**Explain what is JDBC?**

JDBC stands for Java Database Connectivity. As its name implies, it is a Java API used for interacting with relational databases to access, modify and process data using SQL. It utilizes JDBC drivers for interacting with the database. By using JDBC, one can access tabular data from different types of relational databases such as MySQL, [Oracle](https://www.simplilearn.com/steps-to-become-oracle-database-certified-rar343-article), [MS Access](https://www.simplilearn.com/what-is-microsoft-access-article), and so on.

### What is JDBC Driver?

JDBC Driver is a software component that enables Java application to interact with the database. There are 4 types of JDBC drivers:

1. **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.
2. **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.
3. **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.
4. **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.

[More details.](https://www.javatpoint.com/jdbc-driver)

### 3) What are the steps to connect to the database in java?

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.

Class.forName("oracle.jdbc.driver.OracleDriver");

* **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.

public static Connection getConnection(String url)throws SQLException

public static Connection getConnection(String url,String name,String password) throws SQLException

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

Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","root", "root");

* **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.

public Statement createStatement()throws SQLException

consider the following example to create the statement object

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.

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));

}

However, to perform the insert and update operations in the database, executeUpdate() method is used which returns the boolean value to indicate the successful completion of the operation.

* **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();

[More details.](https://www.javatpoint.com/steps-to-connect-to-the-database-in-java)

**Components of JDBC:**

There are four major components of JDBC using which it can interact with a database. They are:

1. **JDBC API**: It provides different methods and interfaces for easier communication with the database. By using this, applications are able to execute SQL statements, retrieve results and make updation to the database. It has two packages as follows which consist of Java SE and Java EE platforms to exhibit Write Once Run Everywhere(WORA) capabilities.
   1. java.sql.\*;
   2. javax.sql.\*;  
      Also, it provides a standard for connecting a database to a client application.
2. **JDBC DriverManager**: It is the class in JDBC API. It loads the JDBC driver in a Java application for establishing a connection with the database. It is useful in making a database-specific call for processing the user request.
3. **JDBC Test suite**: It is used to test the operations like insertion, deletion, updation etc., being performed by JDBC Drivers.
4. **JDBC-ODBC bridge drivers**: It will connect database drivers to the database. JDBC-ODBC bridge interprets JDBC method call to the ODBC function call. It will use sun.jdbc.odbc package, which consists of the native library to access characteristics of ODBC.

### https://s3.ap-south-1.amazonaws.com/myinterviewtrainer-domestic/public_assets/assets/000/000/701/original/JDBC_Components.jpg?1626849763

### 4) What are the JDBC API components?

The java.sql package contains following interfaces and classes for JDBC API.

**Interfaces:**

* **Connection:** The Connection object is created by using getConnection() method of DriverManager class. DriverManager is the factory for connection.
* **Statement:** The Statement object is created by using createStatement() method of Connection class. The Connection interface is the factory for Statement.
* **PreparedStatement:** The PrepareStatement object is created by using prepareStatement() method of Connection class. It is used to execute the parameterized query.
* **ResultSet:** The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points before the first row. The executeQuery() method of Statement interface returns the ResultSet object.
* **ResultSetMetaData:** The object of ResultSetMetaData interface cotains the information about the data (table) such as numer of columns, column name, column type, etc. The getMetaData() method of ResultSet returns the object of ResultSetMetaData.
* **DatabaseMetaData:** DatabaseMetaData interface provides methods to get metadata of a database such as the database product name, database product version, driver name, name of the total number of tables, the name of the total number of views, etc. The getMetaData() method of Connection interface returns the object of DatabaseMetaData.
* **CallableStatement:** CallableStatement interface is used to call the stored procedures and functions. We can have business logic on the database through the use of stored procedures and functions that will make the performance better because these are precompiled. The prepareCall() method of Connection interface returns the instance of CallableStatement.

**Classes:**

* **DriverManager:** The DriverManager class acts as an interface between the user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. It contains several methods to keep the interaction between the user and drivers.
* **Blob:** Blob stands for the binary large object. It represents a collection of binary data stored as a single entity in the database management system.
* **Clob:** Clob stands for Character large object. It is a data type that is used by various database management systems to store character files. It is similar to Blob except for the difference that BLOB represent binary data such as images, audio and video files, etc. whereas Clob represents character stream data such as character files, etc.
* **SQLException** It is an Exception class which provides information on database access errors.

