Java Database Programming

1. Introduction to Relational database and SQL

A *relational database* organizes data in *tables*. A table has *rows* (or *records*) and *columns* (or *fields*). Tables are *related* based on some common columns to ensure data integrity and eliminate data redundancy.

A high-level language called *Structure Query Language* (*SQL*) had been designed for structuring, querying and changing the data in a relational database. SQL defines commands such as `SELECT`, `INSERT INTO`, `DELETE FROM`, `UPDATE`, `CREATE TABLE`, and `DROP TABLE`.

Suppose that we have set up a table called "books" as follows. The table "books" has five columns: `isbn`, `title`, `author`, `price` and `qty`. SQL defines its own data types: `isbn` and `qty` are `INTEGER`; `title` and `author` are variable-length string (`VARCHAR`) of up to 50 characters; price is `FLOAT`.

books
isbn: INTEGER title: VARCHAR(50) author: VARCHAR(50) price: FLOAT qty: INTEGER

Although SQL standard is available. Most of the database implementations (notably MS Access) do not follow the specification strictly.

Suppose that the table books contains the following records:

isbn	title	author	price	qty
1001	Java for Dummies	Tan Ah Teck	11.11	11
1002	More Java for Dummies	Tan Ah Teck	22.22	22
1003	Java ABC	Mohamed Ali	33.33	33
1004	Only Java	Mohamed Ali	44.44	44
1005	A Cup of Java	Kumar	55.55	55
1006	A Teaspoon of Java	Kevin Jones	66.66	66
...	...	...	...	...

SQL statements are intuitive.

1.1 SELECT

SQL `SELECT` command can be used to select records from database table(s). For examples,

```
SELECT title, author, price FROM books
SELECT title, price FROM books WHERE author = 'Tan Ah Teck'
SELECT * FROM books WHERE price < 50 ORDER BY author ASC, price DESC
SELECT * FROM books WHERE author ='Tan Ah Teck' OR author = 'Kumar'
SELECT * FROM books WHERE author IN ('Tan Ah Teck', 'Mohamed Ali', 'Kumar')
SELECT title FROM books WHERE author LIKE '%TA_N%'
```

Syntax:

```
SELECT column1, column2,... | *
  FROM tableName
  WHERE criteria
  ORDER BY columnX ASC|DESC, columnY ASC|DESC,...
```

Notes:

- SQL strings are enclosed by single quotes (instead of double quotes as in Java, C/C++), but many implementations accept both single and double quotes.
- The wildcard '*' in the SELECT clause represents all the columns. Using '*' may not be a good software engineering practice, as you have no control on what you are selecting.
- You can use comparison operators: =, <> (or !=), >, <, >=, <= in the query criteria for comparing numbers as well as string.
- You can use logical operators AND, OR and NOT, on the conditions.
- For string, you can use special LIKE or NOT LIKE operators with wildcards for pattern matching. Wildcard '%' matches 0 or more characters; '_' matches exactly one character.
- SQL keywords and string matching are NOT case sensitive (but it depends on implementation). Access and MySQL are not case sensitive.

1.2 UPDATE

SQL UPDATE command can be used to update records of a table. For example,

```
UPDATE books SET price = 29.99, qty = 88 WHERE isbn = 1002
```

1.3 INSERT INTO

SQL INSERT command can be used to insert a new record into a table. For examples,

```
INSERT INTO books
   VALUES (1001, 'Java for Dummies', 'Tan Ah Teck', 19.99, 99)
INSERT INTO books
   (isbn, title, author) VALUES (1002, 'More Java for Dummies', 'Tan Ah Teck')
```

Syntax:

To insert a *full* record into a table with *N* columns, arrange the columns' value according to the columns in the table.

```
INSERT INTO tableName
  VALUES (column1value, column2value,... columnNvalue)
```

To insert a record with values in specific columns:

```
INSERT INTO tableName
  (column1, column2,...)
  VALUES (column1value, column2value,...)
```

The columns without values specified will be set to a special value NULL. These columns must be configured to accept NULL value.

