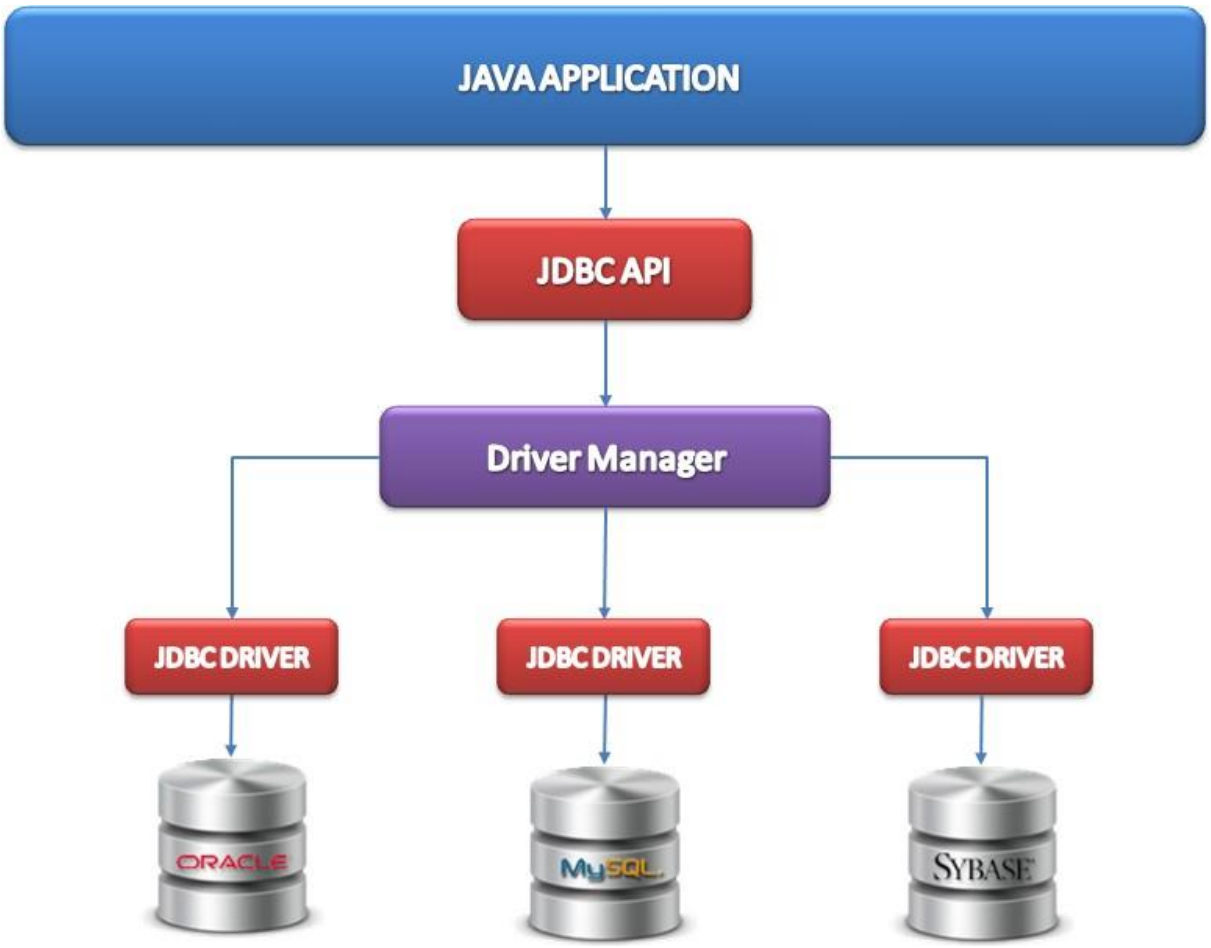
1. **Explain the architecture of JDBC architecture.**



- **Description:**

  a. **Application:** It is a java applet or a servlet which communicates with a data source.
  b. **The JDBC API:** The JDBC API allows Java programs to execute SQL statements and retrieve results. Some of the important classes and interfaces defined in JDBC API are as follows:
     - DriverManager
     - Driver
     - Connection
     - Statement
     - PreparedStatement

- CallableStatement
- ResultSet
- SQL data

c. **DriverManager:** It plays an important role in the JDBC architecture.It uses some database-specific drivers to effectively connect enterprise applications to databases.

d. **JDBC drivers:** To communicate with a data source through JDBC, you need a JDBC driver that intelligently communicates with the respective data source.

