SQL Tutorial

- standard language for storing, manipulating and retrieving data in databases
- **How to print whole database**
  - select * from databasename;
- **Intro to SQL**
  - SQL stands for Structured Query Language
  - SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987
  - RDBMS stands for Relational Database Management System
  - table is a collection of related data entries and it consists of columns and rows
  - Each column contains a field of all records
  - Field contains a record of all fields
  - SQL keywords are not case-sensitive
  - We require a ‘;’ by ending of every sql statement
  - **Most important SQL Commands:**
    - SELECT , UPDATE , DELETE , INSERT INTO , CREATE DATABASE , ALTER DATABASE , CREATE TABLE ,ALTER TABLE , DROP TABLE ,CREATE INDEX, DROP INDEX.
- **If we want to select particular columns in table then**
  - select column1,column2 from tablename;
- **If we want to select all the table and display the table then**
  - Select * from tablename;
- **If we want to select a particular column even if the values are repeated**
  - Select column1 from databasename;
- **If we want to select a particular column and select distinct values of it**
  - select distinct column1 from databasename;
  - This gives all the distinct values of that column
- **To know how many distict columns are there**
  - select count(distinct column_name) from tablename;
- **Where clause**
  - Used to filter records
  - It extracts all the elements which satisfy the specific condition
  - Select column1,column2,…. From table name where condition;
  - Where clause can use in update,delete,statement etc.

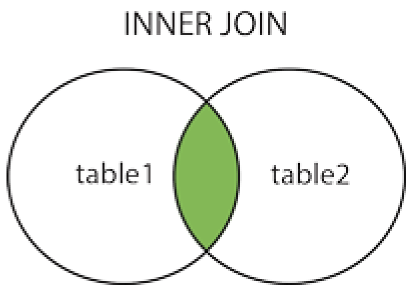
- **Where clause example**
    - Select * from tablename where code=”a”;
- **AND SYNTAX:**
    - Select col1,col2 from tablename where condition1 and condition2 and…….;
- **OR SYNTAX:**
    - SELECT col1,col2 from tablename where condition1 or condition2 or …….;
- **NOT SYNTAX**
    - Select col1,col2 from tablename where not condition;
- You can use “()” for forming complex expressions
- **ORDER BY SYNTAX:**
    - Sorts result by ascending or descending order , ascending is by default
    - If we want In descending order then use DESC keyword.
    - Select col1,col2 from tablename order by col1,col2,….DESC;
    - If we use two columns in order by then first it will check the first column then if any of first col is same with other then second column will be sorted among those same
- **INSERT INTO SYNTAX:**
    - Used to import new records into table
    - Insert into tablename (col1,col2,col3,……) values (val1,val2,val3,….);
    - We need not to give values for auto increment columns
    - If we didn’t give values for all attributes then they will remain null
- **NULL VALUES**
    - Select colname from tablename where colname is null;
    - Select colname from tablename where colname is not null;
- **To know is there a null value or not**
    - Select col1,col2,col3 from tablename where col3 is null;
    - Select col1,col2,col3 from tablename where col3 is not null;
- **Update Statement**
    - Used to modify existing data
    - Upadate tablename set col1=val1,col2=val2,…. Where condition;
    - If we donot use where in the statement then all the records will be updated
    - Ex: update customers set contactname=”jeevan”,city=”Tenali” where customerid=1;
    - Where statement determines how many number of data will be updated
- **Delete Statement**
    - Delete from tablename where condition
    - Delete from tablename
        - Is used to delete all the contents of the table but it stores structure ,attributes ,indexes
        - It just remove the elements of the table but it wont delete the table

- **SQL Select top clause**
  - It gives number of records to return
  - SQL supports top clause
    - Select top number column_name(s) from tablename where condition;
  - MySql supports Limit clause
    - Select column_name from tablename where condition limit number;
  - Oracle supports rownum clause
    - Select column_name(s) from table_name where rownum<=number;
- **SQL Min() Max() Functions**
  - Select min(column_name) from tablename where condition;
    - To select the minimum value of particular column
  - Select max(column_name) from tablename where condition;
    - To select the maximum value of particular column
- **SQL Count()**
  - Select count(column_name) from tablename where condition;
    - This helps to know number of rows that matches the condition
    - Null values are not counted
- **SQL AVG()**
  - Select avg(column_name) from table_name where condition;
    - Null Values are not counted
- **SQL SUM()**
  - Select sum(column_name) from table_name where condition
    - Null values are not counted
- **SQL Like Operator(similarly not like is also present)**
  - Like operator is used in where clause to search for a specified pattern in a column
  - Select col1,col2,….from tablename while column like pattern
  - “a%”: values start with a
  - “%a”: values end with a
  - “%or%” any values that have or in any position
  - “_r%” any value which have r in second postion
  - “a__%” any value which have at least 3 charecters
  - “a%o” starts with ‘a’ and end with ‘o’
- **Wild Cards:**
  - % represents zero or more charecters
  - _ represents single character
  - [] represents a single character within brackets
  - [^] represents any character not in brackets
  - – represents range of charecters
  - Examples:

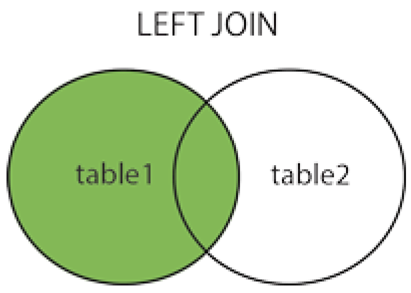
            - Ber%
            - %es%
            - _ondon
            - L_n_on
            - [abc]%
            - [a-c]%
            - [!a-c]%
- **IN OPERATOR**
    - SELECT * FROM TABLENAME WHERE COLUMNNAME IN (VAL1,VAL2,VAL3);
    - SELECT * FROM TABLENAME WHERE COLUMNNAME IN (STATEMENT);
- **Between operator is inclusive**
    - SELECT COLUMNNAMES FROM TABLENAME WHERE COLUMNNAME BETWEEN VALUE1 AND VALUE2;
    - SELECT COLUMNNAMES FROM TABLENAME WHERE COLUMNNAME NOT BETWEEN VALUE1 AND VALUE2;
- **BETWEEN WITH IN EXAMPLE:**
    - SELECT COLUMNNAMES FROM TABLENAME WHERE COLUMNNAME BETWEEN VALUE1 AND VALUE2 AND NOT IN (VALUE1,VALUE2,VALUE3);
    - **EXAMPLE:**
        - SELECT * FROM TABLENAME WHERE COLUMNNAME BETWEEN START(STRING) AND END(STRING);
            - This gives unordered column
        - SELECT * FROM TABLENAME WHERE COLUMNNAME BETWEEN START(STRING) AND END(STRING) ORDERBY COLUMNNAME;
            - This gives ordered column
    - Similarly not between also
- **SQL ALIASES**
    - It gives a temporary name to for a column in a table
    - Aliases exists only duration of query
    - **Alias for column**
        - SELECT COLUMN1 AS [ALIAS1],COLUMN2 AS [ALIAS2] FROM CUSTOMERS;
        - Example:
            - Select singlecolumn,concatcol(col1+',''+col2+','+col3) as newcolname from tablename;
    - **Alias from tables**
        - If we use multiple tables in one command we need to use alias otherwise it would be lengthy code
        - SELECT T1ALIAS.COLUMNNAME,T2ALIAS.COLUMNNAME FROM TABLE1NAME AS T1,TABLE2NAME AS T2 WHERE CONDITION;
- **SQL Joins**

- It joins rows of two tables based on a related column between them
- If we try to find common between two tables based on a column then you have to use inner join
- **INNER JOIN**
  - **Syntax:**
  - Select table1.columnname,table2.columnname,table1.columnname from table1 inner join table2 on table1.columnname=table2.columnname;
  - **Example:**

  -
    ```
    SELECT Orders.OrderID, Customers.CustomerName,
    Orders.OrderDate
    FROM Orders
    INNER JOIN Customers ON Orders.CustomerID=Customers.Custo
    merID;
    ```
- **InnerJoin:**

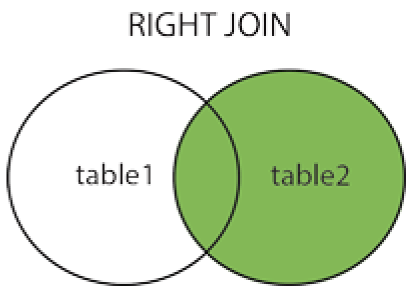
-
  ```
  SELECT column_name(s)
  FROM table1
  INNER JOIN table2
  ON table1.column_name = table2.column_name;
  ```
- 

- **LEFT JOIN**

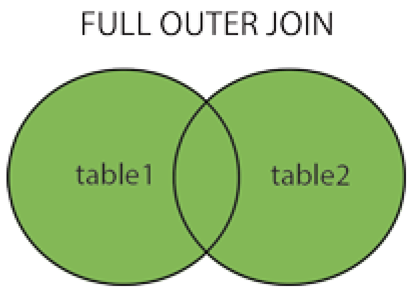
-
  ```
  SELECT column_name(s)
  FROM table1
  LEFT JOIN table2
  ON table1.column_name = table2.column_name;
  ```
- 

- **RIGHT JOIN**

