UNIT: 1 THE J2EE PLATFORM, JDBC (JAVA DATABASE CONNECTIVITY)

CS-25 Advanced Java Programming (J2EE)

- INTRODUCTION TO J2EE
- ENTERPRISE ARCHITECTURE & STYLE (2 TIRE, 3 TIRE, N-TIRE)
- THE J2EE PLATFORM
- INTRODUCTION TO J2EE APIS (SERVLET, JSP, EJB, JMS, JAVAMAIL, JSF, JNDI)
- INTRODUCTION TO CONTAINERS
- TOMCAT AS A WEB CONTAINER
- INTRODUCTION OF JDBC
- JDBC ARCHITECTURE
- DATA TYPES IN JDBC
- PROCESSING QUERIES
- DATABASE EXCEPTION HANDLING
- DISCUSS TYPES OF DRIVERS
- JDBC INTRODUCTION AND NEED FOR JDBC
- JDBC ARCHITECTURE
- TYPES OF JDBC DRIVERS
- JDBC API FOR DATABASE CONNECTIVITY (JAVA.SQL PACKAGE)
- STATEMENT, PREPAREDSTATEMENT, CALLABLE STATEMENT, RESULT SET META DATA, DATABASE META DATA
- OTHER JDBC APIS, CONNECTING WITH DATABASES (MYSQL, ACCESS, ORACLE)

: ASSIGNMENT 1:

1. GIVE FULL FORM OF J2EE
2. GIVE FULL FORM OF ODBC
3. WHAT IS THE NAME OF TYPE -1 DRIVER?
4. GIVE FULL FORM OF JD8C
5. EXPLAIN THIN CLIENT AND THICK CLIENT IN BRIEF.
6. EXPLAIN PREPAREDSTATEMENT WITH EXAMPLE.
7. HOW TO CALL STORED PROCEDURE? EXPLAIN WITH EXAMPLE.
8. JMS STANDS FOR ?
9. WHO CONVERTS THE JAVA DATA TYPE TO THE APPROPRIATE JDBC TYPE BEFORE SENDING IT TO THE DATABASE?
10. EXPLAIN ENTERPRISE ARCHITECTURE OF J2EE.
11. WHAT IS JDBC ? EXPLAIN JDBC ARCHITECTURE.
12. EXPLAIN TYPES OF JDBC DRIVERS.
13. WHICH STATEMENT USED FOR PRECOMPILING SQL STATEMENTS THAT MIGHT CONTAIN INPUT PARAMETERS.
14. WHAT ARE ENTERPRISE ARCHITECTURE STYLES? EXPLAIN IN DETAIL.

PREPARED BY: PROF. N.K.PANDYA (PHD*, MCA, BCA, BPP)
KAMANI SCIENCE COLLEGE, AMRELI(BCA/BSC)

Introduction to J2EE

- The Java 2 platform, Enterprise Edition reduces the cost and complexity of developing multi-tier services.
- It can be rapidly deployed and easily enhanced as the enterprise responds to competitive pressures.
- J2EE is an open standard which is provided by Sun Microsystems for application which run on servers.
- It provides multi-tired architecture for commercial applications. J2EE it is used in order to maintain Speed, Security and Reliability of Server-Side technology.
- It includes J2SE+ most of the other java technologies including JavaMail, Activation, Servlets, JSF, JMS, EJB and others. Most of the API is component- oriented. They are used to provide interfaces for business oriented components for enterprise and distributed internet applications.
- Java has emerged as one of the most of the mature and commonly used programming language for building enterprise software.
- Java has manly Three Platform/Editions:
    - J2SE – stands for Java 2 Standard Edition and is normally used for Desktop Applications
    - J2EE - stands for Java 2 Enterprise Edition for applications which run on server.
    - J2ME - stands for Java 2 Micro Edition for applications which run-on small-scale devices like cell phones.

Enterprise Architecture Types

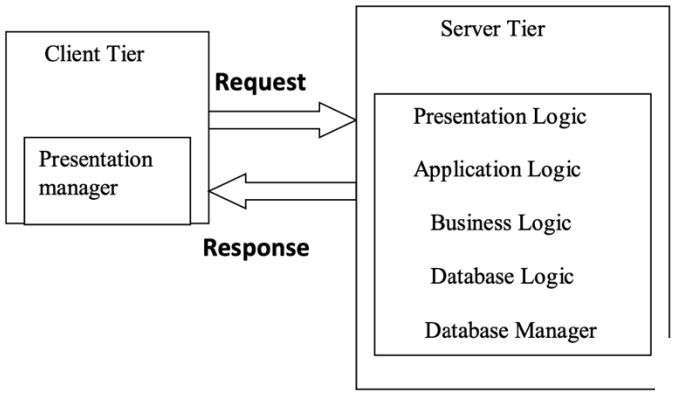
- A Software application composition can be broken down into three fundamental concern or logical layer, The User Interface, The Processing, and Data.
    - The User Interface/ Presentation Layer
    - Processing Logic/Business Layer
    - Database/ Data Access Layer
- **The User Interface:** This is Responsible for displaying interface to the user and collecting data from the user. Often called as the Presentation Layer. Its job is to presents stuff to the user and provides to the software system. The presentation layer includes the part of the Software that creates and controls the user interface and validates the user's action.
- **The Processing / Business layer:** In this layer, the application work and handles the important processing. The logical layer is called the business rules layer.
- **Reading and Writing data/ Data Access Layer**: All nontrivial business application needs to read and store the Data. The part of the Software that is responsible for reading and writing data from whatever source is known as Data Access Layer.
- Whenever we Design any enterprise application, we have to use concept of N-Tier architecture. As we are aware of client-server system which is based on 2-Tier architecture because there is clear separation between the Data and the Logic
- The 2-Tier architecture is generally data driven with application existing entirely on the local machine and the Database deployed at a specific and secure location in the Organization.
- There are four types of architecture which are as follows.
- Single-Tier Architecture
    - 2 -Tier Architecture
    - 3-Tier Architecture
    - N-Tier Architecture

- **The Single Tier Architecture**: Simple Software application is written to run on a single computer such architecture is called as Single tier architecture. Because all of the logical application

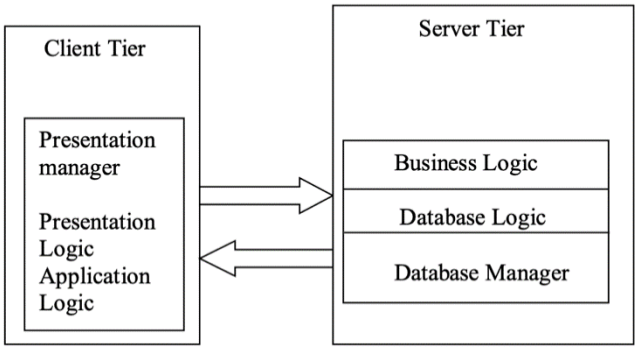
services presentation, the business rules and the data access layer exit in a single computing layer.

- Such systems are relatively easy to manage and Data consistency is simple because the Data is stored at only one single location. The only problem with such systems is it cannot handle multiple users and do not provide easy means of Sharing the Data across the Firm.
- **Advantages** :Single tier system is relatively easy to manage and data consistency is simple because data is stored at only one place.
- **Disadvantages**: Now a day's single location is not sufficient because of changing business needs. We cannot share the data in large amount. It cannot handle multiple users.

