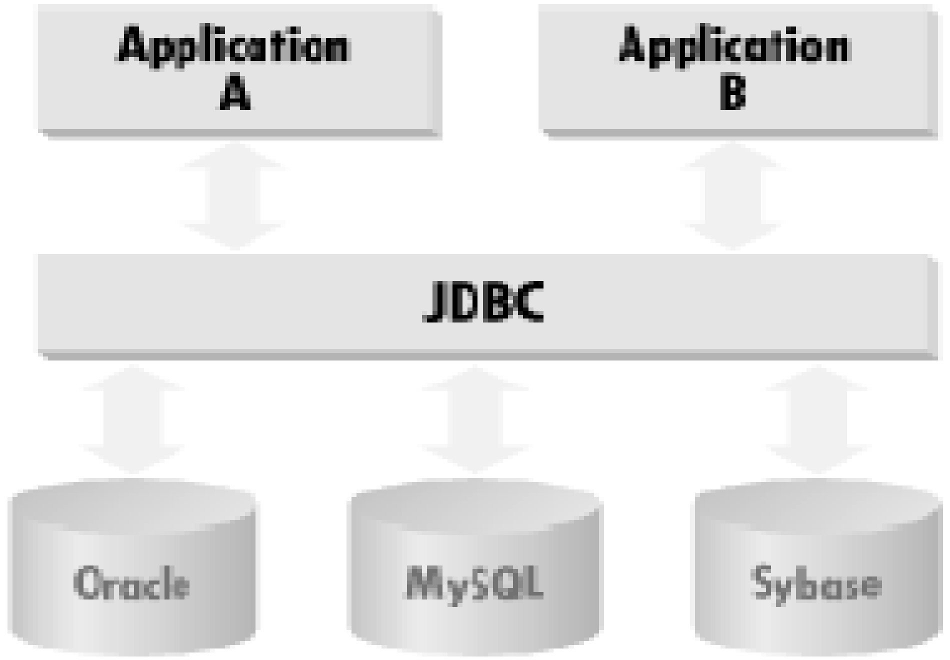
- Chapter 9

Java Database Connectivity

Overview

- JDBC provides a set of interfaces & classes that create a common point at which database applications can talk to database engines
- The set of classes that implement the JDBC interfaces for a particular database engine is called a JDBC driver
- the whole point of JDBC is to hide the specifics implementation of each database



Overview

- There are three major steps to getting access to a database using JDBC
  - Load the database drivers that you need
  - Make the connection to each driver, creating the Connection object
  - Create the Statement object linked to the right Connection object; this object can contain the SQL command to be executed and can be used repeatedly for each desired command.
- Many of the JDBC API concepts are taken from other sources, particularly Microsoft's ODBC (Open Database Connectivity)

Database URL

- The Database URL is the string used by JDBC to connect to the database.
- The general syntax is
  jdbc:subprotocol name:other_stuff
- The subprotocol name is the specific driver used by JDBC to connect to the database, e.g. odbc if the jdbcodbc bridge driver is being used.
- Other_stuff depends on which specific driver is being used, e.g. for odbc it would be the Data Source Name specified in the ODBC Administrator program

Database Driver

- ensures that the application interacts with all databases in a standard and uniform manner
- ensures that the requests made by the application are presented to the database in a language understood by the database
- receives the requests from the client, converts it into the format understandable by the database and then presents it to the database
- receives the response, translates it back to Java data format and presents it to the client application
- All databases follow SQL and hence there is only one JDBC Driver as the Java-to-SQL translator

