~~Executa~~

by

- executeUpdate():
  - This method is used to execute non SELECT statement that modifies the D.B such as INSERT, UPDATE, DELETE or DDL.
  - It returns the integer value representing the number of row affected by the statement.

QBank

22fall 5a

Ans The program to print the total marks of student and find the average marks is:

```
import java.sql.*;
~~import~~

public class MarksFinder
{
  public static void main(String[] args)
  { try{
    //Registering the driver class
    Class.forName("com.mysql.cj.jdbc.driver");

    // creating connection
    String url = "jdbc:mysql://localhost:3306/db-std";
    String username = "localhost";
    String password = "";
```

cook ma of prepared statement!

```
Connection conn = DriverManager.getConnection(url, username,
                                              password);
```

(4)

```
// creating statement
Statement st = conn.createStatement();
String query1 = "SELECT COUNT(*) FROM tbl_std";
String query2 = "SELECT AVG(marks) FROM tbl_std";

// executing the statement
ResultSet rs1 = st.executeQuery(query1);
ResultSet rs2 = st.executeQuery(query2);
while(rs1.next())
{
  System.out.println("Total number of student in the
    table is: " + rs1.getInt(1));
}

while(rs2.next())
  {
    System.out.println("Average marks of
    student is: " + rs2.getInt(1));
  }

// closing the connection
    conn.close();
} catch(exception e)
  {
    e.printStackTrace();
  }
}
}
```

21-fall65

Soln

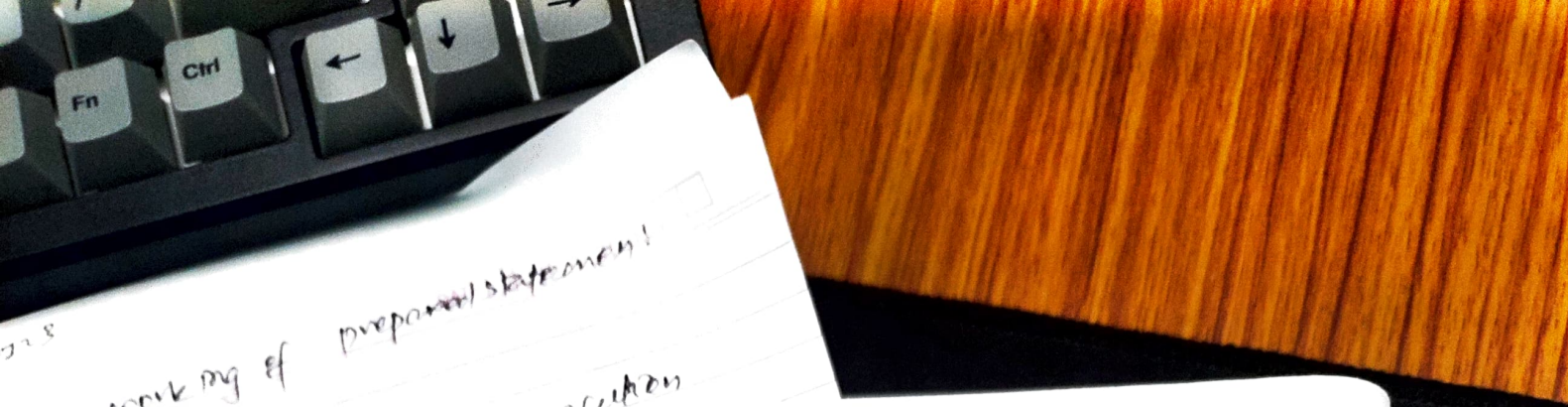
```
import java.sql.*;
import java.util.*;
import java.io.*;

public class UpdateDatabase
{
  public static void main (String[] args)
  { try{
    // registering a driver class
    class.forName("com.mysql.cj.jdbc.driver");

    // creating connection
    String url = "jdbc:mysql://localhost:3306/testdb";
    String username = "localhost";
    String password = "";

    Connection conn = DriverManager.getConnection
        (url, username, password);

    // creating statement
    Statement st = conn.createStatement();
    String userchoice = "y";
    // executing query
    Scanner sc = new Scanner(System.in);
while(userchoice != "n")
    { System.out.println("enter id, name & address of
      the employee: ");
    Int id = sc.nextInt();
    String name = sc.nextLine();
String address = sc.nextLine();
```



main working of prepared statement

```
String query = " INSERT INTO student values
               ( id, name, address )";
```

(86)

```
boolean result = st.execute(query);
if(!result)
{
  System.out.println(" sucessfuly inserted");
}
else
{
  System.out.println(" Insertion is failed");
}
System.out.println(" Dog you want to insert another
        row as well? press y for yes, n for no: ");
userchoice = sc.nextLine();
}
} catch(Exception e)
{
  e.printStackTrace();
}
}
}
```

