LAB # 6 :

SQL WILDCARDS, IN, BETWEEN, ALIASES, AUTOINCREMENT, LIKE COMMANDS

Objective (aim) of the experiment

To practice and implement SQL Commands (SQL WILDCARDS, IN, BETWEEN, ALIASES, AUTOINCREMENT, and LIKE COMMANDS).

Scoring Rubrics for Lab 6:

S#	Task	Weightage	Obtained	Signature and Date
1	Using the 'In' command correctly	20 %		
2	Using the 'Between' statement correctly	15 %		
3	Making use of 'Aliases' correctly	25 %		
4	Using the 'AutoIncrement' command correctly	15 %		
5	Using the 'SQL Wildcards' in conjunction with the 'Like' command correctly	25%		
	Total marks obtained in this lab	**100%**	**%**	

Equipment used

Sl. No.	Facilities Required	Quantity
1	System	1
2	Operating System	Windows 7
3	DBMS	Sql Server Management Studio 2012

TASKS

SQL Wildcard Characters

A wildcard character can be used to substitute for any other character(s) in a string. In SQL, wildcard characters are used with the SQL LIKE operator.

SQL wildcards are used to search for data within a table.

With SQL, the wildcards are:

Wildcard	Description
%	A substitute for zero or more characters
_	A substitute for a single character
[*charlist*]	Sets and ranges of characters to match
[^*charlist*] or [!*charlist*]	Matches only a character NOT specified within the brackets

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Demo Database

In this tutorial we will use the well-known Northwind sample database. Below is a selection from the "Customers" table:

Using the SQL % Wildcard

The following SQL statement selects all customers with a City starting with

"ber": Example

```
SELECT * FROM

Customers WHERE

City LIKE 'ber%';
```

The following SQL statement selects all customers with a City containing the

pattern "es": Example

```
SELECT * FROM

Customers WHERE

City LIKE '%es%';
```

Using the SQL _ Wildcard

The following SQL statement selects all customers with a City starting with any character, followed by "erlin":

Example

```
SELECT * FROM

Customers WHERE

City LIKE '_erlin';
```

The following SQL statement selects all customers with a City starting with "L", followed by any character, followed by "n", followed by any character, followed by "on":

Example

```
SELECT * FROM

Customers WHERE

City LIKE 'L_n_on';
```


Using the SQL [charlist] Wildcard

The following SQL statement selects all customers with a City starting with "b", "s",

or "p": Example

SELECT * FROM

Customers WHERE

City LIKE '[bsp]%';

The following SQL statement selects all customers with a City starting with "a", "b",

or "c": Example

SELECT * FROM

Customers WHERE

City LIKE '[a-c]%';

The following SQL statement selects all customers with a City NOT starting with "b", "s", or "p":

Example

SELECT * FROM

Customers WHERE

City LIKE '[!bsp]%';

Or

SELECT * FROM

Customers WHERE City

NOT LIKE '[bsp]%';

SELECT *column_name(s)*

FROM *table_name*

WHERE *column_name* LIKE *pattern*;

SQL LIKE Syntax

Demo Database

In this tutorial we will use the well-known Northwind sample

database. Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

SQL LIKE Operator Examples

The following SQL statement selects all customers with a City starting with the

letter "s": Example

SELECT * FROM

Customers WHERE

City LIKE 's%';

Tip: The "%" sign is used to define wildcards (missing letters) both before and after the pattern. You will learn more about wildcards in the next chapter.

The following SQL statement selects all customers with a City ending with the letter "s":

Example

SELECT * FROM

Customers WHERE

City LIKE '%s';

The following SQL statement selects all customers with a Country containing the pattern "land":

Example

SELECT * FROM

Customers WHERE

Country LIKE '%land%';

Using the NOT keyword allows you to select records that does NOT match the pattern.