### 5) What are the JDBC statements?

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. |

### 6) What is the return type of Class.forName() method?

The Class.forName() method returns the object of java.lang.Class object.

### 7) What are the differences between Statement and PreparedStatement interface?

Here is a comparison of the Statement and PreparedStatement interfaces in tabular format:

| **Feature** | **Statement** | **PreparedStatement** |
| --- | --- | --- |
| **Definition** | Used to execute simple SQL queries | Used to execute precompiled SQL queries with parameters |
| **Compilation** | SQL query is compiled each time it is executed | SQL query is precompiled and stored in a PreparedStatement object |
| **Performance** | Slower for repeated executions due to recompilation | Faster for repeated executions due to precompilation |
| **Security** | More susceptible to SQL injection attacks | Provides better protection against SQL injection attacks by using parameterized queries |
| **Parameter Handling** | Does not support parameterized queries | Supports parameterized queries using placeholders (?) |
| **Use Case** | Suitable for executing dynamic SQL queries | Suitable for executing static SQL queries or queries that are executed multiple times |
| **Batch Processing** | Supports batch processing | Supports batch processing |
| **Flexibility** | More flexible for dynamic SQL queries | Less flexible for dynamic SQL queries but safer and more efficient for repetitive tasks |
| **Resource Usage** | Higher resource usage for repetitive tasks | Lower resource usage for repetitive tasks |

### What the differences are between execute, executeQuery, and executeUpdate?

| **Feature** | **execute** | **executeQuery** | **executeUpdate** |
| --- | --- | --- | --- |
| **Purpose** | Used to execute any kind of SQL statement | Used to execute SQL SELECT statements | Used to execute SQL INSERT, UPDATE, DELETE, and DDL statements (like CREATE, ALTER, DROP) |
| **Return Type** | boolean indicating if the result is a ResultSet | ResultSet containing the results of the query | int indicating the number of rows affected |
| **Result Handling** | Can handle both ResultSet and update counts | Always returns a ResultSet | Returns the number of rows affected |
| **Use Case** | General-purpose execution of SQL statements | Retrieving data from a database | Modifying data in the database or modifying database structure |
| **Flexibility** | Most flexible, as it can execute any type of SQL statement | Limited to SELECT statements | Limited to DML and DDL statements |
| **Typical Use Cases** | When the type of SQL statement is not known at compile time | When retrieving data is required | When modifying data or schema is required |
| **Result** | Can be ResultSet or update count | Always a ResultSet | Always an update count |

**What are the differences between Statement ,PreparedStatement and callableStatement ?**

| **Feature** | **Statement** | **PreparedStatement** | **CallableStatement** |
| --- | --- | --- | --- |
| **Purpose** | Executes simple SQL queries | Executes precompiled SQL queries with parameters | Executes stored procedures and functions in the database |
| **SQL Query Type** | Static SQL queries | Precompiled and parameterized SQL queries | Stored procedures and functions |
| **Compilation** | SQL query is compiled each time it is executed | SQL query is precompiled and stored in the PreparedStatement object | Stored procedure is precompiled and stored in the database |
| **Performance** | Slower for repeated executions due to recompilation | Faster for repeated executions due to precompilation | Faster for executing stored procedures due to precompilation |
| **Security** | More susceptible to SQL injection | Provides better protection against SQL injection by using parameterized queries | Provides better protection against SQL injection by using parameterized queries |
| **Parameter Handling** | Does not support parameterized queries | Supports parameterized queries using placeholders (?) | Supports input, output, and input-output parameters |
| **Use Case** | Suitable for executing dynamic SQL queries | Suitable for executing static SQL queries or queries executed multiple times | Suitable for executing stored procedures and functions |
| **Batch Processing** | Supports batch processing | Supports batch processing | Supports batch processing |
| **Flexibility** | More flexible for dynamic SQL queries | Less flexible for dynamic SQL queries but safer and more efficient for repetitive tasks | Suitable for complex operations involving stored procedures |
| **Result Handling** | Can return ResultSet or update counts | Typically returns ResultSet for queries and update counts for updates | Can return ResultSet, update counts, and output parameters |
| **Transaction Management** | Supports transaction management | Supports transaction management | Supports transaction management |
| **Resource Usage** | Higher resource usage for repetitive tasks | Lower resource usage for repetitive tasks | Lower resource usage for executing stored procedures |
| **Typical Use Cases** | Ad-hoc queries, dynamic SQL execution | Frequent execution of the same query with different parameters | Complex database operations, business logic encapsulated in stored procedures |

**what is ResultSet and RowSet in jdbc**

### ResultSet

**ResultSet** is an interface in JDBC used to represent a set of results returned by a SQL query. It provides methods to iterate through the rows of data and retrieve column values. A ResultSet object is generated by executing a Statement, PreparedStatement, or CallableStatement.