Loading Driver

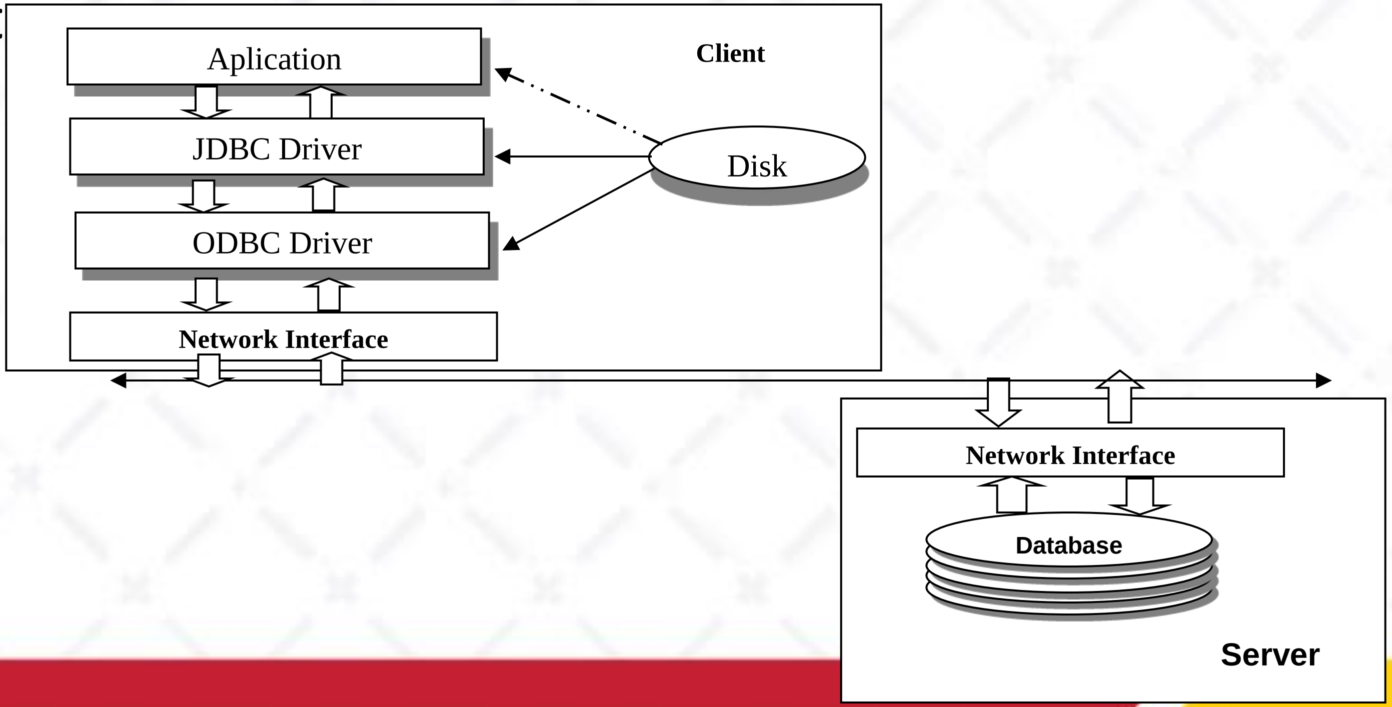
- The DriverManager is the class responsible for loading database drivers and creating a new database connection
- You need to force the loading of the required driver manager code by: ***Class.forName(String);***
  - This returns the object associated with the class with the given string name.
  - Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
- Equivalent to:
  new sun.jdbc.odbc.JdbcOdbcDriver();
  or
  DriverManager.registerDriver(new sun.jdbc.odbc.JdbcOdbcDriver());
- If you have a driver from another vendor, then find out the class name of that driver and load it instead.

JDBC Driver

- There are 4 types of JDBC Driver
  - Type 1: JDBC/ODBC
  - Type 2: Native-API
  - Type 3: Open Protocol-Net
  - Type 4: Proprietary-Protocol-Net
- Types 2 to 4 generally have to be written by the Database vendors. They are more efficient than Type 1, however, performance varies greatly.