Syntax:

```
UPDATE tableName expression
  WHERE criteria
```

1.4 DELETE FROM

SQL `DELETE` command is used to *permanently* delete records from a table. For example,

```
DELETE FROM books WHERE author = 'Tan Ah Teck'
DELETE FROM books            // All records deleted!!!
```

Syntax:

```
DELETE FROM tableName
  WHERE criteria
```

1.5 CREATE TABLE

`CREATE TABLE` commands can be used to create a new table in a database (or schema). For example

```
CREATE TABLE books (
  isbn INTEGER NOT NULL PRIMARY KEY,
  title VARCHAR(50),
  author VARCHAR(50) NOT NULL DEFAULT 'unknown',
  price FLOAT,
  qty INTEGER
)
```

The "`PRIMARY KEY`" and "`NOT NULL`" are known as *column's constraints*. The values in the `PRIMARY KEY` field are to be unique (i.e., no duplicate). `NOT NULL` excludes `NULL` value for this column.

Syntax:

```
CREATE TABLE tableName (
  columnName type [constraint...] [DEFAULT defaultValue] ...
  ......
)
```

2. Setting Up Databases

We have to set up a database before embarking on our database programming. We shall call our database "eshop" which contains only one table called "books", as illustrated below:

books
isbn: INTEGER title: VARCHAR(50) author: VARCHAR(50) price: FLOAT qty: INTEGER

The table "books" has five columns: `isbn`, `title`, `author`, `price` and `qty`. SQL defines it own data types, such as `INTEGER`, `FLOAT`, `CHAR(`*n*`)` (fixed-length string of *n* characters) and `VARCHAR(`*n*`)` (variable-length string up to *n* characters). SQL type `INTEGER` corresponds to Java's `int`; `FLOAT` to Java's `float` or `double`, `CHAR(n)` and `VARCHAR(n)` to Java's `String`.

In this article, I shall describe two database implementations: MS Access and MySQL. MS Access is a personal database and is not meant for business production. Some of the Java's JDBC features do not work on Access (due of the poor JDBC-ODBC bridge driver). MySQL is an industrial-strength open-source database with a native JDBC driver, supporting most (if not all) of the Java's JDBC features.

MS Access

Access has various versions, such as Access 2003 and Access 2007. The file type for Access 2003 (and earlier versions) is "`.mdb`". The file type for Access 2007 is "`.accdb`". These file formats are NOT compatible.

Access 2003

- Launch "MS Access": From "Start" button ⇒ "All Programs" ⇒ "Microsoft Office" ⇒ "Microsoft Office Access 2003".
- Create a new database: From Access's "File" menu ⇒ "New" ⇒ "Blank database..." ⇒ Save as "`eshop.mdb`".
- Create a new table: Select "Create table by entering data" ⇒ Rename the default column names "`field1`" , ..., "`field5`" to "`isbn`", "`title`", "`author`", "`price`" and "`qty`" (by clicking on the field names).
- Create new records: Type in the following records ⇒ Save the table as "books" ⇒ Answer "NO" to "there is no primary key defined" prompt (for simplicity).

isbn	title	author	price	qty
1001	Java for Dummies	Tan Ah Teck	11.11	11
1002	More Java for Dummies	Tan Ah Teck	22.22	22
1003	Java ABC	Mohamed Ali	33.33	33
1004	Only Java	Mohamed Ali	44.44	44
1005	A Cup of Java	Kumar	55.55	55
1006	A Teaspoon of Java	Kevin Jones	66.66	66
...	...	...	...	...

Access 2007

- Launch "MS Access": From "Start" button ⇒ "All Programs" ⇒ "Microsoft Office" ⇒ "Microsoft Office Access 2007".
- Create a new database: From the "Access" button ⇒ "New" ⇒ In "Filename" box, select your working directory and enter "`eshop.accdb`" as the name of the database ⇒ "Create".
- In "Add New Field", creates the following 5 fields (or columns): "`isbn`", "`title`", "`author`", "`price`" and "`qty`".
- Type in the above records. Keep (and ignore) the auto-generated column `ID` for the time being.

Check out the "Design view" (In Access 2003: Right-click on table "books" ⇒ select "Design view". In Access 2007: Select the view menu from the ribbon).

