Database Access

Java DataBase Connectivity [JDBC]

Overview

- Overview of JDBC technology
- JDBC Driver Types
- Seven basic steps in using JDBC
- Using Meta Data
- Using Statement
- Useful Statement methods
- Using prepared and callable statements
- Submitting multiple statements as a transaction

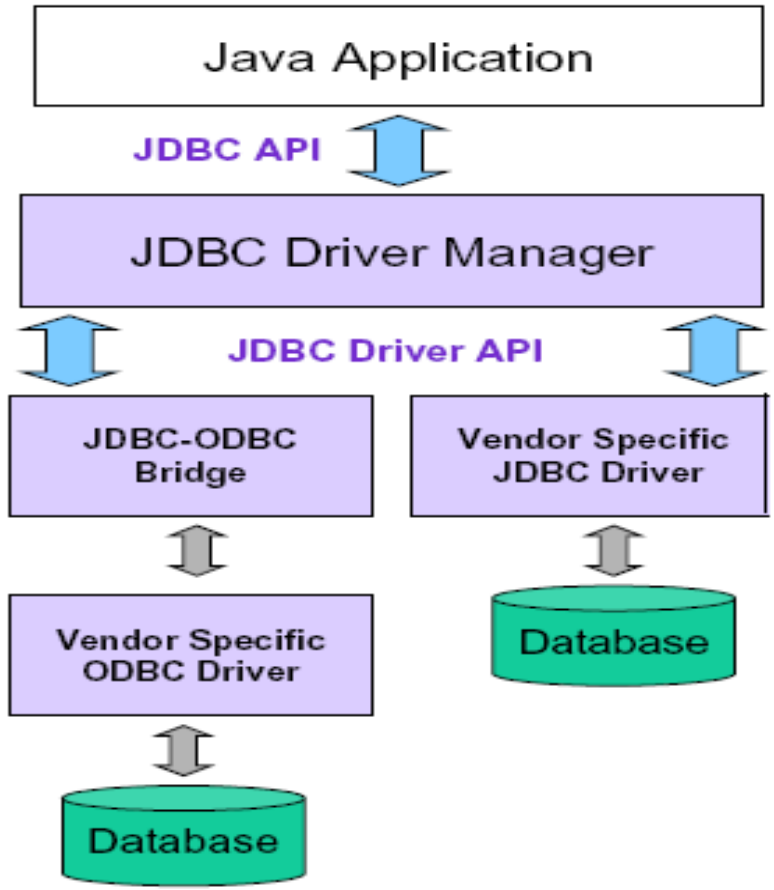
JDBC Introduction

- JDBC provides a standard library (API) for accessing relational databases
  - API standardizes
    - Way to establish connection to database
    - Approach to initiating queries
    - Method to create stored (parameterized) queries
    - The data structure of query result(table)
      - Determining the number of columns
      - Looking up metadata etc..
  - API does not standardize SQL syntax
    - JDBC is not embedded SQL
  - JDBC classes are in **java.sql** package

JDBC Continued..

JDBC Consists of 2 parts

- JDBC API – purely java based API
- JDBC Driver Manager – communicates with vendor specific drivers that perform real communication with database
  - Translation to vendor format is performed on client side
  - Driver (translator) needed on client side



JDBC Driver Types

- **JDBC drivers** are divided into four types or levels
  - **Type 1:** JDBC-ODBC Bridge driver (Bridge)
  - **Type 2:** Native-API/partly Java driver (Native)
  - **Type 3:** All Java/Net-protocol driver (Middleware)
  - **Type 4:** All Java/Native-protocol driver (Pure)

Type 1 JDBC Driver

JDBC-ODBC Bridge driver

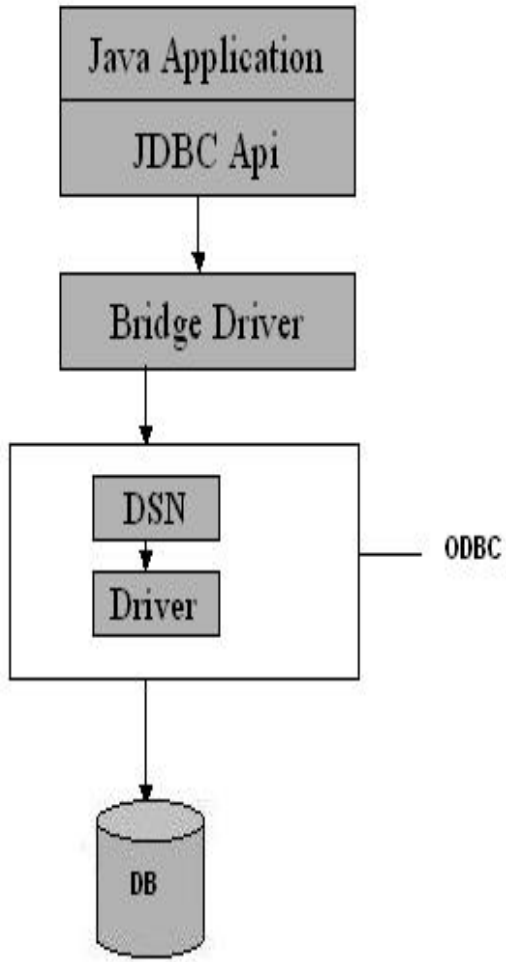
- Translates all JDBC calls into ODBC calls and sends them to the ODBC driver
- ODBC is a generic API
- recommended only for experimental use or when no other alternative is available