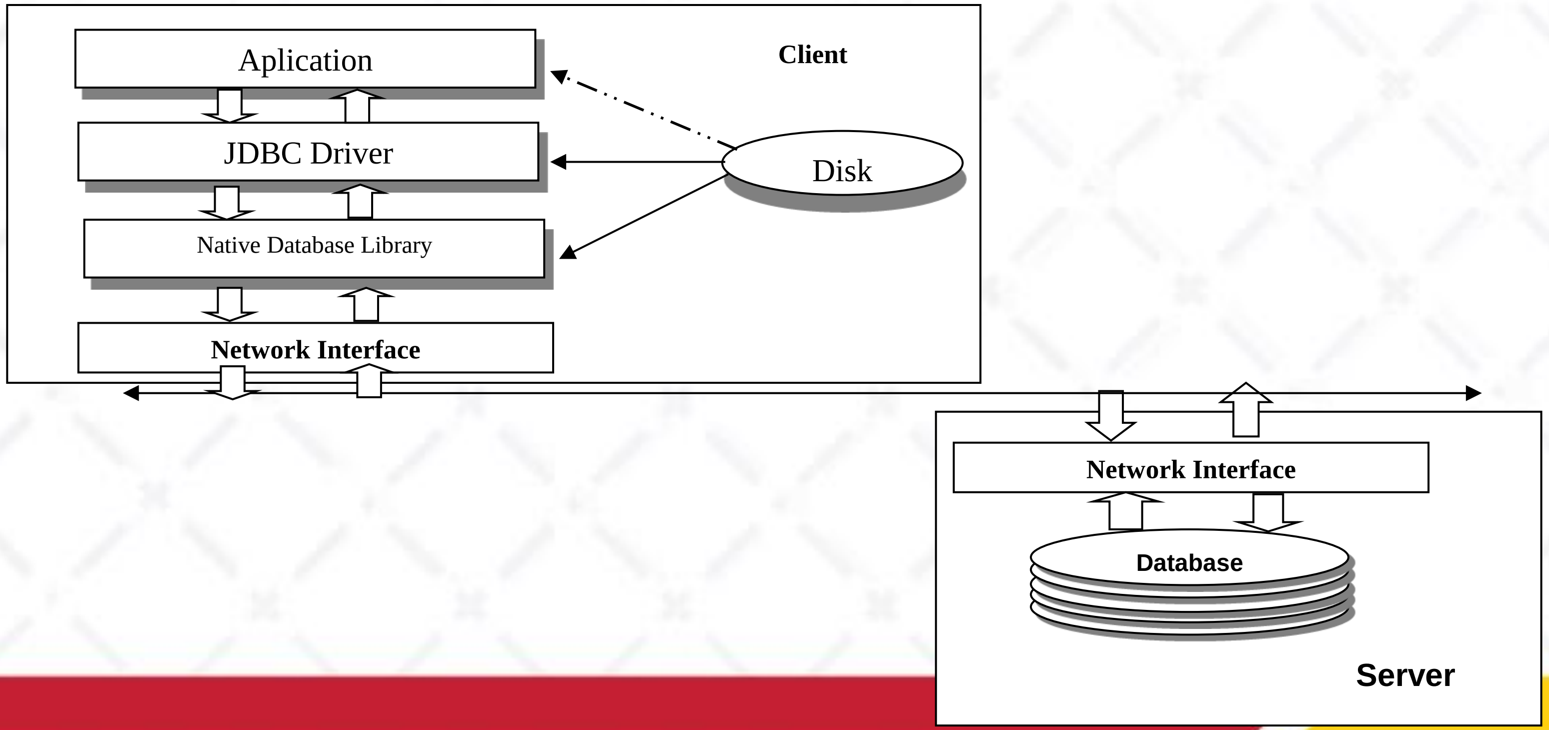
Type I JDBC/ODBC

- jdk supplies the jdbc -odbc bridge.
- Any database with an ODBC driver can be accessed.
- Very flexible, but not efficient. Can connect to many databases this way and gives a lot of portabilit



