Course Number & Title: Using Structured Query language (SQL) Syntax

Course Author(s): Richard Patlan
Author(s) Affiliation: www.dbwebsite.com
Phone: 626-221-8435
Fax:
E-Mail: rpatlan@dbwebsite.com

Note: Don't worry if results do not match exercise results.

Exercise 14:

Table Joins

Types of Joins

By using joins, you can retrieve data from two or more tables based on logical relationships between the tables. Joins indicate how SQL statements should use data from one table to SELECT the rows in another table.

A join condition defines the way two tables are related in a query by:

- Specifying the column from each table to be used for the join. A typical join condition specifies a foreign key from one table and its associated key in the other table.

- Specifying a logical operator (for example, = or <>,) to be used in comparing VALUES from the columns.

INNER Joins

An inner join is a join in which the VALUES in the columns being joined are compared using a comparison operator. This inner join is known as an equi-join. It returns all the columns in both tables, and returns only the rows for which there is an equal VALUE in the join column. Inner joins eliminate the rows that do not match with a row from the other table.

In the SQL-92 standard, inner joins can be specified in either the FROM or WHERE clause. This is the only type of join that SQL-92 supports in the WHERE clause. Inner joins specified in the WHERE clause are known as old-style inner joins.

EXAMPLE:

The following SELECT statement returns the ProductID and Name from the Product table. This is a simple SELECT statement:

Write Exercise Code:

```
SELECT
       ProductID,
       Name
FROM
       Product
```

Result

The following SELECT statement is an example of using INNER JOIN in SELECT statement and returns all data from both Employeer and Contact tables that match ContactID. This is because we did not specify field names and used the asters * which brings back all fields.,

Write Exercise Code:

```
SELECT *
FROM Employee AS e
    INNER JOIN Contact AS c
    ON e.ContactID = c.ContactID
ORDER BY c.LastName
```

Result

The following SELECT statement modifies first SELECT statement and uses an INNER JOIN to return all data from both Productvendor and Product tables that match ProductID:

Write Exercise Code:

```
SELECT
       ProductVendor.VendorID,
       Product.ProductID,
       Name
FROM
       ProductVendor
       INNER JOIN Product ON
       ProductVendor.ProductID = Product.ProductID
```

Result

The following SELECT statement modifies first SELECT statement and uses an INNER join to return all data from both Productvendor and Product tables that match ProductID. It also only returns the top 10 records and includes an ORDER BY clause

Write Exercise Code:

```
SELECT
       Top 10 ProductVendor.VendorID,
       Product.ProductID,
       Name
FROM
       ProductVendor
       INNER JOIN Product ON
       ProductVendor.ProductID = Product.ProductID
ORDER BY
       VendorID
```

Result

In this example we are joining between the SalesOrderDetail and Product tables. The tables are aliased with the following: SOD for SalesOrderDetail and P for Product. The JOIN logic is based on matching records in the SOD.ProductID and P.ProductID columns. The records are filtered by only returning records with the SOD.UnitPrice greater than 1000. Finally, the result set is returned in order with the most expensive first based on the ORDER BY clause and only the highest 100 products based on the TOP clause.

Write Exercise Code:

```
SELECT  TOP 100 P.ProductID,
 P.Name,
 P.ListPrice,
 P.Size,
 P.ModifiedDate,
 SOD.UnitPrice,
 SOD.UnitPriceDiscount,
 SOD.OrderQty,
 SOD.LineTotal
FROM SalesOrderDetail SOD
INNER JOIN Product P
 ON SOD.ProductID = P.ProductID
WHERE SOD.UnitPrice > 1000
ORDER BY SOD.UnitPrice DESC
```

Result

The following SELECT statement is an example of using Implicit JOIN in SELECT statement and returns all data from both Productvendor and Product tables that match ProductID:

Write Exercise Code:

```
SELECT
       ProductVendor.VendorID,
       Product.ProductID,
       Product.Name
FROM
       ProductVendor, Product
WHERE ProductVendor.ProductID = Product.ProductID
```

Result

Exercise 15:

OUTER Joins

Inner joins eliminate the rows that do not match with a row from the other table.

Outer joins, however, return all rows from at least one of the tables or views mentioned in the FROM clause, as long as those rows meet any WHERE or HAVING search conditions. All rows are retrieved from the left table referenced with a left outer join, and all rows from the right table referenced in a right outer join.

EXAMPLE:

To include all customers regardless if they made an order, use a left outer join. Here is the SQL query and results of the left outer join:

Write Exercise Code:

```
SELECT
	Contact.LastName,
	SalesOrderHeader.SalesOrderID
FROM
	Contact Left Outer Join SalesOrderHeader   ON
		Contact.ContactID = SalesOrderHeader.ContactID
```

Result

```
LastName                                           SalesOrderID
-------------------------------------------------- ------------
Mitzner                                            53496
Mitzner                                            58937
Mitzner                                            65244
Mitzner                                            71816
(Not all rows displayed)

(32318 row(s) affected)
```

Exercise 16:

SELF Joins

A table can be joined to itself in a self-join. For example, you can use a self-join to find the Employee and the Manager.

