Introduction to SQL & Database Application Development Using Java

By Alan Andrea

The purpose of this paper is to give an introduction to relational database design and sql with a follow up on how these concepts may be applied to developing database applications using Java and jdbc.

I) Introduction to SQL and relational databases

Part 1 Schema Design

A relational database basically contains tables. Tables are used to store like pieces of information. A table may contain any number of columns. A column corresponds to a unit of information. E.g. customer_name, phone_number, email_address etc. The actual data in a table is called the rows of the table or (tuples) Indexes are used for quick lookup of rows in a table.

EG:

TABLE: EMPLOYEE

Employee_no	*Department_no*	*Employee_name*	*Phone_number*	*Street Address*
123456	101	John Smith	123-456-7890	some street
484884	200	Mary Smith	123-456-7890	some street

In the above table, we have a table called EMPLOYEE with two rows of data in it (2 tuples).

Another important aspect of a table is that, in most cases, you must be able to differentiate one row from another. In other words it is of little use to have a table with duplicate information or rows that repeat. That is because there is no way to tell one row from another. Therefore, the meaning and value of the table and its underlying data becomes suspect.

For example, taking the above EMPLOYEE table, say we have two Employee_no's which are 123456. If this is the case how do you differentiate one employee from another especially when, in a large database, two people may actually have the same name? The primary key (which may itself potentially contain more than one column) is what enforces rows to be unique and differentiable so that we do not repeat rows of data.

In the above example, employee_no, is defined as the **PRIMARY KEY** since it is what uniquely identifies a row in the table. Furthermore, the primary key enforces the data to be unique and does not allow duplicates to be inserted!

A table may also contain what is called a **FOREIGN KEY**. A Foreign Key is used to join one table to another table so that you can pull corresponding information. As an example, you have another table, called DEPARTMENT which has as columns Department_no, Department_name and Department_manager. You may wish to write a query which extracts all employees and who their department managers are. To do this, you must write a query which **JOINS** the Employee Table to the Department Table.

TABLE: DEPARTMENT

Department_no	*Department_name*	*Department_Manager*
101	MIS	Some Person
200	Sales	Jane Doe
300	Accounting	Another Person

Perhaps you are wondering why we do not also include the department table columns on the Employee table!! The reason why we do not is because these two tables contain information which is clearly distinct from one another. We do not want to store, say, department name on the employee table because if that department does not have anyone working in it yet there will be no row for it in the employee table. Similarly if we delete an employee, and that employee is right now the only one in that department then we have lost all of the department information. This concept is called Normalization And is CRITICALLY IMPORTANT when developing a transactional database system that allows for inserts, updates and deletes on rows of your tables on a regular basis. Moreover, it is important that you ***break out*** like groups of information and join to them by foreign keys in order to maintain good Referential Integrity of your database.

In a DataWarehousing application which is not transactional and is ONLY used for reporting I.E. selecting information and **not** deleting or updating, you want the exact opposite. You instead want to have larger NON NORMALIZED TABLES with lots of Columns and fewer rows in order to speed up reporting. This is because if you are selecting many rows of information and doing summarization, you do not want to do joins which slow down queries Since these require nested loops (O(n^2) to complete) whereas with all the information in one Table it is O(n) to complete or less!!!

A final point on building relational databases is that in order to speed up operations you want
To put indexes on primary keys. Indexes behind the scenes create a hash table which allow
For O(1) (Instant) time to do a lookup of any 1 key value per se. I.e. without an index you must
Do O(n) full table scan (i.e. look at all rows in your table) to retrieve the row you are
interested in.

Part 2 WRITING QUERIES TO ACCESS RELATIONAL TABLES

A) *Basic definitions*

Sql- an acronym for Structured Query Language, is a simple language used to essentially ask questions of a database and to manipulate data in a database. It is a means of extracting information as well as updating information in the tables of your relational database.

Basic terminology used to talk about types of SQL statements are as follows:

DDL- Data Definition Language - involves statements which define tables, indexes And how to store data in a relational database.

DML - Data Manipulation Language- involves statements which update, insert or Essentially change information in a database.