- **Types of Architectures:**

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

a. **Two-tier model:** A java application communicates directly to the data source. The JDBC driver enables the communication between the application and the data source. When a user sends a query to the data source, the answers for those queries are sent back to the user in the form of results.
The data source can be located on a different machine on a network to whicha user is connected. This is known as a client/server configuration, where the user's machine acts as a client and the machine having the data source running acts as the server.

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

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

2. What are JDBC drivers and explain the types of JDBC drivers?

JDBC Driver is a software component that enables java applications to interact withthe database. There are 4 types of JDBC drivers:

a. **JDBC-ODBC bridge driver:** The JDBC-ODBC bridge driver uses an ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin drivers.

Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

→ Advantages:
- easy to use.

- can be easily connected to any database.

→ Disadvantages:
- Performance degraded because JDBC method calls are converted intothe ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

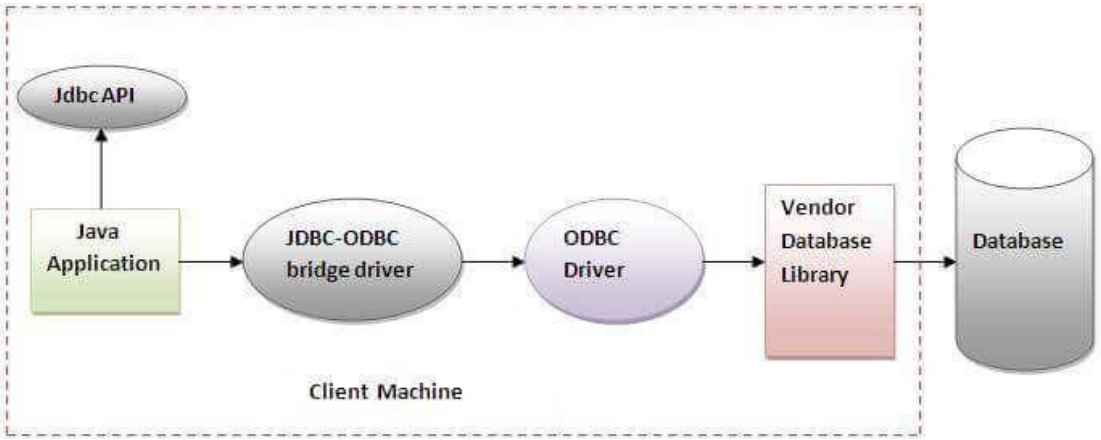


Fig: JDBC ODBC Driver

b. Native-API driver (partially java driver): The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

→ Advantage:
- performance upgraded than JDBC-ODBC bridge driver.

→ Disadvantage:
- The Native driver needs to be installed on each client machine.
- The Vendor client library needs to be installed on the client machine.