}

}

1fall6b

```
import java.sql.*;
import java.util.*;

public class TwentyOnefallSixB {
    public static void main(String[] args)
    {
        String url = "jdbc:mysql://localhost:3306/testdb";
        String userName = "root";
        String pw = "";
```

```
//setting up the DB connection

try
{ //connection setting
  Connection conn = DriverManager.getConnection(url, user
                    ame, password);

  //statement creation
    Statement st = conn.createStatement();

  // creating query
    //Firstly taking user input
    Scanner scanner = new Scanner(System.in);
    char userChoice;
    do
    {
      System.out.println("StudentID: ");
int id= Scanner. nextInt();

      System.out.Print("Student name: ");
      String name= scanner.nextLine();

      System.out.Print("Address: ");
      String Address = scanner.nextLine();

      //query
      String query= "INSERT INTO student (id, name,
                    address) VALUES (?,?,?)";
    PreparedStatement preparedStatement = conn.prepd
      Statement(query);
```

working of prepared statement:

```
preparedStatement.setInt(1, id);
preparedStatement.setString(2, name);
preparedStatement.setString(3, address);

int row = preparedStatement.executeUpdate();

if(row > 0)
{
  Sout("Data inserted successfully");

}
else
  {
  Sout("Insertion failure");
  }

Sout("Continue entering data? (y/n): ");
userChoice = scanner.next().charAt(0);
Scanner.next line();

}while(choice == 'y' || choice = 'Y');

connection.close();
Statement.close();
Scanner.close();

}

Catch(SQLException e)
{
  e.PrintStackTrace();
    }
}
```

JDBC	ODBC
→ stands for Java Database connectivity	→ Stands Open Database Connectivity.
→ Introduced by SUN Microsystem in 1997	⇒ Introduced by Microsoft in 1992.
⇒ JDBC is only applicable for Java.	⇒ ODBC is applicable for C, C++, Java etc.
⇒ JDBC can be used in any platform like windows, mac, Linux.	⇒ ODBC can be used only in windows
⇒ JDBC is object oriented	⇒ ODBC is procedural.
⇒ Codebase is easier to understand	⇒ Codebase is complex to understand
⇒ JDBC acts as a bridge between Java apps and DB s/w.	⇒ ODBC acts as a bridge between Apps of any prog lang and DB
⇒ Commonly used for server side web apps	not commonly used for web dev.

working of prepared statement:

Different steps to connect to DB in Java

① Loading the DB driver

↳ Before creating the connection with DB, we need to load the driver related to specific DB. Each database has its own JDBC driver and we need to include the driver JAR file in the classpath. For exam for mysql we need to load, MYSQL JDBC driver as shown below cod.

② Creating connection

↳ DriverManager.getConnection(); is the method used for establishing the connection where we need to pass the URL, username and password of the localhost for the authentication purpose.

③ Creating Statement / prepared statement

↳ After establishing the connection, we need to create either statement or prepared statement. A statement is used for executing simple SQL queries while ~~complex queries~~ can be easily executed by prepared statement; or parameterised queries.

④ Executing SQL query

↳ We can use statement or prepared statement object to execute the SQL statement.

⑤ Closing the connection:

↳ To prevent from resource leakage, all resources must be closed as well.

Q.SP 4.

Type 1 JDBC driver	Type 4 JDBC driver
Eg:	
(1) It is also known as JDBC - ODBC driver	(1) Also known as Thin driver or Native protocol pure Java driver
(2) Here extra odbc driver needs to be installed on client machine	(2) No any extra setup has to be done in client machine
(3) Here the performance is slow since the JDBC method call has to converted into ODBC function call	(3) performance speed is faster as compared no any call transformation has to be done
(4) It is not written purely in Java	(4) It is written purely in Java programming language. Hence it is called as Native protocol driver
(5) It requires middleware as the ODBC layer	(5) Since no any middleware are need thin driver itself connects with Java apps and DB, so it is called as Thin driver
(6) Its connection is easier and easier to use	(6) Connection is a bit tricky
(7)	(7)

working of prepared statement:

JDBC Arch

JSP file

```
import java.util.*;
import java.sql.*;

public class UpdateClass
{
    public static void main
    (String[] args)
    {
    // registering the class

    String url = "jdbc:mysql://localhost:3306/
                  testdb/employee";

    String username = "root";
    String password = " ";

    try
    {
    //Loading the MYSQL JAB driver
    Class.forName("com.mysql.cj.jdbc.Driver");

    // setting the connection
    Connection conn = DriverManager.getConnection(url,
                      username, password);

    //creating the statement
    Statement st = conn.createStatement();
```

