Lab Manual for Database Management System

Lab No. 1

SQL Basics: SELECT & WHERE Clause

LAB 01: SQL BASICS: SELECT & WHERE CLAUSE

1. INTRODUCTION:

A **database** is an organized collection of data. It is the collection of **schemas, tables, queries, reports, views** and other objects. The data are typically organized to model aspects of reality in a way that supports processes requiring information, such as modelling the availability of rooms in hotels in a way that supports finding a hotel with vacancies.

A **database management system** (DBMS) is a computer software application that interacts with the user, other applications, and the database itself to capture and analyze data. A general-purpose DBMS is designed to allow the **definition, creation, querying, update**, and **administration** of databases.

2. OBJECTIVE:

After completing this lab the student should be able to:

1. *Understand select and where clause*
2. *Understand like / between / In keyword*
3. *Retrieve results using select and where clause*
4. *Understand usage of Wildcards*

3. THEORY

A **SELECT** indicates that we are merely reading information, as opposed to modifying it. What we are selecting is identified by an expression or column list immediately following the **SELECT**. The **FROM** statement specifies the name of the table or tables from which we are getting our data.

Syntax → ***SELECT <column list> [FROM <source table(s)>]***

Example 1

*Select * from Employees;*

Selects all the employees' records from the data base and displays its columns.

Example 2

Select FirstName, LastName from Employees;

Selects data of these two columns from the Employees table.

Then there are some functions that can be used in select statement syntax is:

Syntax → ***SELECT function (<column list>) [FROM <source table(s)>]***

Function are **SUM, COUNT, DISTINCT, AVG, MIN** and **MAX** there are many other function that are also available.

Syntax → ***SELECT function (<column list>) [FROM <source table(s)>] [WHERE]<condition>]***

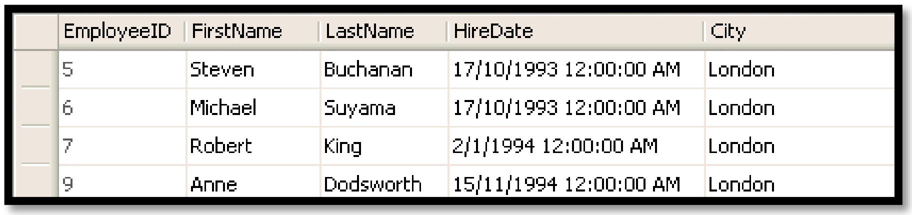
The WHERE clause immediately follows the FROM clause and defines what conditions a record has to meet before it will be shown.

Example 3

SELECT COUNT(*) FROM Orders;

The next thing we want to do is to start limiting, or filtering, the data we fetch from the database. By adding a **WHERE** clause to the **SELECT** statement, we add one (or more) conditions that must be met by the selected data. This will limit the number of rows that answer the query and are fetched. In many cases, this is where most of the "action" of a query takes place.

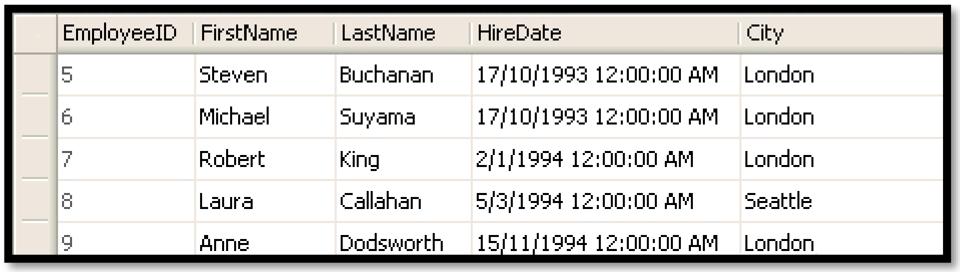
We can continue with our previous query in ***Example 02***, and limit it to only those employees living in London: ***SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees WHERE City = 'London'*** resulting in



EmployeeID	FirstName	LastName	HireDate	City
5	Steven	Buchanan	17/10/1993 12:00:00 AM	London
6	Michael	Suyama	17/10/1993 12:00:00 AM	London
7	Robert	King	2/1/1994 12:00:00 AM	London
9	Anne	Dodsworth	15/11/1994 12:00:00 AM	London

