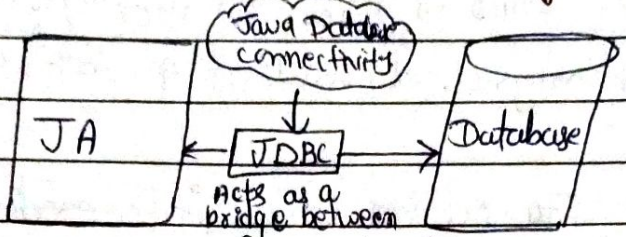
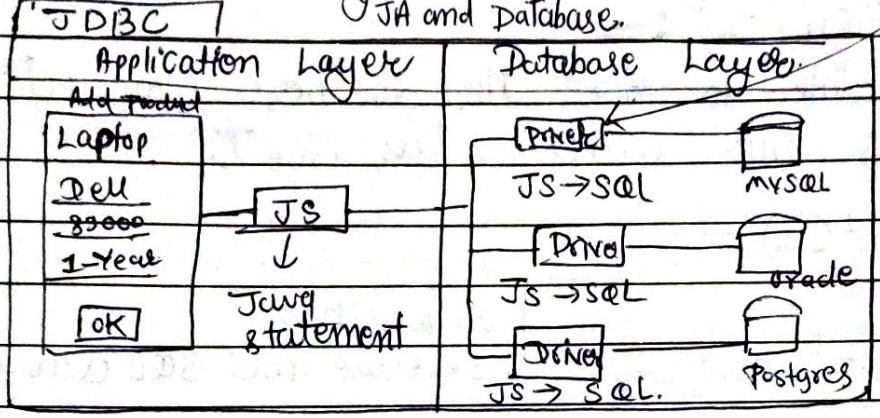
JDBC.

PAGE NO.:

Java Database Connectivity :-





provided by.
MySQL - Developer
MySQL-Connector jar
Java - Archive.
JDBC API
Interfaces
classes
Abstract classes

JDBC :- JDBC is an API that helps application to communicate with database. It allows java programs to connect to database, run queries, retrieve and manipulate data. Bcoz of JDBC, java applications can easily work with different relational database like MySQL, Oracle, PostgreSQL, and more.

 # Types of Driver :-

⇒ Type-1 :- JDBC-ODBC bridge driver.
⇒ Type-2 :- Driver or Native-API driver (partially java driver)
⇒ Type-3 :- driver or Network Protocol driver (fully java driver)
⇒ Type-4 :- Driver or thin driver.

JDBC API :- `import java.sql.*;` Provides the classes and Interfaces to write java code

① → Driver (I) :- convert JS → SQL
② → DriverManager (C) :- responsibility of DM
- to register the driver
- Get a database connection with database.

`[DriverManager.getConnection (String, String, String)]`

Static method.

eg: String 1 → Database URL
String 2 → ds_username
String 3 → ds_password.

* getConnection() → Always return Connection obj.

PAGE NO: XML Configure

Connection con

⇒ DriverManager.getConnection("Jdbc:mysql://localhost:3306/empdb", "root", "root");

[Jdbc through mysql at Localhost on Port 3306 to empdb]

To return the information

③ Connection (I) :-

Responsible to store the database Connection, It returns the child class object.

④ Statement (I)

⑤ PreparedStatement (I) → responsible. To execute the SQL Query.

⑥ Collable statement (I)

⑦ Resultset (I) :-

DDL DML DRL.

execute() (return boolean)

executeUpdate() (return int)

executeQuery() return Resultset.

⇒ It responsible to execute the result return from executeQuery. method. Select query.

String sql = "Select * from employee";
Resultset rs = st.executQuery(sql);

(8) ResultsetMetaData (I):-

⇒ If you want to get the information about resultset, than use resultsetMetaData.

⑨ SQLException (E) :- (checked exception)

→ Application Class loader Read.
→ Boot strap class loader
→ Extension class loader

PAGE NO:

→ Steps to Connect database :-

Step 1 :- Add Driver (.jar) in library path.

Step 2 :- Load Driver Class.

mysql	oracle
class Driver implements Driver { }	class oracleDriver implements Driver { }

Class Loading →

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

→ [Exception :- class not found Exception;]

= Pappu =

Class Loading
- Static → creating obj. → calling static method
- Dynamic → in java, there is a class, name "Class". Class.forName

Step : 3 :- Get Connection;
DriverManager.getConnection.

Step : 4 :- Get st/ps to execute the SQL Query.

Step : 5 :- Close the Connection.

```
Package com.testjdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class TestConnection {
Public static void main (String args[]){
try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    Connection con = DriverManager.getConnection("jdbc:mysql:
        //localhost:3306/empdb", "root", "");
```

```
System.out.println("Database connected : "+con);
  }
Catch( ClassNotFoundException e){
    e.printStackTrace ();
}
catch( SQLException e){
  e.printStackTrace();
}
}
}.
```

"STEPS To Use JDBC"

5 Steps :-

Step : 1:- Load and Register Driver

Load the database driver class so that java knows which database you want to connect to.

Example:-
```
Class.ForName("com.mySQL.cj.jdbc.Driver");
```
This loads the MySQL JDBC driver into memory.

Step : 2 :- Establish the Connection.

Create connection between your java program and the database using the DriverManager class.

Example :-
```
Connection con = DriverManager.getConnection(
"jdbc:mysql://localhost:3306/empdb","root","1234");
```

- **URL :-** Tells where the database is.
  (jdbc: mysql :// localhost: 3306/ empdb)
- **USERNAME & PASSWORD :-** for Authentication

Step :-3 :- Create a Statement

Create a statement or preparedstatement object to send a SQL queries to the Database.

example :- `Statement stmt = Con.createStatement();`

Step :- 4 :- Execute the Query.

