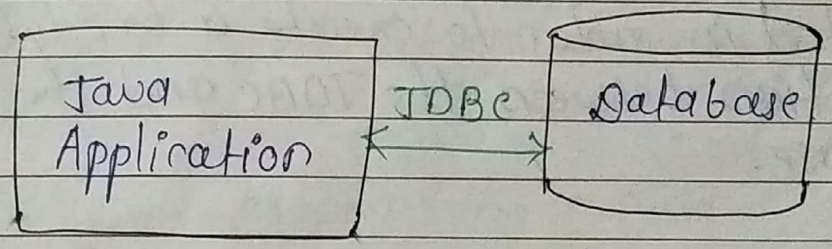
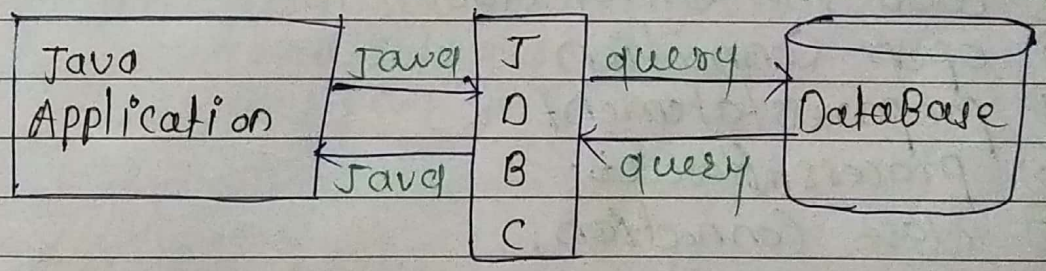
JDBC

* Java DataBase Connectivity

- JDBC stands for Java database Connectivity
- It provides an interface betn the java base appn and the database appn.
- Hence JDBC can be called as an API (Application programming interface).
- There is no other Technology which can help us to communicate betn the Java application and the database.

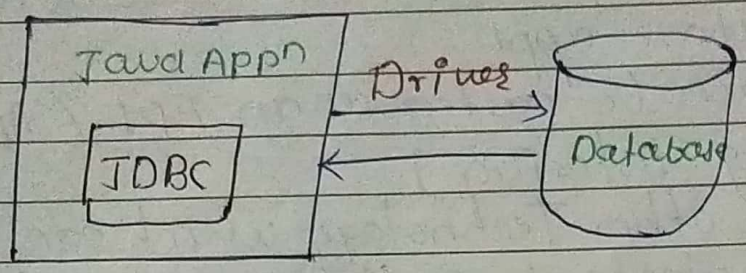




- The Java Appn understands the Java programming language only. whereas the database understands the query language only.
- both these applications are incompatible to understand each other.
- In order to establish communication betn the Java appn & database a mediator is Required.
- JDBC acts as the mediator in this Scenario. It will convert the Java program into

Scanned with CamScanner

the corresponding query language and vice versa.



- The JDBC logic is present inside the Java Appn itself. Hence a driver has to be appointed in order to create a bridge / connection between the JDBC and the Database.

- Steps of JDBC
  1. Load the Driver class
  2. open Connection
  3. prepare statement
  4. process the Result.
  5. close Connection.

way 1) jdbc

```
package com.jspider.jdbc.main;
import java.sql.Statement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException

public class Jdbc {
p s v main(String[]args) {
try {
```

```
// step1: load the driver class
class.forName("com.mysql.cj.jdbc.Driver");

// step2: open the connection
Connection connection = DriverManager.getConnection
                        ("jdbc:mysql://"
                        + "localhost:3306/wejee2?"
                        + "user=root&password=root");

// step3: create / prepare the statement.
Statement statement = connection.createStatement();
ResultSet resultSet = statement.executeQuery
                        ("select * " + "from student");

// step4: process the result.
while (resultSet.next()) {
s.o.pln (resultSet.getString(1) + " || " +
        resultset.getString(2) + " || " +
        resultSet.getString(3));
  }

// Step5: close the connections
  connection.close();
  statement.close();
  resultset.close();
}
  catch (ClassNotFoundException e) {
    e.printStackTrace();
} catch (SQLException e) {
    e.printstackTrace();
  }
}
}
```

2)

```
package com.jspider.jdbc.main;
import java.sql.connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class JdbcInsert {
p s v main (String [] args) {
  try {
  // step 1: load the driver class
class.forName ("com.mysql.cj.jdbc.Driver");

// step 2: open the connection
Connection connection = DriverManager.
                    getConnection ("jdbc:mysql://"
                        + "localhost:3306/weje2?"
                        + "user=root&password=root");

// step 3: Create/prepare the statement
Statement statement = connection.CreateStatement();
int result = statement.executeUpdate
            ("insert into student "
            + "values (2,'Baburao', "
          + "'babu@thetq.com');");

// step 4: process the result.
  s.o.pln( result + "row(s) affected ");

// step 5: close the connections
   connection.close();
   Statement.close();
```

Scanned with CamScanner

```
} catch (classNotFoundException e) {
        e.printStackTrace();
} catch (SQLException e) {
        e.printStackTrace();
}
}
}
```

The database name mentioned in the DBURL should be existing

- Prerequesting for writting JDBC program
  The database connector jar file has to be deploied on the build path of the project.