Advantages

- allows access to almost any database, since the database's ODBC drivers are already available

Disadvantages

- Since the Bridge driver is not written fully in Java, Type 1 drivers are not portable.
- A performance issue is seen as a JDBC call goes through the bridge to the ODBC driver, then to the database, and this applies even in the reverse process. They are the slowest of all driver types.
- The client system requires the ODBC Installation to use the driver.
- 4. Not good for the Web.



Type 2 JDBC Driver

Native-API/partly Java driver

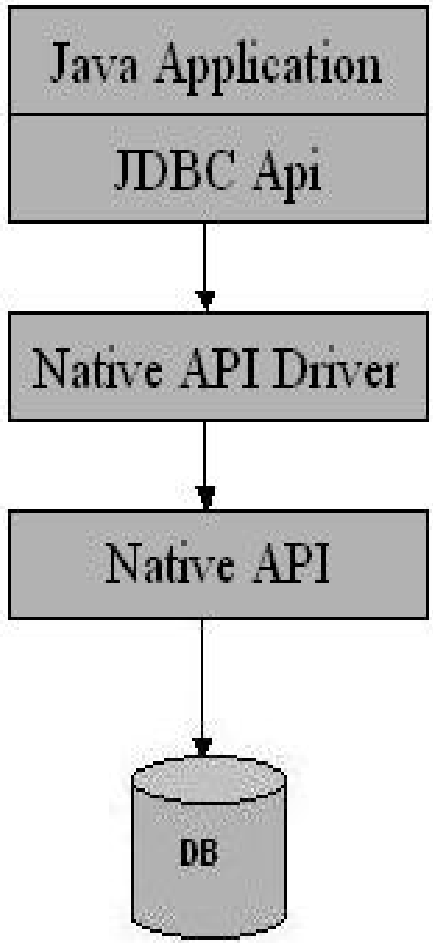
- convert JDBC calls into database-specific calls
  i.e. this driver is specific to a particular database
- Example: Oracle will have oracle native api.

Advantages

- offer better performance than the JDBC-ODBC Bridge as the layers of communication (tiers) are less than that of Type1 Drivers
- uses Native api which is Database specific

Disadvantages

- Native API must be installed in the Client System and hence type 2 drivers cannot be used for the Internet.
- Like Type 1 drivers, it's not written in Java Language which forms a portability issue.
- If we change the Database we have to change the native api as it is specific to a database
- Mostly obsolete now



Type 3 JDBC Driver

All Java/Net-protocol driver

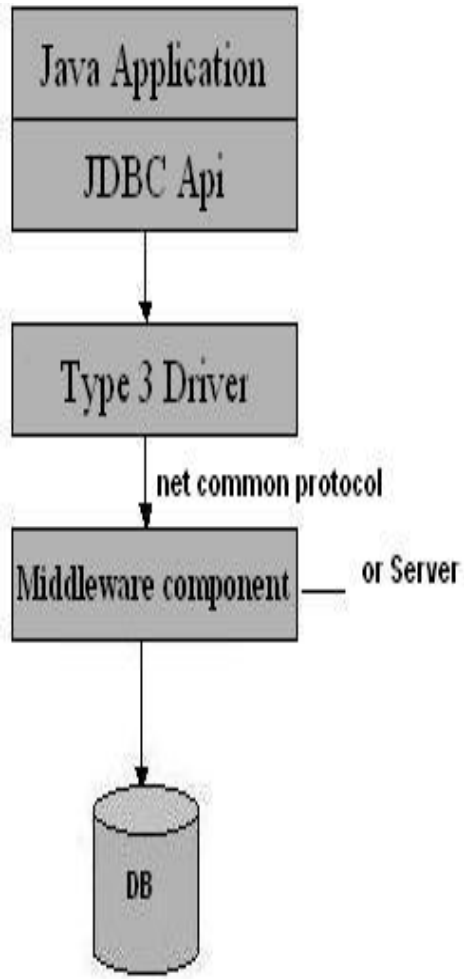
- database requests are passed through the network to the middle-tier server
- middle-tier then translates the request to the database
- middle-tier server can in turn use Type1, Type 2 or Type 4 drivers

Advantages

- server-based, so there is no need for any vendor database library to be present on client machines.
- fully written in Java and hence Portable. It is suitable for the web.
- The type 3 driver typically provides support for features such as caching (connections, query results etc), load balancing, and advanced system administration such as logging and auditing.
- very flexible allows access to multiple databases using one driver.
- They are the most efficient amongst all driver types.