Because this query involves a join of the Employee table with itself, the Employee table appears in two roles. To distinguish these roles, you must give the Employee table two different **aliases** (e1 and 22) in the FROM clause. These aliases are used to qualify the column names in the rest of the query.

EXAMPLE:

This is an example of the self-join statement:

Write Exercise Code:

```
SELECT
      e1.ManagerID AS ManagerID,
      e2.EmployeeID EmployeeID
FROM Employee e1
      INNER JOIN Employee e2
      ON e2.ManagerID = e1.EmployeeID
 ORDER BY e1.ManagerID
```

Result

Now let's add the employee Title

Write Exercise Code:

```
SELECT
      e1.ManagerID,
      e1.Title,
      e2.EmployeeID,
      e2.Title
FROM   Employee e1
       JOIN Employee e2
       ON e2.ManagerID = e1.EmployeeID
 ORDER BY e1.ManagerID
```

Result

Exercise 17:

Multiple Table Joins

Although each join specification joins only two tables, the FROM clauses can contain multiple join specifications. This allows many tables to be joined for a single query.

EXAMPLE:

The ProductVendor table offers a good example of a situation in which joining more than two tables is helpful. The following SQL query finds the names of all products of a particular subcategory and the names of their vendors where ProductSubCategoryID = 15:

Write Exercise Code:

```
SELECT
       p.Name As ProductName,
       v.Name AS VendorName
FROM
       Product p
       INNER JOIN ProductVendor pv
              ON p.ProductID = pv.ProductID
       INNER JOIN Vendor v
              ON pv.VendorID = v.VendorID
WHERE
       ProductSubcategoryID = 15
ORDER BY
       v.Name
```

Result

```
ProductName                                VendorName
------------------------------------------ --------------------
LL Mountain Seat/Saddle                     Chicago City Saddles
ML Mountain Seat/Saddle                     Chicago City Saddles
HL Mountain Seat/Saddle                     Chicago City Saddles
(Not all rows displayed)
```

Joins with Aggregate Functions

To retrieve a list of all customers and the number of orders that each has placed we can use a join with an aggregate function

EXAMPLE:

Write Exercise Code:

```
Select
   c.ContactID,
   c.LastName,
   Count(s.SalesOrderID) AS OrderID
FROM
       Contact c Inner Join SalesOrderHeader s ON
       c.ContactID = s.ContactID
Group BY
   c.LastName,
   c.ContactID
```

Result

contactID	LastName	OrderID
1	Achong	7
2	Abel	4
3	Abercrombie	12
4	Acevedo	11
5	Ackerman	4
6	Adams	12

(Not all rows displayed)

Exercise 18:

Combined Queries

Some queries are combined using the UNION operator. The UNION combines the results of two or more queries INTO a single result set that includes all the rows that belong to all queries in the union. The UNION operation is different from using joins that combine columns from two tables.

The following are basic rules for combining the result sets of two queries by using UNION:

- The number and the order of the columns must be the same in all queries.
- The data types must be compatible.

EXAMPLE:

The first SELECT statement returns all products with a price of more than 42.2100 The second SELECT statement uses the IN to find all products made by vendors 1001 and 1002. By default the UNION operator removes duplicates. To add duplicates just add the word ALL after UNION

To combine these two statements do the following. First write the first SELECT statement and execute:

Write Exercise Code:

```
SELECT
       v.ProductID,
       v.VendorID,
       v.StandardPrice
FROM
       ProductVendor v
Where v.StandardPrice <= 42.2100
```

Result

Then write second SELECT statement and execute:

Write Exercise Code:

```
SELECT
       v.ProductID,
       v.VendorID,
       v.StandardPrice
FROM
       ProductVendor v
Where v.VendorID IN (50,83)
```

When you combine the two statements using the UNION operator you basically are removing any duplicate records. To sort when you use the UNION operator you have to add only one ORDER BY clause at the end of the second statement. Otherwise you will get an error. To combine these two statements do the following:

Write Exercise Code:

```
SELECT
       v.ProductID,
       v.VendorID,
       v.StandardPrice
FROM
       ProductVendor v
Where v.StandardPrice <= 42.2100
UNION
SELECT
       v.ProductID,
       v.VendorID,
       v.StandardPrice
FROM
       ProductVendor v
Where v.VendorID IN (50,83)
order by v.StandardPrice asc
```

Result

To return all records when you combine the two statements you use the UNION ALL operator. To combine these two statements do the following:

Write Exercise Code:

```
SELECT
       v.ProductID,
       v.VendorID,
       v.StandardPrice
FROM
       ProductVendor v
Where v.StandardPrice <= 42.2100
UNION ALL
SELECT
       v.ProductID,
       v.VendorID,
       v.StandardPrice
FROM
       ProductVendor v
Where v.VendorID IN (50,83)
order by v.StandardPrice asc
```

Result

To write the same UNION query above using multiple WHERE conditions you write it as follows:

Write Exercise Code:

```
SELECT
       v.ProductID,
       v.VendorID,
       v.StandardPrice
FROM
       ProductVendor v
WHERE v.StandardPrice <= 42.2100
OR v.VendorID IN (50,83)
ORDER BY v.StandardPrice asc
```

Result