- Procedure to deploy MySQL connector Jar file
- Search for maven Repository
  - In maven Repository type my-SQL connector and look for mysql connector /J
  - Select the table version of mysql connector
  - Download the Jar file for the stable version

Note - Next step need to perform in Eclipse
- Go to the package explorer in the IDE

- Right click on the target project go to build path and click on configure ~~the~~ field path
- In the configuration window go to the library tab
- click on add external Jars (JARS)
- In the selection window open the folder where the JAR file is located and select the JAR file
- once the JAR file is added to the build path click on apply and close.

In JDBC Next page

- once the JAR file is loaded we can see one added structure in project Named referenced Libraries
- Now the classes and Interfaces required for JDBC will be available with in the project

2) procedure - the database name mentioned in the DBURL should be existing.

* DBURL
- DBURL is the string value used to locate the Database application in a system.
- format of DBURL is as follows.

protocol : subprotocol : // host-name : port-no / db-name ? user = username & password = password

1* protocol -
- It indicates the technology that is being implimented
- In this case it is JDBC

2* Subprotocol -
- It indicates the database technology which is under use, in this case it is MySQL

3* host Name -
- It indicates that the database application is present in which system.
- It can be either 'local host' if the database application is in the same system where

JDBC is being implimented. or it has to be the ip address of the system.

4) port no -
- It indicates the exact location of the database application in the located system.

5) database name -
- It indicates the database required for execution

3 of connection on JDBC
- In JDBC program the connection betn the java application and the database can be done in 3 diff. ways.
- The getConnection() method from the DriverManager class is responsible to establish the connection. It is the overloaded method with 3 different method signatures.

① getConnection (String url)
② getConnection (String url, String user, String pass.)
③ getConnection (String url, properties info)

2nd way of establishing connection.
③

jdbc1
```
package com.jspider.jdbc.main;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException
import java.sql.Statement
import java.sql.
```

```
public class Jdbc1 {
public static void main (String [] args) {
  try {
  // step1: load the driver class
   Class.forName ("com.mysql.cj.jdbc.Driver");

// step 2: open the connection
Connection connection = DriverManager.
    getConnection ("jdbc:mysql://localhost:
                    3306/jweje2" "root", "root");

// step 3: Create/prepare the statement.
Statement statement = Connection.CreateStatement();
ResultSet resultSet = statement.executeQuery
                    ("Select *" + "from student");

// step 4: process the Result.
while (resultSet.next()) {
s.o.pln( resultset.getString(1) + " " +
         resultset.getString(2) + " " +
         resultSet.getString(3) );
 }
//
// step5: Close the connections
    Connection.close();
    Statement.close();
    resultSet.close();
 }
    catch (classNotFoundException e) {
      e.printStackTrace();
 } catch (SQLException e) {
3 }   e.printStackTrace();
```

select

(4) A code standerdization

```
package com.jspiders.jdbc1.select;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class JdbcSelect {
private static Connection connection;
private static Statement statement;
private static ResultSet resultSet;

public static void main (String [] args) {
  try {
   class.forName("com.mysql.cj.jdbc.Driver");
 connection = DriverManager.getConnection
            ("jdbc:mysql://localhost:3306/weje29"
          +"user=root&password=root");
 Statement = connection.CreateStatement();
 resultSet = statement.executeQuery
            ("select * from student");
 while (resultSet.next()) {
 S.o.pln (resultSet.getInt(1) + " | " +
          resultSet.getString(2) + " | " +
          resultSet.getString(3));
  }
 } catch (ClassNotFoundException e) {
      e.printStackTrace();
 } catch (SQLException e) {
    e.printStackTrace();
 }
```

Scanned with CamScanner

```
finally {
    if (connection != null) {
    try {
        connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
if (statement != null) {
    try {
        statement.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
if (resultSet != null) {
    try {
    statement.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
if (resultSet != null) {
    try {
        resultset.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}
}
}
```