-
  ```
  SELECT column_name(s)
  FROM table1
  RIGHT JOIN table2
  ON table1.column_name = table2.column_name;
  ```
-



-
- **FULL OUTER JOIN**



-

-
  ```
  SELECT column_name(s)
  FROM table1
  FULL OUTER JOIN table2
  ON table1.column_name = table2.column_name
  WHERE condition;
  ```
- **SELF JOIN:**
- SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City
- FROM Customers A, Customers B
- WHERE A.CustomerID <> B.CustomerID
- AND A.City = B.City
- ORDER BY A.City;
- **SQL UNION**
- Used to combine two or more select statements

-
  ```
  SELECT column_name(s) FROM table1
  UNION
  SELECT column_name(s) FROM table2;
  ```
- Union selects only distinct values but union all selects duplicate values also
- Alias only exists only in the duration of query
- **SQL GROUP BY STATEMENT**

-
  ```
  SELECT column_name(s)
  FROM table_name
  WHERE condition
  GROUP BY column_name(s)
  ORDER BY column_name(s);
  ```
- GROUP BY statement groups the table based on a column name
- This has some aggregate properties like (count,max,min,sum,avg) to group results by one or more columns
- **Example:**

-
  ```
  SELECT COUNT(CustomerID), Country
  FROM Customers
  GROUP BY Country
  ORDER BY COUNT(CustomerID) DESC;
  ```
- **SQL Having Clause:**
- Where keyword couldnot be used for aggregate functions so Having is introduced in sql

-
  ```
  SELECT column_name(s)
  FROM table_name
  WHERE condition
  GROUP BY column_name(s)
  HAVING condition
  ORDER BY column_name(s);
  ```
- **Example:**

-
  ```
  SELECT COUNT(CustomerID), Country
  FROM Customers
  GROUP BY Country
  HAVING COUNT(CustomerID) > 5
  ORDER BY COUNT(CustomerID) DESC;
  ```

-
  ```
  SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOr
  ders
  FROM (Orders
  INNER JOIN Employees ON Orders.EmployeeID =
  Employees.EmployeeID)
  GROUP BY LastName
  HAVING COUNT(Orders.OrderID) > 10;
  ```

-
  ```
  SELECT Employees.LastName,COUNT(Orders.OrderID) as
  Numberoforders from orders innerjoin employees on
  orders.emploeeid=employees.employeeid where
  lastname=”Kanaparthi” or lastname=”Jeevan” groupby lastname
  having count(Orders.orderid)>25;
  ```

- **SQL ANY ALL OPERATORS**
  - ANY ALL operators are used with a having or where clause
  - Any operator returns true if any of subquery values meet the condition
  - All operator returns true if all the subquery values meet the condition
  - **ANY SYNTAX**

  -
    ```
    SELECT column_name(s)
    FROM table_name
    WHERE column_name operator ANY
    (SELECT column_name FROM table_name WHERE condition);
    ```
  - standard comparison operator (=, <>, !=, >, >=, <, or <=)
  - **ANY EXAMPLES:**

  -
    ```
    SELECT ProductName
    FROM Products
    WHERE ProductID
    = ANY (SELECT ProductID FROM OrderDetails WHERE Quantity
    = 10);
    ```
  - **ALL SYNTAX**

  -
    ```
    SELECT column_name(s)
    FROM table_name
    WHERE column_name operator ALL
    (SELECT column_name FROM table_name WHERE condition);
    ```
  - **ALL EXAMPLES**

  -
    ```
    SELECT ProductName
    FROM Products
    WHERE ProductID
    = ALL (SELECT ProductID FROM OrderDetails WHERE Quantity
    = 10);
    ```

- **SQL SELECT INTO STATEMENT**
  - Select into statement copies data from one table to a new table