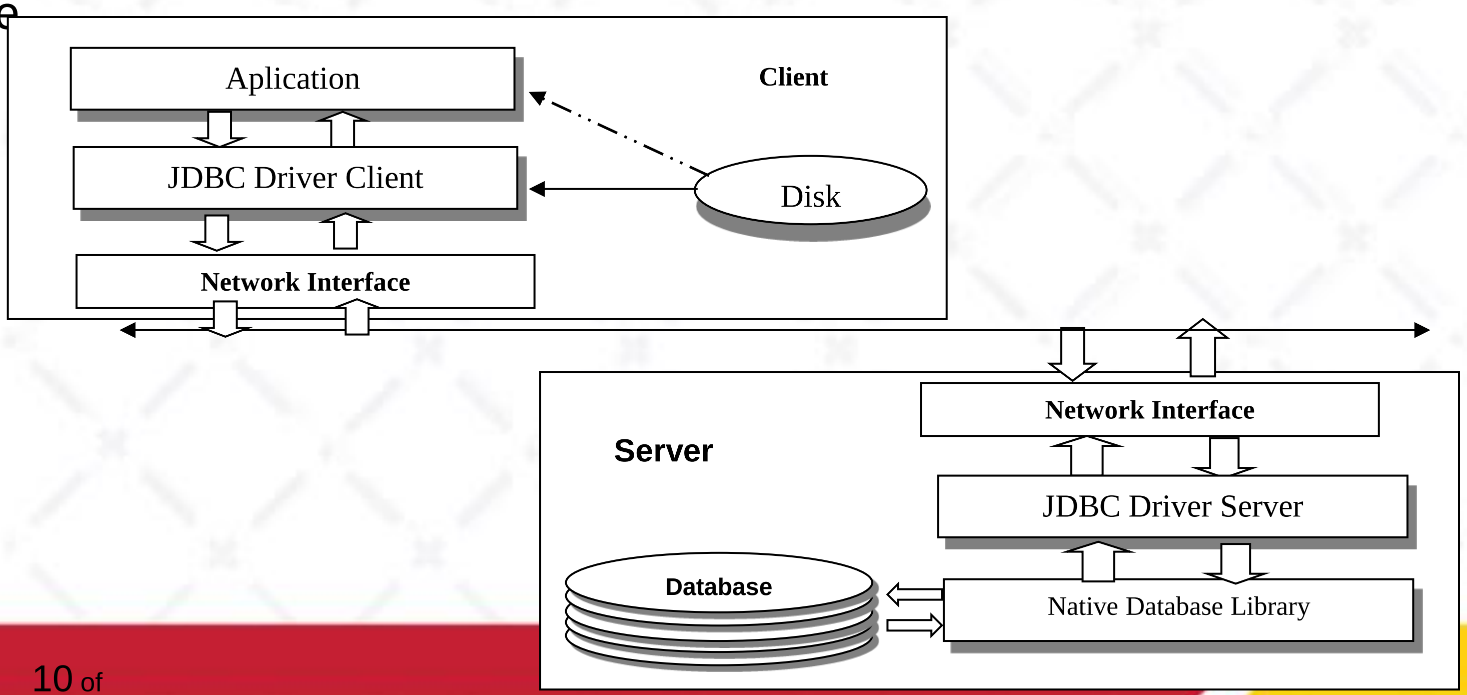
Type 2: Native-API

- Rather than using ODBC Drivers, this option enables the JDBC driver to interface directly with the Vendor specific driver or database API such as Oracle’s OCI (Oracle Call Interface)



