UNIT 14: INTRODUCTION TO JDBC

UNIT STRUCTURE

14.1 Learning Objectives

14.2 Introduction

14.3 JDBC

14.3.1 Basic Steps to JDBC

14.4 JDBC API

14.5 JDBC Drivers

14.6 Connection Management

14.7 JDBC Design Considerations

14.8 Two Tier and Three Tier Client Server Model

14.9 ResultSet

14.10 Prepared Statement and Callable Statement

14.10.1 Prepared Statement

14.10.2 Callable Statement

14.11 ResultSet Meta Data Object

14.12 Let Us Sum Up

14.13 Further Reading

14.14 Answer to Check Your Progress

14.15 Model Questions

14.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- describe JDBC
- explain the basic steps in JDBC
- describe JDBC API and JDBC drivers
- describe the steps for connections in JDBC
- describe the design of JDBC
- explain the structure of JDBC
- describe Two-tier and Three-tier architecture
- define ResultSet
- describe ResultSetMetaData,

- describe PreparedStatement Interface and CallableStatement Interface

14.2 INTRODUCTION

In the earlier unit, we had been introduced to AWT and Swings. Topics like AWT basics, AWT components, event handling in AWT, and application and Menus were discussed in the previous unit. In addition to these, concepts related to Swings, swing components, event handling in Swings, etc. were also described in the previous unit.

In this unit, you will be introduced to JDBC. Topics like JDBC API and JDBC drivers along with the design and structure of JDBC. In addition to these, concepts related to two-tier and three-tier architecture and ResultSet are also described in this unit. In the next unit, we will explore topics related to database programming using Java.

14.3 JAVA DATABASE CONNECTIVITY (JDBC)

JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases. It gives the advantage of connecting the database to java API. All the classes of JDBC The JDBC classes are contained in java.sql and javax.sql packages. JDBC has its API(JDBC API) that uses JDBC drivers. To create the connectivity JDBC follows some steps from loading and registering the driver to closing the connection. JDBC uses four different drivers to connect different types of the data source. It Follows the two-tier and three-tier architecture. A resultset is the return data of the SQL statements that read data from a database query. Users can navigate and update rows and columns in the resultset. JDBC can prepare PreparedStatement and CallableStatement.

14.3.1 Basic Steps to JDBC

In the Java programming language, Java Database Connectivity (JDBC) is an application interface (API) that defines how a client

may access any kind of tabular data, especially a relational database. It gives the service of a middle-layer interface between java applications and databases. The classes and interface of JDBC handle the request made by users to the specified database.

The JDBC classes are contained in java.sql and javax.sql packages.

Why Use JDBC?

Initially, ODBC API was the database API to connect and execute the query with the database. But, the ODBC driver of ODBC API is written in C language. So it is platform dependant and unsecured to overcome this limitation JDBC is introduced against ODBC. It has its API(JDBC API) that uses JDBC drivers (in Java language). JDBC uses its java Native calls.

Fundamental Steps

The fundamental steps involved in the process of connecting to a database and executing a query consist of the following:

- Import JDBC packages.
- Load and register the JDBC driver.
- Open a connection to the database.
- Create a statement object to perform a query.
- Execute the statement object and return a query resultset.
- Process the resultset.
- Close the resultset and statement objects.
- Close the connection.

14.4 JDBC API

The Java Database Connectivity (JDBC) API provides universal data access from the Java programming language. Using the JDBC API, users can virtually access any data source, from relational databases to flat file spreadsheets. JDBC technology provides a common base on which tools and alternate interfaces can be built.

The JDBC API is comprised of two packages:

- java.sql

- javax.sql

To use the JDBC API with a particular database management system, to mediate between JDBC technology and the database, need a JDBC technology-based driver. A driver might be written purely in the Java programming language or a mixture of the Java programming language and Java Native Interface (JNI) native methods. To obtain a JDBC driver for a particular database management system

14.5 JDBC DRIVERS

Client-side adaptors are called JDBC drivers that convert requests from Java programs to a protocol that the DBMS can understand. There are 4 types of JDBC drivers:

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

JDBC-ODBC bridge driver

Type-1 driver or JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver makes the ODBC function calls from JDBC method calls. It is also called a Universal driver because it can be used to connect to any of the databases.

