2016

JDBC MATERIAL

1. What is the JDBC?

Java Database Connectivity (**JDBC**) is a standard Java API to interact with relational databases from Java. **JDBC** has set of classes and interfaces which can use from Java application and talk to database without learning RDBMS details and using Database Specific JDBC Drivers.

2. What are the new features added to JDBC 4.0?

The major features added in JDBC 4.0 include:

a) Auto-loading of JDBC driver class
b) Connection management enhancements
c) Support for RowId SQL type
d) DataSet implementation of SQL using Annotations
e) SQL exception handling enhancements
f) SQL XML support

3. Explain Basic Steps in writing a Java program using JDBC?

JDBC makes the interaction with RDBMS simple and intuitive. When a Java application needs to access database:

a) Load the RDBMS specific JDBC driver because this driver actually communicates with the database (Incase of JDBC 4.0 this is automatically loaded).
b) Open the connection to database which is then used to send SQL statements and get results back.
c) Create JDBC Statement object. This object contains SQL query.
d) Execute statement which returns resultset(s). ResultSet contains the tuples of database table as a result of SQL query.
e) Process the result set.
f) Close the connection.

4. Explain the JDBC Architecture.

The JDBC Architecture consists of two layers:

a) The JDBC API, which provides the **application-to-JDBC Manager** connection.
b) The JDBC Driver API, which supports the **JDBC Manager-to-Driver** Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases. The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases. The location of the driver manager with respect to the JDBC drivers and the Java application is shown in Figure 1.

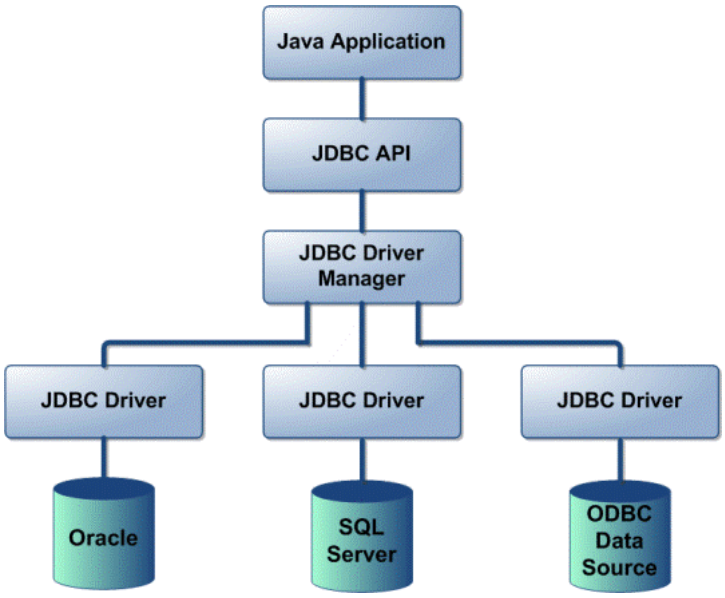

Figure 1: JDBC Architecture

5. What are the main components of JDBC?

The life cycle of a JDBC consists of the following phases:

a) **DriverManager**: Manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain sub protocol under JDBC will be used to establish a database Connection.

b) **Driver**: The database communications link, handling all communication with the database. Normally, once the driver is loaded, the developer need not call it explicitly.

c) **Connection:** Interface with all methods for contacting a database. The connection object represents communication context, i.e. all communication with database is through connection object only.

d) **Statement**: Encapsulates an SQL statement which is passed to the database to be parsed, compiled, planned and executed.

e) **ResultSet**: The ResultSet represents set of rows retrieved due to query execution.

6. How the JDBC application works?

A JDBC application can be logically divided into two layers:

a) Driver layer

b) Application layer

I. Driver layer consists of DriverManager class and the available JDBC drivers.

II. The application begins with requesting the DriverManager for the connection.

III. An appropriate driver is chosen and is used for establishing the connection. This connection is given to the application which falls under the application layer.

IV. The application uses this connection to create Statement kind of objects, through which SQL commands are sent to backend and obtain the results.

