Introducing JDBC

✓ According to Sun, JDBC is not an acronym, but is commonly misinterpreted to mean Java DataBase Connectivity
✓ Supports ANSI SQL 92 Entry Level
The Standard Query Language (SQL)

✓ Composed of two categories:
  – Data Manipulation Language (DML)
    • used to manipulate the data
      – select
      – delete
      – update
  – Data Definition Language (DDL)
    • create database
    • create table
    • drop database
Data Manipulation Language

- SELECT - query the database
  - select * from customer where id > 1001

- INSERT - adds new rows to a table.
  - Insert into customer values (1009, ‘John Doe’)

- DELETE - removes a specified row
  - delete

- UPDATE - modifies an existing row
  - update customers set amount = 10 where id > 1003
Data Definition Language

✔ CREATE DATABASE - allows you to create a database
✔ CREATE TABLE - allows you to create a table definition in a database
✔ DROP TABLE - removes a table from a database
✔ ALTER TABLE - modifies the definition of a table in a database
JDBC Framework

✓ The JDBC driver manager
✓ The JDBC driver
The JDBC Driver Manager

- Management layer of JDBC, interfaces between the client and the driver.
- Keeps a hash list of available drivers
- Manages driver login time limits and printing of log and tracing messages
- Secure because manager will only allow drivers that come from local file system or the same initial class loader requesting a connection
- Most popular function:
  - `Connection getConnection(url, id, passwd);`
JDBC Driver Types

- Type 1 (JDBC-ODBC Bridge Technology)
- Type 2 (JNI drivers for C/C++ connection libraries)
- Type 3 (Socket-level Middleware Translator)
- Type 4 (Pure Java-DBMS driver)
speed to small degree
Translation Layer compromises execution

Client
ODBC binary code must exist on every

the DBMS
redirects ODBC calls to an ODBC driver on

JDBC driver translates call into ODBC and

JDBC-ODBC Bridges

Type 1 Drivers
Type 2 Drivers
Native-API + Java Driver

- Java driver makes JNI calls on the client API (usually written in C or C++)
  - eg: Sybase dblib or ctlib
  - eg: Oracle Call Interface libraries (OCI)
- Requires client-side code to be installed
- Often the fastest solution available
- Native drivers are usually delivered by DBMS vendor
- Bug in driver can crash JVMs
- Example: JDBC=>Sybase dblib or ctlib
Type 3 Drivers
JDBC-Middleware Pure Java Driver

✔ JDBC driver translates JDBC calls into a DBMS-independent protocol
✔ Then, communicates over a socket with a middleware server that translates Java code into native API DBMS calls
✔ No client code need be installed
✔ Single driver provides access to multiple DBMSs, eg. WebLogic Tengah drivers
✔ Type 3 drivers auto-download for applets.
Type 4 Drivers
Pure Java Drivers

✔ Java drivers talk directly to the DBMS using Java sockets
✔ No Middleware layer needed, access is direct.
✔ Simplest solution available.
✔ No client code need be installed.
✔ Example: JConnect for Sybase
✔ Type 4 drivers auto-download for applets
Result Sets and Cursors

- Result Sets are returned from queries.
- Number of rows in a RS can be zero, one, or more.
- Cursors are iterators that iterate through a result set.
- JDBC 2.0 allows for backward as well as forward cursors, including the ability to go to a specific row or a relative row.
A JDBC Primer

✔ First, load the JDBC Driver:

- call new to load the driver’s implementation of Driver class (redundant--
  Class.forName does this for you automatically) and call
  DriverManager.RegisterDriver()
- add driver to the jdbc.drivers property - DriverManager will load these
  automatically
  
  • eg: ~/.hotjava/properties:
    - jdbc.drivers=com.oracle.jdbc.OracleDriver:etc;
  
  • or programatically:
    - String old = sysProps.getProperty("jdbc.drivers");
    - drivers.append(":" + oldDrivers);
    - sysProps.put("jdbc.drivers", drivers.toString());
- call Class.forName and pass it the classname for the driver
  implementation
Create a Connection to the database via the driver

- Call the getConnection method on the DriverManager object.
- Connection conn = DriverManager.getConnection(url, login, password)
- url: jdbc:subprotocol:host:port[/database]
  - registered subprotocol: sybase, odbc, msqql, etc.
  - eg: jdbc:sybase:Tds:limousin:4100/myDB
- Only requirement: The relevant Drivers must be able to recognize their own URL
SQL Statements

✔ Create some form of Statement

– Statement
  • Represents a basic SQL statement
  • Statement stmt = conn.createStatement();

– PreparedStatement
  • A precompiled SQL statement, which can offer improved performance, especially for large/complex SQL statements

– Callable Statement
  • Allows JDBC programs access to stored procedures
Execute the Statement

✔ executeQuery(): execute a query and get a ResultSet back
✔ executeUpdate(): execute an update and get back an int specifying number of rows acted on
  - UPDATE
  - DELETE
✔ execute(): execute unknown SQL and returns true if a resultSet is available:
  - Statement genericStmt = conn.createStatement();
  - if( genericStmt.executeUpdate(String)) {
    • ResultSet rs = genericStmt.getResultSet(); process(); }
  - else {
    • int updated = genericStmt.getUpdateCount(); processCount();
  }
  • etc.
Result Sets

✓ ResultSet rs = stmt.executeQuery("select id, price from inventory");
   - rs.next() - go to next row in ResultSet
     • call once to access first row: while(rs.next()) {}
   - getXXX(columnName/indexVal)
     • getFloat()
     • getInt()
     • getDouble()
     • getString() (highly versatile, inclusive of others; automatic conversion to String for most types)
     • getObject() (returns a generic Java Object)
   - rs.wasNull() - returns true if last get was Null
Prepared Statements

