EXPERIMENT NO. 14

1. What is JDBC?

Ans:

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database.

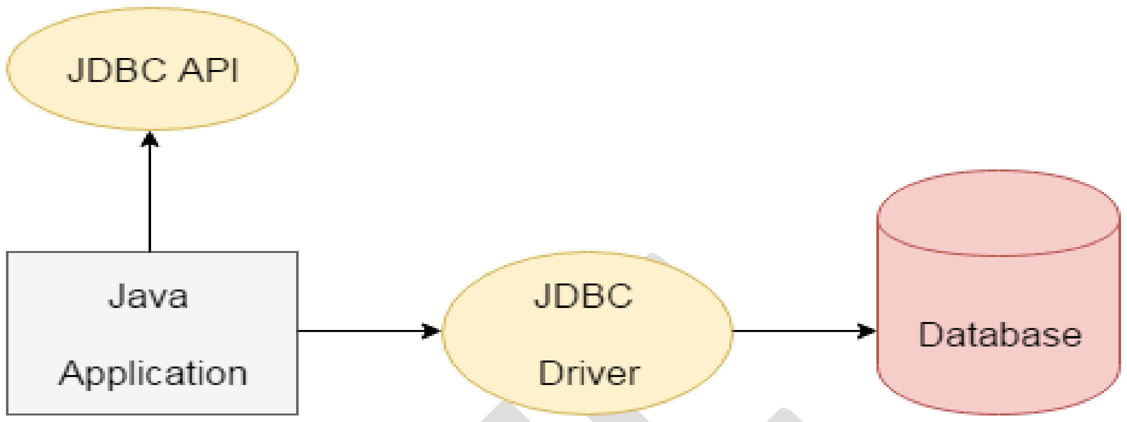
2. What are the JDBC driver types? which is most useful type ? and why?

Ans:

There are four types of JDBC drivers:

• JDBC-ODBC Bridge	…(Type 1)
• Native-API partly JAVA	…(Type 2)
• Net-protocol All-java Driver	…(Type 3)
• Native-Protocol All-Java Driver	…(Type 4)

- **JDBC-ODBC bridge driver:** The JDBC-ODBC bridge driver uses the ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of the thin driver. It is easy to use and can be easily connected to any database.
- **Native-API driver (partially java driver):** The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in Java. Its performance is better than JDBC-ODBC bridge driver. However, the native driver must be installed on each client machine.
- **Network Protocol driver (fully java driver):** The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is entirely written in Java. There is no requirement of the client-side library because of the application server that can perform many tasks like auditing, load balancing, logging, etc.
- **Thin driver (fully java driver):** The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as the thin driver. It is entirely written in Java language. Its performance is better than all other drivers however these drivers depend upon the database.



If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4.

If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.

Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.

The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.

Type 4 is most useful because of it is purely made in java and doesn't required and other driver to connect to the database through java rather the vendor specified driver.

3.What are different methods are available to load database driver?

Ans:

In short, you have two options:

- registerDriver() method
- Class.forName()

1) registerDriver() method - provides the functionality to register the drivers so that they can function properly and don't have to be changed or installed again and again

2) Class.forName() - is having the value as string and used to load the JDBC driver class.

4. What kind of information present in database connection string?

Ans:

Below given information is present in the database connection string,

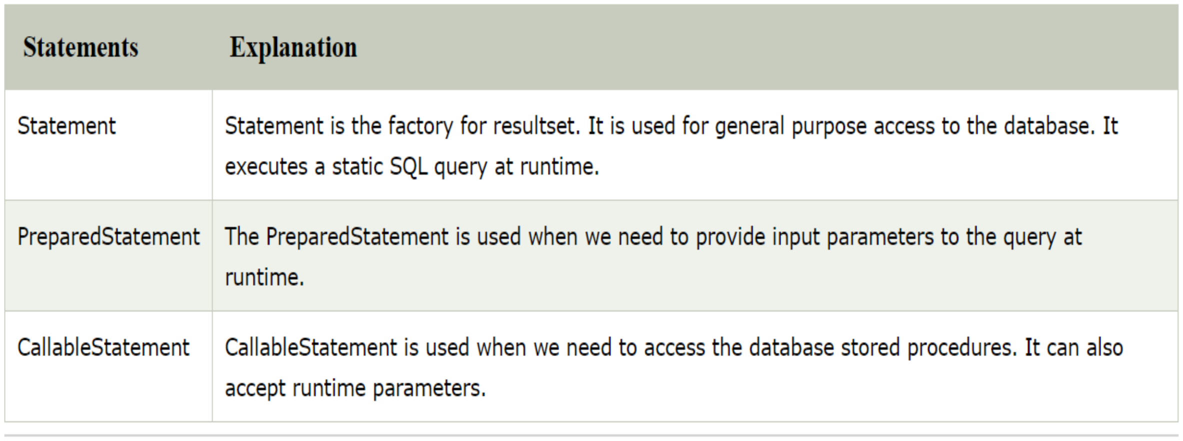
1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/db** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and db is the database name. We may use any database, in such case, we need to replace the db with our database name.
3. **Username:** The default username for the mysql database is **root**.
4. **Password:** It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

5. What are the different types of statement ? Mention the benefits of each type of statement?

Ans:

In JDBC, Statements are used to send SQL commands to the database and receive data from the database. There are various methods provided by JDBC statements such as execute(), executeUpdate(), executeQuery, etc. which helps you to interact with the database.