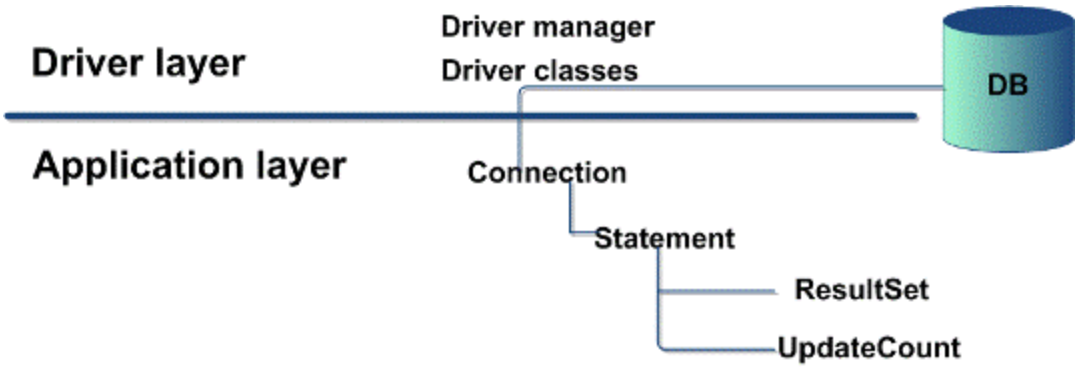

Figure 2: JDBC Application

7. How do I load a database driver with JDBC 4.0 / Java 6?
Provided the JAR file containing the driver is properly configured, just place the JAR file in the classpath. Java developers no longer need to explicitly load JDBC drivers using code like Class.forName () to register a JDBC driver. The DriverManager class takes care of this by automatically locating a suitable driver when the DriverManager.getConnection () method is called. This feature is backward-compatible, so no changes are needed to the existing JDBC code.

8. What is JDBC Driver interface?
The JDBC Driver interface provides vendor-specific implementations of the abstract classes provided by the JDBC API. Each vendor driver must provide implementations of the java.sql.Connection, Statement, PreparedStatement, CallableStatement, ResultSet and Driver.

9. What does the connection object represents?
The connection object represents communication context, i.e., all communication with database is through connection object only.

10. What is Statement?
Statement acts like a vehicle through which SQL commands can be sent through the connection object we create statement kind of objects.

```
Statement stmt = conn.createStatement();
```

This method returns object which implements statement interface.

11. What is PreparedStatement?
A prepared statement is an SQL statement that is precompiled by the database. Through precompilation, prepared statements improve the performance of SQL commands that are executed multiple times. Once compiled, prepared statements can be customized prior to each execution by altering predefined SQL parameters.

```
PreparedStatement pstmt = conn.prepareStatement ("UPDATE EMPLOYEES SET SALARY = ? WHERE ID = ?");
                          pstmt.setBigDecimal (1, 153833.00);
                          pstmt.setInt (2, 110592);
```

*Here: **conn** is an instance of the Connection class and "**?**" represents parameters. These parameters must be specified before execution.*

12. What is the difference between a Statement and a PreparedStatement?

Statement	PreparedStatement
A standard Statement is used to create a Java representation of a literal SQL statement and execute it on the database.	A PreparedStatement is a precompiled statement. This means that when the PreparedStatement is executed, the RDBMS can just run the PreparedStatement SQL statement without having to compile it first.
Statement has to verify its metadata against the database every time.	While a prepared statement has to verify its metadata against the database only once.
If you want to execute the SQL statement once go for STATEMENT	If you want to execute a single SQL statement multiple number of times, then go for PREPAREDSTATEMENT. PreparedStatement objects can be reused with passing different values to the queries

13. What are callable statements?
Callable statements are used from JDBC application to invoke stored procedures and functions.

14. How to call a stored procedure from JDBC?
PL/SQL stored procedures are called from within JDBC programs by means of the prepareCall () method of the Connection object created. A call to this method takes variable bind parameters as input parameters as well as output variables and creates an object instance of the CallableStatement class.