- Use for complex queries or repeated queries
- Features:
  - precompiled at database (statement usually sent to database immediately on creation for compilation)
  - supply with new variables each time you call it (repeatedly eg.)
- eg:
  - PreparedStatement ps = conn.prepareStatement("update table set sales = ? Where custName = ?");
- Set with values (use setXXX() methods on PreparedStatement:
  - ps.setInt(1, 400000);
  - ps.setString(2, "United Airlines");
- Then execute:
  - int count = ps.executeUpdate();
Using the JDBC MetaData Interface

✔ ResultSet: ResultSetMetaData getMetaData()

✔ ResultSetMetaData provides information about the types and properties of the DDL properties of a ResultSet object

✔ ResultSetMetaData provides various methods for finding out information about the structure of a ResultSet:
  - getColumnClassName(int col): gets fully-qualified Java class name to which a column value will be mapped; eg. Java.lang.Integer, etc.
  - getColumnCount(): gets the number of columns in the ResultSet
  - getColumnDisplaySize(int col): gets the normal maximum width in characters for column
  - getColumnName(int col): gets the name of column
  - int getColumnType(int col): gets the JDBC type (java.sql.Types) for the value stored in col; eg. Value 12 = JDBC VARCHAR, etc.
  - getPrecision(int col): for numbers, gets the mantissa length, for others, gets the number of bytes for column
JDBC Transactions

✔ A Transaction’s ACID properties are:
  - **Atomic**: The entire set of actions must succeed or the set fails
  - **Consistent**: consistent state transfer from one state to the next
  - **Isolated**: A transaction is encapsulated and unmodifiable until the execution of the transaction set is *complete*
  - **Durable**: Changes committed through a transaction survive and tolerate system failures.

✔ Classic Example 1: Bank Transfer from one account to another
  - Step 1: withdrawal from Account A
  - Step 2: deposit into Account B
Using Transactions

✔ Step 1: turn off autocommit:
  – conn.setAutoCommit(false);
✔ Step 2: create and execute statements like normal
✔ Step 3: fish or cut bait: commit or rollback
  – if all succeeded:
    • conn.commit();
  – else, if one or more failed:
    • conn.rollback();
✔ Step 4 (Optional): turn autocommit back on:
  – conn.setAutoCommit(true);
Rolling Back Transactions

✔ When you get a SQLException, you are not told what part of the transaction succeeded and what part failed (this should be irrelevant)
✔ Best Practice:
  – *try* to rollback() (may throw new SQLException)
  – start over
✔ Example:
  – catch( SQLException e) {
    • try {
      – conn.rollback();
    • } catch (SQLException e) { checkPlease(); }  
  –  }

Transactions and Performance Implications

✔ Favor Transactions:
  – Disabling auto-commit means fewer commits over the wire (from driver to DBMS) which may cut down on IO overhead at the dataserver

✔ Favor Autocommit:
  – enabling autocommit may improve performance when multiple users are vying for database resources because locks are held for shorter periods of time
    • locks are only held per transaction. In autocommit mode, each statement is essentially a transaction
    • locks may be either page-level or row-level locks, the latter being more efficient (Oracle)
Transaction Isolation Modes

- **TRANSACTION_NONE**
  - Transactions are disabled or unsupported

- **TRANSACTION_READ_UNCOMMITTED**
  - Open policy that allows others to read uncommitted segments of a transaction, high potential for *dirty reads*

- **TRANSACTION_READ_COMMITTED**
  - Closed policy that disallows others’ reading uncommitted segments. They must block until a commit is received, dirty reads are forbidden.

- **TRANSACTION_REPEATABLE_READ**
  - Subsequent read transactions always get same set regardless of alteration until they call commit(), after which they get the changed data

- **TRANSACTION_SERIALIZABLE**
  - As above but also adds row insertion protection as well. If a transaction reads, and another transaction adds a row, and the first transaction reads again, it will get the original set without seeing the new row.

- `Conn.setTransactionIsolation(TX_READ_COMMITTED)`
Stored Procedures

✓ A Stored Procedure is written in a metalanguage defined by the DBMS vendor
✓ Used to batch or group multiple SQL statements that are stored in executable form at the database
✓ Written in some internal programming language of the DBMS:
  – Oracle’s PL/SQL
  – Sybase’s Transact-SQL
✓ THESE LANGUAGES ARE NON-PORTABLE from one DBMS to another (with the exception of the SQLJ standard, which allows you to write SQL in standard Java and have that understood by any DBMS that supports the SQLJ standard).
Incompatibilities

✔ Oracle Example:

- CREATE PROCEDURE sp_select_min_bal
  @balance IN FLOAT,
  AS
  SELECT account_id
  WHERE balance > @balance

✔ Sybase Example:

- create proc sp_select_min_bal
  (@balance real)
  as
  select account_id
  where balance > @balance
  return
Why Use Stored Procedures?

- ✔ Faster Execution of SQL (compiled and in-memory stored query plan)
- ✔ Reduced Network Traffic
- ✔ Modular Programming
- ✔ Automation of complex or sensitive transactions
- ✔ Syntax checking at time of creation of SP
- ✔ Syntax supports if, else, while loops, goto, local variables, etc., all of which dynamic SQL doesn’t have
Using Stored Procedures

✓ Create a CallableStatement (using prepareCall which is similar to prepareStatement)

- CallableStatement stmt =
  
  • conn.prepareCall("{call sp_setBalance(?,?)}")
  • stmt.registerOutParameter(2, Types.FLOAT);
  • stmt.setInt(1, custID);
  • stmt.setFloat(2, 213432.625);
  • stmt.execute();
  • Float newBalance = stmt.getFloat(2);

- Always register OUT or INOUT parameters in stored procedures using registerOutParameter()