- As a common driver is used to interact with different databases, the data transferred through this driver is not so secure.
- The ODBC bridge driver is needed to be installed in individual client machines.
- The type-1 driver isn't portable.
- This driver software is built-in with JDK so no need to install it separately.
- It is a database-independent driver.

Native -API driver

The Native API driver uses the client-side libraries of the database. This driver form native calls of the database API from JDBC method calls. With the help of local API, this driver interacts with different databases, so

data transfer is more secure than the type-1 driver.

- The driver needs to be installed separately in individual client machines
- The Vendor client library needs to be installed on the client machine.
- The type-2 driver isn't written in java, that's why it isn't a portable driver
- It is a database-dependent driver.

Network Protocol driver

The Network Protocol driver uses middleware (application server). And this middleware converts JDBC calls directly or indirectly into the vendor-specific database protocol. All the database connectivity drivers are present in a single server so no need for individual client-side installation.

- Type-3 drivers are fully written in Java, hence they are portable drivers.
- No client-side library is required because the application server can perform many tasks like auditing, load balancing, logging, etc.
- Network support is required on the client machine.
- Maintenance of the Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.
- Switch facility to switch over from one database to another database.

Thin driver

The type-4 driver is also called the native protocol driver. This driver interacts directly with the database. It does not depend on any native database library, so it is also known as Thin Driver.

- 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.

Which Driver to use when?

- If you are accessing one type of database, then preferred 4 type driver.
- If you are accessing multiple types of databases at the same time, type 3 is the preferred driver.
- Type 3 or type 4 driver is not available for your database, then type 2 drivers are useful.
- The type 1 driver is used for development and testing purposes only.

14.6 CONNECTION MANAGEMENT

Java Database Connectivity is done with the following five steps:

- Register for the Driver class
- Create connection
- Create statement
- Execute queries
- Close connection

1. Register for the driver class

The forName() method of Class is used to register the driver class. And it's used to dynamically load the driver class.

Syntax of forName() method:

```
public static void forName(String className)throws
ClassNotFoundException
```

2. Create the connection object

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

Syntax of getConnection() method:

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

3. Create the Statement object

The createStatement() method of the Connection interface is used to create a statement. This statement is responsible to execute queries with the database.

Syntax of createStatement() method:

```
public Statement createStatement()throws SQLException
```

4. Execute the query

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

Syntax of executeQuery() method:

```
public ResultSet executeQuery(String sql)throws
SQLException
```

5. Close the connection object

Closing connection object statement automatically closed the ResultSet. The close() method of Connection interface is used to close the connection.

Syntax of close() method:

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



CHECK YOUR PROGRESS

Q 1. What are the packages related to JDBC classes?

Q 2. What are the steps related to JDBC process?

Q 3. How many drivers are available in JDBC?

Q 4. What is the use of JDBC API?

Q 5. How can you manage JDBC connection?

14.7 JDBC DESIGN CONSIDERATIONS

JDBC and database drivers work to access spreadsheets and databases. The design of JDBC defines the components of JDBC, which are used for connecting to the database.

Components of JDBC

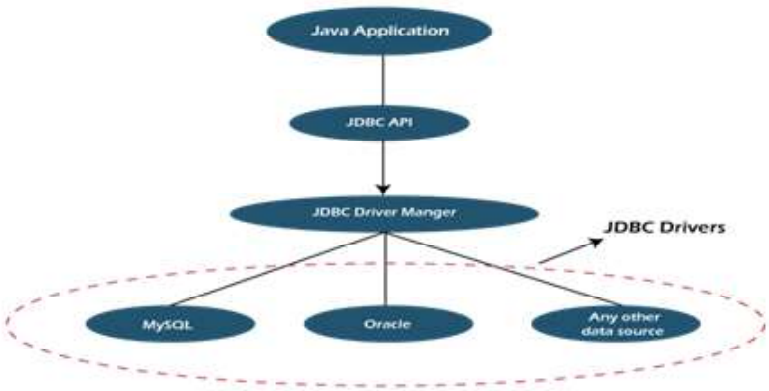


Figure 14.1: Component of JDBC

JDBC has four major components that are used for the interaction with the database.