The following SQL statement selects all customers with a Country NOT containing the pattern "land":

Example

SELECT * FROM Customers

WHERE Country NOT LIKE

'%land%'; The IN Operator

The IN operator allows you to specify multiple values in a WHERE

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...);
```

Demo Database

In this tutorial we will use the well-known Northwind sample

database. Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

IN Operator Example

The following SQL statement selects all customers with a City of "Paris" or "London": Example

```
SELECT * FROM Customers
WHERE City IN
('Paris','London');
```

The BETWEEN operator is used to select values within a range. The SQL BETWEEN Operator

The BETWEEN operator selects values within a range. The values can be numbers, text, or dates.

SQL BETWEEN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Demo Database

In this tutorial we will use the well-known Northwind sample database. Below is a selection from the "Products" table:

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	1	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	1	2	36 boxes	21.35

BETWEEN Operator Example

The following SQL statement selects all products with a price BETWEEN 10 and 20: Example

```
SELECT * FROM Products
```

WHERE Price BETWEEN 10 AND 20;

NOT BETWEEN Operator Example

To display the products outside the range of the previous example, use NOT

BETWEEN: Example

SELECT * FROM Products

WHERE Price NOT BETWEEN 10 AND 20;

BETWEEN Operator with IN Example

The following SQL statement selects all products with a price BETWEEN 10 and 20, but products with a CategoryID of 1,2, or 3 should not be displayed:

Example

SELECT * FROM Products

WHERE (Price BETWEEN 10

AND 20) AND NOT CategoryID

IN (1,2,3);

BETWEEN Operator with Text Value Example

The following SQL statement selects all products with a ProductName beginning with any of the letter BETWEEN 'C' and 'M':

Example

SELECT * FROM Products WHERE ProductName BETWEEN

'C' AND 'M'; NOT BETWEEN Operator with Text Value Example

The following SQL statement selects all products with a ProductName beginning with any of the letter NOT BETWEEN 'C' and 'M':

Example

SELECT * FROM Products

WHERE ProductName NOT BETWEEN 'C' AND 'M';

Sample Table

Below is a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	7/4/1996	3
10249	81	6	7/5/1996	1
10250	34	4	7/8/1996	2
10251	84	3	7/9/1996	1
10252	76	4	7/10/1996	2

BETWEEN Operator with Date Value Example

The following SQL statement selects all orders with an OrderDate BETWEEN '04-July-1996' and '09-July-1996':

Example

SELECT * FROM Orders

WHERE OrderDate BETWEEN #07/04/1996# AND #07/09/1996#;

Notice that the BETWEEN operator can produce different result in different databases! In some databases, BETWEEN selects fields that are between and excluding the test values. In other databases, BETWEEN selects fields that are between and including the test values. And in other databases, BETWEEN selects fields between the test values, including the first test valexcluding the last test value.
Therefore: Check how your database treats the BETWEEN operator!

SQL
Aliases

SQL aliases are used to temporarily rename a table or a column heading.

SQL aliases are used to give a database table, or a column in a table, a temporary name. Basically aliases are created to make column names more readable.

SQL Alias Syntax for Columns

SELECT *column_name* AS *alias_name*

FROM *table_name;*

SQL Alias Syntax for Tables SELECT

column_name(s)

FROM *table_name* AS *alias_name;*

Demo Database

In this tutorial we will use the well-known Northwind sample

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK

And a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10354	58	8	1996-11-14	3
10355	4	6	1996-11-15	1
10356	86	6	1996-11-18	2

Alias Example for Table Columns

The following SQL statement specifies two aliases, one for the CustomerName column and one for the ContactName column. **Tip:** It requires double quotation marks or square brackets if the column name contains spaces:

Example

```
SELECT CustomerName AS Customer, ContactName AS [Contact

Person]  FROM Customers;
```

In the following SQL statement we combine four columns (Address, City, PostalCode, and Country) and create an alias named "Address":

Example

```
SELECT CustomerName, Address+', '+City+', '+PostalCode+', '+Country

AS Address  FROM Customers;
```

Note: To get the SQL statement above to work in MySQL use the following:

```
SELECT CustomerName, CONCAT(Address,', ',City,', ',PostalCode,', ',Country) AS
Address