Observe the data types of the columns: `isbn` and `qty` are "Number (Long Integer)"; `price` is "Number (Double)"; `title` and `author` are "Text (255)". Access uses its own name for data types (which is NOT conforming to the SQL specification). Access's "Number (Long Integer)", "Number (Double)" and "Text" correspond to Java's `int`, `double` and `String`, respectively. You could change the data type and remove the additional `ID` column (Access 2007) in the design view.

<u>The correct way to create a database in Access is to create all the columns (with proper types and attributes) in the "Design View" before entering or importing the records.</u>

Close the Access before proceeding to the next step to define ODBC connection (otherwise, you may get an error "invalid directory or file path", as the database is currently opened in the *exclusive* mode and not *shared* mode).

ODBC Connection

An *ODBC connection* is needed to connect to the Access database. Define an ODBC connection called "eshopODBC", which selects the database we have just created, (i.e., "`eshop.mdb`" or "`eshop.accdb`"), as follows:

From the "Control Panel" ⇒ "Administrator Tools" ⇒ "Data Source (ODBC)" ⇒ Choose "System DSN" (System Data Source Name) or "User DSN" (User Data Source Name for the current login user only) ⇒ "Add" ⇒ select "Microsoft Access Driver (*.accdb) or Microsoft Access Driver (*.mdb)" ⇒ click "Finish" ⇒ Enter "eshopODBC" in "Data Source Name" and use the "Select" button to select "`eshop.accdb`" or "`eshop.mdb`".

MySQL

(Read http://www3.ntu.edu.sg/home/ehchua/programming/howto/MySQL_HowTo.html on how to setup MySQL. Assume that a user "`javauser`" with password "`javauser`" is already created.)

Start a monitor, login as "`javauser`" and issue these commands:
(You need to include "`--defaults-file=..\my.ini`" as the *first* command-line option if your MySQL "option file" is not kept in the "`Windows`" directory but in $MYSQL_HOME.)

```
> mysql -u javauser -p
Enter password: *******

mysql> create database if not exists eshop;

mysql> use eshop;
mysql> CREATE TABLE books (isbn INTEGER NOT NULL PRIMARY KEY,
       title VARCHAR(50), author VARCHAR(50), price FLOAT,
       qty INTEGER);

mysql> INSERT INTO books VALUES (1001, 'Java for dummies',
       'Tan Ah Teck', 11.11, 11);

mysql> INSERT INTO books VALUES (1002, 'More Java for dummies',
       'Tan Ah Teck', 22.22, 22);

mysql> INSERT INTO books VALUES (1003, 'Java ABC',
       'Mohamed Ali', 33.33, 33);

mysql> INSERT INTO books VALUES (1004, 'Only Java',
       'Mohamed Ali', 44.44, 44);

mysql> INSERT INTO books VALUES (1005, 'A Cup of Java',
       'Kumar', 55.55, 55);

mysql> INSERT INTO books VALUES (1006, 'A Teaspoon of Java',
       'Kevin Jones', 66.66, 66);

mysql> exit
```

Alternatively, you could use GUI Tool "MySQL Administrator" to set up the database.

3. Basic JDBC Programming

Java Database Connectivity (JDBC) is the Java's interface to the relational databases. A JDBC program has the following steps:
Step 1: Load the JDBC driver.
Step 2: Create a `Connection` object.
Step 3: Create a `Statement` object, from the `Connection` object created.
Step 4: Execute a SQL SELECT query by calling the `executeQuery()` method of the `Statement` object, which returns the query result in a `ResultSet` object; or

Execute a SQL INSERT|UPDATE|DELETE command by calling the `executeUpdate()` method of the `Statement` object, which returns an `int` indicating the number of rows affected.

Step 5: Process the query result.

Step 6: Free the resources by closing the `ResultSet` (for `executeQuery()`), `Statement` and `Connection`.