- **2-Tier Architecture:** Application which is divided into two separate tiers namely Client machine and database Server machine called as Two Tier Architecture. This is the traditional method of enterprise development. It includes presentation and business logic. In a 2- Tier application the processing load is given to the client while to the server simply controls the Traffic between the application and the Data.



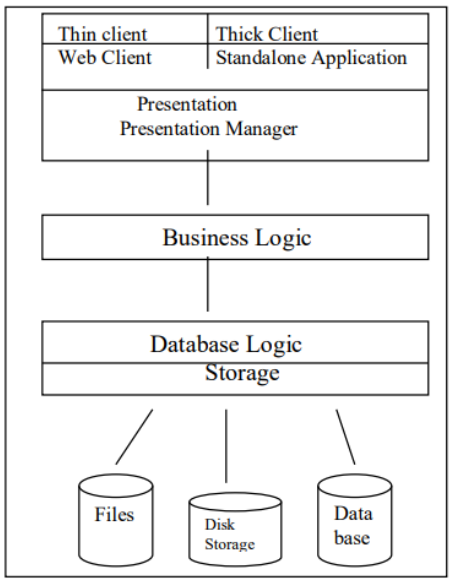
- **Thin Client:** With 2 tier architecture, if the presentation manager resides only with client tier then the client is called as thin client. other presentation logic, application logic, business logic, database logic and database manager resides with the server side.



- **Thick Client:** With 2 tier architecture, If the presentation manager, presentation logic resides with client tier then the client is called as thick client other like business logic, database logic and database manager resides with the server side.
- **Advantages**: Any changes made in data access logic will not affect the presentation and business logic. With 2 tier architecture it is easy to develop an application.
- **Disadvantages**:
- In case of limited resource pc, the performance would be suffered due to pc resources.
- It supports a limited number of users.
- Each client requires its own connection and each connection requires CPU and memory.
- As number of connection increase, the database performance degrades.
- If the application requires more than one database request at a time it is not
- possible for the server to manage it. This architecture requires high maintenance.
- When the data is shared among various users, it will caused problem what is to be displayed to which user regarding the latest data.

3-Tier architecture:

Application which is divided into 3 tiers namely Client tier, middle tier, and database (Enterprise Information System-EIS) tier is known as Three Tier Architecture. Logic physically separates the business rules. Presentation layer and logic runs on Client machine, application and business logic runs on J2EE server and database logic is there with database layer.



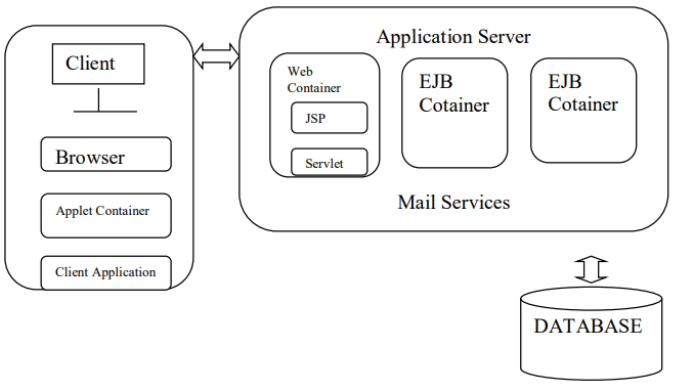
Thin Client: With 3 tier architecture, if the presentation manager resides only with client tier, then the client is called as thin client. other presentation logic, application logic, business logic, database logic and database manager reside with the EIS tier(database).

Thick Client: With 3 tier architecture, if the presentation manager, presentation, application logic resides with client tier then the client is called as thick client. Business logic is only with the business tier. Database logic and database manager resides with the EIS Tier(database).

Advantages: It improves scalability since the application can be deployed on many machines. The database no longer required a connection from every client-it only requires connections from a smaller number of application server. It provides better reusability because the same logic can be initiated from many clients or applications. It provides security because client does not have direct access the database.

Disadvantages: It increases the complexity because to develop the 3-tier application is more difficult than developing a 2-tier application.

N-Tier Architecture: An application which is divided into more than 3 tier can be called as N Tier architecture. In N tier architecture it is not decided how many tiers can be there, it depends on computing and network hardware on which application is deployed. Basically, it is divided into four layers: Application layer, Client tier, business tier (EJB) and database tier (EIS).



Client Tier: It consists of the user interface for user request and print the response. It runs on client machine. It basically uses the browser or applet as client-side application.

Web Tier: It consists of JSP and Servlet dynamic webpages to handle HTTP specific request logons, sessions, accesses the business services and finally constructs response and send back to the client.

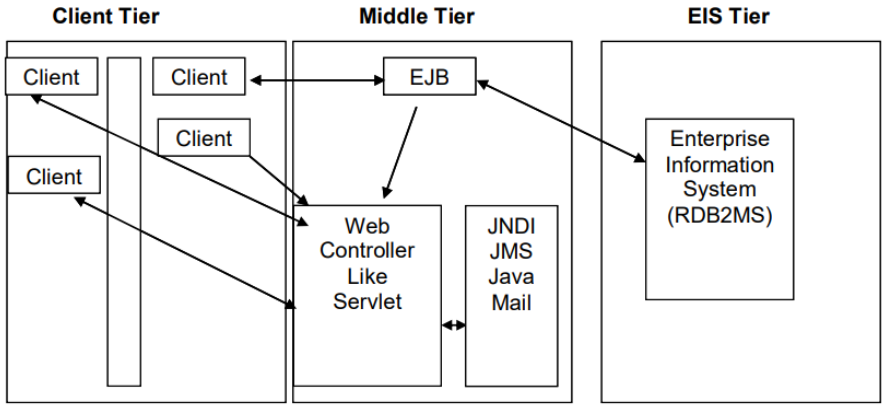
Business Tier: It consists of the business logic for the J2EE application. For Example: EJB (Enterprise Java Beans) the benefit of having a centralized business tier is same business logic can support different type of client like browser, WAP, other standalone application.

EIS Tier: It consists of the DBMS/RDBMS. It handles the users SQL request and generates appropriate response base on queries. It is responsible for communicating with external recourses such as legacy systems, ERP systems, messaging systems like MQSeries etc. It also stores all persistent data in the database. J2EE is based on N tier architecture application. J2EE makes easy to develop the Enterprise application based on 2, 3, more application layers. Here it also proves that J2EE is distributing computing framework and multi-tiered application.

Advantages 1) Improved maintainability 4) Provides consistency 2) Interoperability 5) Flexibility 3) Scalability 6) Security

Disadvantages: It is having complex structure and difficult to setup and maintain all separated layers.

Enterprise Architecture of J2EE: The architecture of J2EE platform uses a multi-tier distributed application model. The application logic, according to the function will be divided into different component and application components make up a J2EE application. The J2EE application consists of three or four tier. J2EE multi-tier application because the architecture has defined three tiers namely client-tier, middle tier and back end tier. This means that the application logic is distributed over different location.



