DAILY ASSIGNMENT – 17/08/2021

SQL INJECTION

SQL injection is the technique to extract the database information through web application or in other words we can say that inserting SQL code in a query via user inputted data. It can occur in any applications using relational databases like Oracle, MySQL, PostgreSQL and SQL Server.
It happens when an attacker successfully tampers with the input of a web application, gaining the ability to execute arbitrary SQL queries on that application. SQL injection can be used to manipulate the application web server by malicious users.

SQL injection generally occurs when we ask a user to input their username/userid instead of a name or id, the user gives us an SQL statement that we will unknowingly run on our database.

EXAMPLE :-

We create a SELECT statement by adding a variable "samUserID" to select a string.

```
samUserId = getrequestString("UserId");

samSQL = "SELECT * FROM users

WHERE UserId = " +samUserid;
```

Types of SQL Injection

1. **Boolean based SQL Injection :-**
   It uses a Boolean expression that evaluates to true or false.

2. **Union based SQL Injection :-**
   SQL union operator combines data from two different queries with the same number of columns. In this case, the union operator is used to get data from other tables
   Input Data: 2 union select username, password from tableuser
   Query: SELECT first_name, last_name
   FROM tbl_employee
   WHERE empId = 2 union select username, password from tableuser
3. **Time based SQL Injection :-**
   This attack slows down the database server. It can bring down your application by affecting the database server performance.
4. **Error based SQL Injection :-**
   This variation, the attacker tries to get information like an error code and a message from the database.

STATEMENT :-

It uses this for general purpose access to your database. It is useful when we are using static SQL statements at runtime.

The statement interface cannot accept parameter.

(i) **create statement :-**
In java JDBC statement first we need to create a statement
Statement statement = connection.createStatement();

(ii) **executing a query via a statement :-**
Once we have created a Java Statement object, you can execute a query against the database.by calling its

executequery() method, passing an SQL statement as parameter.

```
String sql = "SELECT * FROM people";

ResultSet result = statement.executeQuery(sql);

while(result.next())
{
 String name = result.getString("name");
 long   age  = result.getLong  ("age");
}
```

(iii) **executing an update via a statement :-**

Execute an update of database in a java JDBC statement instance, for instance execute an SQL insert, update, or delete via statement instance.

```
Statement statement = connection.createStatement();
String    sql = "update people set name = 'John'
WHERE id = 123";
int rowsAffected = statement.executeUpdate(sql);
```

(iv) **closing a statement:-**

Once we are finished with a Statement instance we need to close it. We close a Statement instance by calling its close() method.

```
statement.close();
```

Prepare Statement :-

Java JDBC Prepared Statement is a special kind of Java JDBC Statement object with some useful additional features.

The Java JDBC Prepared Statement primary features are:

Easy to insert parameters into the SQL statement. Easy to reuse the Prepared Statement with new parameter value

```
String sql = "update people set firstname = ?, lastname = ?

where id = ?";

PreparedStatement preparedStatement  =
connection.prepareStatement(sql);

preparedStatement.setString(1, "Gary");

preparedStatement.setString(2, "Larson");

preparedStatement.setLong  (3, 123);

int rowsAffected = preparedStatement.executeUpdate();
```

creating a prepared statement:

```
String sql = "SELECT * FROM people WHERE id=?";

PreparedStatement preparedStatement =
connection.prepareStatement(sql);
```

Inserting Parameters into a PreparedStatement:

Everywhere you need to insert a parameter into your SQL, you write a question mark (?). For instance:

```
String sql = "SELECT * FROM people WHERE id=?" ;
```

executing the preparedstatement:

the executeupdate() method is used when updating the database

```
String sql = "update people set firstname = ? , lastname = ? where id =
?";

PreparedStatement preparedStatement =
connection.prepareStatement(sql);

preparedStatement.setString(1, "Gary");

preparedStatement.setString(2, "Larson");

preparedStatement.setLong  (3, 123);

int rowsAffected = preparedStatement.executeUpdate();
```

resuing a preparedstatement:

once a preparestatement is prepared it can be reused after the execution.

```
String sql = "update people set firstname=? , lastname=? where id=?";

PreparedStatement preparedStatement =
connection.prepareStatement(sql);

preparedStatement.setString(1, "Gary");

preparedStatement.setString(2, "Larson");

preparedStatement.setLong  (3, 123);


int rowsAffected = preparedStatement.executeUpdate();

preparedStatement.setString(1, "Stan");

preparedStatement.setString(2, "Lee");

preparedStatement.setLong  (3, 456);

int rowsAffected = preparedStatement.executeUpdate();
```

callable statement :-

The CallableStatement interface provides methods to execute the stored procedures. Since the JDBC API provides a stored procedure SQL escape syntax, we can call stored procedures of all RDBMS in single standard way.