Exercise 3.1: SQL SELECT

Try out the following JDBC program which issues an SQL `SELECT`. (If you are using MySQL, modify Step 1 and 2 accordingly to load the correct database driver and create the connection.)

```
import java.sql.*;    // uses classes in java.sql package
public class TestSQLSelect {
   public static void main(String[] args) {

      try {
         // Step 1: Load JDBC Driver
         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");   // For Access
//       Class.forName("com.mysql.jdbc.Driver");          // For MySQL

         // Step 2: Create a database Connection object
         Connection conn = DriverManager.getConnection(
            "jdbc:odbc:eshopODBC");                       // For Access
//       Connection conn = DriverManager.getConnection(
//          "jdbc:mysql://127.0.0.1/eshop",
//          "javauser", "javauser");                      // For MySQL

         // Step 3: Create a Statement object thru the Connection
         Statement stmt = conn.createStatement();

         // Step 4: Execute a SQL SELECT Query, the query result
         //  is returned in a ResultSet object.

         String sqlStr = "SELECT title, price, qty FROM books";
         // For debugging
         System.out.println("The SQL query is:\n" + sqlStr);

         ResultSet rset = stmt.executeQuery(sqlStr);

         // Step 5: Process the ResultSet by scrolling the cursor.
         //  Use the column's name to retrieve the content of each
         //  cell with getString(), getInt() or getDouble().
         System.out.println("The records selected are:");
         int rowCount = 0;
         while(rset.next()) {   // Move the cursor to the next row
            String title = rset.getString("title");
            double price = rset.getDouble("price");
            int qty      = rset.getInt("qty");
            System.out.println(title + ", " + price + ", " + qty);
            rowCount++;
         }
         System.out.println("Total number of records = " + rowCount);
         rset.close();          // Close the ResultSet

         // Step 6: Free resources
         stmt.close();
         conn.close();
      } catch(ClassNotFoundException ex) {  // for Class.forName()
         ex.printStackTrace();
      } catch(SQLException ex) {            // for java.sql methods
         ex.printStackTrace();
      }
   }
}
```

Note that the JDBC operations are carried out through Connection, Statement, and ResultSet interfaces (defined in package java.sql). However, you need not know the details, but merely the public methods defined in the Application Program Interface (API). You also need not re-invent the wheels by creating these classes yourself (which will take you many months?!). "Re-using" software component is a main strength of OOP. Further notice that there is little programming involved in using JDBC. You only have to choose the appropriate database driver, specify the database URL, write the SQL query, and process the query result. The rest of the codes are kind of "standard program template". Again, this is because the wheels have been invented.

Try:

Try the following SQL SELECT statements and display all the columns retrieved. Make sure you modify the ResultSet processing to process only the columns retrieved (otherwise, you will get a "Column not found" error).

```
SELECT * FROM books

SELECT title, author FROM books WHERE price < 30

SELECT title, author, price, qty FROM books
    WHERE author = 'Tan Ah Teck' OR price >= 30
    ORDER BY price DESC, isbn ASC

SELECT title FROM books
    WHERE author LIKE 'T_n%' OR author LIKE '%Ali'
```

Exercise 3.2: SQL UPDATE

To execute SQL UPDATE, you have to call the method executeUpdate() (via the Statement object), which returns an int indicating the number of records affected. Recall that for SQL SELECT, we use executeQuery(), which returns a ResultSet object.

```
// Step 4 & 5: execute a SQL UPDATE statement via executeUpdate()
//   which returns an int indicating the number of row affected.
String sqlStr1 = "UPDATE books "
      + "SET price = 12.34, qty = 88 WHERE isbn = 1001";
System.out.println("The SQL query is:\n" + sqlStr1);  // For debugging
int count1 = stmt.executeUpdate(sqlStr1);
System.out.println(count1 + " records affected");
```

Notes:

- You do not need rset.close(), as no ResultSet was produced in executeUpdate().
- Open the database to observe the changes. (In Access, you may need to close the table and re-open the table to refresh the records.)

