Chapter-11 Database Handling

You have to know about API to understand this chapter.

API *imp*

- Application Programming Interface
- It is a software intermediary that allows two applications to talk to each other.
- It is a collection of communication protocols and subroutines used by various programs to communicate between them.
- API facilitates the programmers with an efficient way to develop their software programs.
- API helps two programs or applications to communicate with each other by providing them with necessary tools and functions.
- API takes the request from the user and sends it to the service provider and then again sends the result generated from the service provider to the desired user.
- API is similar to a GUI (Graphical User Interface) with one major difference. Unlike GUI's, an API helps the software developers to access the web tools while a GUI helps to make a program easier to understand by the users.

When you use an application on your mobile phone, the application connects to the Internet and sends data to a server. The server then retrieves that data, interprets it, performs the necessary actions and sends it back to your phone. The application then interprets that data and presents you with the information you wanted in a readable way. This is what an API is - all of this happens via API.

Example to Understand API

Imagine you're sitting at a table in a restaurant with a menu of choices to order from. The kitchen is the part of the "system" that will prepare your order. What is missing is the critical link to communicate your order to the kitchen and deliver your food back to your table. That's where the waiter or API comes in. The waiter is the messenger – or API – that takes your request or order and tells the kitchen – the system – what to do. Then the waiter delivers the response back to you; in this case, it is the food.

Advantages of APIs ***imp***

- **Efficiency:** API produces efficient, quicker and more reliable results than the outputs produced by human beings in an organization.
- **Flexible delivery of services:** API provides fast and flexible delivery of services according to developer's requirements.
- **Integration:** The best feature of API is that it allows movement of data between various sites and thus enhances integrated user experience.
- **Automation:** As API makes use of robotic computers rather than humans, it produces better and automated results.
- **New functionality**: While using API the developers find new tools and functionality for API exchanges.

Disadvantages of APIs ***imp***

- **Cost:** Developing and implementing API is costly at times and requires high maintenance and support from developers.
- **Security issues:** Using API adds another layer of surface which is then prone to attacks, and hence the security risk problem is common in API's.

11.1 Server side database drivers and tools: JDBC, ODBC and SQL

ODBC

- ❖ Open Database Connectivity (ODBC)
- ❖ It is an open standard API (Application Programming Interface) for accessing a database. In 1992, Microsoft partners with Simba to build the world's first ODBC driver; SIMBA.DLL
- ❖ Using ODBC statements in a program, we can access files in a number of different common databases.
- ❖ ODBC is a specification for a database API. This API is independent of any one DBMS or operating system
- ❖ ODBC API is language-independent.
- ❖ ODBC is designed to expose database capabilities, not supplement them.

What is an ODBC Driver?

- *An ODBC driver uses the Open Database Connectivity (ODBC) interface by Microsoft that allows applications to access data in database management systems (DBMS) using SQL as a standard for accessing the data.*

- ODBC permits maximum interoperability, which means a single application can access different DBMS. Application end users can then add ODBC database drivers to link the application to their choice of DBMS.

The ODBC solution for accessing data led to ODBC database drivers, which are dynamic-link libraries on Windows and shared objects on Linux/UNIX. These drivers allow an application to gain access to one or more data sources. ODBC provides a standard interface to allow application developers and vendors of database drivers to exchange data between applications and data sources.

JDBC

- Java Database Connectivity
- One of the server side database driver tools for java application
- Application programming interface (API) for the programming language Java, which defines how a client may access any kind of tabular data, especially relational database.

  OR

  DBC is a standard API specification developed in order to move data from frontend to backend.
- It acts as a middle layer interface between java applications and database.
- JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database
- It's an advancement for ODBC (Open Database Connectivity).
- JDBC provides the same capabilities as ODBC
- The JDBC classes are contained in the Java Package **java.sql** and **javax.sql**. JDBC helps you to write Java applications that manage these three programming activities:
  - Connect to a data source, like a database.
  - Send queries and update statements to the database
  - Retrieve and process the results received from the database in answer to your query

Use of JDBC

- JDBC helps to write Java applications that manage these three programming activities:
  - Connect to a data source, like a database.