J2EE Components: J2EE has been developed with a group of various components. With the help of these components our programs will work on internet.

- **J2EE Runtime**: The J2EE architecture provides the uniform mean of accessing platform level services via its runtime environment such services include messaging, security, distributed transaction etc. The runtime infrastructure of it specifies the role and interface for the application and the run time on which the application could be deployed.
- **J2EE API:** In the enterprise services, the distributed services require access. These enterprise services include fraction processes, managing, multi-threading, database access to such services in its services APIs. The application programs in the J2EE can access their APIs through the container.

Java Database Connectivity (JDBC 2.0): The JDBC API lets you invoke SQL commands from the java programming method. This API is useful to connect our java database application with any relational database like ORACLE, MYSQL, MSACCESS. You can also use the JDBC API from a servlet or JSP page to access your database directly. The JDBC API is divided into two parts. An application-level interface used by the application component to access the database. Service provider interface to attach a JDBC driver to the J2EE platform.

EJB: (Enterprise Java Beans) An enterprise beans is a body of code with field and method to implement modules of the business log. An enterprise bean can be considered as building block that can be used alone or with other enterprise beans to execute business logic on J2EE server. It is useful to define server-side component.

There are kinds of enterprise beans.

- Session beans
- Entity beans
- Message beans

Java Servlet (JS) : This API provides object oriented approach for building dynamic web application. The Java servlet technology lets you to define the http specific servlet classes. A servlet class extends the capability of server that host the application.

- **Java Server Page (JSP 2.1)**: This API provides easy way for building dynamic web application. The JSP technology allows you to put the combines snippets (general UDF) of java programming language code with static markup text-based documents. A JSP page is a document which contains two types of text.
  - Static template data in text-based format like html, xml.
  - JSP elements that determine how the page constructs the dynamic contain.

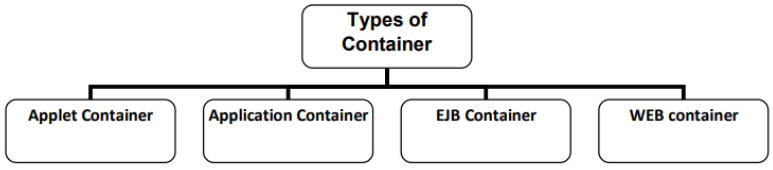
- **Java Server Faces (JSF)**: JSF is the relatively the new technology that attempts to provide a rich user interface for the web application used in conjunction with servlet and JSP.

- **Java Message Services (JMS 1.1):** The JMS API is a messaging standard that allows J2EE application components to create, send, receive and read messages. It enables distributed communication that is loosely coupled, reliance and asynchronous.

- **Java Transaction API (JTA 1.0):** The JTA API provides the interface for the transaction. The J2EE architecture provides a default auto-committee to handle the transaction committee and roll back.

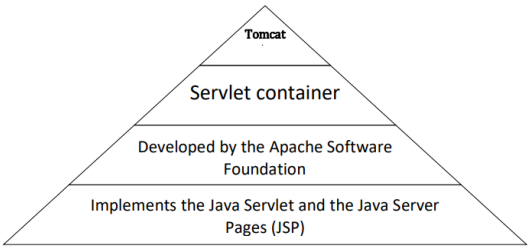
- **Java Mail API (1.3)**: Many applications need to send e-mail notification for that J2EE platform includes the Java Mail API with Java Mail Service provider that application component can use to send an internet e-mail. It has two parts.
  - An application-level interface used by the application to send mails.
  - A services provider interface.
- **Java Naming and Directory Interface (JNDI):** JNDI provides the naming & directory functionality. It provides the application with the methods for performing standard directory operations such as a J2EE application can store & receive any type of named java object.

- **Introduction to J2EE Containers**:
- "Containers are interface between a component and client or platform-oriented functionality which supports component. Container provides communication platform between client and component.'
- Containers are the central thing of J2EE architecture container are like the rooms in house, people and thing exists in a room and interface with the infrastructure through well-defined interface.



- **Applet Container**: Applet Container is a client container which is used to manage an execution of the applet on the browser.
- **Application Container**: Application Container is a client container which is used to manage application client and their container.
- **EJB Container**: EJB Container is a server container which is used to manage Enterprise bean component.
- **WEB container**: WEB container is a server container which is used to manage an execution of the JSP and Servlet Components.

- **Tomcat as a Web Container**



  - Apache Tomcat is a servlet container developed by the Apache Software Foundation developed by the Apache Software Foundation (ASF).
  - Tomcat implements the Java Servlet and the Java Server Pages (JSP) specifications from Sun Microsystems, and provides a "pure Java" HTTP web server environment for Java code to run.
  - Tomcat should not be confused with the Apache web server, which is a C implementation of an HTTP web server; these two web servers are not bundled together.
- Apache Tomcat includes tools for configuration and management, but can also be configured by editing XML configuration files.

J2EE as Application Server

An application server can be either a software framework that provides a generalized approach to creating an application-server implementation, regard to what the application functions are, or the server portion of a specific implementation instance. In either case, the server's function is dedicated to the efficient execution of procedures (programs, routines, scripts) for supporting its applied applications.

Most Application Server Frameworks contain a comprehensive service layer model. An application server acts as a set of components accessible to the software developer through an API defined by the platform itself. For Web applications, these components are usually performed in the same running environment as its web server(s), and their main job is to support the construction of dynamic pages.

However, many application servers target much more than just Web page generation: they implement services like clustering, fail-over, and load-balancing, so developers can focus on implementing the business logic. In the case of Java application servers, the server behaves like an extended virtual machine for running applications, transparently handling connections to the database on one side, and, often, connections to the Web client on the other. Other uses of the term may refer to the services that a server makes available or the computer hardware on which the services run.

Java Platform, Enterprise Edition or Java EE (was J2EE) defines the core set of API and features of Java Application Servers. The Web modules include servlets, and JavaServer Pages. Enterprise JavaBeans are used to manage transactions.

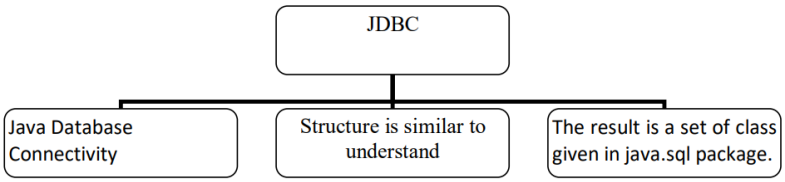
Some Java Application Servers leave off many Java EE features like EJB and JMS including Tomcat from Apache, and Jetty from Eclipse Foundation. Their focus is more on Java Servlets and JavaServer Pages.

There are many open source Java application servers that support Java EE including JOnAS from Object Web, JBoss AS from JBoss (division of Red Hat), Geronimo from Apache, TomEE from Apache, Resin Java Application Server from Caucho Technology, Blazix from Desiderata Software, Enhydra Server from Enhydra.org, and GlassFish from Oracle.

Commercial, non open-source, Java application servers have been dominated by WebLogic Application Server by Oracle and WebSphere Application Server from IBM.