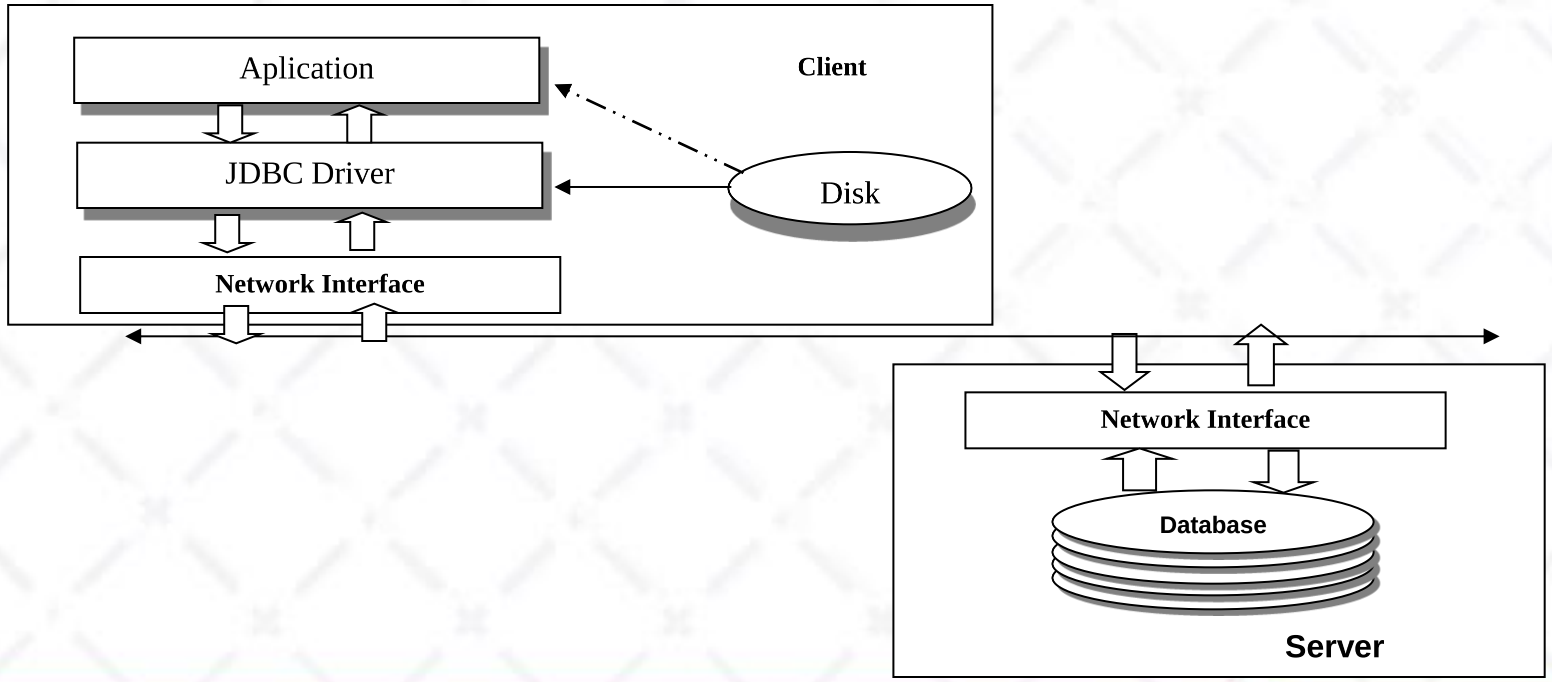
Type 3: Open Protocol-Net

- Drivers
  - can forward database requests to a remote data source
  - can interface to multiple database types
  - are not vendor specific
  - all Java code



Type 4: Proprietary-Protocol Net

- 100% java
- Capable of communicate directly to DBMS without any translation



Java.sql package

- contains a set of interfaces and classes defined by JDBC API that are used for communicating with the database
- Classes
  - DriverManager
  - Date, Time
  - Timestamp
  - Types
- Interfaces of **java.sql** package include:
  - Driver
  - Connection
  - DatabaseMetaData
  - Statement
  - PreparedStatement
  - CallableStatement
  - ResultSet
  - ResultSetMetaData

The Statement object

- the Connection object is a direct link to the database.
- Use the Connection object to create the Statement object
  - Statement s = con.createStatement();
- You do not actually assign SQL to the Statement until you are ready to send the SQL to the database by the **executeQuery(String)** or **executeUpdate(String)** method
- JDBC uses a different method for sending queries than for sending updates.The key difference is the fact that the method for queries returns an instance of ResultSet, while the method for nonqueries returns an integer.
- The same Statement object can be used for many different, unrelated queries.

The Statement object

- There are three statement execution methods in the Statement class.
  - executeQuery()
  - executeUpdate()
  - execute()
- The executeQuery()
  - takes a SQL String as an argument
  - returns a ResultSet object.
  - This method should be used for any SQL calls that expect to return data from the database.
- ResultSet rs = s.executeQuery(“SELECT * FROM Books”);

The Statement object

- The executeUpdate() method
  - Used for update statements,
  - returns the number of affected rows by the UPDATE, INSERT, or DELETE.
  - when executeUpdate return 0 , it can mean one of two things:
    - (1) the update statement affected zero rows, or
    - (2) the statement executed was a DDL statement.
- The execute() method
  - for situations in which you do not know whether the SQL being executed is a query or update.
  - This usually happens when the application is executing dynamically created SQL statements.
  - If the statement returns a row from the database, the method returns true. Otherwise it returns false. The application can then use the getResultSet() method to get the returned row.