Disadvantages

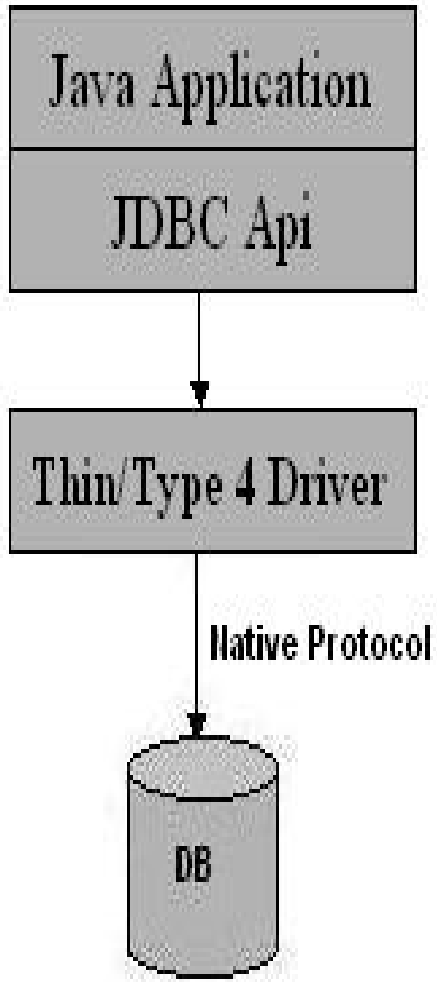
- It requires another server application to install and maintain.
- Traversing the recordset may take longer, since the data comes through the backend server.



Type 4 JDBC Driver

Native-protocol/all-Java driver

- uses java networking libraries to communicate directly with the database server.



Advantages

- The major benefit of using a type 4 jdbc drivers are that they are completely written in Java to achieve platform independence and eliminate deployment administration issues. It is most suitable for the web.
- Number of translation layers is very less i.e. type 4 JDBC drivers don't have to translate database requests to ODBC or a native connectivity interface or to pass the request on to another server, performance is typically quite good.
- You don’t need to install special software on the client or server. Further, these drivers can be downloaded dynamically.

Disadvantages

- the user needs a different driver for each database .

7 basic steps in using JDBC

1. Load the driver
2. Define the Connection URL
3. Establish the Connection
4. Create a Statement object
5. Execute a query
6. Process the results
7. Close the connection

Using MetaData

- System-wide data
  - connection.getMetaData().getDatabaseProductName();
  - connection.getMetaData().getDatabaseProductVersion();
- Table-specific data
  - resultSet.getMetaData().getColumnCount();
    - When using the result, the index starts at 1 not 0.
  - resultSet.getMetaData().getColumnName();

Using Statement

- **`Statement`** object is used for sending SQL statements to the database
- Three types of **`Statement`** objects are available
  - `Statement`
    - for executing a simple SQL statement
  - `PreparedStatement`
    - For executing precompiled SQL statement passing in parameters
  - `CallableStatement`
    - For executing database stored procedure

Useful Statement Methods

- executeQuery
  - Executes the SQL query and returns the data in a table in the form of a ResultSet object
- executeUpdate
  - Used to execute for INSERT, UPDATE or DELETE SQL statements.
  - The return value is number of rows that were affected in database
  - Supports Data Definition Language (DDL) statements [CREATE TABLE, DROP TABLE and ALTER TABLE]
- execute
  - Generic method for executing stored procedures and prepared statements
  - The statement execution may or may not return a ResultSet

Useful Statement Methods…..

- getMaxRows()/setMaxRows()
  - Determines the maximum number of rows a ResultSet may contain
  - Unless explicitly set, the number of rows in unlimited [return value of 0]
- getQueryTimeout()/setQueryTimeout()
  - Specifies the amount of time a driver will wait for a STATEMENT to complete before throwing a SQLException

Prepared Statements (Precompiled Queries)

- Idea behind Prepared Statements
  - If you are going to execute similar SQL statements multiple times, using “prepared” [or ‘parameterized’] statements can be more efficient
  - Create a statement in standard for that is sent to the database for compilation before actually being used
  - Each time you use it, you simply replace some of the marked parameters using setXxx methods
- The following execute methods of prepared statement have no parameters
  - execute()
  - executeQuery()
  - executeUpdate()
- Other useful prepared statement methods
  - setXxx – sets the indicated parameter (?) in the SQL statement to value
  - clearParameters – clears all set parameter values in the statement