There is three type of JDBC statements given in the following table.



Statements	Explanation
Statement	Statement is the factory for resultset. It is used for general purpose access to the database. It executes a static SQL query at runtime.
PreparedStatement	The PreparedStatement is used when we need to provide input parameters to the query at runtime.
CallableStatement	CallableStatement is used when we need to access the database stored procedures. It can also accept runtime parameters.

Statement: Use for the general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.

Prepared Statement: Use the when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.

CallableStatement: Use the when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters.

6. Why db code is surrounded by try-catch block ?

Ans:

Basically, any checked exception needs to be either **handled** (option 1) or to be **specified** (option 2) in the method signature (look here on **Catch or Specify Requirement**).

Now, specific to, DriverManager's getConnection method, it throws a checked exception (i.e., SQLException) and you can look for the API below (or here). So you can handle it using try-catch block or specify it (using throws cluase in the method signature).

For e.g.

public static Connection getConnection(String url) throws SQLException

Attempts to establish a connection to the given database URL. The DriverManager attempts to select an appropriate driver from the set of registered JDBC drivers.

Returns: a connection to the URL

Throws: SQLException - if a database access error occurs or the url is null

7. What are the steps to connect to Oracle/MySQL DB using TYPE 4 driver.

Ans:

The following steps are used in database connectivity.

- **Registering the driver class:**

The forName() method of the Class class is used to register the driver class. This method is used to load the driver class dynamically. Consider the following example to register OracleDriver class.

1. Class.forName("oracle.jdbc.driver.OracleDriver"); …(For oracle Database)
2. Class.forName("com.mysql.jdbc.Driver"); …(For MySQL Database)

- **Creating connection:**

The getConnection() method of DriverManager class is used to establish the connection with the database. The syntax of the getConnection() method is given below.

1) **public static** Connection getConnection(String url)**throws** SQLException
2) **public static** Connection getConnection(String url,String name,String password)
1. **throws** SQLException

Consider the following example to establish the connection with the Oracle database.

1.Connection con=DriverManager.getConnection(
2."jdbc:oracle:thin:@localhost:1521:database_identifier","system","password");

Similarly for MySQL database ,

1. Connection con=DriverManager.getConnection(
2. "jdbc:mysql://localhost:3306/database_name","root","password");

- **creating the statement:**

The createStatement() method of Connection interface is used to create the Statement. The object of the Statement is responsible for executing queries with the database.

1.**public** Statement createStatement()**throws** SQLException

consider the following example to create the statement object

1.Statement stmt=con.createStatement();

- **Executing the queries:**

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method is given below.

1. **public** ResultSet executeQuery(String sql)**throws** SQLException

Example to execute the query

```
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

- **Closing connection:**

By closing connection, object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method is given below.

```
public void close()throws SQLException
```

Consider the following example to close the connection.

```
con.close();
```

8. How to connect to MongoDB from java?

Ans:

The steps for the connection to MongoDB are as follows,

1. Download the MongoDB Java Driver
Version (2.11.1) …latest stable
2. Connection to MongoDB using MongoClient

The **MongoClient** class is used to make a connection with a MongoDB server and perform database-related operations.
Here are some examples:
• Creating a MongoClient instance that connects to a default MongoDB server running on, localhost and default port:

```
MongoClient mongoClient = new MongoClient();
•Connecting to a named MongoDB server listening on the default port (27017):

MongoClient mongoClient = new MongoClient("localhost");
Or:
MongoClient mongoClient = new MongoClient("db1.server.com");
•Connecting to a named MongoDB server listening on a specific port:

MongoClient mongoClient = new MongoClient("localhost", 27017);
Or:
MongoClient mongoClient = new MongoClient("db1.server.com", 27018);
```

•Connecting to a replica set of servers:

```
List<ServerAddress> seeds = new ArrayList<ServerAddress>();
seeds.add(new ServerAddress("db1.server.com", 27017));
seeds.add(new ServerAddress("db2.server.com", 27018));
seeds.add(new ServerAddress("db3.server.com", 27019));

MongoClient mongoClient = new MongoClient(seeds);
```

After the connection is established, we can obtain a database and make authentication (if the server is running in secure mode), for example:

```
MongoClient mongoClient = new MongoClient();
DB db = mongoClient.getDB("test");

char[] password = new char[] {'s', 'e', 'c', 'r', 'e', 't'};
boolean authenticated = db.authenticate("root", password);

if (authenticated) {
    System.out.println("Successfully logged in to MongoDB!");
} else {
    System.out.println("Invalid username/password");
}
```

By default, MongoDB server is running in trusted mode which doesn't require authentication

3.Using MongoDB connection URL

Syntax of the URI is as follows:

mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN[:portN]]][/[database][?options]]

```
Connecting to the MongoDB server running on localhost at the default port:
mongodb://localhost

Connecting to the admin database on a named MongoDB server db1.server.com running on port 27027 with user root and password secret:
mongodb://root:secret@db1.server.com:27027

Connecting to the users database on server db2.server.com:
mongodb://db2.server.com/users

Connecting to the products database on a named MongoDB server db3.server.com running on port 27027 with user tom and password secret:
mongodb://tom:secret@db3.server.com:27027/products

Connecting to a replica set of three servers:
mongodb://db1.server.com,db2.server.com,db3.server.com
```