Run your SQL query using the statement object.

- for Select (read) queries.

```
ResultSet rs = stmt.executeQuery("SELECT * from Student");
```

- for insert / update / delete queries.

```
stmt.executeUpdate(" inset into student values(1,'Ncha','BCA');
```

Step : 5 :- Process the results

If you need used a SELECT query, you get results inside a ResultSet. You can read data row by row like this.

```
while( rs.next()){
System.out.println( rs.getInt(1)+ "   "+rs.getString(2));
```

Step : 6 :- Close the Connection

After finishing, always close your Connection to free up the resources :-

```
Con.close();
```

How to establish Connection :-

```
package Com.testJdbc;
Public class TestConnection{
public . . . . . .
try{
    Class.ForName("Com.mysql.cj.Jdbc.Driver");
    Connection con = DriverManager.getConnection("jdbc:
    mysql://localhost:3306/empdb", "root", "root");
    S.o.p("Database Connected :" + con);
} Catch (ClassNotFoundException e){
        e.printStackTrace(); }
catch ( SQLException e){
    e.printStackTrace();
}
}
}
```

#

```
Public static void main(String args[]){
try ( Connection con = GetConnection.getConnection(); ){
    Scanner sc=new Scanner (System.in);
    S.o.p ("Enter Name");
    String employeeName = sc.Next();
    S.o.p("Enter Salary");
    int salary = sc.CnextInt();
    S.o.p("Center Skill");
    String Skill = sc.nextInt();
Statement st = Con.Createstatement();
String sql = "Insert into employee (name, Salary,
    Skill) values ('" + employeeName + "'," + Salary + ",
    '" + Skill + "')";
int x = St.executeUpdate(sql);
```

```
if (x1 = 0)
 S.o.p ("Record Inserted");
else{
   S.O.P ("Record not Inserted");
}
Catch (Exception e) {
    e.printStackTrace();
  }
 }
}
```

SQL Injection :- Because generally we can Contacte any parameters, that why this error occures "SQL Injection".

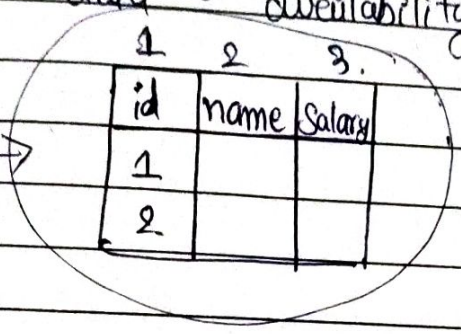
Statement	Prepared Statement
1) whenever you want to execute non-parameterised SQL query.	1) want to execute parameterised sql query.
2) Statement st = con.createStatement(); st.executeUpdate(sq) :- DML st.execute(sq) :- DDL ⇒ (Har bar Compile, har bar run).	2) PrepareStatement PS = con.prepareStatement(sq); PS.executeUpdate(); executeQuery(); PS.execute(); ⇒ 1 bar Compile, har bar run.
3) SQL Injection ✓	3) SQL Injection ✗

```
PSVM( ){
try{
(Connection con = GetConnection.getConnection();){
  Scanner sc = new Scanner( System.in);
  S.O.P(" Enter name");
  String name = sc.next();
  S.O.P(" Enter Salary");
  int Salary = sc.nextInt();
  S.O.P(" Enter skill");
  String skill = sc.next();

  String sql = " insert into employee(name, skill, salary
                values(?,?,?)";
 PreparedStatement ps = con.prepareStatement(sql);
  ps.setString(1, name);
  ps.setString(2, skill);
  ps.setInt(3, Salary);

 int x= ps.executeUpdate();
if( x!=0){
  S.O.P(" Record Inserted...");}
else{
 S.O.P(" not Inserted...");
 }
}
Catch( Exception e){
  e.printStackTrace();
 }
}
}
```

Before accessing the data from result set, first check the data availability ⇒ next();



⇒

MVC (multi layer Architecture)

Model view Controller

[frontend UI]

Model :- Model represent data in a application. (Simply java class).

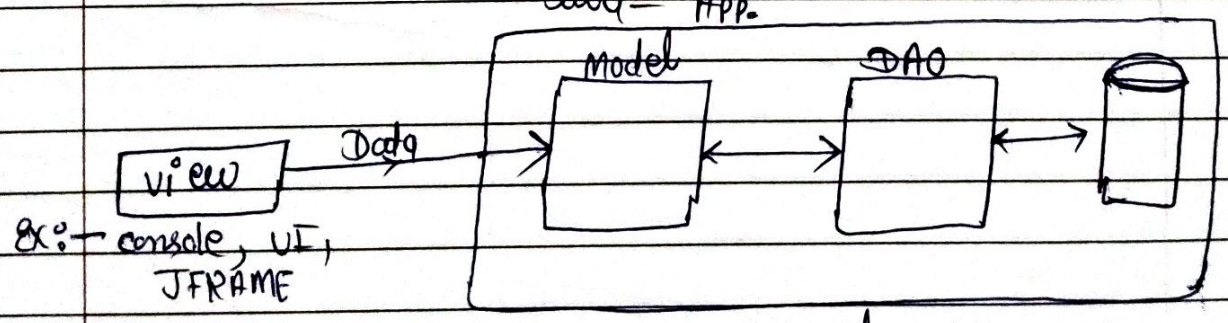
Ex:- employee

id	name	salary
1		
2		
3		

→

Model

```
class employee{   //Pojo class.
int id;
String name;
String salary;
}.
```



~~DAO~~ ⇒ ~~Database~~ Access object.