ResultSet

- A ResultSet is one or more rows of data returned by a database query.
- The ResultSet class handles only a single row from the database at any given time,
- Using the the next( ) method for making it reference the next row of a result set.
  - If next() returns true, you have another row to process and any subsequent calls you make to the ResultSet object will be in reference to that next row.
  - If there are no rows left, it returns false.
- To examine a ResultSet object, an iterative loop can be set up as follows:

```
while (rs.next()){
// examine a row from the results
}
```

ResultSet

- A ResultSet object is initially positioned before its first row. Execution of next() advances the cursor to point to the next row
  - There are no other ways to move around a ResultSet object with JDBC1.0 - note this inflexibility.
- The class provides a series of methods for retrieving columns from the results of a database query. The methods for getting a column all take the form:
  - type get type(int | String)
    - int columnNumber | String columnName
    - type can be int, double, String, Date, etc.
  - These methods return the value of the column with a given column number or name.
  - String isbn = rs.getString(1); // Column 1
  - float price = rs.getDouble(“Price”);

ResultSet Metadata

- Program can also makes use of the ResultSetMetadata interface.
- From this an object can be used to find out about the types and properties of the columns in a ResultSet.
- Its is used to find out sizing details in the data retrieved from different table and to resize the window to fit the data sizes
- ***ResultSet rs = stmt.executeQuery(SQLString);***
  ResultSetMetaData rsmd = rs.getMetaData();
  int numberOfColumns = rsmd.getColumnCount();
- ***getColumnName(int column)***

Prepared Statements

- To execute a Statement object many times, it will reduce execution time to use PreparedStatement object
- PreparedStatement object
  - unlike a Statement object, it is given an SQL statement when it is created.
  - The advantage to this is that in most cases, this SQL statement will be sent to the DBMS right away, where it will be compiled.
  - As a result, the PreparedStatement object contains not just an SQL statement, but an SQL statement that has been precompiled.
  - This means that when the PreparedStatement is executed, the DBMS can just run the PreparedStatement 's SQL statement without having to compile it first

Prepared Statements

- PreparedStatement can be used for SQL statements that take parameters.
  - you can use the same statement and supply it with different values each time you execute it.
- PreparedStatement updateAddr = con.prepareStatement(
  "UPDATE Customers SET Address = ? WHERE CustNo= ?");
- You need to supply values to be used in place of the question mark placeholders,
  - do this by calling one of the setXXX methods defined in the class PreparedStatement
    updateAddr.setString(1, “Danang”);
    updateAddr.setInt(2,1001);
- Once a parameter has been set with a value, it will retain that value until it is reset to another value or the method clearParameters is called

Callable Statement

- A CallableStatement object provides a way to call stored procedures in a standard way for all RDBMSs.
- A stored procedure is stored in a database;
- The syntax for invoking a stored procedure in JDBC
  - **{Call procedure_name(arg1, arg2, ...)}**
  - **{?= call procedure-name arg1,arg2, ...}**
- or using DBMS syntax as in SQL Server (not recommend)
  - **Exec procedure_name arg1, arg2,..**
- Both forms may have a variable number of parameters used for input (IN parameters), output (OUT parameters), or both (INOUT parameters).
  - A question mark serves as a placeholder for a parameter

Creating a CallableStatement Object

- CallableStatement
  - inherits Statement methods, which deal with SQL statements in general,
  - it also inherits PreparedStatement methods, which deal with IN parameters.
  - All of the methods defined in CallableStatement deal with OUT parameters.
- CallableStatement objects are created with the Connection method prepareCall

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

  CallableStatement cstmt2=con.prepareCall("Exec sp_Acc ?,?,?,?");
- Whether the ? placeholders are IN, OUT, or INOUT parameters depends on the stored procedure