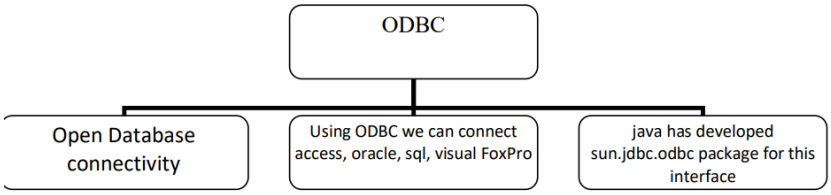
A Java Server Page (JSP) executes in a Web container. JSPs provide a way to create HTML pages by embedding references to the server logic within the page. HTML coders and Java programmers can work side by side by referencing each other's code from within their own. The application servers mentioned above mainly serve Web applications, and services via RMI, EJB, JMS and SOAP. Some application servers target networks other than web-based ones: Session Initiation Protocol servers, for instance, target telephony networks.

Introduction and Need for JDBC



- JDBC Stands for the Java Database Connectivity.
- The Role of the JDBC is to most important with respect to java database application.
- If you are familiar with SQL and relation Database.
- The structure of JDBC API is similar and easy to understand.
- JDBC API provides connectivity and Data access across the range of relational database.
- JDBC generalize the most common database access function by abstracting the vendor specific details of the database.
- The result is a set of class given in java.sql package.
- JDBC enables java application & applet common to the database.
- The main idea behind this is we have been able to have different application to the different database. The java.sql.package gives us to facility to connect to every database.

JDBC-ODBC



ODBC stands for the Open Database connectivity, where we want to our regular database like access, oracle, sql, visual FoxPro etc. we have to use the JDBC ODBC driver, ODBC is standard which is purposed by Microsoft and we know that Microsoft is widely used in computer technology so sun Microsystems has accepted ODBC as Universal driver and java has developed sun.jdbc.odbc package for this interface. To connect JDBC by using ODBC we have to develop one ODBC driver by using the Control panel 32-bit ODBC program from your window application.

The Steps for the Creating DSN are as follows.

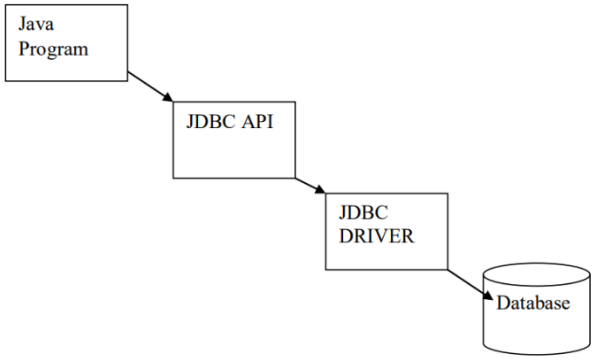
- Double click on 32-ODBC Program
- Click on New Button
- Give the DSN Name
- Select appropriate Driver
- Click on to Complete Preparation

JDBC V/S ODBC:

- **ODBC** cannot be directly used with Java because it uses C interface. Calls from Java to native C code have number of drawbacks in the security, implementation, robustness and automatic portability of applications.
- **ODBC** make use of Pointer which has been totally removed from java.
- **ODBC** mixes simple advanced features together and has complex options for simple queries. But **JDBC** is designed to keep things simple while allowing advanced capabilities when required.
- **ODBC** requires manual installation of the ODBC driver manager and driver on all client machine.
- **JDBC** drivers are written in Java and JDBC code is automatically installable, secure and portable on all java platform.
- **JDBC API** is a natural Java interface and is built on ODBC JDNC retains some of the basic features of ODBC like SQL call level interface.

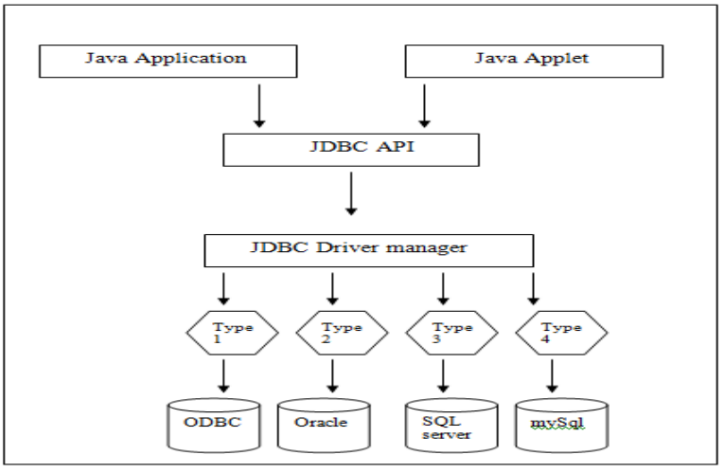
JDBC API

JDBC API Is collection of methods, classes and interfaces that enable java applications to communicate with database. For this database should be installed on the computer and RDBMS should provide a driver. JDBC classes and interfaces defined in the package java.sql constitute the JDBC API. The relations between a java program, JDBC API, the JDBC driver and database is shown in figure.



- Using JDBC API a java application can perform insert, delete, update and select rows in a database. It can also retrieve the data from the database and show it in a proper format to the user according to the SQL query.
- The JDBC API can be used in 3 tier database architecture. In 2 tier architecture, java programs invoke the method of JDBC API and communicate with the database server. so, there is a direct interaction with the database server.

JDBC Architecture: JDBC Architecture describes the interaction of JDBC API with java application and java applet. JDBC API consists of several call level interfaces for interaction with JDBC Driver Manager and JDBC driver for defining database. JDBC API is responsible for transferring data between an application and a database.



JDBC Data types:

- The JDBC driver converts the Java data type to the appropriate JDBC type before sending it to the database. It uses a default mapping for most data types. For example,
- a Java int is converted to an SQL INTEGER. Default mappings were created to provide consistency between drivers.
- The following table summarizes the default JDBC data type that the Java data type is converted to when you call the setXXX() method of the PreparedStatement or CallableStatement object or the ResultSet.updateXXX() method.

SQL	JDBC/Java	setXXX	updateXXX
VARCHAR	java.lang.String	setString	updateString
CHAR	java.lang.String	setString	updateString
LONGVARCHAR	java.lang.String	setString	updateString
BIT	boolean	setBoolean	updateBoolean
NUMERIC	java.math.BigDecimal	setBigDecimal	updateBigDecimal
TINYINT	byte	setByte	updateByte
SMALLINT	short	setShort	updateShort
INTEGER	int	setInt	updateInt
BIGINT	long	setLong	updateLong
REAL	float	setFloat	updateFloat
FLOAT	float	setFloat	updateFloat
DOUBLE	double	setDouble	updateDouble
VARBINARY	byte[ ]	setBytes	updateBytes
BINARY	byte[ ]	setBytes	updateBytes
DATE	java.sql.Date	setDate	updateDate
TIME	java.sql.Time	setTime	updateTime
TIMESTAMP	java.sql.Timestamp	setTimestamp	updateTimestamp
CLOB	java.sql.Clob	setClob	updateClob
BLOB	java.sql.Blob	setBlob	updateBlob
ARRAY	java.sql.Array	setARRAY	updateARRAY
REF	java.sql.Ref	SetRef	updateRef
STRUCT	java.sql.Struct	SetStruct	updateStruct

- JDBC 3.0 has enhanced support for BLOB, CLOB, ARRAY, and REF data types. The
- ResultSet object now has updateBLOB(), updateCLOB(), updateArray(), and updateRef() methods that enable you to directly manipulate the respective data on the server.
- The setXXX() and updateXXX() methods enable you to convert specific Java types to specific JDBC data types. The methods, setObject() and updateObject(), enable you to map almost any Java type to a JDBC data type.