Exercise 3.3: SQL INSERT INTO

Similarly, use the executeUpdate() to execute SQL INSERT INTO. The method returns an int indicating the number of records affected.

```
// Step 4 & 5: execute a SQL INSERT statement via executeUpdate()
//   which returns an int indicating the number of row affected.
// For inserting a whole record
String sqlStr2 = "INSERT INTO books "          // need a space
      + "VALUES (3001, 'Web Programming', 'Kumar', 55.66, 55)";
System.out.println("The SQL query is:\n" + sqlStr2);  // For debugging
int count2 = stmt.executeUpdate(sqlStr2);
System.out.println(count2 + " records affected");
```

```
// For inserting only specific fields of a record
String sqlStr3 = "INSERT INTO books (isbn, title, author) "
      + "VALUES (3002, 'Fishing 101', 'Kumar')";
System.out.println("The SQL query is:\n" + sqlStr3);  // For debugging
int count3 = stmt.executeUpdate(sqlStr3);
System.out.println(count3 + " records affected");
```

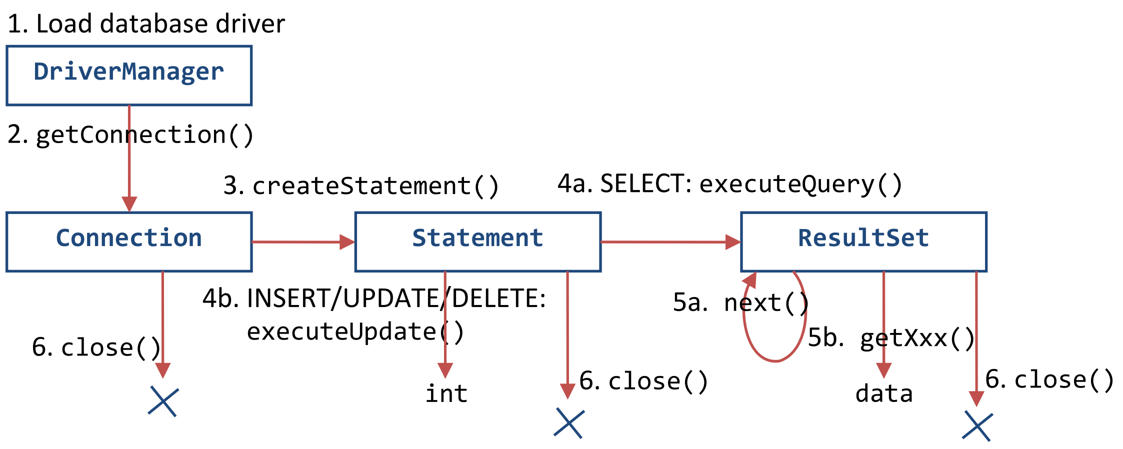
- You cannot insert duplicate records with the same key (e.g., isbn) if the field is configured to be UNIQUE. In other words, you cannot run the above codes twice without changing the isbn number. (Access 2007 signals a "General Error" for duplicate records?!)
- If you insert a partial record, make sure that the missing fields are configured to accept NULL value.

Exercise 3.4: SQL DELETE FROM

Again, use executeUpdate() to execute SQL DELETE statements. The method returns an int indicating the number of records affected. For example,

```
// Step 4 & 5: execute a SQL DELETE statement via executeUpdate()
//   which returns an int indicating the number of row affected.
String sqlStr4 = "DELETE FROM books WHERE isbn = 3002";
System.out.println("The SQL query is:\n" + sqlStr4);  // For debugging
int count4 = stmt.executeUpdate(sqlStr4);
System.out.println(count4 + " records affected");
```

4. JDBC Cycle



5. More JDBC

Exercise 5.1: Atomic Transaction (COMMIT/ROLLBACK)

An *atomic transaction* is *a set of SQL statements*, which either *all* succeed or *none* succeeds. This is to prevent *partial update* to the database. To manage transaction in JDBC, we first disable the default *auto-commit* (which commits every SQL statement), issue a few SQL statements, and then decide whether to issue a commit() to commit all the changes or rollback() to discard all the changes since the last commit. For example:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");   // For Access

Connection conn =
   DriverManager.getConnection("jdbc:odbc:eshopODBC");

// Disable auto-commit, which commits every SQL statement.
conn.setAutoCommit(false);

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

```java
// Issue two UPDATE statements thru executeUpdate()
stmt.executeUpdate("UPDATE books SET qty = qty - 1"
     + " WHERE isbn = 1001");
stmt.executeUpdate("UPDATE books SET qty = qty + 10"
     + " WHERE isbn = 1002");
conn.commit();     // Commit changes

// Issue two UPDATE statements thru executeUpdate()
stmt.executeUpdate("UPDATE books SET qty = qty - 11"
     + " WHERE isbn = 2001");
stmt.executeUpdate("UPDATE books SET qty = qty + 100"
     + " WHERE isbn = 2002");
conn.rollback();   // Discard all changes since the last commit

stmt.close();
conn.close();
```

Notes:

- Before closing the connection (i.e., `conn.close()`), you need to issue a `conn.commit()` or `conn.rollback()`. Otherwise, you will get an error "Invalid Transaction state".