The following line of code illustrates this:

```
CallableStatement stproc_stmt = conn.prepareCall("{call procname(?,?,?)}");
```

Here conn is an instance of the Connection class.

15. What are types of JDBC drivers?
There are four types of drivers defined by JDBC as follows:

a) **Type 1: JDBC/ODBC**— These require an ODBC (Open Database Connectivity) driver for the database to be installed. This type of driver works by translating the submitted queries into equivalent ODBC queries and forwards them via native API calls directly to the ODBC driver. It provides no host redirection capability.
b) **Type2: Native API (partly-Java driver)** — This type of driver uses a vendor-specific driver or database API to interact with the database. An example of such an API is Oracle OCI (Oracle Call Interface). It also provides no host redirection.
c) **Type 3: Open Protocol-Net**— This is not vendor specific and works by forwarding database requests to a remote database source using a net server component. How the net server component accesses the database is transparent to the client. The client driver communicates with the net server using a database-independent protocol and the net server translates this protocol into database calls. This type of driver can access any database.
d) **Type 4: Proprietary Protocol-Net(pure Java driver)**—This has a same configuration as a type 3 driver but uses a wire protocol specific to a particular vendor and hence can access only that vendor's database. Again this is all transparent to the client.

Note: *Type 4 JDBC driver is most preferred kind of approach in JDBC.*

16. Which type of JDBC driver is the fastest one?

JDBC Net pure Java driver (Type IV) is the fastest driver because it converts the JDBC calls into vendor specific protocol calls and it directly interacts with the database.

17. Does the JDBC-ODBC Bridge support multiple concurrent open statements per connection?
No. You can open only one Statement object per connection when you are using the JDBC-ODBC Bridge.

18. Which is the right type of driver to use and when?

a) Type I driver is handy for prototyping
b) Type III driver adds security, caching, and connection control
c) Type III and Type IV drivers need no pre-installation

19. What are the standard isolation levels defined by JDBC?
The values are defined in the class java.sql.Connection and are:

a) TRANSACTION_NONE
b) TRANSACTION_READ_COMMITTED
c) TRANSACTION_READ_UNCOMMITTED
d) TRANSACTION_REPEATABLE_READ
e) TRANSACTION_SERIALIZABLE

Any given database may not support all of these levels.

20. What is resultset?
The ResultSet represents set of rows retrieved due to query execution.

```
ResultSet rs = stmt.executeQuery(sqlQuery);
```

21. What are the types of resultsets?
The values are defined in the class java.sql.Connection and are:

a) **TYPE_FORWARD_ONLY** specifies that a resultset is not scrollable, that is, rows within it can be advanced only in the forward direction.
b) **TYPE_SCROLL_INSENSITIVE** specifies that a resultset is scrollable in either direction but is insensitive to changes committed by other transactions or other statements in the same transaction.
c) **TYPE_SCROLL_SENSITIVE** specifies that a resultset is scrollable in either direction and is affected by changes committed by other transactions or statements within the same transaction.

***Note**: A TYPE_FORWARD_ONLY resultset is always insensitive.*

22. What's the difference between TYPE_SCROLL_INSENSITIVE and TYPE_SCROLL_SENSITIVE?

TYPE_SCROLL_INSENSITIVE	TYPE_SCROLL_SENSITIVE
An insensitive resultset is like the snapshot of the data in the database when query was executed.	A sensitive resultset does NOT represent a snapshot of data, rather it contains points to those rows which satisfy the query condition.
After we get the resultset the changes made to data are not visible through the resultset, and hence they are known as insensitive.	After we obtain the resultset if the data is modified then such modifications are visible through resultset.
Performance not effected with insensitive.	Since a trip is made for every '**get'** operation, the performance drastically get affected.

22. What is rowset?
A RowSet is an object that encapsulates a set of rows from either Java Database Connectivity (JDBC) result sets or tabular data sources like a file or spreadsheet. RowSets support component-based development models like JavaBeans, with a standard set of properties and an event notification mechanism.