- ResultSet object provides corresponding getXXX() method for each data type to retrieve column value. Each method can be used with column name or by its ordinal position.

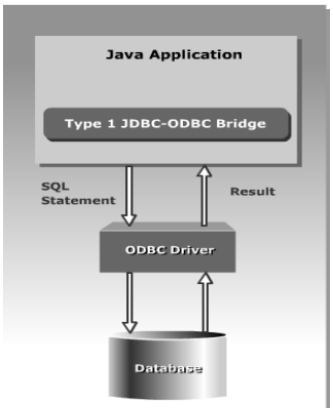
Handling NULL Values:

- SQL's use of NULL values and Java's use of null are different concepts. So how do you handle SQL NULL values in Java?
- There are few strategies you can use:
  - Avoid using getXXX( ) methods that return primitive data types.
  - Use wrapper classes for primitive data types, and use the ResultSet object's wasNull( ) method to test whether the wrapper class variable that received the value returned by the getXXX( ) method should be set to null.

JDBC Database Drivers: JDBC Drivers specification classifies JDBC drivers into four groups. Groups are referred to as JDBC Drivers types and address a specific need for Communicating with various DBMS.

Type 1 : JDBC to ODBC Driver

The JDBC-ODBC bridge driver converts all JDBC calls into ODBC calls and sends them to ODBC Driver. The ODBC driver then converts the call to the database server. For example: MS ACCESS do not provide JDBC Driver driver. Instead, they provide ODBC Driver. One can access such database by using ODBC-JDBC Bridge.



Advantages:

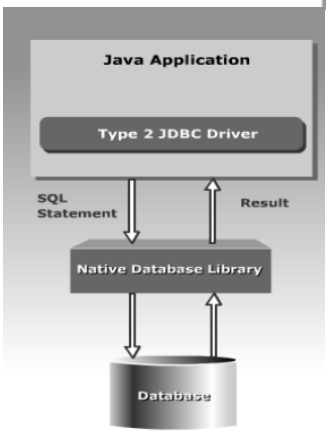
It allows access to any database

Disadvantages:

- This driver not fully written in java so not portable.
- Performance is slow compare to all drivers because JDBC calls are converted to ODBC
- calls and ODBC calls are converted to database specific calls.
- Same step in reverse order when query is obtained.
- Client system requires ODBC installation to use the drivers.
- Not good for web application or for large amount of database-based application.

Type 2: NATIVE API Driver

The Java/Native Code driver Communicates directly with database server. Therefore the vendor database library needs to be loaded on each client computer. It converts JDBC calls into database specific calls for databases. For example: Oracle will have native API



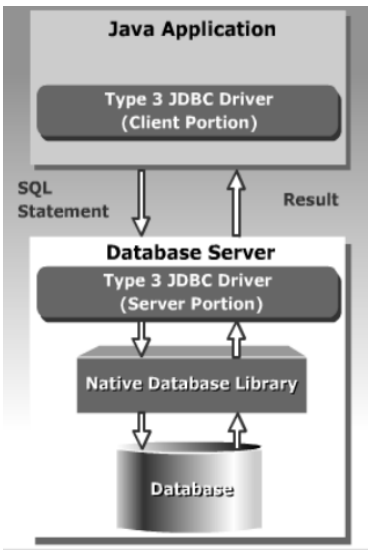
Advantages: They offer better performance than JDBC-ODBC Bridge. Because as the layers of communication are less than type-1 driver and also use native API which is database specific.

Disadvantages

- Native API must be installed in client system and hence type-2 driver cannot be used for internet.
- It is not written in java which forms a portability issue
- If we change the database, we have to change Native API as it is specific to database.
- Mostly outdated because it is not thread safe

Type-3 JDBC Drivers/NET Protocol Driver:

This driver follows a 3 tired approach. In 3 tired approach JDBC requests are passed to a middle tier server. The middle Tier server then translates request to the database specific native connectivity interface and forward the request to the database server. If middle server is written in java can use type-1 or type-2 driver to forward the request to the database server. It is used to connect client application/applet to database server over TCP/IP connection. The presence of any vendor database library on client computer is not required because type-3 driver is server based.



Advantages:

- It is sever based so no need for any vendor database library to be present on client machine.
- This is fully written in java and hence portable, so it is suitable for web application.
- It provides portability, performance, security and scalability.
- Net protocol can be designed to make client JDBC driver very small and fast to load.
- It provides support for feature such as caching, load balancing, advanced
- system administration such as logging and auditing.
- Very flexible and allows access to multiple databases using one driver
- They are efficient among all drivers.

Disadvantages:

Type-3 driver requires database specific coding to be done on middle tier and it requires additional server for that. Additionally traversing the record set may take longer because the data comes-through the backend server or backend database.

Type-4 JDBC Driver/Native Protocol:

Type-4 driver are completely written in java to achieve platform independent. It converts JDBC calls into vendor specific DBMS protocol. Therefore, client application can communicate directly with the database server when type-4 driver is used. Type-4 driver is better than other driver because type-4 driver does not have to translate database request to ODBC calls or a native connectivity interface or pass on to another server.

Disadvantages:

Different driver is needed for each database.

Advantages:

- They are written in java to achieve platform independence.it reduces deployment administration issues. It is most suitable for web application.
- Number of layers is very less Type-4 do not translate database request to
- ODBC or native connectivity interface or to pass to the server so performance is good.
- You do not need to install special software in the client machine.
- These drivers can be downloaded dynamically.

JDBC Process:

Processing SQL Statements with JDBC

In general, to process any SQL statement with JDBC, you follow these steps:

- Establishing a connection.
- Create a statement.
- Execute the query.
- Process the ResultSet object.
- Close the connection.

Creating Statements

A Statement is an interface that represents a SQL statement. You execute Statement objects, and they generate ResultSet objects, which is a table of data representing a database result set. You need a Connection object to create a Statement object.

There are three different kinds of statements:

Statement: Used to implement simple SQL statements with no parameters.

PreparedStatement: (Extends Statement.) Used for precompiling SQL statements that might contain input parameters.

CallableStatement: (Extends Prepared Statement.) Used to execute stored procedures that may contain both input and output parameters.

Executing Queries

To execute a query, call an execute method from Statement such as the following:

execute: Returns true if the first object that the query returns is a ResultSet object. Use this method if the query could return one or more ResultSet objects. Retrieve the ResultSet objects returned from the query by repeatedly calling Statement.getResultSet.

executeQuery: Returns one ResultSet object.

executeUpdate: Returns an integer representing the number of rows affected by the SQL statement. Use this method if you are using INSERT, DELETE, or UPDATESQL statements.

