Webucator's Free SQL Tutorial

Lesson: Advanced SELECTs

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

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

Lesson Goals

- Learn to use `SELECT` statements to retrieve calculated values.
- Learn to work with aggregate functions and grouping.
- Learn to work with SQL's data manipulation functions.

Calculated Fields

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

Concatenation

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

	(No column name)
1	Nancy Davolio
2	Andrew Fuller
3	Janet Leverling
4	Margaret Peacock
5	Steven Buchanan
6	Michael Suyama
7	Robert King
8	Laura Callahan
9	Anne Dodsworth

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

Code Sample:

AdvancedSelects/Demos/Concatenate-SqlServer.sql

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

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

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

Code Sample:

AdvancedSelects/Demos/Concatenate-Oracle.sql

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

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

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

Code Sample:

AdvancedSelects/Demos/Concatenate-MySQL.sql

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

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

Mathematical Calculations

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

Mathematical
Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

Code Sample:

AdvancedSelects/Demos/MathCalc.sql

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

The above SELECT statement will return the following results:

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

Aliases

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

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;
```

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

Calculating Fields

Duration: 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.

- 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.

	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.

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' || Extension
 || '.' 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', Extension,
 '.') AS ContactInfo
FROM Employees;
```

Aggregate Functions and Grouping

Aggregate Functions

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

Common Aggregate Functions

Aggregate Function	Description

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

Code Sample:

AdvancedSelects/Demos/Aggregate-Count.sql

```
-- Find the Number of Employees
```

Returns 9.

Code Sample:

AdvancedSelects/Demos/Aggregate-Sum.sql

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

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

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

Returns 328.

Code Sample:

AdvancedSelects/Demos/Aggregate-Avg.sql

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

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

Returns 28.8663.

Code Sample:

AdvancedSelects/Demos/Aggregate-MinMax.sql

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

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

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

Grouping Data

GROUP BY

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

Code Sample:

AdvancedSelects/Demos/Aggregate-GroupBy.sql

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

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

Since the GROUP BY clause is used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the results by a column (or more than one column), there's no need for GROUP BY if there is no aggregate function in the query.

HAVING

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

Code Sample:

AdvancedSelects/Demos/Aggregate-Having.sql

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

The above SELECT statement will return the following results:

	City	NumEmployees
1	London	4
2	Seattle	2

Order of Clauses

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

Code Sample:

AdvancedSelects/Demos/Aggregate-OrderOfClauses.sql

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

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

The above SELECT statement will return the following results:

	City	NumEmployees
1	London	3

Grouping Rules

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

Selecting Distinct Records

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

Code Sample:

AdvancedSelects/Demos/Distinct.sql

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

SELECT DISTINCT City
FROM Employees
ORDER BY City
```

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

Code Sample:

AdvancedSelects/Demos/Distinct-Count.sql

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

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

Working with Aggregate Functions

Duration: 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.

Solution:

AdvancedSelects/Solutions/Grouping.sql

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`
			`FORMAT(num,2)` or

Formatting numbers to two decimal places	`CAST(num AS decimal(8,2))`	`CAST(num AS decimal(8,2))`	`CAST(num AS decimal(8,2))`

Code Sample:

AdvancedSelects/Demos/Functions-Math1.sql

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

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

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

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

Code Sample:

AdvancedSelects/Demos/Functions-Math2.sql

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

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

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

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

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

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

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

Common String Functions

Common String Functions

Description	SQL Server	Oracle	MySQL
Convert characters to lowercase	LOWER	LOWER	LOWER
Convert characters to uppercase	UPPER	UPPER	UPPER
Remove trailing blank spaces	RTRIM	RTRIM	RTRIM
Remove leading blank spaces	LTRIM	LTRIM	LTRIM
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
*/
```

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

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

Code Sample:

AdvancedSelects/Demos/Functions-String2.sql

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

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

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

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`

Code Sample:

AdvancedSelects/Demos/Functions-Date1.sql

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

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

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

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

-- In MySQL 4.1.1 and later, DATEDIFF() returns the number of days between
-- two dates. You can then divide and floor to get age.
SELECT LastName, BirthDate, HireDate, FLOOR(DATEDIFF(HireDate,BirthDate)/3
65) AS HireAge
FROM Employees
ORDER BY HireAge;
```

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

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

And like this in Oracle:

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

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

Code Sample:

AdvancedSelects/Demos/Functions-Date2.sql

```
-- Find the Birth month for every employee

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

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

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

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

Data Manipulation Functions

Duration: 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 USERS ONLY: In an earlier demo, 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.
4. Create a report that shows the contact title in all lowercase letters of each customer contact.

Solution:

AdvancedSelects/Solutions/Functions.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');

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;
```

Next Lesson: Subqueries, Joins and Unions ➞
(https://www.webucator.com/tutorial/learn-sql/subqueries-joins-unions.cfm)

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