- Execute / Send queries and update statements to the database
- Retrieve and process the results received from the database in answer to executed query

❖ JDBC API is Standard API. We can communicate with any Database without rewriting our Application i.e. it is Database Independent API.

❖ JDBC Drivers are developed in Java and hence JDBC Concept is applicable for any Platform. I.e. JDBC Is Platform Independent Technology.

❖ By using JDBC API, we can perform basic CRUD Operations very easily.

C ➔ Create (Insert)

R ➔ Retrieve (Select)

U ➔ Update (Update)

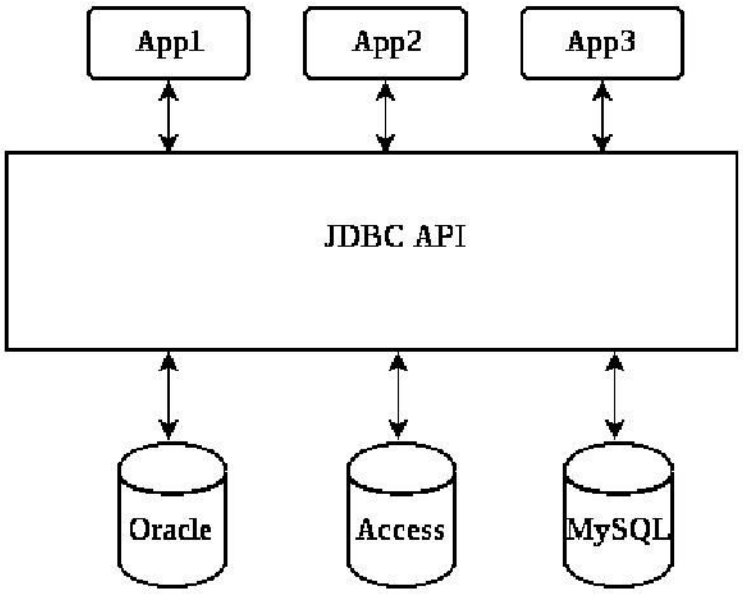
D ➔ Delete (Delete)

❖ We can also perform Complex Operations (like Inner Joins, Outer Joins, calling Stored Procedures etc.) very easily by using JDBC API.

❖ JDBC API supported by Large Number of Vendors and they developed multiple Products based on JDBC API

❖ Different Java executables such as Java applications, java applets, java servlets, and Java server Pages (JSPs) etc. able to use a JDBC driver to access a database.

JDBC STURUCTURE



JDBC Drivers *imp*

- JDBC drivers are client-side adapters (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand.
- There are **4 types** of JDBC drivers

1. JDBC-ODBC bridge driver / Type-1 driver
2. Native-API driver / Type-2 driver
3. Network Protocol driver / Type-3 driver
4. Thin driver / Type-4 driver

Type-1 driver

- Uses ODBC driver to connect to the database.
- It converts JDBC method calls into the ODBC function calls.
- It is also called Universal driver because it can be used to connect to any of the databases.
- As a common driver is used in order to interact with different databases, the data transferred through this driver is not so secured.
- This driver is needed to be installed in individual client machines.
- This driver isn't written in java, that's why it isn't a portable driver.
- ***The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.***

Type-2 driver

- It uses the client -side libraries of the database.
- This driver converts JDBC method calls into native calls of the database API.
- In order to interact with different database, this driver needs their local API, that's why data transfer is much more secure as compared to type-1 driver.
- Type-2 driver needs to be installed separately in individual client machines
- The Vendor client library needs to be installed on client machine.
- Type-2 driver isn't written in java, that's why it isn't a portable driver
- ***Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.***

Type-3 driver

- This driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. Here all the

database connectivity drivers are present in a single server, hence no need of individual client-side installation.