Processing ResultSet Objects

You access the data in a ResultSet object through a cursor. Note that this cursor is not a database cursor. This cursor is a pointer that points to one row of data in the ResultSet object. Initially, the cursor is positioned before the first row. You call various methods defined in the ResultSet object to move the cursor.

Closing Connections

When you are finished using a Statement, call the method Statement. close to immediately release the resources it is using. When you call this method, its ResultSetobjects are closed.

JDBC Exception Handling:

Exception handling allows you to handle exceptional conditions such as programdefined errors in a controlled fashion. When an exception condition occurs, an exception is thrown. The term thrown means that current program execution stops, and control is redirected to the nearest applicable catch clause. If no applicable catch clause exists, then the program's execution ends.JDBC Exception handling is very similar to Java Excpetion handling but for JDBC, the most common exception you'll deal with is java.sql.SQLException.

A SQLException can occur both in the driver and the database. When such an exception occurs, an object of type SQLException will be passed to the catch clause. The passed SQLException object has the following methods available for retrieving additional information about the exception:

Method	Description
getErrorCode( )	Gets the error number associated with the exception.
getMessage( )	Gets the JDBC driver's error message for an error handled by the driver or gets the Oracle error number and message for a database error.
getSQLState( )	Gets the XOPEN SQLstate string. For a JDBC driver error, no useful information is returned from this method. For a database error, the five-digit XOPEN SQLstate code is returned. This method can return null.
getNextException( )	Gets the next Exception object in the exception chain.
printStackTrace( )	Prints the current exception, or throwable, and its backtrace to a standard error stream.
printStackTrace(PrintStream s)	Prints this throwable and its backtrace to the print stream you specify.
printStackTrace(PrintWriter w)	Prints this throwable and its backtrace to the print writer you specify.

JDBC API FOR DATABASE CONNECTIVITY

Connection Interface:
This defines connection to different database. This provides following methods.

1. **createStatement()**
   Create a Statement: From the connection interface, you can create the object for this interface. It is generally used for general–purpose access to databases and is useful while using static SQL statements at runtime.
   For example: Statement statement = connection.createStatement();
   Implementation: Once the Statement object is created, there are three ways to execute it.
   boolean **execute**(String SQL): If the ResultSet object is retrieved, then it returns true else false is returned. Is used to execute SQL DDL statements or for dynamic SQL.
   int **executeUpdate**(String SQL): Returns number of rows that are affected by the execution of the statement, used when you need a number for INSERT, DELETE or UPDATE statements.
   ResultSet **executeQuery**(String SQL): Returns a ResultSet object. Used similarly as SELECT is used in SQL.
2. **preparedStatement()** is used to execute dynamic or parameterized SQL queries. PreparedStatement extends Statement interface. You can pass the parameters to SQL query at run time using this interface. It is recommended to use PreparedStatement if you are executing a particular SQL query multiple time. It gives better performance than Statement interface. Because, PreparedStatement are precompiled and the query plan is created only once irrespective of how many times you are executing that query. Prepared Statement represents a recompiled SQL statement, that can be executed many times. This accepts parameterized SQL queries. In this, “?” is used instead of the parameter, one can pass the parameter dynamically by using the methods of PREPARED STATEMENT at run time.

Example:

```
String query = "INSERT INTO people(name, age)VALUES(?, ?)";
Statement pstmt = con.prepareStatement(query);
pstmt.setString(1,"Ayan");
ptstmt.setInt(2,25);
```

Implementation: Once the PreparedStatement object is created, there are three ways to execute it:

execute(): This returns a boolean value and executes a static SQL statement that is present in the prepared statement object.

executeQuery(): Returns a ResultSet from the current prepared statement.

executeUpdate(): Returns the number of rows affected by the DML statements such as INSERT, DELETE, and more that is present in the current Prepared Statement.

3. **PrepareCall():**CallableStatement is used to execute the stored procedures. CallableStatement extends PreparedStatement. Usng CallableStatement, you can pass 3 types of parameters to stored procedures. They are : IN – used to pass the values to stored procedure, OUT – used to hold the result returned by the stored procedure and IN OUT – acts as both IN and OUT parameter. Before calling the stored procedure, you must register OUT parameters using registerOutParameter() method of CallableStatement. The performance of this interface is higher than the other two interfaces. Because, it calls the stored procedures which are already compiled and stored in the database server.

   Example:

   CallableStatement cstmt = con.prepareCall("{call Procedure_name(?, ?}");

   Implementation: Once the callable statement object is created

   execute() is used to perform the execution of the statement.

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

4. **Close()**:It is used to release connection do not wait for it to release it automatically.
   con.close (): This method having the following syntax
   Public void close() throws SQLException
5. **Commit():** It is used to save all the state of the database which are changed by transaction. It can be written as follows:
   con.commit();
   This method having the following syntax
   - Public void commit() throws SQLException

6. **rollback()**: It is used to cancel the effect of current running transaction.

   It can be written as follows: con.rollback ();
   This method having the following syntax
   - Public void rollback() throws SQLException
7. **SetAutoCommit()**: It is used to do auto commit all the transaction .if we set auto commit with connection all SQL statements executed and committed together as individual transaction.It has Boolean type parameter .If we pass true then it enables the auto commit and if we pass false it disables the auto commit.
8. **getAutoCommit():** It is used to get curruent commited state. It can be written as follows:
   Boolean b=con.getAutoCommit ();
   This method having the following syntax
   - Public boolean getAutoCommit () throws SQLException
9. **isClosed():** It is used to check whether the connection has been closed or not.
   Boolean b=con.isClosed();
   This method having the following syntax
   Public boolean isClosed () throws SQLException

10. **getMetaData():**It is used to get metadata(data about data)related to current connection with database.it provides information like table driver name,driver type,its version and so on.It ses DatabaseMetaData interface object to get information about database drivers. For example: Connection con; con=DriverManager.getConnection (url,"", ""); DatabaseMetadata ds=con.getMetadata ();

DriverManager Class

DriverManager class belongs to java.sql package.It consists of static method to manage JDBC Drivers. Each and every driver must register with DriverManager class. There are many JDBC Drivers used for different JDBC servers. For example: JDBC Drivers for MS ACCESS is different from ORACLE Driver. It controls interface between application and JDBC Drivers.

DriverManager class has one method getConnection() to establish connection with different database server.It has 3 syntax:
Public static Connection getConnection (String url) throws SQLException
Public static Connection getConnection (String url, Properties info) throws SQLException
Public static Connection getConnection (String url, String username, String password) throws SQLException

Statement Interface

Statement object is used to execute SQL statements and obtain the result produced after excuting the SQL statements.it is having following methods.

