**JDBC(API)**

A screenshot of a cell phone

Description generated with very high confidence

**Query Processing System:**

A screenshot of a computer

Description generated with very high confidence

Definition 1: JDBC is a step by step procedure to interact with database from java applications to perform database operations from java applications.

Definition 2: JDBC is a technology which provides a good environment to connect with database and to perform operations.

Definition 3: JDBC is an API [collection of classes and interfaces].

Definition 4: JDBC is an abstraction developed by sun micro systems, vendors implement this as per their requirement to interact with the database.

**Driver:** used to map Java API to database API.

* Driver is the interface provided by developers and its implementation is provided by database vendors.

A screenshot of a cell phone

Description generated with high confidence

A screenshot of a cell phone

Description generated with very high confidence

* There are 5 types of drivers.

1. **Type-1 Driver:**  Provide by Sun microsystems as a reference to all database vendors.
   * Sun.jdbc.odbc.jdbcOdbcDriver
   * Also called as **jdbc Odbc driver or bridge driver.**
   * Prepared on the basis on the Microsoft product ODBC (open database connectivity) driver.
   * ODBC is a specification provided by Microsoft based on its own native implementation which provides a good environment to interact with any type of database from JDBC-ODBC driver.
   * If we want to use this driver in JDBC applications, we must configure ODBC driver in our system.
   * Suggested for only standalone applications not for distributed applications.
   * Limitations:
     1. Less Portability (almost platform dependent)
     2. Requires two conversions (Slower driver)
     3. Need to install to use the driver.

A close up of a map

Description generated with high confidence

1. **Type-2 Driver:**

* Also called as **“part java part native driver” or native driver**. Because designed based on database vendor provided native library and java implementations.
* More portable than type-1(dependent on database library/ less dependent)
* One conversion is sufficient which in turn increases the performance than type-1.
* This is also suggested for standalone applications not for distributed application.
* In order to use this driver in JDBC application we must install DB vendor provided native library.
* Cost full driver.

A screenshot of a social media post

Description generated with very high confidence

1. **Type-3 Driver:**
   * Also called **Middleware database server access driver or Network driver.**
   * Only for distributed applications.
   * We must provide application server environment.
   * More portable when compared to type-1 and type-2.
   * Faster when compared to type-1 and 2 because driver will connect with database on the basis of DB systems IP address and Port number.
   * Provides environment to connect multiple databases from multiple java applications.
   * This driver requires some middleware component, which says that application server is mandatory to use this type of driver.

A screenshot of a social media post

Description generated with very high confidence

1. **Type-4 Driver:**
   * Also called as “pure java driver” or “thin driver”.
   * Suggestable for both standalone and distributed applications.
   * More portable driver (designed completely based on java)
   * Faster driver. (good performance)
   * Frequently used driver.
   * Economical driver.

A screenshot of a cell phone

Description generated with very high confidence

Java has predefined libraries in java.sql package and javax.sql package

* Driver
* DriverManager
* Connection
* Statement
* PreparedStatement
* callableStatement
* ResuktSet
* DatabaseMetaData
* ResultSetMetaData

In javax.sql DataSource and PooledConnection.

**Steps to prepare first JDBC application:**

1. Load and register driver.
2. Establish connection between java application and database.
3. Create either statement or prepared statement or callable statement object.
4. Write and execute SQL queries.
5. Close the resources like connection and statement.
6. **Load and register driver:**
   1. Set class Path variable. (No need to set class Path variable for type-1 driver as it is part of java libraries.)
   2. Load driver class batch code to the memory.
      1. Ex: Class c = Class.forName(“sun.jdbc.odbc.jdbcodbcDriver”);
   3. By this loading a static block will be executed which calls **DriverManager,registerDriver(--) method., means driver is avtivated.**
   4. JVM collects the meta data of driver class and store in the heap memory with help of class type variable.
   5. To use type-1 driver we need to configure odbc driver, means provide DSN (data source Name) we get identity to interact with database.
   6. Java removed type-1 driver from 1.8 version. If you still use you get ClassNotFoundException.(**Refer session 10**)

**A screenshot of a cell phone

Description generated with very high confidence**

1. **Establish connection between java application and database.**
   1. Public static Connection getConnection(String driver\_URL, String db\_User\_Name, String db\_Password) throws SQLException