Passing IN parameter

- Passing in any IN parameter values to a CallableStatement object is done using the setXXX methods inherited from PreparedStatement.
- The type of the value being passed in determines which setXXX method to use
  - setFloat(1,var1) to pass in a float value,
  - setBoolean(2,var2) to pass in a boolean, and so on.
- The parameter number does not refer to literal parameters that might be supplied to a stored procedure call.
- Of the programs that use parameters, the vast majority use only IN parameters

Register OUT Parameter

- If the stored procedure returns OUT parameters, the JDBC type of each OUT parameter must be registered before the CallableStatement object can be executed.
- This is necessary because some DBMSs require the SQL type (which the JDBC type represents), not because JDBC requires it.
- Registering the JDBC type is done with the method registerOutParameter.
    - registerOutParameter uses a JDBC type (so that it matches the data type that the database will return)
    - getXXX casts this to a Java type.
- After the statement has been executed, getXXX() methods can be used to retrieve OUT parameter values.

INOUT parameters

- an INOUT parameter requires
  - a call to the appropriate setXXX method (inherited from PreparedStatement)
  - a call to the method registerOutParameter.

  stmt1.setString(1,"00000");

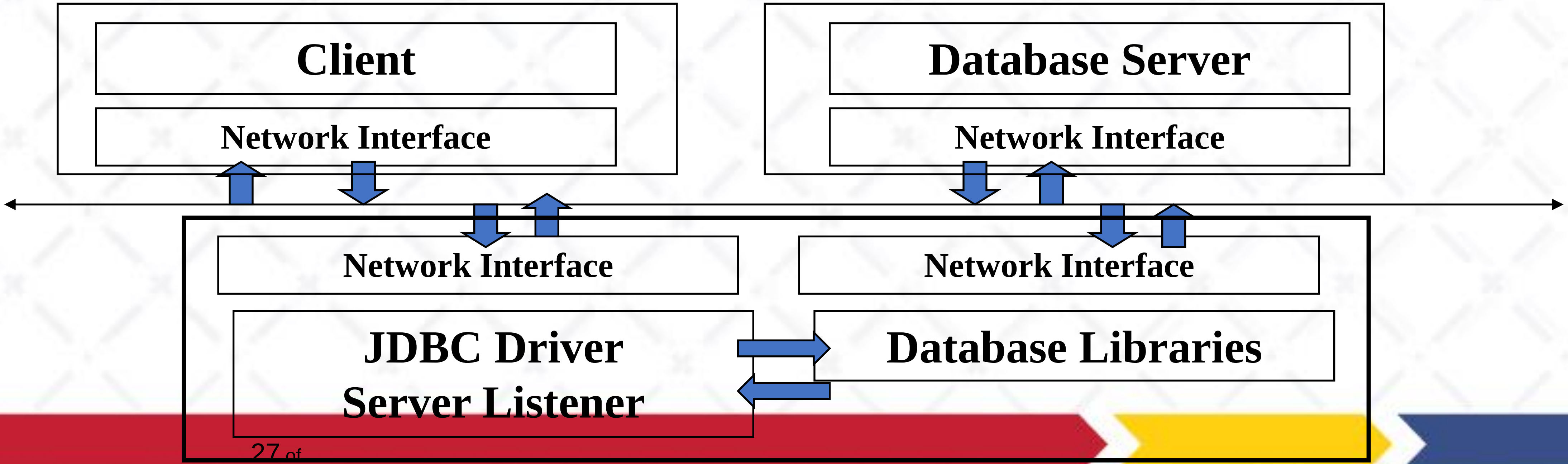
  stmt1.registerOutParameter(1,Types.VARCHAR);
- to retrieve the output value, a corresponding getXXX method is used.
  - values from OUT parameters can be retrieved (using CallableStatement.getXXX methods).
  - the only way to know if a value of 0 or false was originally JDBC NULL is to test it with the method wasNull(),
- In practice, stored procedure is not approriate in a heterogeneous and distributed enviroment as it couple very closely an application with a specific database.