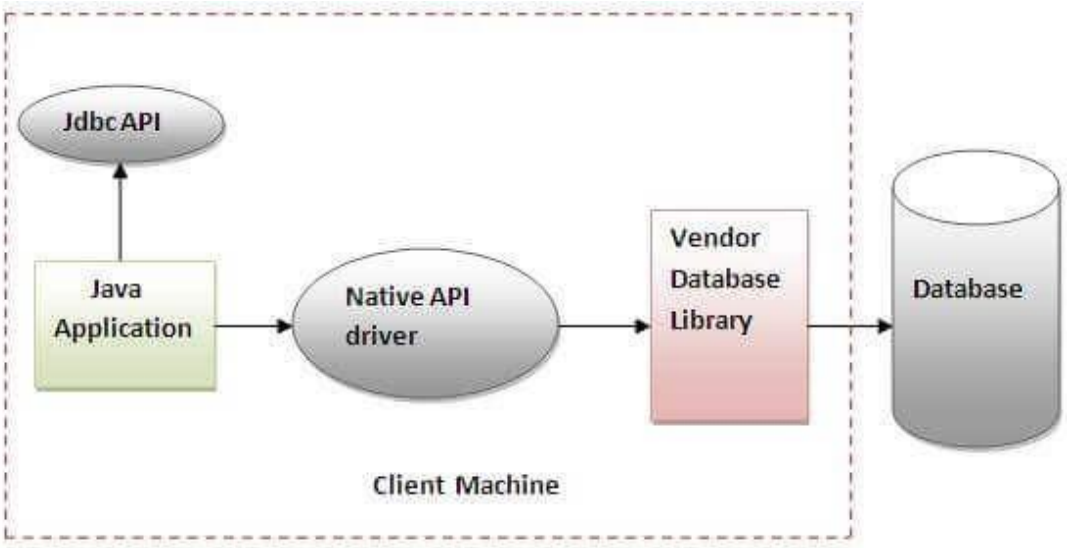


Fig: Native API Driver

c. **Network Protocol driver (fully java driver):** The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

→ Advantage:
- No client side library is required because of the application server thatcan perform many tasks like auditing, load balancing, logging etc.

→ Disadvantages:
- Network support is required on client machines.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

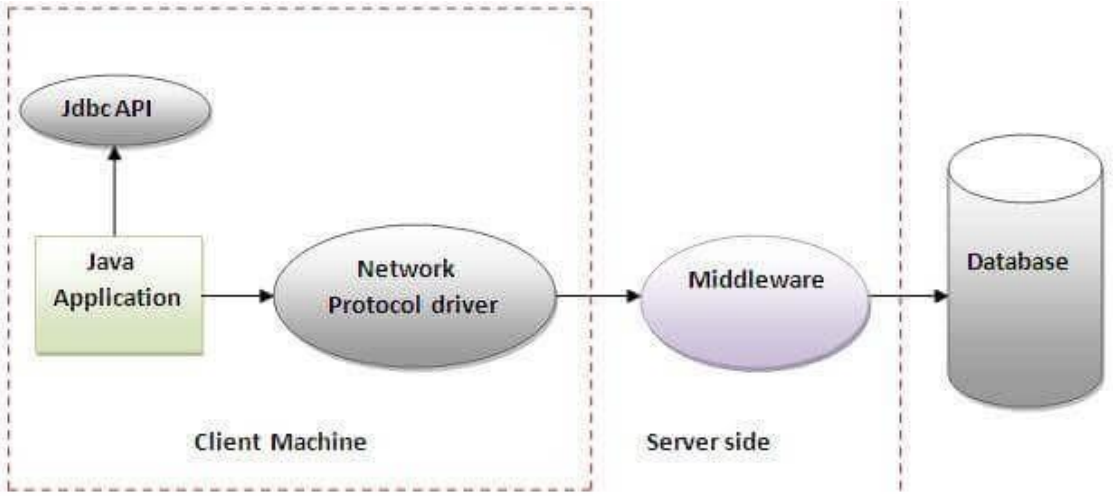


Fig: Network Protocol Driver

d. **Thin driver (fully java driver):** The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as a thin driver. It is fully written in Java language.

→ Advantage:
- Better performance than all other drivers.
- No software is required at client side or server side.

→ Disadvantage:
- Drivers depend on the Database.

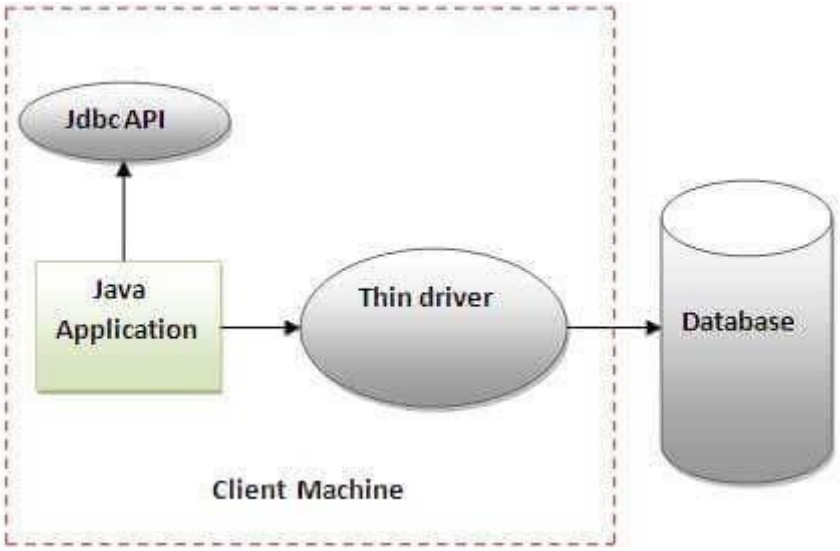


Fig: Thin Driver

3. **What is meant by ResultSet and what are methods associated with the ResultSet?**