B) *elements of a sql Select Statement:*

Sql Statements are comprised of the following elements:

SELECT	Enumerates which columns you wish to extract from a database. EG: Select customer_number, customer_name, ssn
FROM	Enumerates what Tables you wish to use to extract Information
WHERE	Allows you to specify column joins from one table to Another and Also allows you to place constraints Or restrictions to limit what data is returned by your Query
GROUP BY	Allows like rows to be grouped. Used in conjunction With aggregation functions like min,max,sum,count etc
HAVING	Allows you to place a constraint on an aggregated column in your query.
ORDER BY	Allows you to sort the rows that are returned in Your query

Examples of SQL:

E.G. 1)

Given Above tables **Employee** and **Department**, We want to write a query which Selects Employee number, Employee Name , Department Name and Manager for all employees whose department number is 200 and sort it Alphabetically by Employee Name:

```
Select  Employee_no, Employee_name,  Department_name, Department_manager
From   Employee,  Department
Where Employee.Department_no   =  Department.Department_no
And     Employee.Department_no  =  '200'
Order by Employee_name  asc;
```

In this query, I have JOINED the employee Table to the Department table via
The column department_no (which is contained in both tables and allows them to be joined)

E.G. 2)

Now we want to write a query which lists all departments and how many employees belong
To each department but only for departments which have greater than 1000 employees:

```
Select Department_name, count(employee_no)
From  Employee A,   Department B
Where  A.department_no  = B.department_no
Group by Department_name
Having   count(employee_no)  > 1000
```

C) elements of a DML statement:

Dml Statements may be used to do Manipulation of data in a relational database.

Such statements include Inserts, Updates and Deletes.

Examples:

1) We wish to insert a new employee into the Employee table:

```
Insert into Employee   Values('999999','500','Some Name','203-343-4030','Some Address');
```

2) We wish to update the employee table to change the Name of employee no 123456
From John Smith to Tom Smith:

```
Update Employee
 Set Employee_name = 'Tom Smith'
Where  Employee_no  = '123456';
```

3) We wish to delete all employees with the name Mary Smith from the Employee Table:

```
Delete from Employee where Employee_name  = 'Mary Smith';
```

II) Introduction to developing applications with java and JDBC

The First half of this paper discusses well formed relational database design. If you are using java or any other language to develop a transactional system which goes against a relational database, it is of **PARAMOUNT** importance that you have good referential integrity and normalization. This cannot be Stressed enough. You must provide primary keys for all tables to make sure that rows are not repeated. That being said, let us now look at database application development using Java.

Firstly, why do you want to use java to connect to a database and why do you **NOT** want to use java to connect to a database? The advantage of java is that it is basically portable meaning that you can pretty much compile it once and run it anywhere. Moreover, you have the ability to create a java applet and distribute it via the web running in the virtual machine of a web browser. From a programming point of view java is much easier to debug and get working than say c or c++ since you need not directly worry about memory allocation and garbage collection. Also with tools like Borland Jbuilder and DBSwing, you can develop jdbc enabled java applets and applications **without** having to do **extensive** coding. Therefore, the main advantage of java is ease of use, portability and ease of deployment. The downside of java is that when you compile it, you are generating byte code and not machine level instructions. Byte code must be interpreted by the virtual machine into actual machine instructions. This results in somewhat of a performance reduction. Therefore, you do not want to use java in mission critical real time applications which require speed and must process transactions immediately with little delay. For example, you do not want to use java in an embedded system which controls a jet fighter. Instead you must use c, c++ or assembly code. If on the other hand you are developing a web enabled application for recording customer satisfaction information or perhaps an e-commerce catalog application deployed to the internet, java might then be a valid choice. Again, java is portable across ***equivalent*** versions of the jdk. Another words you can run a java applet written in version 1.22 of the jdk on a 1.22 virtual machine on say windows NT and the same code will also run on Motif Running 1.22 virtual machine. However, **you of course cannot** run an Applet developed with 1.22 on a virtual machine written for 1.0 of the jdk. An important disclaimer, however, is that the actual look and feel may vary on different virtual machines running an equivalent version of the jdk. For example, the look of your applet on Motif may be somewhat different from the look on Windows 95. However, they should both basically function the exact same way.

Jdbc is an API (Application Program Interface) which allows you to extract information from a database (via sql statements) and Also to Update a database Via DML statements. It comes in several flavors from Type I through Type IV. Type I uses the jdbc odbc Bridge which requires odbc to be installed on the client. Moreover, the driver itself is not written in java but instead is a binary. Type II Utilizes a native proprietary driver installed on the client to tie into the database. Type III drivers let you send sql statements to an application server (which may reside on another computer) and the application server then forwards over to the database. The client may be written in All java. Type IV drivers are entirely written in Java and connect directly into the database engine or network layer itself. Types III and IV, therefore, are the preferred way of connecting to a database. This is because they require no binaries or separate programs to be installed on a client thereby allowing for easy deployment of your application.