Exercise 5.2: `ResultSetMetaData`

Each `ResultSet` object is associated with a *header* (called *metadata*), which contains information about the `ResultSet` object, such as the number of columns, the name and type of the columns etc. The metadata is stored in a `ResultSetMetaData` object. You can use method `rset.getMetaData()` to retrieve the associated meta data object of a `ResultSet` `rset`.

`ResultSetMetaData` is useful in *dynamically* processing the `ResultSet`. You could retrieve the number of columns and use `rset.getXxx(`*`columnNumber`*`)` to retrieve the content of a particular column number in the current row. Note that column number begins at 1 (not 0). For example,

```java
// Step 5: Process the ResultSet of a SQL SELECT query
//  Use the ResultSetMetaData to retrieve the columns' name
//  and number of columns
ResultSetMetaData rsetMD = rset.getMetaData();
int numColumns = rsetMD.getColumnCount();

// Print columns' name
for (int i = 1; i <= numColumns; i++) {
   System.out.printf("%-25s", rsetMD.getColumnName(i));
}
System.out.println();

// Print columns' class name
for (int i = 1; i <= numColumns; i++) {
   System.out.printf("%-25s",
      "(" + rsetMD.getColumnClassName(i) + ")");
}
System.out.println();

// Print columns' content for all the rows
while(rset.next()) {
   for (int i = 1; i <= numColumns; i++) {
      System.out.printf("%-25s", rset.getString(i));
   }
   System.out.println();
}
```

Exercise 5.3: `PreparedStatement`

Suppose that you are going to insert 1000 records into a database. Using `Statement` object, you need to prepare 1000 different SQL `INSERT` strings as argument to `stmt.executeUpdate(sqlStr)`. Running a thousand SQL INSERT is inefficient!

JDBC provides a class called `PreparedStatement`, which allows you to pass parameters to a SQL statement and execute the same SQL statement multiple times. A `PreparedStatement` is a pre-compiled SQL statement that is more efficient than calling the same `Statement` over and over. In `PreparedStatement`, "?" indicates a *place holder* for parameter. A set of `setXxx()` methods can be used to fill in the parameters. For example,

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");   // For Access

Connection conn =
   DriverManager.getConnection("jdbc:odbc:eshopODBC");

PreparedStatement pstmt = conn.prepareStatement(
      "INSERT INTO books VALUES (?, ?, ?, ?, ?)");  // five place holders

pstmt.setInt(1, 4001);                     // set value for 1st place holder
pstmt.setString(2, "Web Programming");     // set value for 2nd place holder
pstmt.setString(3, "Kumar");               // set value for 3rd place holder
pstmt.setDouble(4, 55.66);                 // set value for 4th place holder
pstmt.setInt(5, 55);                       // set value for 5th place holder
int rowAffected = pstmt.executeUpdate();   // execute statement

pstmt.setInt(1, 4002);                     // change value for 1st place holder
pstmt.setString(2, "Fishing");             // change value for 2nd place holder
// same value for 3rd, 4th, and 5th place holders
rowAffected = pstmt.executeUpdate();

pstmt.close();
conn.close();
```

Notes: Once a parameter has been defined for a given `PreparedStatement`, it can be used for multiple executions, until it is cleared by a call to `pstmt.clearParameter()`. In the above example, the 3rd, 4th and 5th parameters of the 2nd statement are set in the 1st statement.

6. More on SQL SELECT (Optional)

`SELECT` is the most important, and also most complex, SQL statement.

6.1 Filtering Operators

Many operators can be used in the `WHERE` clause of a `SELECT` statement to filter the selection:

- **`LIKE`**, **`NOT LIKE`**: Pattern matching for string. Wildcard '%' matches zero or more characters, '_' matches one single character. For example,

```
SELECT * FROM books
  WHERE author LIKE 'Tan%' OR author LIKE 'k__'
```

- **`IS NULL`**, **`IS NOT NULL`**: to check for `NULL` value.