Scanned with CamScanner

* For deleting record from database

```
package com.jspider.jdbc1.delete;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SqlException;
import java.sql.Statement;
public class JdbcDelete {
 private static Connection connection;
 private static Statement statement;
 private static int result;

 public static void main (String [] args) {
   try {
   Class.forName ("com.mysql.cj.jdbc.Driver");
 connection= DriverManager.getConnection
          ("jdbc:mysql://localhost:3306/wejez?"
          +"user=root&password=root");

 Statement = connection.CreateStatement();

 result = statement.executeUpdate
          ("delete from student where id = 1");
 S.o.pln (result + " row(s) affected ");
 } catch (ClassNotFoundException e) {
     e.printStackTrace();
 }
 catch (SQLException e) {
     e.printStackTrace();
 }
   Finally {
    if (connection != null) {
     try {
```

```
        connection.close();
    }
    catch (SQLException e) {
        e.printStackTrace();
    }
}
if (statement != null) {
    try {
        statement.close();
    }
    catch (SQLException e) {
        e.printStackTrace();
    }
}
}
}
```

* for updating

6 * ~~delete~~ update

```
package com.jspider.jdbc1.update;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.SQLException;
import java.sql.DriverManager;

public class JdbcUpdate {
private static Connection connection;
private static Statement statement;
private static int result;

public static void main(String[] args) {
try {class.forName("com.mysql.cj.jdbc.Driver");
Connection = DriverManager.getConnection
("jdbc:mysql://localhost:3306/weje?
user=root & password=root");
```

Scanned with CamScanner

```
statement = Connection.CreateStatement();
result = statement.executeUpdate ~~();~~
        ("update student set SName='Arzoo' where SId=3");
s.o.pln(result + " row(s) affected ");

} catch (ClassNotFoundException e) {
        e.printStackTrace();
} catch (SQLException e) {
        e.printStackTrace();
} finally {
  ~~try~~ {
        if (connection != null) {
        try {
            connection.close();
        }
        catch (SQLException e) {
            e.printStackTrace();
        }
    }
        if (statement != null) {
        try {
            statement.close();
        }
        catch (SQLException e) {
        e.printStackTrace();
        }
    }
  }
}
}
```

Scanned with CamScanner

- why to use 3rd way of connection
- While using the first two ways of establishing connection in the JDBC program the database credentials that is username and password are exposed this may cause database security bridge. In order to avoid this we can make use of the third way of establishing connection.
  their, we need to create a property file which would act as the resource for our JDBC program. this property file will contain the database credentials in the form of key-value pair
- The extension of the property file should be '.properties'

* procedure to create the properties file
- Go to the project and create new General folder.
- Name the folder as Resources.
- Under the Resources folder create a new file and name it as "db-info.properties"
- Inside the properties file declare the properties of user and password.
- To load the properties file we need the absolute path of the file.

db-info.properties:-

```
user = root
password = root
```

3rd way

```
package com.jspider.jdbc2.select;

import java.io.FileReader;
import java.io.IoException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.properties;

Public class JdbcSelect {

private static Connection connection;
private static statement statement;
private static ResultSet resultset;
private static properties properties = new properties();
private static FileReader fileReader;

public static void main (String [] args) {
  try {
   Class.forName("com.mysql.cj.jdbc.Driver");

FileReader = new FileReader ("D:\\WEJE2\\"
                + "jdbc2\\resources\\"
                + "db-info.properties");
properties.load (fileReader);

connection = DriverManager.getConnection
          ("jdbc:mysql://localhost:3306/wejee2"
          , properties);
```

```
statement = Connection.createStatement();
resultSet = Statement.executeQuery
            ("select * from student");
while (resultSet.next()) {
  s.o.pln (resultset.getInt(1)
          + " | " + resultSet.getString(2)
          + " | " + resultSet.getString(3));
}
} catch (ClassNotFoundException e) {
    e.printStackTrace();
} catch (SQLException e) {
    e.printStackTrace();
} catch (IoException e) {
    e.printStackTrace();
}
  finally {
  if (connection != null) {
    try {
    connection.close();
    }
    catch (SQLException e) {
    e.printStackTrace();
} }
  if (statement != null) {
    try {
    statement.close();
    } catch (SQLException e) {
    e.printStackTrace();
    }
}
```

Scanned with CamScanner

```
if (resultSet != null) {
    try {
        resultSet.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}
```

methods in JDBC

Method Name	class / Interface	member	signature	return type	Access Modifier
forName(String className)	class. class	static	forName(String className)	class	public
getConnection (String url)	DriverManager	static	getConnection(String url)	connection	public
getConnection (String url, String user, String pass)	DriverManager	static	getConnection (String url, String user, String password)	connection	public
getConnection (String url, properties)	DriverManager	static	getConnection (String url, properties info)	Connection	public
CreateStatement()	Connection	non-static	CreateStatement()	Statement	public
executeQuery (String url)	statement. class	non-static	executeQuery (String url)	ResultSet	public