1. JDBC API
2. JDBC Test Suite
3. JDBC Driver Manager
4. JDBC ODBC Bridge Driver

1) JDBC API: JDBC API provides various interfaces and methods to establish easy connections with different databases.

- javax.sql.*;
- java.sql.*;

2) JDBC Test suite: JDBC Test suite facilitates the programmer to check the various operations such as deletion, updating, and insertion which are executed by the JDBC Drivers.

3) JDBC Driver manager: JDBC Driver manager loads the database-specific driver into an application to establish the connection with the database. The JDBC Driver manager processes the request by converting the database-specific call to the database.

4) JDBC-ODBC Bridge Drivers: JDBC-ODBC Bridge Drivers establish the connection between the database drivers and the database. The bridge translates the JDBC method calls into the ODBC method call. It makes the usage of the native library to access the ODBC (Open Database Connectivity) characteristics.

14.8 TWO-TIER AND THREE-TIER CLIENT-SERVER MODEL

The JDBC architecture consists of two-tier and three-tier processing modes to access a database. They are as follows:

1. **Two-tier model:** In this architecture, a java application communicates directly with the data source. The JDBC driver establishes the communication between the application and the data source. The driver sent the answers for user queries back the result to the user after getting a query from the user to the data source.

The user can connect data sources that can be located on a different machine on a network. This is known as a **client/server configuration,** where the user's machine treat as a client, and the machine has the data source acting as the server.

2. **Three-tier model:** In this, the middleware services accept the user's queries from which the commands are again sent to the data source. The results are sent back to the middle tier, and the middle tier sends them to the user.

This type of model is found very useful by management information system directors.

14.9 RESULTSET

A ResultSet is the return data of the SQL statements that read data from a database query. The SELECT statement is the standard approach to select rows from a database and view them in a result set. The java.sql.ResultSet interface represents the result set of a database query.

A ResultSet object maintains a cursor to point to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object.

The methods of the ResultSet interface consist of three categories?

- Navigational methods ? Used to move the cursor around.
- Get methods ? Used to view the data in the columns of the current row being pointed by the cursor.
- Update methods ? Used to update the data in the columns of the current row from the underlying database.

To create statements with desired ResultSet, JDBC provides the following connection methods ?

- createStatement(int RSType, int RSConcurrency);
- prepareStatement(String SQL, int RSType, int RSConcurrency);
- prepare all(String sql, int RSType, int RSConcurrency); where the first argument shows the type of a ResultSet object and the second argument is one of two ResultSet constants for specifying whether a result set is

read-only or updatable.

Type of ResultSet:

The possible ResultSet Types are as given below:

Table 14.1: Types of ResultSet

Type	**Description**
ResultSet.TYPE_FORWARD_ONLY	The cursor can only move forward in the result set.
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.
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.

Important point: If you do not specify any ResultSet type, you will automatically get one that is TYPE_FORWARD_ONLY.

Viewing a Result Set:

The ResultSet interface contains methods for getting the data of the current row.

There is a get method for each of the possible data types with two versions:

- One that takes in a column name.
- One that takes in a column index.

For example, to view the column content which contains an int, need to use one of the getInt() methods of ResultSet -

```
public int getInt(String columnName) throws SQLException
returns the int in the current row in the column named columnName
```

Or

```
public int getInt(int columnIndex) throws SQLException
```

which returns the int in the current row in the specified column index. The column index starts at 1, meaning the first column of a row is 1, the second column of a row is 2, and so on.

Updating a Result Set:

For updating the data of a result set, the ResultSet interface contains a collection of update methods. There are two update methods for each data type ?

- One that takes in a column name.
- One that takes in a column index.

For example, to update a String column of the current row of a result set, you can use one of the following updateString() methods -

```
public void updateString(int columnIndex, String s) throws SQLException
```

which changes the String in the specified column to the value of s.

or

```
public void updateString(String columnName, String s) throws SQLException
```

Similar to the previous method, except that the column is specified by its name instead of its index.