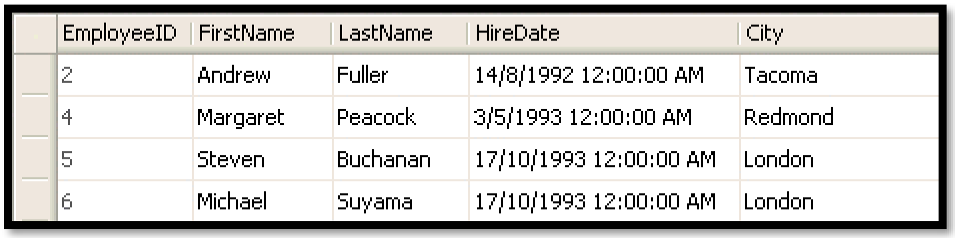
If you wanted to get the opposite, the employees who do *not* live in London, you would write: ***SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees WHERE City <> 'London'***

It is not necessary to test for equality; you can also use the standard equality/inequality operators that you would expect. For example, to get a list of employees who were hired on or after a given date, you would write: ***SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees WHERE HireDate >= '1-july-1993'*** and get the resulting rows:



EmployeeID	FirstName	LastName	HireDate	City
5	Steven	Buchanan	17/10/1993 12:00:00 AM	London
6	Michael	Suyama	17/10/1993 12:00:00 AM	London
7	Robert	King	2/1/1994 12:00:00 AM	London
8	Laura	Callahan	5/3/1994 12:00:00 AM	Seattle
9	Anne	Dodsworth	15/11/1994 12:00:00 AM	London

Of course, we can write more complex conditions. The obvious way to do this is by having multiple conditions in the **WHERE** clause. If we want to know which employees were hired between two given dates, we could write: ***SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees WHERE (HireDate >= '1-june-1992') AND (HireDate <= '15-december-1993')*** resulting in:



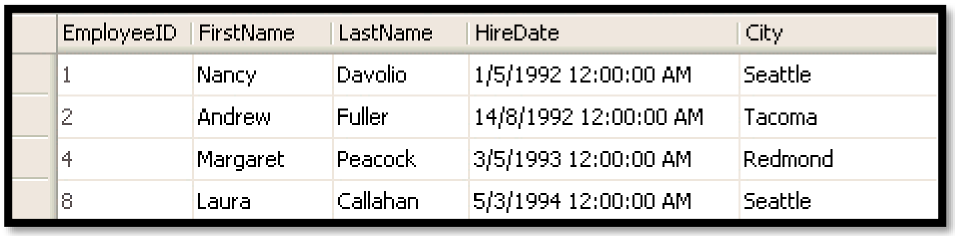
	EmployeeID	FirstName	LastName	HireDate	City
	2	Andrew	Fuller	14/8/1992 12:00:00 AM	Tacoma
	4	Margaret	Peacock	3/5/1993 12:00:00 AM	Redmond
	5	Steven	Buchanan	17/10/1993 12:00:00 AM	London
	6	Michael	Suyama	17/10/1993 12:00:00 AM	London

Note that SQL also has a special **BETWEEN** operator that checks to see if a value is between two values (including equality on both ends). This allows us to rewrite the previous query as: ***SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees WHERE HireDate BETWEEN '1-june-1992' AND '15-december-1993'***

We could also use the **NOT** operator, to fetch those rows that are *not* between the specified dates: ***SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees WHERE HireDate NOT BETWEEN '1-june-1992' AND '15-december-1993'.***

If we want to check if a column value is equal to more than one value? If it is only two values, then it is easy enough to test for each of those values, combining them with the **OR** operator and writing something like: ***SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees WHERE City = 'London' OR City = 'Seattle'***

However, if there are three, four, or more values that we want to compare against, the above approach quickly becomes messy. In such cases, we can use the **IN** operator to test against a set of values. If we wanted to see if the City was either Seattle, Tacoma, or Redmond, we would write: ***SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees WHERE City IN ('Seattle', 'Tacoma', 'Redmond')*** producing the results shown below.



	EmployeeID	FirstName	LastName	HireDate	City
	1	Nancy	Davolio	1/5/1992 12:00:00 AM	Seattle
	2	Andrew	Fuller	14/8/1992 12:00:00 AM	Tacoma
	4	Margaret	Peacock	3/5/1993 12:00:00 AM	Redmond
	8	Laura	Callahan	5/3/1994 12:00:00 AM	Seattle

As with the **BETWEEN** operator, here too we can reverse the results obtained and query for those rows where City is not in the specified list: ***SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees WHERE City NOT IN ('Seattle', 'Tacoma', 'Redmond')***

Finally, the **LIKE** operator allows us to perform basic pattern-matching using wildcard characters. For Microsoft SQL Server, the wildcard characters are defined as follows:

Wildcard	Description
_ *(underscore)*	*matches any single character*
%	*matches a string of one or more characters*
[^]	*matches any single character not within the specified range (e.g. [^a- f]) or set (e.g. [^abcdef]).*
[ ]	*matches any single character within the specified range (e.g. [a-f]) or set (e.g. [abcdef]).*

A few **examples** should help clarify these rules.