24. What are the different types of RowSet?
There are two types of RowSet are there. They are:

a) **Connected** - A connected RowSet object connects to the database once and remains connected until the application terminates.
b) **Disconnected -** A disconnected RowSet object connects to the database, executes a query to retrieve the data from the database and then closes the connection. A program may change the data in a disconnected RowSet while it is disconnected. Modified data can be updated in the database after a disconnected RowSet reestablishes the connection with the database.

25. What is the need of BatchUpdates?
The BatchUpdates feature allows us to group SQL statements together and send to database server in one single trip.

26. What is a DataSource?
A DataSource object is the representation of a data source in the Java programming language. In basic terms,

a) A DataSource is a facility for storing data.
b) DataSource can be referenced by JNDI.
c) Data Source may point to RDBMS, file System, any DBMS etc.

27. What are the advantages of DataSource?
The few advantages of data source are:

a) An application does not need to hardcode driver information, as it does with the DriverManager.
b) The DataSource implementations can easily change the properties of data sources. *For example*: There is no need to modify the application code when making changes to the database details.
c) The DataSource facility allows developers to implement a DataSource class to take advantage of features like connection pooling and distributed transactions.

28. What is connection pooling? What is the main advantage of using connection pooling?
A connection pool is a mechanism to reuse connections created. Connection pooling can increase performance dramatically by reusing connections rather than creating a new physical connection each time a connection is requested.

29. Explain differences among java.util.Date, java.sql.Date, java.sql.Time, and java.sql.Timestamp?
As shown below all the sql Date classes extend the util Date class.

a) java.util.Date - class supports both the Date (i.e. year/month/date etc) and the Time (hour, minute, second, and millisecond) components.
b) java.sql.Date - class supports only the Date (i.e. year/month/date etc) component. The hours, minutes, seconds and milliseconds of the Time component will be set to zero in the particular time zone with which the instance is associated.
c) java.sql.Time - class supports only Time (i.e. hour, minute, second, and millisecond) component. The date components should be set to the "zero epoch" value of January 1, 1970 and should not be accessed.
d) java.sql.TimeStamp : class supports both Date (i.e. year/month/date etc) and the Time (hour, minute, second, millisecond and nanosecond) components.

30. Difference between DataSource vs DriverManager
A DataSource 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 can reside on a remote server, or it can be on a local desktop machine. Applications access a data source using a connection, and a DataSource object can be thought of as a factory for connections to the particular data source that the DataSource instance represents. The DataSource interface provides two methods for establishing a connection with a data source.

Using a DataSource object is the preferred alternative to using the DriverManager for establishing a connection to a data source. They are similar to the extent that the DriverManager class and DataSource interface both have methods for creating a connection, methods for getting and setting a timeout limit for making a connection, and methods for getting and setting a stream for logging.

Their differences are more significant than their similarities, however. Unlike the DriverManager, a DataSource object has properties that identify and describe the data source it represents. Also, DataSource object works with a Java Naming and Directory Interface (JNDI) naming service and is created, deployed, and managed separately from the applications that use it. A driver vendor will provide a class that is a basic implementation of the DataSource interface as part of its JDBC 2.0 or 3.0 driver products. What a system administrator does to register a DataSource object with a JNDI naming service and what an application does to get a connection to a data source using a DataSource object registered with a JNDI naming service are described later in this chapter.

Being registered with a JNDI naming service gives a DataSource object two major advantages over the DriverManager. First, an application does not need to hardcode driver information, as it does with the DriverManager. A programmer can choose a logical name for the data source and register the logical name with a JNDI naming service. The application uses the logical name, and the JNDI naming service will supply the DataSource object associated with the logical name. The DataSource object can then be used to create a connection to the data source it represents.

The second major advantage is that the DataSource facility allows developers to implement a DataSource class to take advantage of features like connection pooling and distributed transactions. Connection pooling can increase performance dramatically by reusing connections rather than creating a new physical connection each time a connection is requested. The ability to use distributed transactions enables an application to do the heavy duty database work of large enterprises.

Although an application may use either the DriverManager or a DataSource object to get a connection, using a DataSource object offers significant advantages and is the recommended way to establish a connection.