  -
    ```
    SELECT column1, column2, column3, ...
    INTO newtable [IN externaldb]
    FROM oldtable
    WHERE condition;
    ```
  - **To copy a table into a specific database then use the following code**

  -
    ```
    SELECT * INTO CustomersBackup2017 IN 'Backup.mdb'
    FROM Customers;
    ```
  - You can add where or any other clause into this statement

- Copying data from more than one table to a new table

-
  ```
  SELECT Customers.CustomerName, Orders.OrderID
  INTO CustomersOrderBackup2017
  FROM Customers
  LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
  ```
- To create a empty table with same datatypes as older table but not the previous data

-
  ```
  SELECT * INTO newtable
  FROM oldtable
  WHERE 1 = 0;
  ```

- **INSERT INTO SELECT STATEMENT**
  - To copy all columns from one table to another table

  -
    ```
    INSERT INTO table2
    SELECT * FROM table1
    WHERE condition;
    ```
  - **Examples**

  -
    ```
    INSERT INTO Customers (CustomerName, ContactName, Address,
    City, PostalCode, Country)
    SELECT SupplierName, ContactName, Address, City,
    PostalCode, Country FROM Suppliers;
    ```
- **SQL CASE STATEMENT**
  - **SYNTAX:**

  -
    ```
    CASE
        WHEN condition1 THEN result1
        WHEN condition2 THEN result2
        WHEN conditionN THEN resultN
        ELSE result
    END;
    ```
  - SQL goes through conditions and returns a value when the first condition is met
  - **EXAMPLE:**

  -
    ```
    SELECT CustomerName, City, Country
    FROM Customers
    ORDER BY
    (CASE
        WHEN City IS NULL THEN Country
        ELSE City
    END);
    ```
- **IFNULL() STATEMENT**
  - If there is a column in which some of its values are null values so when doing calculations on such columns we will get a result null for the whole table
  - So to escape from that we use ifnull () statement
  - **Example**

- SELECT ProductName, UnitPrice * (UnitsInStock + IFNULL(UnitsOnOrder, 0))
  FROM Products;

- **PROCEDURES**
  - Stored procedures are prepared sql code that you can save, so that you can save the code now and can use the code over and over again it is similar to function we can pass parameters to a procedure
  - **SYNTAX**
  - CREATE PROCEDURE *procedure_name*
    AS
    sql_statement
    GO;
  - **Execute a stored procedure**
  - EXEC *procedure_name*;
  - **Example:Procedure with multiple parameter**
  - CREATE PROCEDURE SelectAllCustomers @City nvarchar(30), @PostalCode nvarchar(10)
    AS
    SELECT * FROM Customers WHERE City = @City AND PostalCode = @PostalCode
    GO;
  - **Excecuting procedure**
  - EXEC SelectAllCustomers @City = "London", @PostalCode = "WA1 1DP";
- **SQL Comments**
  - -- Single Line Comment
  - Multi-line comments start with /* and end with */

CREATE DATABASE:
1)Creates new sql database
2)CREATE DATABASE *databasename*;
3)if you want to check whether the database is created or not then use the following statement
4)SHOW DATABASES;

DROP DATABASE:
1)this statement drop the mentioned database
2)DROP DATABASE testDB;

BACKUP DATABASE:
1)this helps to backup the database and store in mentioned location
2)BACKUP DATABASE *databasename*
TO DISK = '*filepath*';

3)This statement helps us to backup the database from the last backup means if the data is aldready present then it will just update the database

```
4)BACKUP DATABASE databasename
TO DISK = 'filepath'
WITH DIFFERENTIAL;
```

CREATE TABLE

1)used to create a new table in the database

```
2)CREATE TABLE table_name (
     column1 datatype,
     column2 datatype,
     column3 datatype,
    ....
);
```

3)datatype declaration is must needed for the column because we have to mention what kind of data we will store in that column and it is also declared for memory location also

4)Example

```
5)CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

6)varchar means each variable consits of charecters of max length 255

7)Now the empty table will be created

8)we can create it by using insert into statement

CREATING 2ND TABLE USING 1ST TABLE USING AS

```
1)CREATE TABLE new_table_name AS
      SELECT column1, column2,...
     FROM existing_table_name
     WHERE ....;
```

DROP DATABASE

1)drop an existing table of a database

```
2)DROP TABLE table_name;
```

TRUNCATE TABLE

1)used to delete data in table but not the table

```
2)TRUNCATE TABLE table_name;
```