1. ***WHERE FirstName LIKE '_im'*** finds all three-letter first names that end with 'im' (e.g. Jim, Tim).

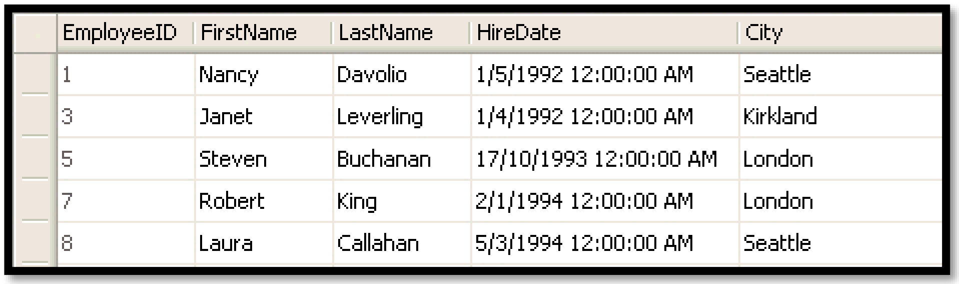
2. ***WHERE LastName LIKE '%stein'*** finds all employees whose last name ends with 'stein'.

3. ***WHERE LastName LIKE '%stein%'*** finds all employees whose last name includes 'stein' anywhere in the name.

4. ***WHERE FirstName LIKE '[JT]im'*** finds three-letter first names that end with 'im' and begin with either 'J' or 'T' (that is, *only* Jim and Tim).

5. ***WHERE LastName LIKE 'm[^c]%'*** finds all last names beginning with 'm' where the following (second) letter is not 'c'.

Here too, we can opt to use the NOT operator: to find all of the employees whose first name does not start with 'M' or 'A', we would write: ***SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees WHERE (FirstName NOT LIKE 'M%') AND (FirstName NOT LIKE 'A%')*** resulting in:



EmployeeID	FirstName	LastName	HireDate	City
1	Nancy	Davolio	1/5/1992 12:00:00 AM	Seattle
3	Janet	Leverling	1/4/1992 12:00:00 AM	Kirkland
5	Steven	Buchanan	17/10/1993 12:00:00 AM	London
7	Robert	King	2/1/1994 12:00:00 AM	London
8	Laura	Callahan	5/3/1994 12:00:00 AM	Seattle

4. ACTIVITY TIME BOXING

Activity Name	Activity Time	Total Time
Instruments Allocation + Setting up Lab	10 mints	10 mints
Walk through Theory & Tasks (Lecture)	60 mints	60 mints
Implementation & Practice time	90 mints	80 mints
Evaluation Time	20 mints	20 mints
	Total Duration	180 mints

5. EXERCISE:

PERFORMANCE TASKS:

1. ***All Example Tasks.***
2. *Find Title of employee Nancy.*
3. *Display all the orders where unit price lies in the range of 10$ to 40$.*
4. *In Customer table, display all cities that ends with the letter 'a'.*
5. *Write a query to list employees whose address does not contain Rd.*
6. *List order whose ShipRegion is not Null.*

LAB FILE TASKS:

1. *Get Order id, Product id, Unit price from Order Details.*
2. *Get the price of an order (by multiplying unit price by quantity).*
3. *Display all cities that employees belong to but don't allow repetition.*
4. *Find complete name of all employees.*
5. *Display data of all employees those working as Sales Representative.*
6. *Display complete name of employees those lives in London.*
7. *Display product name whose unit price are greater than 90$.*
8. *List the name of all employees whose first name starts with the letter 'A'.*
9. *Display names of all employees whose name contain 'an'.*
10. *Display the company name where Region is NULL in Customer Table.*
11. *Write a query to list employees who live in London, Seattle or Redmond*
12. *Write a query to list employees whose address contains 3 numbers in its start.*
13. *Write a query to list those employees whose TitleofCourtesy does not starts with M.*
14. *List all products where UnitPrice is between 10 and 15 and QuantityPerUnit contains "bottles".*
15. *List all products where UnitPrice is not in 10,12,15,17 or 19.*