Scanned with CamScanner

method name	class/Interface	Signature	return type	Access modifier	member
executeUpdate (String url)	Statement class	executeUpdate (String url)	ResultSet	public	non-static
load (Reader, reader)	properties class	load (Reader reader)	void	public	non-static
close ()	connection class	close ()	void	public	non-static
getString (int column index)	ResultSet class	getString (int columnIndex)	String	public	non-static
getBoolean (int columnIndex)	ResultSet class	getBoolean (int columnIndex)	boolean	public	non-static
getByte (int columnIndex)	ResultSet class	getByte (int columnIndex)	byte	public	non-static
getShort (int columnIndex)	ResultSet class	getShort (int columnIndex)	Short	public	non-static
getInt (int columnIndex)	ResultSet class	getInt (int columnIndex)	int	public	non-static
getFloat (int columnIndex)	ResultSet class	getFloat (int columnIndex)	Float	public	non-static

Scanned with CamScanner

getLong (int columnIndex)	ResultSet .class	getLong (int columnIndex)	long	public	non-static
getDouble (int columnIndex)	ResultSet .class	getDouble (int columnIndex)	double	public	non-static

Dynamic Query

- dynamic query help us to compile a query without the values & provide a feature to assign the values at runtime.
- The statement interface is not capable of handling dynamic queries, Hence we need to make use of an interface called as PreparedStatement. unlike the statement interface the prepared statement interface is capable of handling both static as well as dynamic queries. This is possible because preparedStatement interface is a child interface of statement.

① package com.jspiders.jdbc3.insert;

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IoException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Properties;
```

jdbc1 (insert)

```
public class JdbcInsert {

private static Connection connection;
private static preparedStatement preparedStatements
private static int result;
private static properties properties = new
                Properties();
private static FileReader fileReader;
private static String filepath =
    "D:\\KIEJE2\\jdbc3\\resources\\ db-info.
        properties";
private static String query;

public static void main (String [] args) {
    try {
  fileReader = new FileReader (filepath);
   properties.load (FileReader);
   class.forName (properties.getproperty("driverpath"))
connection = DriverManager.getConnection
            (properties.getproperty ("dburl"),
                properties);
query = "insert into student values (?, ?, ?)";

preparedStatement = connection.preparestatement
                        (query);
preparedstatement.setInt(1, 3);
preparedStatement.SetString (2, "Babita");
preparedStatement.SetString(3, "babita@iyer.com")
result = preparedstatement.executeUpdate();
S.o.pln (result + "row(s) affected.");
```

```
} catch (ClassNotFoundException e) {
    e.printStackTrace();
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IoException e) {
    e.printStackTrace();
} catch (SQLException e) {
    e.printStackTrace();
}
Finally {
if (connection != null) {
try {
connection.close();
} catch (SQLException e) {
    e.printStackTrace();
}
}
if (preparedStatement != null) {
try {
    preparedStatement.close();
} catch (SQLException e) {
    e.printStackTrace();
}
}
}
}
```

Scanned with CamScanner

* Methods in Dynamic query.

Method Name	class / Interface	Signature	member	return type.	Access modifier
getproperty (String key)	properties. class	getproperty (String key)	non-static	String	public
getProperty (String key, String default value)	properties. class	getproperty (String key, String defaut value)	non-static	String	public
prepareStatement (String sql)	connection. class	prepareStatement (String sql)	non-static	prepared Statement	public
executeUpdate()	Prepared statement	executeUpdate()	non-static	int	public
executeQuery()	prepared statement	executeQuery()	non-static	ResultSet	public

Static query	Dynamic query
• Static query is that do not change at runtime	• Dynamic query is constructed at runtime.
• In static query the methods executeQuery and executeUpdate will accept an argument in the string format	• In dynamic query the methods executequery & execute will not accept any argument.
• In static query createStatement ()	• In dynamic query

Scanned with CamScanner

is required for preparing the statement.	prepareStatement (String sql) present in Connection interface will required to prepare Statement.
· static query will get initialized with values first and do not change at the Runtime.	· Dynamic query will get compile first and can execute it during runtime with passing values.

* Stored procedure

· Like methods in java we can create stored procedure in database we can declare one or multiple queries as the body of the procedure. This procedure can be called for execution whenever Required.

Syntax of creating stored procedure.