- Type-3 drivers are fully written in Java, hence they are portable drivers.
- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.
- Network support is required on client machine.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.
- ***If any Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.***

Type-4 driver

- It is also called native protocol driver.
- This driver interact directly with database.
- It does not require any native database library that is why it is also known as Thin Driver.
- It does not require any native library and Middleware server, so no client-side or server-side installation.
- It is fully written in Java language, hence they are portable drivers.
- ***If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is type-4.***

SQL

- Continued commitment of Microsoft towards interoperability, Microsoft provides a Java Database Connectivity (JDBC) driver for use with SQL Server, and Azure SQL Database.
- The driver is available at no additional charge and provides Java database connectivity from any Java application, application server, or Java-enabled applet.
- This driver is a Type 4 JDBC driver that provides database connectivity through the standard JDBC application program interfaces (APIs).
- The Microsoft JDBC Driver for SQL Server has been tested against major application servers such as IBM WebSphere, and SAP NetWeaver

Difference between ODBC and JDBC *v imp*

ODBC	JDBC
1. ODBC Stands for Open Database Connectivity.	JDBC Stands for java database connectivity.

2. Introduced by Microsoft in 1992.	Introduced by SUN Micro Systems in 1997.
3. We can choose ODBC only windows platform.	We can Use JDBC in any platform.
4. We can use ODBC for any language like C,C++,Java etc.	We can use JDBC only for Java languages.
5. Mostly ODBC Driver developed in native languages like C, C++.	JDBC Stands for java database connectivity.
6. For Java applications it is not recommended to use ODBC because performance will be down due to internal conversion and applications will become platform Dependent.	For Java application it is highly recommended to use JDBC because there we no performance & platform dependent problem.
7. ODBC is procedural.	JDBC is object oriented.

11.2 Creating Connection to Database

- It is almost impossible to find a java-compactible database without using a JDBC driver.
- During the development of java application there may use of any database like Oracle, MYSQL, MS-Access database etc.
- To communicate with these database developer need to connect that particular database using JDBC driver tool.

Steps to connect to database in Java

1. Import the package

- Include the packages containing the JDBC classes needed for database programming

2. Register Driver class

- Initialize a driver so you can open a communication channel with the database.
- For this, use ***forName ()*** method to register driver class. This method forName() will load database driver class dynamically

Syntax:

Class.forName ("string that specifies the type of database that you are going to use")

For example if you are going to use oracle database then you need to register the oracle driver class then

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

3. Create Connection Object

- When the driver of database is loaded in your program then you need to create the connection with the database
- To create connection you have to use getConnection () method.
- For this, you need to create the object of connection class and then you need to use DriverManager . getConnection ("path of the database", "username_of _database", "password_of_database")

Syntax:

Connection con = DriverManager . getConnection ("path of the database", "username_of _database", "password_of_database");

4. Create statement object

- Statement object are used to execute the query in database. For this you have to make an object of Statement class.

Syntax:

Statement stmnt = con . CreateStatement();

5. Execute the query

- When the statement is created then you have to execute that statement.
- For the execution of the query you have to create the object of result set (when the query is executed then query returns some date, such data are called result set). Then execute the query with the help of that statement created in step-3.

Syntax:

ResultSet rs= stmnt . executeQuery("Query")

Eg. ResultSet rs= stmnt . executeQuery("Select *From TableName");

5. Process Query and extract data from result set

- Use the appropriate *ResultSet.getXXX()* method to retrieve the data from the result set.

6. Close Connection or clean up the environment

- Requires explicitly closing all database resources versus relying on the JVM's garbage collection.
- If you perform the any job on database conccetion then you need to open that connection and closing of the database is necessary after completing the desired task.

Syntax:

Con.Colse();

Question- Write a step to connect database using JDBC and explain with example. ***V imp***

Answer:

Steps to connect database using JDBC are as follows.

1. Import packager
2. Register driver class / register JDBC driver
3. Create connection object / open connection
4. Execute query
5. Process query / extract data from result set
6. Close the connection / clean up the environment

Note: explain the step as per Question's Wage.