1. **ExecuteQuery()**: It is used to fetch records from database using select query.It passes SQL statement as a parameter.It returns single Resultset Objects that contains rows, columns and metadata that represent data requested by query. Example:
   Statement st;
   St=con.createStatement ();
   ResultSet rs=st.executeQuery ("select * from tablename");
   Syntax:
   Public static ResultSet executeQuery (String url) throws SQLException
2. **ExecuteUpdate()**:It is used to perform insert, update, and delete operation on record of database.It returns integer value indicating no. of records updated in database.It passes SQL as a parameter.
   Statement st;
   St=con.createStatement ();
   ResultSet rs=st.executeUpdate ("insert into tablename values (val1, val2..., valn");
   Syntax:
   Public int executeUpdate (String url) throws SQLException
3. **execute():** The execute () Statement is used when they returns Multiple result. It returns Boolean indication that if it returns true it defines that next result is ResultSet object and if it returnd false then it defines no of records updated or no more results.
   Statement st;
   st=con.createStatement ();
   String sql="create table tablename (fieldname type (size), fieldname type(size))";
   st.execute()
   Syntax:
   Public Boolean execute (String url) throws SQLException

Prepared Statement interface

Prepared Statement interface is derived from Statement interface. It uses a template to create SQL request and uses a Prepared Statement object to send precompiled SQL statements with one or more parameters. It is more convenient to use a Prepared Statement object for sending SQL statements to database. If you want to execute SQL statements many times Prepared Statement object reduces execution time in comparison to other Statement object.

Main feature of it is SQL statements is given to Prepared Statement object when SQL statements is created.so advantage of Prepared Statement object is that in many of the cases SQL statements is sent to database immediately, where it is compiled. It also used to execute precompiled statements.

Creating object of Prepared Statement

String sql="insert into tablename(field1, field1) values(?,?)";
St.execute (sql);
PreparedStatement pstm=con.prepareStatement(sql);

SetXXX() methods

- **setInt(int parameterindex,int x)**
- **setString(int parameterindex,int x)**
- **setChar(int parameterindex,int x)**
- **setFloat(int parameterindex,int x)**
- **setDouble(int parameterindex,int x)**
- **setLong(int parameterindex,int x)**
- **setShort(int parameterindex,int x)**
- **setDate(int parameterindex,int x)**
- **setByte(int parameterindex,int x)**
- **setBlob(int parameterindex,int x)**

Here parameter index is first parameter 1,second 2 and so on. X-is the object containing input parameter value

Execute methods:

1. **executeQuery():**
   It is used to fetch records from database using select query.It passes SQL statement as a parameter.It returns single Resultset Objects that contains rows, columns and metadata that represent data requested by query.
   Example:
   String sql="select * from tablename where fieldname=? ";
   setString(1,"C01");
   PreparedStatement pst =con.prepareStatement ();
   ResultSet rs=pst.executeQuery ();
   Syntax: public static ResultSet executeQuery throws SQLException
2. **ExecuteUpdate():**
   It is used to perform insert, update, and delete operation on record of database. It returns integer value indicating no. of records updated in database. It passes SQL as a parameter.
   Statement st = con.createStatement ();
   Int n=st.executeUpdate ("insert into tablename values (val1,val2,.....,valn");
   Syntax: Public int executeUpdate throws SQLException
3. **execute():**
   It is used to execute SQL statements which returns multiple results.It returns the Boolean indicating that value if it returns true then it defines that next result is ResultSet obkect and if it returns false then it defines that number of records updated or no more results is there.It passes any sql statements as parameter.

Packages to import

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
```

Callable Statement

- The callable statement is useful to called stored procedure from within a J2EE object. Stored procedure is a block of code and is identified by a unique name. The type and style of code depend on dbms vendor and can be return in PL/SQL. The stored procedure is invoked by giving name of stored procedure.

- The callable statement uses three types of parameters when calling a stored procedure. These parameters are IN, OUT & INOUT.
- The IN parameter contains any data that needs to be passed to the stored procedure and whose value is assign using the setXxx ().
- The OUT parameter contains the value return by the stored procedure if any OUT parameter must be registered using the register OUT parameter () and then his later retrieve by J2EE component by using getXxx ().
- The INOUT parameter is a single parameter that is use for both that is to pass information to stored procedure and to retrieve information from stored procedure.
- In above program, the statement of try block creates a query that will call the stored procedure last no. order which retrieve the most recently used order number.
- The stored procedure requires one parameter that is represented by " ? ".
- The parameter is an OUT parameter that will contain last order no. following the execution of stored procedure.
- Next the preparedCall () of connection object is called and it pass the query. This method return callable statement which is called a since an OUT parameter is used by stored procedure. The parameter must be register using registeroutParameter () of callable Statement object.
- The registeroutParameter () requires two parameters. The first parameter is an integer that represents no. of the parameter. Here one means, the first parameter of stored procedure and second parameter is data type of value return by the stored procedure.
- The execute () of callable statement will be useful to execute the query. After the stored procedure is executed, getString() is called to return the value of specified parameter of stored procedure.
- Syntax for stored procedure with IN parameters: {call procedure_name [(?,?,.....)]}
- Syntax for stored procedure with IN and OUT parameters: { ? = call procedure_name [(?,?,.....)]}
- Syntax for stored procedure with no parameters: {call procedure_name}
- CallableStatement sctm=con.prepareCall ("{call insertdata(?,?)}");

ResultSet

It is used to access the data of the table from database. Resultset oject stores the data of the table by executing the query.It also maintains the cursor position for navigation of the data.Cursor can ove on first() , last() , previous() ,next() from the current row position.It also fetch data from the table using getXXX() method depends on columnType(getString(),getInt()etc.).It can fetch data either using getXXX().Index number always starts with one.

There are six methods of ResultSet object that are used to position the virtual cursor. They are first() , last() , previous() , next(),absolute() , relative() , and getRow().

Fields of result set interface:

The Statement object created using creates Statements three constants.

TYPE_SCROLL_INSENSITIVE: It defines that the cursor can scroll but cannot be modified or updated.
TYPE_SCROLL_SENSITIVE: It defines that the cursor can scroll and also be modified or updated.
TYPE_FORWARD_ONLY: It defines that the cursor from the current row moves forward only.
CONCURE_READ_ONLY: It defines that cursor can scroll and also can be modified or updated.

Methods of ResultSet Interface

- **getString()** : It is used to fetch the string type of records from table.It returns the String object.
- **getInt()** : It is used to fetch the integer type of records from table.It returns the String object.
- **getBoolean()** : It is used to fetch the boolean type of records from table.It returns either true/false.
- **getDouble()** : It is used to fetch the decimal type of records from table.It returns the double value.

- **getFloat()** : It is used to fetch the float type of records from table.It returns the String object.
- **getDate()** : It is used to fetch the date type of records from table.
- **getLong()** : It is used to fetch the long type of records from table.
- **getShort()** : It is used to fetch the short type of records from table.
- **getByte()** : It is used to fetch the string type of records from table.
- **getBlob()** : It is used to fetch binary representing large object from table can be image file,audio file,video file.

Example:

```
ResultSet rs=st.executeQuery ("select * from emp"); Int id=re.getInt(1);
String name=rs.getString(2);
                    OR
ResultSet rs=st.executeQuery("select * from emp"); Int id=re.getInt("eno");
String name=rs.getString("ename");
```

Navigation Method

- **first()**: It is used to move the cursor position on the first record of the table.
- **last()**:It is used to move the cursor position on the last record of the table.
- **next()**:It is used to move the cursor position on the next record of the table.
- **previous()**:It is used to move the cursor position on the previous record of the table.
- **afterLast()**:It is used to move the cursor on after the last row.
- **beforeFirst()**:It is used to move the cursor on before the first row.
- **relative(int row)**: It is used to move the cursor on relative number of rows.
- **absolute(int rows)**: It is used to move the cursor on absolute row.

Meta Data

Metadata is about data (data about data) . There are two types of metadata interface are available (1) ResultsetMetadata and (2) Database Metadata. The DatabaseMetaData interface is used to retrieve information about databases, tables, columns and indexes among other information about the DBMS.

ResultSetMataData:

This interface is used to obtain information about type and properties of the column in a ResultSetMataData are used to store following information of Resultset object:

Method	Description
getColumnCount()	Returns the number of columns contained in the ResultSet
getColumnName(int n)	Returns the name of the column specified by the column number
getColumnType(int n)	Returns the data type of the column specified by the column number.
isNullable(int column);	Indicates the null ability of values in the designated columns.

DatabaseMetaData Interface

This interface describes the database as whole. Many methods of DatabaseMetaData interface return list of information in the form of Resultset objects.It provides many methods that represent comprehensive information of the database.

Method	Description
getDatabaseProductname()	Returns the product name of the database
getUserName()	Returns the username
getURL()	Returns the URL of the database
getSchemas()	Returns all the schema names available in this database.
getPrimaryKeys()	Returns primary keys
getProcedures()	Returns stored procedure names
getTables()	Returns names of tables in the database.
getDriverName()	Retrives the name of JDBC Driver
getDriverVersion()	Retrives the version of JDBC Driver
isReadOnly()	Retrives whether this database is read only.

Connecting with Databases

The programming involved to establish a JDBC connection is fairly simple. Here are these simple four steps:

- **Import JDBC Packages**: Add import statements to your Java program to import required classes in your Java code.
- **Register JDBC Driver**: This step causes the JVM to load the desired driver implementation into memory so it can fulfil your JDBC requests.
- **Database URL Formulation**: This is to create a properly formatted address that points to the database to which you wish to connect.
- **Create Connection Object**: Finally, code a call to the DriverManager object's getConnection ( ) method to establish actual database connection.

Import JDBC Packages:
The Import statements tell the Java compiler where to find the classes you reference in your code and are placed at the very beginning of your source code. To use the standard JDBC package, which allows you to select, insert, update, and delete data in SQL tables, add the following imports to your source code:

```
import java.sql.* ;
```

Register JDBC Driver:
You must register your driver in your program before you use it. Registering the driver is the process by which the Oracle driver's class file is loaded into memory so it can be utilized as an implementation of the JDBC interfaces. You need to do this registration only once in your program. You can register a driver in one of two ways.

1. Class.forName():
   The most common approach to register a driver is to use Java's Class.forName() method to dynamically load the driver's class file into memory, which automatically registers it. This method is preferable because it allows you to make the driver registration configurable and portable.

```
try {
Class.forName ("oracle.jdbc.driver.OracleDriver"); }
catch (ClassNotFoundException ex) {
System.out.println ("Error: unable to load driver class!"); System.exit (1);
}
```

   You can use getInstance () method to work around noncompliant JVMs, but then you'll have to code for two extra Exceptions as follows:

```
try {
Class.forName ("oracle.jdbc.driver.OracleDriver").newInstance ();
}
catch(ClassNotFoundException ex) {
System.out.println("Error: unable to load driver class!"); System.exit(1);
catch(IllegalAccessException ex) {
System.out.println("Error: access problem while loading!"); System.exit(2);
catch(InstantiationException ex)
{ System.out.println("Error: unable to instantiate driver!"); System.exit(3);
}
```

2. **DriverManager.registerDriver()**
   The second approach you can use to register a driver is to use the static DriverManager.registerDriver() method. You should use the registerDriver() method if you are using a non-JDK compliant JVM, such as the one provided by Microsoft.
   The following example uses registerDriver() to register the Oracle driver:

```
try {
Driver myDriver = new oracle.jdbc.driver.OracleDriver(); DriverManager.registerDriver( myDriver);
}
catch(ClassNotFoundException ex) {
System.out.println("Error: unable to load driver class!");
System.exit(1); }
```

Database URL Formulation:
After you've loaded the driver, you can establish a connection using the DriverManager.getConnection() method. For easy reference, let me list the three overloaded DriverManager.getConnection() methods:

- getConnection(String url)
- getConnection(String url, Properties prop)
- getConnection(String url, String user, String password)

Here each form requires a database URL. A database URL is an address that points to your database.

Formulating a database URL is where most of the problems associated with establishing a connection occur.
Following table lists down popular JDBC driver names and database URL.

RDBMS	JDBC driver name	URL format
MySQL	com.mysql.jdbc.Driver	**jdbc:mysql://**hostname/ databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	**jdbc:oracle:thin:@**hostname:port Number:databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	**jdbc:db2:**hostname:port Number/databaseName
Sybase	com.sybase.jdbc.SybDriver	**jdbc:sybase:Tds:**hostname: port Number/databaseName

All the highlighted part in URL format is static and you need to change only remaining part as per your database setup.

Create Connection Object:
Using a database URL with a username and password:
I listed down three forms of DriverManager.getConnection () method to create a connection object. The most commonly used form of getConnection () requires you to pass a database URL, a username, and a password: Assuming you are using Oracle's thin driver, you'll specify a host:port:databaseName value for the database portion of the URL.
If you have a host at TCP/IP address 192.0.0.1 with a host name of localhost, and your Oracle listener is configured to listen on port 1521, and your database name is EMP, then complete database URL would then be:
jdbc:oracle:thin:@localhost:1521:EMP

Now you have to call getConnection() method with appropriate username and password to get a Connection object as follows:
String URL = "jdbc:oracle:thin:@localhost:1521:EMP"; String USER = "username";
String PASS = "password"
Connection conn = DriverManager.getConnection(URL, USER, PASS);

Using only a database URL: A second form of the DriverManager.getConnection( ) method requires only a database URL:
DriverManager.getConnection(String url);

However, in this case, the database URL includes the username and password and has the following general form: jdbc:oracle:driver:username/password@database

So the above connection can be created as follows:
String URL = "jdbc:oracle:thin:username/password@localhost:1521:EMP"; Connection conn =DriverManager.getConnection(URL);

Using a database URL and a Properties object:
A third form of the DriverManager.getConnection( ) method requires a database URL and a Properties object:
DriverManager.getConnection(String url, Properties info);
A Properties object holds a set of keyword-value pairs. It's used to pass driver properties to the driver during a call to the getConnection() method.
To make the same connection made by the previous examples, use the following code:

```
import java.util.*;
String URL = "jdbc:oracle:thin:@amrood:1521:EMP"; Properties info = new Properties( );
info.put( "user", "username" );
info.put( "password", "password" );
Connection conn = DriverManager.getConnection(URL, info);
```

Closing JDBC connections:

- At the end of your JDBC program, it is required explicitly close all the connections to the database to end each database session. However, if you forget, Java's garbage collector will close the connection when it cleans up stale objects.
- Relying on garbage collection, especially in database programming, is very poor programming practice. You should make a habit of always closing the connection with the close() method associated with connection object.
- To ensure that a connection is closed, you could provide a finally block in your code. A finally block always executes, regardless if an exception occurs or not.
- To close above opened connection you should call close() method as follows:
  conn.close();