JDBC Application Design Consideration

- Two-Tier Client-Server Model
  - Advantage
    - The most simplest to implement and management
    - Maintain a persistent connection between client and database server
    - Faster than three-tier implementation
  - Disadvantage
    - Requires the native libraries be loaded on client machine
    - Local configuration must be maintain as required by some driver
    - Applets can only open up connections to the server from which they were downloaded

JDBC Application Design Consideration

- Three-Tier Client-Server Model
  - A third server is employed to handle request from client and pass them off to database server
  - The third server acts as proxy for all client request
  - Eliminate the need for RDBMS to be located on the same server as Web server



JDBC Application Design Consideration

- Three-Tier Client-Server Model
  - Advantage
    - Client do not need to have native libraries loaded locally
    - Drivers can be managed centrally
    - Database Server is not visible to the InterNet
  - Disadvantage
    - Client does not maintain a persistent connection
    - May require a separate proxy server
    - Increase network trafic

Transaction management

Autocommit mode off

- By default, JDBC commits each SQL statement as it is sent to the database; this is called autocommit.
- For more robust error handling, you can set up a Connection that changes have no effect on the database until you expressly send a commit.
  - Each Connection is separate, and a commit on one has no effect on the statements on another.
- The Connection class provides the setAutoCommit( ) method for turning on/off auto-commit
- con.setAutoCommit(false);
  s = con.createStatement();
- s.executeUpdate(SQLString)
  con.commit( ); or rollback();

Database Metadata

- One of the very powerful interfaces is DatabaseMetadata
- Provide comprehensive information about the database as a whole.
- Many of the methods here return lists of information in the form of ResultSet objects.
  - can use the normal ResultSet methods such as getString and getInt to retrieve the data from these ResultSets.
  - If a given form of metadata is not available, these methods should throw an SQLException.
- The metadata enables you to create generalized programs which can accurately connect and process data from different types of relational databases and deal uniformly with different and multiple structures

JDBC 2.0 Enhancements

- Two parts
  - Core Java feature enhancements. In java.sql package
  - Extended features. In javax.sql package. Extra download
- Enhancements
  - Scrollable Result Sets(in core API)
  - Batch Updates (in core API)
  - Advanced Data Types (in core API)
  - JNDI for connecting to a database (in extension API)
  - Connection pooling (in extension API)
  - Rowsets (an additional Result type) (in extension API)
  - Distributed transaction support (in extension API)

Scrollable Result Sets

- ResultSet: three types of result sets
  - methods for moving the cursor to a particular row or to a relative position (either forward or backward)
  - previous(), beforeFirst( ), first( ), last( ), absolute( ), and relative( ).
  - methods for ascertaining the current position of the cursor
  - constants indicating the scrollability of a result set
- Connection
  - new versions of the methods for creating Statement, PreparedStatement, and CallableStatement objects that make the result sets they produce scrollable
  - con.createStatement(ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY);
- DatabaseMetaData
  - method indicating whether the DBMS and driver support scrollable result sets

Batch Updates

- The new batch update facility provides the ability to send multiple updates to the database to be executed as a batch rather than sending each update separately
  - Statement, PreparedStatement, and CallableStatement have methods for adding update statements to a batch, clearing all update statements, and executing a batch .
- Statement s = connection.createStatement();
  s.addBatch(“INSERT …. “);
  s.addBatch(“INSERT …. “);
  s.executeBatch();
  - can queue up SQL and execute in one call, improving efficiency
- BatchUpdateException:
  - thrown exception when an error occurs in a batch update

Programmatic Updates

- Programmatic updates provide the ability to make updates using the JDBC API rather than SQL statements. The following interfaces have new methods and constants that support programmatic updates
- ResultSet
  - an updateXXX method for updating each data type
  - methods for inserting, deleting, or updating a row
  - methods indicating a row was inserted, deleted, or updated
  - method for cancelling a row update
  - constants indicating the updatability of a result set
- DatabaseMetaData have methods
  - indicating the visibility of changes to a result set
  - indicating whether a result set detects inserts, deletes, or updates
  - indicating whether the DBMS and driver support updatable result sets