FROM Customers;
```

Alias Example for Tables

The following SQL statement selects all the orders from the customer with CustomerID=4 (Around the Horn). We use the "Customers" and "Orders" tables, and give them the table aliases of "c" and "o" respectively (Here we have used aliases to make the SQL shorter):

Example

SELECT o.OrderID, o.OrderDate,

c.CustomerName FROM Customers AS c,

Orders AS o

WHERE c.CustomerName="Around the Horn" AND

c.CustomerID=o.CustomerID; The same SQL statement without aliases:

Example

SELECT Orders.OrderID, Orders.OrderDate,

Customers.CustomerName FROM Customers, Orders

WHERE Customers.CustomerName="Around the
Horn" AND
Customers.CustomerID=Orders.CustomerID;

Aliases can be useful when:

- There are more than one table involved in a query
- Functions are used in the query
- Column names are big or not very readable
- Two or more columns are combined together

AUTO INCREMENT a Field

Auto-increment allows a unique number to be generated when a new record is inserted into a table.

Very often we would like the value of the primary key field to be created automatically every time a new record is inserted.

We would like to create an auto-increment field in a

table. Syntax for MySQL

The following SQL statement defines the "ID" column to be an auto-increment primary key field in the "Persons" table:

```
CREATE TABLE Persons
(
ID int NOT NULL AUTO_INCREMENT,
LastName varchar(255) NOT
NULL, FirstName varchar(255),
Address varchar(255),
City
varchar(255),
PRIMARY KEY
(ID)
)
```

MySQL uses the AUTO_INCREMENT keyword to perform an auto-increment feature. By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement:

```
ALTER TABLE Persons AUTO_INCREMENT=100
```

To insert a new record into the "Persons" table, we will NOT have to specify a value for the "ID" column (a unique value will be added automatically):

```
INSERT INTO Persons
(FirstName,LastName) VALUES
('Lars','Monsen')
```

The SQL statement above would insert a new record into the "Persons" table. The "ID" column would be assigned a unique value. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".

Syntax for SQL Server

The following SQL statement defines the "ID" column to be an auto-increment primary key field in the "Persons" table:

```
CREATE TABLE Persons
(
ID int IDENTITY(1,1) PRIMARY KEY,
LastName varchar(255) NOT
NULL, FirstName varchar(255),
Address
varchar(255), City
varchar(255)
)
```

The MS SQL Server uses the IDENTITY keyword to perform an auto-increment feature.

In the example above, the starting value for IDENTITY is 1, and it will increment by 1 for each new record.

Tip: To specify that the "ID" column should start at value 10 and increment by 5, change it to IDENTITY(10,5).

To insert a new record into the "Persons" table, we will NOT have to specify a value for the "ID" column (a unique value will be added automatically):

```
INSERT INTO Persons (FirstName,LastName)
VALUES ('Lars','Monsen')
```

The SQL statement above would insert a new record into the "Persons" table. The "ID" column would be assigned a unique value. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".

Syntax for Access

The following SQL statement defines the "ID" column to be an auto-increment primary key field in the "Persons" table:

```
CREATE TABLE Persons
(
ID Integer PRIMARY KEY AUTOINCREMENT,
LastName varchar(255) NOT
NULL, FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

The MS Access uses the AUTOINCREMENT keyword to perform an auto-increment feature.

By default, the starting value for AUTOINCREMENT is 1, and it will increment by 1 for each new record.

Tip: To specify that the "ID" column should start at value 10 and increment by 5, change the autoincrement to AUTOINCREMENT(10,5).

To insert a new record into the "Persons" table, we will NOT have to specify a value for the "ID" column (a unique value will be added automatically):

```
INSERT INTO Persons (FirstName,LastName)
VALUES ('Lars','Monsen')
```

The SQL statement above would insert a new record into the "Persons" table. The "P_Id" column would be assigned a unique value. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".

Syntax for Oracle

In Oracle the code is a little bit more tricky.

You will have to create an auto-increment field with the sequence object (this object generates a number sequence).

Use the following CREATE SEQUENCE syntax:

```
CREATE SEQUENCE seq_person

MINVALUE 1

START WITH 1

INCREMENT BY 1

CACHE 10
```

The code above creates a sequence object called seq_person, that starts with 1 and will increment by 1. It will also cache up to 10 values for performance. The cache option specifies how many sequence values will be stored in memory for faster access.

To insert a new record into the "Persons" table, we will have to use the nextval function (this function retrieves the next value from seq_person sequence):

```
INSERT INTO Persons (ID,FirstName,LastName)

VALUES (seq_person.nextval,'Lars','Monsen')
```

The SQL statement above would insert a new record into the "Persons" table. The "ID" column would be assigned the next number from the seq_person sequence. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".

EXPECTED DELIVERABLE

A spool file showing all executions of the above queries.