code Sample:

SubqueriesJoinsUnions/Demos/OuterJoins-Inner.sql

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

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

The above SELECT statement will return the following results:

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

Left Joins

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

Syntax

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

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

Code Sample:

SubqueriesJoinsUnions/Demos/OuterJoins-Left.sql

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

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.
*/
```

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.
*/
```

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

Unions

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

Code Sample:

SubqueriesJoinsUnions/Demos/Unions.sql

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

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

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

UNION ALL

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

UNION Rules

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

Working with Unions

Duration: 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.

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

```
/*******************************
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;
```

Next Lesson: Conditional Processing with CASE ➞
(https://www.webucator.com/tutorial/learn-sql/conditional-processing-with-case.cfm)

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

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

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

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

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

Programming Training
(https://www.webucator.com/programming-training/index.cfm)

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

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

Business Skills Training
(https://www.webucator.com/business-skills-training/index.cfm)

Web Application Server Training
(https://www.webucator.com/web-application-servers-training/index.cfm)