Type -1 Sun Microsystems:

Driver class : sun.jdbc.odbc.JdbcOdbcDriver

Driver URL: jdbc:odbc:dsn\_Name

**Type -4 Driver Oracle Corp:**

Driver class: oracle.jdbc.OracleDriver

**DriverURL : jdbc:oracle:thin:@localhost:1521/xe** // oracle XE

**DriverURL : jdbc:oracle:thin:@localhost:1521:ORCL** // oracle /10g11g

Path Location : C:\oraclexe\app\oracle\product\11.2.0\server\jdbc\lib\ojdbc6.jar

**Type-4 Driver MySQL : (Sql installation :video 23 )**

Driver Class : com.mysql.jdbc.Driver[MySQL5.x]

Driver Class : com.mysql.cj.jdbc.Driver[MySQL8.x]

DriverURL : jdbc:mysql://localhost:3306/db\_Name

Download jar for 5.X versions for 8.X versions it is available in connector files

**Format: Main\_protocol\_Name : Sub\_protocol\_Name : DB\_Name**

**Main\_protocol\_Name is fixed “jdbc”.**

Connection con = DRivermanager.getConnection(“jdbc:odbc:nag”,”system”,”saikumar”):

* Inside getConnection method connect method will be executed which creates a virtual socket connection between database and java application by creating a connection object which is stored in con variable.
* Connection is an interface here, it creates an object for implemented class of connection interface given by database vendor

A screenshot of a cell phone

Description generated with very high confidence

1. **Create either Statement or PreparedStatement or CallableStatement**

* We use statement when we want to execute SQL queries individually. (means performs tokenization, parsing, optimization and execution)
* We use Prepared statement when we want to execute same SQL query in the next sequence to improve the performance.
* We use callable Statement when we want to access stored procedures and functions which are at database side.

Public statement createStatement() throws SQLException

Ex: Statement st = con.createStatement();// this will return the statement object of the implemented class.

1. **Write and execute SQL queries.**

* executeQuery() – select sql queries
  + public ResultSet executeQuery(String query) throws SQLException
  + we can also use non-select sql queries, but raising an exception is completely depends upon the type of driver(method implementation). Here, to get row count use getUpdateCount()/
* executeUpdate() – non select sql queries
  + public int executeUpdate(String query) throws SQLExceprion
  + returns number of rows effected by update query.(depends on the type of driver used)
  + if you use select query here, raising an exception completely depends upon type of driver using. Type -1 raises an error but type-4 doesn’t.
* execute() – Both select and non select Sql Queries.
  + Public Boolean execute(String query) throws SQLException
  + Returns true when it executes select sql queries.
  + Returns false when it executes non-select sql queries.
  + To get ResultSet object reference we should use method getResultSet().
  + While using non-select Sql queries if we want row count we should use getUpdateCount() method.

A screenshot of a social media post

Description generated with very high confidence

**Note: Loading and registering driver for type-4 driver of Oracle 11g is optional.**

**A screenshot of a cell phone

Description generated with high confidence**

**A screenshot of a social media post

Description generated with very high confidence**

**Database metadata:**

* Data about the database product in our jdbc application.
* To represent the metadata object jdbc API has predefined interface **java.sql.databaseMetadata,** where implementation classes were provided by database vendors.
* This about able to provide the information like database name, product version, minor and major versions, supported sql keywords, supported string functions and numeric functions.
* Use the following method from connection object.
  + DatabaseMetaData md = con.getMetaData();

**ResultSet MetaData:**

* Data about ResultSet object.
* Predefined interface name is **java.sql.ResultSetMetaData**
* Provides details like no of columns, column names, column sizes and data types etc.
* Use the following method of java.sql.ResultSet
  + ResultSetMetaData rmd = rs.getMetaData();

**A screenshot of a social media post

Description generated with very high confidence**

**Result Sets :**

* **As per ResultSet cursor movement:**

**Forward Only ResultSet:**  can read data only in forward direction

* public static final int TYPE\_FORWARD\_ONLY;