EXAMPLE:-

```
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class CallableStatementExample {
   public static void main(String args[]) throws
SQLException {

      DriverManager.registerDriver(new
com.mysql.jdbc.Driver());


      String mysqlUrl = "jdbc:mysql:
      Connection con =
DriverManager.getConnection(mysqlUrl, "root",
"password");
      System.out.println("Connection
established......");
  3
      CallableStatement cstmt =
con.prepareCall("{call myProcedure(?, ?, ?)}");

      cstmt.setString(1, "Raghav");
      cstmt.setInt(2, 3000);
      cstmt.setString(3, "Hyderabad");

      cstmt.setString(1, "Kalyan");
      cstmt.setInt(2, 4000);
      cstmt.setString(3, "Vishakhapatnam");

      cstmt.setString(1, "Rukmini");
      cstmt.setInt(2, 5000);
```

```
        cstmt.setString(3, "Delhi");

        cstmt.setString(1, "Archana");
        cstmt.setInt(2, 15000);
        cstmt.setString(3, "Mumbai");

        cstmt.execute();
        System.out.println("Rows inserted ....");
    }
}
```

INNER CLASSES IN JAVA :-

1. **NESTED INNER CLASS :-**

   Can access any private instance variable of outer class. Like any other instance variable, we can have access modifier private, protected, public and default modifier

   EXAMPLE:-

```
class Outer {

       class Inner {

          public void show() {

               System.out.println("In a nested class
method");

          }

       }
    }
    class Main {

       public static void main(String[] args) {

           Outer.Inner in = new Outer().new Inner();

           in.show();
       }
    }
```

2. **<u>METHOD LOCAL INNER CLASSES :-</u>**
   Inner class can be declared within a method of an outer class.

   <u>EXAMPLE :-</u>

```
class Outer {

    void outerMethod() {

        System.out.println("inside outerMethod");


        class Inner {

            void innerMethod() {

                System.out.println("inside 
innerMethod");

            }
        }

        Inner y = new Inner();

        y.innerMethod();

    }
}

class MethodDemo {

    public static void main(String[] args) {

        Outer x = new Outer();

        x.outerMethod();

    }
    }
```

3. **<u>STATIC NESTED CLASSES :-</u>**
   Static nested classes are not an inner class. They are like a static member of outer class.

   <u>EXAMPLE :-</u>

```
class Outer {

       private static void outerMethod() {

         System.out.println("inside outerMethod");

       }

       static class Inner {
        public static void main(String[] args) {

            System.out.println("inside inner class
Method");

            outerMethod();

        }

    }
     }
```

4. **<u>ANONYMOUS INNER CLASS :-</u>**
   Anonymous inner classes are declared with name to all created in two ways

   (i) As subclass of specified type

<u>EXAMPLE :-</u>

```
class Demo {

    void show() {

        System.out.println("i am in show method of
super class");

    }
  }
```

```
    class Flavor1Demo { 2.

        static Demo d = new Demo() {

            void show() {

                super.show();

                System.out.println("i am in
Flavor1Demo class");

            }

        };

        public static void main(String[] args){

            d.show();

        }
    }
```

(ii) As implementer of the specified interface :-

EXAMPLE :-

```
class Flavor2Demo {

    static Hello h = new Hello() {

        public void show() {
            System.out.println("i am in anonymous
class");
        }
    };

    public static void main(String[] args) {
        h.show();
    }
}


interface Hello {
    void show();
}
```

RELATIONSHIPS IN JAVA

1. **IS-A RELATIONSHIP :-**
   Is-A relationship one class is obtaining the features of another class by using inheritance concept with extends keywords

   EXAMPLE :-

```java
class Faculty
{
float salary=30000;
}
class Science extends Faculty
{
float bonous=2000;

public static void main(String args[])
{
Science obj=new Science();
System.out.println("Salary is:"+obj.salary);
System.out.println("Bonous is:"+obj.bonous);
}
```

2. **<u>HAS-A RELATIONSHIP :-</u>**

   Has-A relationship an object of one class is created as data member in another class the relationship between these two classes is Has-A.

<u>EXAMPLE :-</u>

```
class Employee
{
float salary=30000;
}
class Developer extends Employee
{
float bonous=2000;
public static void main(String args[])
{
Employee obj=new Employee();
System.out.println("Salary is:"+obj.salary);
}
}
```

3. **<u>USER-A RELATIONSHIP :-</u>**

method of one class is using an object of another class the relationship between these two classes is known as user a relationship.

<u>EXAMPLE :-</u>

```
class Employee
{
float salary=20000;
}
class Salary extends Employee
{
void disp()
{
float bonous=1000;
Employee obj=new Employee();
float Total=obj.salary+bonous;
System.out.println("Total Salary is."+Total);
]
}
class Developer
[
public static void main(Striing args[])
{
salary s=new Salary();
s.disp()
}
}
```