```
delimiter /
    create procedure proc-name()
    begin
        statement 1;
        statement 2;
        :
        statement n;
    end /
delimiter;
```

Scanned with CamScanner

* Syntax to call the procedure for execution

```
Call proc-name ();
```

① Create procedure proc1() {delimiter / }
begin
select * from Student;
insert into student values (4, 'undertaker', 'deadman@wwe.com');
select * from student;
end /

delimiter ;

* Callable Statement

- In order to execute a stored procedure from JDBC program we cannot make use of the statement or prepared statement interface. we need to use callable statement interface.
- We can achieve the object of callable statement from a method named 'prepareCall' which is present in the connection interface.
  the procedure calling statement / query has to be given as an argument to prepare call method.

② package

```
public class JdbcCall {
private static Connection connection;
private static CallableStatement callableStatement;
private static ResultSet resultset;
private static Properties properties = new
                    properties ();
private static FileReader Filereader;
private static String Filepath = "D:\\WEJE2\\jdbc3
                \\resources\\db-info.properties";
private static String query;

public static void main(String [] args) {
  try {

    FileReader = new FileReader (Filepath);
    properties.load (FileReader);
    Class.ForName (properties.getproperty
                    ("driverpath"));

    Connection = DriverManager.getConnection
                (properties.getproperty ("dburl")
                , properties);

    query = " call proc2 () ";

    CallableStatement = Connection.prepareCall
                    (query);

    resultSet = callableStatement.executeQuery ();
```

Scanned with CamScanner

```
while (resultSet.next()) {
  S.o.pln (resultSet.getInt(1)
          + " | " + resultSet.getString(2)
          + " | " + resultSet.getString(3));
  }
} catch (FileNotFoundException e) {
        e.printStackTrace();
} catch (IoException e) {
        e.printStackTrace();
} catch (classNotFoundException e) {
        e.printStackTrace();
} catch (SQLException e) {
        e.printStackTrace();
}
  finally {
   if (connection != null) {
     try {
        connection.close();
   } catch (SQLException e) {
        e.printStackTrace();
   }
}

 if (resultSet != null) {
   try {
      resultSet.close();
 } catch (SQLException e) {
      e.printStackTrace();
   }
}
 if(callableStatement != null) {
    try {
```

```
callableStatement.close();
}
catch (SQLException e) {
    e.printStackTrace();
}
}
}
}
```

② package com.jspider.jdbc4.entity;

```
public class Student {
private int id;
private String name;
private String email;

public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
public string getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public String getEmail() {
    return email;
}
```

Scanned with CamScanner

```
    public void setEmail (String email)
    {
        this.email = email;
    }
}
```

Dynamic select *:

```
public class JdbcSelect {
private static Connection connection;
private static preparedStatement preparedStatement;
private static ResultSet resultset;
private static properties properties =
                    new properties();
private static FileReader FileReader;
private static String filepath = "D:\\WEJE2\\
                    jdbc3\\resources\\db-info.
                    properties";
private static String query;

public static void main (String[] args) {
    try {
    FileReader = new FileReader (filepath);
properties.load (FileReader);
    class.forName (properties.getproperty
                    ("driverpath"));

Connection = DriverManager.getConnection
            (properties.getproperty ("dburl")
            , properties);

query = "select * from student";
```

```
prepared Statement = Connection.preparedStatement
                                    (query);
resultSet = preparedStatemen.executeQuery();

 while (resultSet.next()){
   Student student = new Student();
 Student.setId (resultSet.getInt(1));
 Student.setName (resultSet.getString(2));
 Student.setEmail (resultSet.getString(3));

 s.o.pln ( Student );
 }
} catch (FileNotFoundException e) {
        e.printStackTrace();
} catch ( IoException e)
        e.printStackTrace();
} catch (ClassNotFoundException e)
        e.printstackTrace();
} catch (SQLException e)
      e.printStackTrace();
 }
  finally {
  if (connection != null){
     try {
     connection.close();
} catch (SQLException e) {
     e.printStackTrace();
 }
}
  if (resultSet != null){
     try {
```

```
        resultSet.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    } if (preparedStatement != null) {
        try { preparedStatement.close();
    } catch (SQLException e) { e.printStackTrace(); }
    }
```

* Drawbacks of Jdbc:

- while writing a Jdbc program the java programer is forced to take care of the sql related properties and credentials
- Queries need to be written explicitely by the programmer.
- In Jdbc the conversion from java class to database table is not possible. there is no relationship betn the java objects and the data in the database.

Note- due to the drawbacks Jdbc has become a deprecated technology.

Scanned with CamScanner