Class.forName("com.mysql.cj.jdbc.Driver");

Class.forN
url = "
String
int r
if (
{
el
{
}
}
catch
}
}

```
Class.forName("com.mysql.cj.jdbc.Driver");
url= "jdbc:mysql://localhost:3306/testdb
String updateQuery = "UPDATE employee set salary
                     = 50000 where post = 'manager'";

int rowAffected = st.executeUpdate(updateQuery);

if (rowAffected >0)
  {
    System.out.println("salary is updated for "+
      rowAffected + "managers");

  }
else
  {
   System.out.println("failed to updatesalary");

  }

}
catch (SqlException e)
   {
    e.printStackTrace();

   }

 }

}
```

working of prepared statement.

Simple statement	prepared statement
(a) It is used when sql query has to be executed only once	(a) Is used when sql query ha to be executed multiple time.
(b) Parameter can't be passed at run time	(b) We can pass parameters at run time
(c) It is only used for DDL statements.	(c) It is used for dynamic sql queries execution
(d) It has low performance speed.	(d) Performance speed is high than simple statement
(e) It is used to execute Normal sql queries	(e) It is used for any sql queries
(f) Binary Protocol is not used for communication	(f) Binary protocol is used for communication.
(g) We can't use statement for writing binary data.	(g) We can use prepared statement for writing binary data.

18SQ9 Benifit of using prepared statement in Java

↳ Prepared statement is a kind of statement that is used for performing the CRUD operation, that is specially comes under use when we need execute multiple sql query multiple times.

The prepared statement helps to improve the performance of query execution and keeps the DB secure from threat like SQL injection and many more.

simply when we execute regular SQL statement directly, DB needs to parse, compile and optimize the query each time it executed which leads to unnecessary overhead and performance drop. At the same time, with prepared statement, query is precompiled and optimized by the DB once and stored in statement object. At this stage place holder for parameter in the query is left unfilled and when we wanna perform the operation, we need to provide the attribute and DB execute the statement without recompiling it.

So, it is the main pros of prepared statement over statement. Some of the other advantage are:

(a) It is capable for Dynamic query execution,
(b) Improvement in performance
(c) Security level enhance, it prevents the DB from SQL injection attacks when using placeholder for parameter, DB treats the parameter values as data not executable code, prevent SQL malicious injection attack
(d) Readibility is high
(e) uses the Binary protocol for communication.

ain working of preparedstatement!

17 sep

(7c) Result set

↳ A result set object is a table of data representing a database result set, which is usually generated by executing a statement that queries the DB.

A resultset maintains a cursor pointing to the row of the table. Here initially the cursor points before the row. The default result set ~~~~ object is not update and has the cursor which moves forward only.

⇒ ResultSet is an Interface located inside JDBC API

16 fall 6a What is resultset Metadata. Provide a sample code to illustrate its usage.

↳ ResultSet MetaData provides information about the obtained resultset objects like the no. of columns names of column, data types of column, name of table etc.

Hence ResultSetMetaData is an Interface that provides the ~~information about~~ methods to get MetaData from

the Result

some met

⇒ getColum
from the

⇒ getColumn
↳ If re
from

⇒ getColu
n from

⇒ getT

the ResultSet object.

Some methods of ResultSet Metadata are:

⇒ getColumnCount(): It retrieves the number of column from the obtained resultset object.

⇒ getColumnLabel(): It retrieves the suggested name of the column for use.

⇒ getColumnName(): It retrieves the name of the column from the obtained resultset.

⇒ getTableName(): It retrieves the name of the table

in working of prepared statement:

Resultset usage in code

```
import java.mysql.*;

public class ResultSetDemo
{
    Public static void main(String [] args)
    {
        String url= "jdbc:mysql://localhost:3306/lab_
                    exam";
        String userName = "root";
        String password = "";
        String query = "select * from lab_students where age>=25";
        Connection conn=null;
        Statement st=null;
        ResultSet resultset =null;

        try
        {//creating connection
        Conn = DriverManager.getConnection(url, userNa-
                me, password);

        // Statement creation
        st= conn.createStatement();

        // executing query
            result= conn.executeQuery(query);

        //iterating throughout the resultset to store the
        table info in variable and displaying in console
        while( resultset.next())
            { int s_id= result.getInt("id");
              String s_name= resultset.getString("name");
              String address= resultset.getString("address");
```

```
sout( s_id + "  " + s_name + "  " + s_address );
}
}catch(SqlException e)
{
e.printStackTrace();
}finally
{
try
{ if(resultset!=null)
{
resultset.close();
}
}catch(SQLException e)
{
e.printStackTrace();
}
}
}
}
```