Jdbc enables you to easily access and move through the results of a Sql Select Statement and to also run dml and ddl directly on a database. Furthermore, you have the ability to extract metadata (which is information about columns and their data types) as well as extract information about a database itself.

Using JDBC in Java Applications Part I: Connecting to a data source

In order to use the jdbc api, you must first import the jdbc package into your class:

Import java.sql.*

The package java.sql contains all of the methods which will allow you to work with a relational database.

Next, you must register the driver(s) that you will be using to connect to databases With:

Class.forName (driver);

Where driver is the name of the driver. Many database vendors provide their own Drivers that let you connect directly to a database via the type IV method. If you wish to connect to a database via jdbc-odbc bridge (such as access, sql server etc) there is a standard driver provided by sun which allows jdbc-odbc. This driver is called : sun.jdbc.odbc.JdbcOdbcDriver

Basically what the above is doing is loading the java packages which implement Java.sql.*

E.G. Class.forName (sun.jdbc.odbc.JdbcOdbcDriver) loads the jdbc-odbc bridge driver.

Finally, You need to specify a connection which essentially specifies a url where your database is located (the hostname) as well as the username and password to logon. It may potentially encapsulate other important information needed to connect to your particular database as well. To connect you use the following syntax:

Connection con = DriverManager.getConnection (url);

At this point, you have imported the package which has all the methods you need for the jdbc api, you have registered the driver(s) that you will be using and have opened up connection(s) to the database. Note: you may have an application which requires multiple simultaneous connections to different databases. This is **perfectly** allowable!!

Important Note: when you register a driver with Class.forName, you must Enclose that code in a try catch block and test for ClassNotFoundException errors. Similarly, when you connect to the database with the getConnection method or use any other jdbc methods, you must catch any SQLException errors.

Using JDBC in Java Applications Part 2: Using the jdbc API for running Selects

In this section, we will examine running select statement queries which bring back rows of data into to your java program. In order to facilitate running queries, java gives you three main classes: Statement, Resultset and ResultsetMetadata. Statement is an object that contains several methods for executing queries, fetching query return codes, finding out how many rows were returned into a resultset etc. It provides the foundation for working with resultsets and resultsetmetatdata And allows for a "holder" for your sql statements (whether they are for selects, updates deletes etc) Resultset is a kind of pointer to the results that get returned by your query. It is initialized when you call statement.executeQuery(). Resultset allows you to scroll forward one row at a time through the rows that are returned. In version of the API, you can only move forward using the next method; however, in version 2.0, you have **much** more flexability since you can also move backward or to any row which you choose. Furthermore, you can do updates and deletes in-situ. ResultsetMetadata provides information about the columns in your select statement. It is useful for determining information such as data type of a column, length of the column etc. Another important use of ResultsetMetadata is to determine how many columns there are in your query which is useful when displaying the results of dynamic queries.

Lets now look at an example of a using jdbc. We will use the Tables That I have illustrated In the first section on sql. I will write the java code to implement the query below:

We want to write a query which lists all departments and how many employees belong to each department but only for departments which have greater than 1000 employees:

First we want to Register our driver:

```
try
  {
     System.out.println(" Attempting to register driver ");
      Class.forName (sun.jdbc.odbc.JdbcOdbcDriver );
    System.out.println(" Driver successfully registered");
  }
   catch (ClassNotFoundException c) { System.out.println(" Error Registering Driver"); }
```

In The above code, we catch ClassNotFoundException. This might arise if The driver that you specify is not found.