ALTER TABLE

1)used to add delete or modify columns of a table

2)Add column

```
     1)ALTER TABLE table_name
     ADD column_name datatype;
```

3)Drop Column

```
     1)ALTER TABLE table_name
     DROP COLUMN column_name;
```

4)Modify column of table

```
    1)ALTER TABLE table_name
    ALTER COLUMN column_name datatype;
```

SQL CONSTRAINTS

```
1)CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
   ....
);
2)used to specify rules for the data in table
3)ALL Constraints
```

- **NOT NULL** - Ensures that a column cannot have a NULL value
- **UNIQUE** - Ensures that all values in a column are different
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- **FOREIGN KEY** - Uniquely identifies a row/record in another table
- **CHECK** - Ensures that all values in a column satisfies a specific condition
- **DEFAULT** - Sets a default value for a column when no value is specified
- **INDEX** - Used to create and retrieve data from the database very quickly

NOTNULL CONSTRAINT

1)this command forces the column not to accept null values or any sql statement is accepted only if the speified column doesnt have null value

```
2)CREATE TABLE Persons (

    ID int NOT NULL,

     LastName varchar(255) NOT NULL,

    FirstName varchar(255) NOT NULL,

    Age int

);

3)ALTER TABLE Persons

MODIFY Age int NOT NULL;
```

UNIQUE CONSTRAINT

1)ensures all values are different in mentioned column

2)Unique and primarykey both work in same manner

3)we can use as many as unique constraints in table but only one primary key per table

```
4)CREATE TABLE Persons (

     ID int NOT NULL UNIQUE,

    LastName varchar(255) NOT NULL,

    FirstName varchar(255),

    Age int

);
```

```
5)ALTER TABLE Persons

ADD UNIQUE (ID);
```

To add a specific constraint on multiple columns

```
6)ALTER TABLE Persons

ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
```

To drop the constraint at a time but using index

```
7)ALTER TABLE Persons

DROP INDEX UC_Person;
```

- **PRIMARY KEY**
  - Identifies each record in a table
  - Contains only unique values no null values
  - A table can have only one primary key

  -
    ```
    CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
        PRIMARY KEY (ID)
    );
    ```

  -
    ```
    ALTER TABLE Persons
    ADD PRIMARY KEY (ID);
    ```

  -
    ```
    ALTER TABLE Persons
    DROP PRIMARY KEY;
    ```
- **FOREIGN KEY**
  - SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created

-
  ```
  CREATE TABLE Orders (
      OrderID int NOT NULL,
      OrderNumber int NOT NULL,
      PersonID int,
      PRIMARY KEY (OrderID),
      FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
  );
  ```

-
  ```
  ALTER TABLE Orders
  DROP FOREIGN KEY FK_PersonOrder;
  ```

➢ **CHECK CONSTRAINT**

- used to limit the value range that can be placed in a column

-
  ```
  CREATE TABLE Persons (
      ID int NOT NULL,
      LastName varchar(255) NOT NULL,
      FirstName varchar(255),
      Age int,
      CHECK (Age>=18)
  );
  ```

-
  ```
  ALTER TABLE Persons
  ADD CHECK (Age>=18);
  ```

-
  ```
  ALTER TABLE Persons
  DROP CHECK CHK_PersonAge;
  ```

➢ **DEFAULT CONSTRAINT**

-
  ```
  CREATE TABLE Persons (
      ID int NOT NULL,
      LastName varchar(255) NOT NULL,
      FirstName varchar(255),
      Age int,
      City varchar(255) DEFAULT 'Sandnes'
  );
  ```

-
  ```
  ALTER TABLE Persons
  ALTER City SET DEFAULT 'Sandnes';
  ```

-
  ```
  ALTER TABLE Persons
  ALTER City DROP DEFAULT;
  ```

➢ **CREATE INDEX STATEMENT**

- used to create indexes in tables
- used to speed up searches/queries

-
  ```
  CREATE INDEX index_name
  ON table_name (column1, column2, ...);
  ```

-
  ```
  CREATE UNIQUE INDEX index_name
  ON table_name (column1, column2, ...);
  ```
- Create index for a particular column

  -
    ```
    CREATE INDEX idx_lastname
    ON Persons (LastName);
    ```

- **Create index on combination of columns**

    -
      ```
      CREATE INDEX idx_pname
      ON Persons (LastName, FirstName);
      ```
- **Drop index**