Example for JDBC Connection ***V imp***

Consideration for the following example are:

1. We are connecting MYSQL Database using JDBC driver
2. We have a database of name = "EECJDBC"
3. Username for DB = "EEC" Password for DB = "EEC"
4. We have one table having table name = "Everest" and 3 row and row 1 contains integer data row 2 contains string data and row 3 contains string data.

```
//Step-1 Import the packager.
import java.sql.*;

class MysqlCon
{
   public static void main(String args[])
   {
      try
      {
//Step-2 Register Driver Class, using forname() method.

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

//step-3 creating connection object. EECJDBC is database name, EEC is username
and password is also EEC & jdbc:mysql://localhost:8099/EECJDBC is path of DB.
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:8099/EECJDBC", "EEC",
"EEC");

//step-4 Createing statement object

Statement stmt = con.createStatement();

//step-5 executing query and creating the object of ResultSet.

ResultSet rs = stmt.executeQuery("select * from Everest");

// While loop is used for formatting Output

while (rs.next())
System.out.println(rs.getInt(1) + "  " + rs.getString(2) + "  " + rs.getString(3));

//Step-6 Closing the connection

con.close();

      }
      catch (Exception e) { System.out.println(e); }
   }
}
```

11.3 Statement / Statement interface

- After the successful connection with desired database application can interact with the respective database.
- Statement interface provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.
- Statement also defines methods that help bridge data type differences between Java and SQL data types used in a database.
- Statement interface resides in **java.sql** package and it is used to execute a static SQL statement and returning the result of the executed query.
- Statement interface is used to execute normal SQL queries. You can't pass the parameters to SQL query at run time using this interface.
- This interface is preferred over other two interfaces if you are executing a particular SQL query only once. The performance of this interface is also very less compared to other two interfaces.
- In most of time, Statement interface is used for DDL statements like **CREATE**, **ALTER**, **DROP** etc.
  For example:
  Statement stmt = con.createStatement(); // creating the statement object

- Statement interface has two sub-interfaces
  1. CallableStatement
  2. PreparedStatement.

Statement	PreparedStatement	CallableStatement
1. It is used to execute normal SQL queries.	**1**. It is used to execute parameterized or dynamic SQL queries.	**1**. It is used to call the stored procedures.
2. It is preferred when a particular SQL query is to be executed only once.	**2**. It is preferred when a particular query is to be executed multiple times.	**2.** It is preferred when the stored procedures are to be executed.
3. You cannot pass the parameters to SQL query using this interface.	**3**. You can pass the parameters to SQL query at run time using this interface.	**3**. You can pass 3 types of parameters using this interface. They are – IN, OUT and IN OUT.
4. This interface is mainly used for DDL statements like CREATE, ALTER, DROP etc.	**4**. It is used for any kind of SQL queries which are to be executed multiple times.	**4.** It is used to execute stored procedures and functions.

5. Performance of this interface is very low.	**5**. Performance of this interface is better than the Statement interface (when used for multiple execution of same query).	**5**. Performance of this interface is high.

Mostly used Statement interface

Mostly, we use execute methods of the java statement interface to execute queries.

1) ***public ResultSet executeQuery(String sql):***
   - ❖ Used to execute SELECT query. It returns the object of ResultSet.
2) ***public int executeUpdate(String sql):***
   - ❖ Used to execute specified query, it may be create, drop, insert, update, delete etc.
3) ***public boolean execute(String sql):***
   - ❖ Used to execute queries that may return multiple results.
4) ***public int[] executeBatch():***
   - ❖ Used to execute batch of commands.

Creating Statement

- ❖ In order to use a Java JDBC `Statement` you first need to create a `Statement`.
- ❖ Example: Statement stmnt = Connection.CreateStatement();
- ❖ After creating java statement object (stmnt in above example) you can execute a query against the database by calling executeQuery() method, by passing an SQL statement as parameter.
- ❖ Statement executeQuery() method returns a Java **JDBC** ResultSet which can be used to navigate the response of the query.