**Scrollable ResultSet :**  can read data in both forward and reverse directions

* Make sure ResultSet cursor is at the end of the ResultSet object. For each record check for the previous record is existed.
* Use public Boolean previous throws SQLException
* Need to use rs.refreshNow() method to refresh the result set.(**It is not supported by Type-4 driver of oracle database**)
* Only reflects the changes if it is commited.
* **Scroll Sensitive ResultSet :** Modifications done to the database reflect in this type of ResultSet(listens to changes in database).
  + public static final int TYPE\_SCROLL\_SENSITIVE
  + **Type-4 driver provided by oracle doesn’t support Scroll sensitive ResultSet**.
* **Scroll Insensitive ResultSet : :** Modifications done to the database does not reflect in this type of ResultSet
  + public static final int TYPE\_SRCOLL\_INSENSITIVE
  + **Type-1 driver provided by Sun Micro Systems doesn’t support Scroll InSensitive ResultSet**.
  + **Type-4 driver provided by oracle doesn’t support Scroll Insensitive ResultSet**.
  + **Type-4 driver of MYSQL doesn’t support Scroll Insensitive ResultSet**
* **As per ResultSet concurrency:**

**Read Only ResultSet:**  allows only to read

* + Public static final int CONCUR\_READ\_ONLY

**Updateable ResultSet:** allow to read as well as update

* + Public static final int CONCUR\_UPDATEABLE

**Conclusion :**

* + **Both oracle and Microsoft provided type-4 driver doesn’t support Scroll Insensitive ResultSet. Also type-1 of sun microsystems**
  + **Oracle provided type-4 driver doesn’t support both sensitive and Insensitive results sets.**

**A screenshot of a cell phone

Description generated with very high confidence**

**Default Type : Read only and Forward only.(**given by jdbc**)**

Can create customizable ResultSet using passing those parameters to create statement constructor.

Ex: public Statement createStatement(int forward\_Only/Scroll\_Sensitive/Scroll\_Insensitive,

int Read\_only/Updateable ) throws SQLException.

A screenshot of a computer

Description generated with very high confidence

**A picture containing screenshot

Description generated with high confidence**

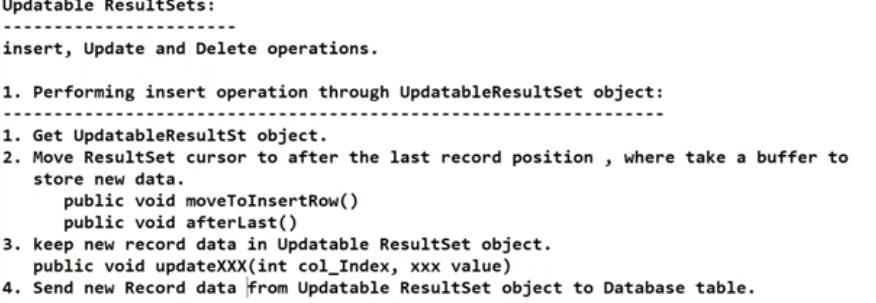
**A screenshot of a cell phone

Description generated with very high confidence**

**A picture containing indoor

Description generated with very high confidence**

**Updateable ResultSets:**

****

**A picture containing screenshot, text

Description generated with high confidence**

**A screenshot of a cell phone

Description generated with very high confidence**

**A close up of a device

Description generated with high confidence**

**To delete a row using updateable resultset.**

**A screenshot of a social media post

Description generated with very high confidence**

**Connecting to database using properties object:**

**A screenshot of a cell phone

Description generated with very high confidence**

**Getting connection through file:**

* We can use Properties class of util package to get the key value pairs of text file.
* If we use this no need to compile the class for each change related to username , password and driverURL and driver class.

A screenshot of a cell phone

Description generated with very high confidence

**Getting connection through Factory Class:** instead of creating multiple connection objects for different methods we can use single connection object by using this.

**A screenshot of a cell phone

Description generated with high confidence**

**A screenshot of a social media post

Description generated with very high confidence**

**A screenshot of a cell phone

Description generated with very high confidence**

**Prepared Statement:**

To improve the performance of executing a sql query that is to skip QT, QP, QOP every time we use prepared statement instead of Statement.

A picture containing screenshot, text

Description generated with very high confidence





Here, “?” is the place holder stating the parameter indexes. No of “?”’s creates the no of parameters in prepared statement object. **Here the parameter index is important**.

**Inserting Current system date and a particular date using Java.SQL.Date:**

**A screenshot of a cell phone

Description generated with very high confidence**