    -
      ```
      ALTER TABLE table_name
      DROP INDEX index_name;
      ```

➢ **AUTO INCREMENT FIELD**

-
  ```
  CREATE TABLE Persons (
      Personid int NOT NULL AUTO_INCREMENT,
      LastName varchar(255) NOT NULL,
      FirstName varchar(255),
      Age int,
      PRIMARY KEY (Personid)
  );
  ```
- AUTO INCREMENT STARTS FROM 1 AS DEFAULT
- We can modify that using the following code
- `ALTER TABLE Persons AUTO_INCREMENT=100;`

➢ **DATE DATA TYPES**

- **Format for date data types**

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
- YEAR - format YYYY or YY
- Selecting data with a certain date
    - `SELECT * FROM Orders WHERE OrderDate='2008-11-11'`
    - You must not include time portion in date to avoid the problems in fetching that select statement only searches for that date

➢ **SQL VIEWS**

-
  ```
  CREATE VIEW view_name AS
  SELECT column1, column2, ...
  FROM table_name
  WHERE condition;
  ```
- **EXAMPLE FOR CREATING A VIEW:**

    -
      ```
      CREATE VIEW [Brazil Customers] AS
      SELECT CustomerName, ContactName
      FROM Customers
      WHERE Country = "Brazil";
      ```
- **To display the view:**
    - `SELECT * FROM [Brazil Customers];`
- **Example:** creates a view that selects every product in the "Products" table with a price higher than the average price

-
  ```
  CREATE VIEW [Products Above Average Price] AS
  SELECT ProductName, Price
  ```

    FROM Products
    WHERE Price > (SELECT AVG(Price) FROM Products);

- SELECT * FROM [Products Above Average Price];
- **VIEW CAN BE UPDATED AS:**

  -
    ```
    CREATE OR REPLACE VIEW view_name AS
    SELECT column1, column2, ...
    FROM table_name
    WHERE condition;
    ```
- **DROPPING VIEW**
  - DROP VIEW *view_name*;

➢ **SQL INJECTION**

- code injection technique that might destroy your database
- one of the most common web hacking techniques
- SQL injection is the placement of malicious code in SQL statements, via web page input
- SUPPOSE YOU THINK THAT THIS IS THE FOLLOWING CODE

-
  ```
  txtUserId = getRequestString("UserId");
  txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
  ```
- now from down we describe how much danger the above code can be affected
- **Example 1: based on 1=1**
- SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1;
- If user enters 105 or 1=1 then the user gets all the data related to other users so the privacy is lost
- **Example 2: based on “”=””**
- **Suppose you have two textboxes to enter the data so if you enter the data Username as " or ""=" and password as " or ""="**
- **It is modifird as**
- SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
- This results in leakage of data of all the existing users
- **Example 3: based on batched sql statements**
- Batched sql statements means group of two or more sql statements separated by semicolon
- **Statements:**

-
  ```
  txtUserId = getRequestString("UserId");
  txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
  ```
- **How to inject:**
- Just enter 105; Drop table tablename;
- This results in return all the data and clear all the table data

➢ **How to protect from sql injection**

- We can escape from sql injection using sql parameters these paramenters are added when on excecution time in controlled manner

- **Example:**

-
  ```
  txtNam = getRequestString("CustomerName");
  txtAdd = getRequestString("Address");
  txtCit = getRequestString("City");
  txtSQL = "INSERT INTO Customers (CustomerName,Address,City)
  Values(@0,@1,@2)";
  db.Execute(txtSQL,txtNam,txtAdd,txtCit);
  ```
- **INSERT INTO STATEMENT USING PHP**

-
  ```
  $stmt = $dbh->prepare("INSERT INTO Customers
  (CustomerName,Address,City)
  VALUES (:nam, :add, :cit)");
  $stmt->bindParam(':nam', $txtNam);
  $stmt->bindParam(':add', $txtAdd);
  $stmt->bindParam(':cit', $txtCit);
  $stmt->execute();
  ```

- My SQL Function
  - https://www.w3schools.com/sql/sql_ref_mysql.asp

- **SQL HOSTING:**
  - If your web server is hosted by an Internet Service Provider (ISP), you will have to look for SQL hosting plans
  - most common SQL hosting databases are MS SQL Server, Oracle, MySQL, and MS Access
  - MySQL is an inexpensive alternative to the expensive Microsoft and Oracle solutions