Updating a row in the result set changes the columns of the current row in the ResultSet object, but not in the underlying database. To update your changes to the row in the database, you need to invoke one of the following methods.

Table 14.2: Different options for updating the row of ResultSet

S.N.	Methods & Description
1.**public void updateRow()**	Updates the current row by updating the corresponding row in the database.
2.**public void deleteRow()**	Deletes the current row from the database

3.**public void refreshRow()**	Refreshes the data in the result set to reflect any recent changes in the database.
4.**public void cancelRowUpdates()**	Cancels any updates made on the current row.
5.**public void insertRow()**	Inserts a row into the database. This method can only be invoked when the cursor is pointing to the insert row.

Concurrency of ResultSet:

The possible ResultSetConcurrency are given below.

Table 14.3: Different options for Concurrency of ResultSet

Concurrency	**Description**
ResultSet.CONCUR_READ_ONLY	Creates a read-only result set. This is the default. It is the default setting for concurrency
ResultSet.CONCUR_UPDATABLE	Creates an updateable result set.

Navigating a Result Set:

There are some methods to moving the cursor in the ResultSet are given below:

Table 14.4: Different options for Navigating a ResultSet

S.N.	**Methods & Description**
1	**public void beforeFirst() throws SQLException** Moves the cursor just before the first row.
2	**public void afterLast() throws SQLException** Moves the cursor just after the last row.
3	**public boolean first() throws SQLException** Moves the cursor to the first row.
4	**public void last() throws SQLException** Moves the cursor to the last row.
5	**public boolean absolute(int row) throws SQLException** Moves the cursor to the specified row.

6	**public boolean relative(int row) throws SQLException** Moves the cursor the given number of rows forward or backward, from where it is currently pointing.
7	**public boolean previous() throws SQLException** Moves the cursor to the previous row. This method returns false if the previous row is off the result set.
8	**public boolean next() throws SQLException** Moves the cursor to the next row. This method returns false if there are no more rows in the result set.
9	**public int getRow() throws SQLException** Returns the row number that the cursor is pointing to.
10	**public void moveToInsertRow() throws SQLException** Moves the cursor to a special row in the result set that can be used to insert a new row into the database.
11	**public void moveToCurrentRow() throws SQLException** Moves the cursor back to the current row if the cursor is currently at the insert row; otherwise, this method does nothing

14.10 PREPARED STATEMENT AND CALLABLE STATEMENT

14.10.1 Prepared Statement

To execute the precompiled SQL statement for more than one time, it is used statements extending from the PreparedStatement interface. This accepts parameterized SQL quires but user can pass 0 or more parameters to this query or we can pass '?' in initial stage. String sql="insert into emp values(?,?,?)"; in this example the argument '?' set by calling setter method.

Creating a PreparedStatement:

The prepareStatement() method of Connection interface is used to return the object of PreparedStatement.

Syntax for the PrepareStatement():

```
public PreparedStatement prepareStatement(String query)throws SQLException{}
```

Methods of PreparedStatement interface

The important methods of PreparedStatement interface are given below:

Table 14.5: Different methods from PreparedStatement Interface

Method	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

Executing the Prepared Statement:

After execution of prepareStatement() you can execute it using one of the execute() methods of the PreparedStatement interface

- **execute():** This method executes normal static SQL statements in the current prepared statement object and returns a boolean value.
- **executeQuery():** This method executes the current prepared statement and returns a ResultSet object.
- **executeUpdate():** This method executes SQL DML statements such as insert update or delete in the current Prepared statement. It returns

an integer value representing the number of rows affected.

Why use PreparedStatement?

The performance of the application will be improve after using PreparedStatement interface because query is compiled only once.

14.10.2 Callable Statement

The CallableStatement interface provides methods to execute the stored procedures. you can call stored procedures of all RDBMS in single standard way as the JDBC API provides a stored procedure SQL escape syntax. CallableStatement interface is used to call the stored procedures and functions. It gives better performance because these are precompiled.