Next we want to connect to the database, run the query, and traverse the resultset to print out the rows:

```
 Connection con;

try
  {

    // Open Connection to the database:
     System.out.println(" Attempting to Connect to database ");
     con = DriverManager.getConnection (JDBC:ODBC:webintel, "", "");
     System.out.println("Connected to database Successfully ");

   //   Create a query statement for this connection:
```

```
    Statement stmt = conn.createStatement ();
    Statement stmt = conn.createStatement ();


   // Define the query that we will run:
    query  =            "Select Department_name, count(employee_no)   ";
    query  =  query +  "From  Employee A,   Department B ";
    query  =  query + "Where  A.department_no  = B.department_no ";
    query  =  query + "Group by Department_name ";
    query  =  query + "Having   count(employee_no)  > 1000 ";


  // use the executeQuery method to run the query on the database and
  //  define a resultset  to handle traversing rows:

    ResultSet rset = stmt.executeQuery (query);

 //   Define a ResultSetMetadata handle to get information about the query we are running:
     ResultSetMetaData mdata = rset.getMetaData();

// Print the Result:
      while (rset.next ())
        {
          System.out.println("\n");
            for ( x=1; x<= mdata.getColumnCount(); ++x )
              System.out.println (   rset.getString (x) + "\t"  );
        }

   // Close the  connection to the database:
    conn.close();

  }
    catch (SQLException ex) {
         System.out.println ("\n*** SQLException caught ***\n" + ex.getMessage()+"   "+ex.getSQLState() );  }
```

In the above program, I register my driver (which is in this case sun's native jdbc odbc driver Bridge) , open up a connection to my database and create a statement to allow me to execute my query, retrieve a pointer to the resultset and also retrieve the resultset metadata. I then traverse that resultset by using the next method. Everytime the next method is called, I advance to the next row that got returned. But for each row that gets returned, I want to print out each column of data. This is where resultsetmetadata comes in handy as it tells me how many columns there are in my query. Then I can just simply loop through each column and retreive each value by means of rset.getString(x) where X is the number of the column that you are on. Then once everything has been traversed And printed out, I simply close my connection to the database since I am finished with it.

Using JDBC in Java Applications Part 2: Using the jdbc api for updating

In the last section, we looked at using java for reporting purposes in the sense that we are running a select statement to bring back rows of data. We will finish up now looking at how we may accomplish transactional processing. That is doing inserts,updates and deletes on a database. Again, as in the previous example, we start out with the statement class. statement has another method called executeQuery() which takes as an argument a string. This string represents the insert,update or delete that we wish to do.

Eg:

```
Statement stmt = conn.createStatement ();
Stmt.executeUpdate( "Update Employee Set Employee_name = 'Tom Smith'Where Employee_no = '123456' ");
```

The above statement will implement the example of an update that I presented in the last section.

An Important Note about Transactions:

The default behavoir of jdbc is autocommit after every sql statement. This can be turned off by using a method associated with Connection. You simply Call con.SetuAutoCommit(false) Then when you are ready to commit your series of transactions simply call con.commit(). This is important when you want to only update the database after say an insert and update were successfully and not anytime before.

I have covered the basics of doing Selects and transactions with inserts, updates and deletes. The jdbc 2.0 api also provides for batch updates and many other useful features for more sophisticated batch query and transactional processing.

Conclusion

In conclusion, I have covered the basic elements of relational databases and sql. I have stressed the importance of good database design and refererential integrity in a transactional system so that you maintain good quality of data throughout the ever changing lifecycle of your system. In designing a system that is meant for constant updates, (such as an e-commerce system) these concepts are of paramount importance. I have also stated that in a datawarehousing application (which is a system that does not update on a regular basis) you may want the opposite. In this case you want more columns in one table and perhaps fewer tables and ultimately fewer joins. In this type of application you will be selecting large amounts of data at a time and you do not want to deal with lots of joins which will slow down your query. In a transactional system, however, you are usually only dealing with few or just 1 row at a time. In this case the important consideration is to maintain good relationships in your database and not lose information. Here speed is not the issue in the sense you are not querying for thousands of rows of data But are instead interested in updating inserting or deleting just one.

In the second section, I have introduced jdbc. Jdbc is an api containing many powerful objects which facilitate sql processing and obtaining information about databases and columns. I have presented some of the low level details of using jdbc, however, many tools for developing java programs today (such as Borland's Jbuilder) actually take care of many of these details for you. This frees you up from having to implement all of the details. However, you must still understand important concepts like transaction processing and when you want to commit your data (if everything was successfully accomplished) or roll-it back if everything was not. Furthermore, it is always useful to understand the basics and details of how things work as this helps to explain how higher level components work behind the scenes and actually helps you to better understand their nature. Applying good

technical rigor will help you to use high level components and understand them better!

If you would like to download the complete working code that I have presented above, Please visit my website: www.webintel-systems.com I may also be reached at Aandrea@webintel-systems.com and atandrea@bellatlantic.net. Also on my website, you will find live working examples of java applets which connect Real time to a mysql database.