Example : calling java JDBC statement executeQuery() abd navigating the ResultSet

Note: (Table "**Everest**" is used in this example same as ResultSet below)

```
String sql = "select * from Everest";

ResultSet result = statement.executeQuery(sql);

while(result.next()) {
```

```
tring name = result.getString("Name");
   long   age  = result.getLong  ("Age");

}
```

- **Update of the database** can be execute via java JDBC Statement instance
- For any instance, execution of an SQL insert, update or delete via a statement instance
- **Example** of executing database via java JDBC instance is follows.

```
Statement stmnt = connection.createStatement();

String   sql      = "update Everest set name='Muku' where id=1";

int rowsAffected    = stmnt.executeUpdate(sql);

// The rowsAffected returned by the stmnt.executeUpdate(sql) call, tells how many records in the database were affected by the SQL statement.
```

- Once you are finished with a Statement instance you need to close it. You close a Statement instance by calling its close()
- Example: Statement.Close()

Result Set *imp*

- Java JDBC *ResultSet* interface represents the result of a database query.
- Text about queries shows how the result of a query is returned as a java.sql.ResultSet and this ResultSet is then iterated to inspect the result
- A JDBC ResultSet contains records. Each records contains a set of columns.
- Each record contains the same amount of columns, although not all columns may have a value. A column can have a null value.
- The SQL statements that read data from a database query, return the data in a result set.
- The SELECT statement is the standard way to select rows from a database and view them in a result set.

- A ResultSet object maintains a cursor that points to the current row in the result set.
- The term "result set" refers to the row and column data contained in a ResultSet object.
- ResultSet need to be close when you are done with it.

Type of the ResultSet

1. **ResultSet.TYPE_FORWARD_ONLY**
   - The cursor can only move forward in the result set.
2. **ResultSet.TYPE_SCROLL_INSENSITIVE**
   - The cursor can scroll forward and backward, and the result set is not sensitive to changes made by others to the database that occur after the result set was created.
3. **ResultSet.TYPE_SCROLL_SENSITIVE**.
   - The cursor can scroll forward and backward, and the result set is sensitive to changes made by others to the database that occur after the result set was created.

Note: if nothing is specified then default result set is **TYPE_FORWARD_ONLY**

Concurrency of ResultSet

A ResultSet can have one of two concurrency levels:

1. ResultSet.CONCUR_READ_ONLY→ means that the ResultSet can only be read.
2. ResultSet.CONCUR_UPDATABLE →means that the ResultSet can be both read and update

- **Example**: name of below table is "Everest"

Name	RollNo	Department
Ram	25	CMP
Geeta	23	IT
Hari	24	CIVIL

This ResultSet has 3 different columns (Name, RollNo, and Department) and 3 records with different values for each column.

- **Create a ResultSet** by executing a statement or prepared statement like this
  Statement stmnt= connection. createStatement();
  ResultSet rs= stmnt.executeQuery("select * from Everest");

  OR

```
String sql = "select * from Everest";
PreparedStatement stmnt = connection.prepareStatement(sql);

ResultSet rs = stmnt.executeQuery();
```

Data Source Object

V imp

- Data Source object is the representation of a data source in the Java programming language.
- In basic terms, a data source is a facility for storing data.
- It can be as sophisticated as a complex database for a large corporation or as simple as a file with rows and columns
- A Data Source object provides a new way for JDBC clients to obtain a DBMS connection.
- To create a DataSource object you define it with an entry in the weblogic.properties file. This DataSource entry then points to a connection pool that is also defined in the weblogic.properties file.
- To use the DataSource objects, import the following classes in your client code:

```
import java.sql.*;
import java.util.*;
import javax.naming.*;
```

- DataSource objects can be defined with or without Java Transaction Services (JTS) enabled.
- Creating a DataSource object with JTS enabled causes the connection to behave like a JTS connection, with support for transactions.

END