Creating a CallableStatement:

To create an object of the CallableStatement (interface) we can use the prepareCall() method of the Connection interface. This method accepts a string variable representing a query to call the stored procedure and returns a CallableStatement object.

```
CallableStatement cstmt=
con.prepareCall("{callmyProcedure(?, ?, ?)}");
```

Execution of Callable Statement

The CallableStatement object you can execute it using following execute() method.

```
cstmt. execute();
```

28llo Java Program for Beginners

14.11 RESULTSET META DATA OBJECT

The metadata means data about data If you have to get metadata of a table like total number of column, column name, column type etc., ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.

Commonly used methods of ResultSetMetaData interface

Table 14.6: Different methods from ResultSetMetaData Interface

Method	Description
public int getColumnCount()throws SQLException	it returns the total number of columns in the ResultSet object.
public String getColumnName(int index)throws SQLException	it returns the column name of the specified column index.
public String getColumnTypeName(int index)throws SQLException	it returns the column type name for the specified index.
public String getTableName(int index)throws SQLException	it returns the table name for the specified column index.



CHECK YOUR PROGRESS

Q 6. What is the function of JDBC Driver Manager?

Q 7. What is the specialty of Three-tier architecture?

Q 8. What is ResultSet?

Q9. Write two differences between Statement and PreparedStatement interface?



14.12 LET US SUM UP

- ODBC abstraction is defined as C native functions. Therefore ODBC calls are C function calls.
- JDBC generally referred as Java Database Connectivity is one of the service API of Java platform.
- JDBC defines a standard protocol for Java Application to access tabular data stores.
- JDBC API needs to be supported by JDBC Driver to communicate wih the respective database.
- JDBC driver is a set of classes that implement the JDBC specification.
- JDBC drivers are responsible to drive the queries from java application to DBMS.

- There are 4 different drivers are related to JDBC. All have different responsibility.
- JDBC API is mainly divided into two packages jav.sql and javax.sql
- The statement object is used to execute the normal SQL statement without IN and OUT parameters.
- The resultset object is used to represent data the DB Cursor to java application, using JDBC to read the database.
- The PreparedStatement object is used to execute SQL statement with IN parameter. This supports the positional parameter, represent using '?'
- The callable statement object is used to execute Db stored procedure that supports IN and OUT parameters.
- The DatabaseMetaData object is used to get the database and JDBC driver details.



14.13 FURTHER READING

1) Seth, A. & Juneja, B. L. (2017), Java One Step Ahead(1st), Oxford University Press
2) Kumar, S. K.et al (2014) JDBC, Servlet, and JSP includes JSF and Design Patterns Black Book, DreamTech Press

14.14 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1. Java.sql and javax.sql.

Ans to Q No 2. Import JDBC packages, Load and register the JDBC driver, Open a connection to the database, Create a statement object to perform a query, Execute the statement object and return a query resultset, Process the resultset, Close the resultset and statement objects, Close the connection.

Ans to Q No 3. There are four different drive are related to JDBC.

Ans to Q No 4. The JDBC API can virtually access any data source, from relational databases to flat file spreadsheets.

Ans to Q No 5. Register the Driver class, Create connection, Create statement, Execute queries, Close connection.

Ans to Q No 6. JDBC Driver manager loads the database-specific driver into an application in order to establish the connection with the database. It process the user request.

Ans to Q No 7. In three-tier architecture, middleware services accept the user's queries from which the commands are again sent to the data source.

Ans to Q No 8. A ResultSet is the return data of the SQL statements that read data from a database query. The SELECT statement is the standard approach to select rows from a database and view them in a result set.

Ans to Q No 9. Statement used when SQL query is execute only once but Prepared statement executed when SQL query execute for multiple time.

In statement cannot parameter at runtime but it is possible in case of Prepared statement.



14.15 MODEL QUESTIONS

Q 1. What is JDBC Driver?

Q 2. What are the steps to connect to the database in java?

Q 3. What are the JDBC API components?

Q 4. What are the JDBC statements?

Q 5. What are the difference between PreparedStatement and CallableStatement?

Q 6. What is ResultSet?

Q 7. Describe is design structure of JDBC.

Q 8. What are the differences between execute, executeQuery, and executeUpdate?

Q 9. What is the difference between stored procedures and functions?

Q 10. Describe the design components of JDBC.

Q 11. Describe the two-tier and three-tier architecture of JDBC.

*** ***** ***