#### **Key Features:**

* **Types of ResultSet**:
  + **TYPE\_FORWARD\_ONLY**: The cursor can only move forward.
  + **TYPE\_SCROLL\_INSENSITIVE**: The cursor can scroll forward and backward but is not sensitive to changes made by others.
  + **TYPE\_SCROLL\_SENSITIVE**: The cursor can scroll forward and backward and is sensitive to changes made by others.
* **Concurrency Types**:
  + **CONCUR\_READ\_ONLY**: The ResultSet cannot be updated.
  + **CONCUR\_UPDATABLE**: The ResultSet can be updated.
* **Cursor Navigation**:
  + Methods like next(), previous(), first(), last(), absolute(int row), relative(int rows) to navigate through the ResultSet.
* **Data Retrieval**:
  + Methods like getString(), getInt(), getFloat(), etc., to retrieve column values by column name or index.

#### Example:

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM users");

while (rs.next()) {

String username = rs.getString("username");

System.out.println("Username: " + username);

}

### RowSet

**RowSet** is a subinterface of ResultSet in JDBC. It provides more functionality and flexibility compared to ResultSet. RowSet objects are typically used for disconnected RowSets and can operate without being connected to a database, which makes them suitable for transferring data between different layers of an application.

* **Disconnection**: CachedRowSet and its extensions can operate in a disconnected state, reducing the load on the database.
* **Flexibility**: Provides additional methods for event listeners, XML serialization, and filtering.
* **Scrolling and Updating**: Supports bi-directional scrolling and updating of data.

#### Example:

RowSetFactory factory = RowSetProvider.newFactory();

CachedRowSet crs = factory.createCachedRowSet();

crs.setUrl("jdbc:mysql://localhost:3306/mydb");

crs.setUsername("username");

crs.setPassword("password");

crs.setCommand("SELECT \* FROM users");

crs.execute();

while (crs.next()) {

String username = crs.getString("username");

System.out.println("Username: " + username);

}

### What are the functions of the JDBC Connection interface?

The **Connection interface** maintains a session with the database. It can be used for transaction management. It provides factory methods that return the instance of Statement, PreparedStatement, CallableStatement, and DatabaseMetaData.

### What are CLOB and BLOB data types in JDBC?

**BLOB:** Blob can be defined as the variable-length, binary large object which is used to hold the group of Binary data such as voice, images, and mixed media. It can hold up to 2GB data on MySQL database and 128 GB on Oracle database. BLOB is supported by many databases such as MySQL, Oracle, and DB2 to store the binary data (images, video, audio, and mixed media).

**CLOB:** Clob can be defined as the variable-length, character-large object which is used to hold the character-based data such as files in many databases. It can hold up to 2 GB on MySQL database, and 128 GB on Oracle Database. A CLOB is considered as a character string.

### What is the major difference between java.util.Date and java.sql.Date data type?

The major difference between java.util.Date and java.sql.Date is that, java.sql.Date represents date without time information whereas, java.util.Date represents both date and time information.

**What is database connection pooling? What are the advantages of connection pool?**

* Connection pooling means database connections will be stored in the cache and can be reused when future requests to the database are required. So in this mechanism, the client need not make new connections every time for interacting with the database. Instead of that, connection objects are stored in the connection pool and the client will get the connection object from there.
* Advantages of using a connection pool are:
  + It is faster.
  + Easier to diagnose and analyze database connections.
  + Increases the performance of executing commands on a database.

**6. What is “Dirty read” in terms of database?**

* Dirty read implies the meaning “read the value which may or may not be correct”. In the database, when a transaction is executing and changing some field value, at the same time another transaction comes and reads the changed field value before the first transaction could commit or rollback the value, which may cause an invalid value for that particular field. This situation is known as a dirty read.
* Consider an example given below, where Transaction 2 changes a row but does not commit the changes made. Then Transaction 1 reads the uncommitted data. Now, if Transaction 2 goes for roll backing its changes (which is already read by Transaction 1) or updates any changes to the database, then the view of the data may be wrong in the records related to Transaction 1. But in this case, no row exists that has an id of 100 and an age of 25.