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

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

The methods of the ResultSet interface can be broken down into three categories −

a. **Navigational methods** − There are several methods in the ResultSet interfacethat involve moving the cursor, including −

Method	Description
public void beforeFirst() throws SQLException	Moves the cursor just before the first row.
public void afterLast() throws SQLException	Moves the cursor just after the last row.
public boolean first() throws SQLException	Moves the cursor to the first row.
public void last() throws SQLException	Moves the cursor to the last row.
public boolean absolute(int row) throws SQLException	Moves the cursor to the specified row.

public boolean relative(int row) throws SQLException	Moves the cursor the given number of rows forward or backward, from where it is currently pointing.
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.
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.
public int getRow() throws SQLException	Returns the row number that the cursor is pointing to.
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. The current cursor location is remembered.
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

b. Get methods − Used to view the data in the columns of the current row being pointed by the cursor. The ResultSet interface contains dozens of methods for getting the data of the current row. There is a get method for each of the possible data types, and each get method has two versions −

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

For example, if the column you are interested in viewing contains an int, you need to use one of the getInt() methods of ResultSet −

Method	Description
public int getInt(String columnName) throws SQLException	Returns the int in the current row in the column named columnName.
public int getInt(int columnIndex) throws SQLException	Returns the int in the current row in the specified column index. The column index starts at 1.

Similarly, there are get methods in the ResultSet interface for each of the eight Java primitive types, as well as common types such as java.lang.String, java.lang.Object, and java.net.URL.

There are also methods for getting SQL data types java.sql.Date, java.sql.Time, java.sql.TimeStamp, java.sql.Clob, and java.sql.Blob. Check the documentation for more information about using these SQL data types.

c. **Update methods** − Used to update the data in the columns of the current row.The updates can then be updated in the underlying database as well. The ResultSet interface contains a collection of update methods for updating the data of a result set. As with the get 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 would use one of the following updateString() methods −

Method	Description
public void updateString(int columnIndex,String s) throws SQLException	Changes the String in the specified column to the value of s.
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.

There are update methods for the eight primitive data types, as well as String, Object, URL, and the SQL data types in the java.sql package.

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.

Method	Description
public void updateRow()	Updates the current row by updating the corresponding rowin the database.
public void deleteRow()	Deletes the current row from the database
public void refreshRow()	Refreshes the data in the result set to reflect any recent changes in the database.
public void cancelRowUpdates()	Cancels any updates made on the current row.
public void insertRow()	Inserts a row into the database. This method can only beinvoked when the cursor is pointing to the insert row.

4. Explain the steps involved in JDBC connection.

There are 5 steps to connect any java application with the database using JDBC.These steps are as follows:

a. Register the Driver class: The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

→ Syntax of forName() method:
public static void forName(String className)throws ClassNotFoundException

→ Example:
Class.forName("oracle.jdbc.driver.OracleDriver");

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

→ Syntax of getConnection() method:
public static Connection getConnection(String url)throws SQLException
public static Connection getConnection(String url, String name, String password) throws SQLException

→ Example:
```
Connection con = DriverManager.getConnection(
        "jdbc:oracle:thin:@localhost:1521:xe", "system", "password");
```

c. Create statement
The createStatement() method of Connection interface is used to create statements. The object of statement is responsible to execute queries with the database.
Syntax of createStatement() method

→ Syntax of createStatement() method:
public Statement createStatement()throws SQLException

→ Example:
Statement stmt=con.createStatement();

d. Execute queries

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

→ Syntax of executeQuery() method:

public ResultSet executeQuery(String sql)throws SQLException

→ Example:

```
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next()){
    System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

e. Close connection

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

→ Syntax of close() method:

public void close()throws SQLException

→ Example:

```
con.close();
```