```
SELECT * FROM books
  WHERE qty IS NULL OR price IS NOT NULL
```

- **`IN`**, **`NOT IN`**: Check if a value is contained in a set of values. For example,

```
SELECT * FROM books
   WHERE author IN ('Tan Ah Teck', 'Mohamed Ali', 'Kumar')
```

- **BETWEEN v1 AND v2**, **NOT BETWEEN v1 AND v2**: Check if the value is between two values. For example,

```
SELECT * FROM books
   WHERE price BETWEEN 33.33 AND 55.55
```

6.2 SELECT Sub-Query

For example,

```
SELECT * FROM books
  WHERE author IN
    SELECT author FROM books WHERE title LIKE 'Java%'
```

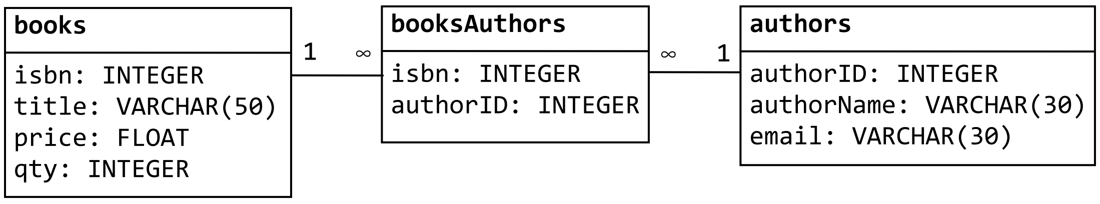
6.3 SQL Functions

You could use these functions in your SELECT statements: **AVG**, **COUNT**, **MIN**, **MAX**, **SUM**. For example,

```
SELECT count(*) FROM books
SELECT min(price) max(price) FROM books
```

7. Relating Tables (Optional)

The power of a relational database is tables can be related via some common columns to eliminate data redundancy and ensure data integrity. To illustration the relation between tables, we have to first create a few related tables, in a database called "BigEShop", as follows:



A book could be written by one or more authors. An author can publish one (possibly zero) or more books. Is it possible to capture these requirements in a single table with a fixed number of columns, and without duplicating any piece of information?!

To fulfill these requirements, we need to design 3 tables:
A table books which maintains information about the books (such as isbn, title, price, qty), with isbn as the *primary key*.

```
+------+------------------+-------+------+
| isbn | title            | price | qty  |
+------+------------------+-------+------+
| 1001 | Java for Dummies | 11.11 |   11 |
| 1002 | Only Java        | 22.22 |   22 |
| 1003 | Java ABC         | 33.33 |   33 |
| 1004 | Java 123         | 44.44 |   44 |
+------+------------------+-------+------+
```

A table authors which maintains information about authors (such as authorID, authorName, email) with authorID as the *primary key*.

```
+----------+-------------+-------+
| authorID | authorName  | email |
+----------+-------------+-------+
|        1 | Tan Ah Teck | NULL  |
|        2 | Mohamed Ali | NULL  |
|        3 | Kumar       | NULL  |
|        4 | Kelvin Jones | NULL  |
+----------+-------------+-------+
```

A table `booksAuthors` which relates the books and authors. Suppose a book (says `isbn=1001`) has three authors (says `authorID={1, 2, 3}`), there will be three records in the `booksAuthors` table for this book. On the other hand, if an author (says `authorID=1`) has published two books (`isbn={1001, 1002}`), there will be 2 records in the `booksAuthors` table about this author.

```
+------+----------+
| isbn | authorID |
+------+----------+
| 1001 |        1 |
| 1001 |        2 |
| 1001 |        3 |
| 1002 |        1 |
| 1002 |        3 |
| 1003 |        2 |
| 2001 |        2 |
+------+----------+
```

The columns `isbn` and `authorID` in the `booksAuthors` table is known as *foreign keys*, which reference the *primary key* in the tables `books` and `authors`, respectively.

The following SQL commands were issued from MySQL. But you should have no difficulty to figure out how to create and populate the 3 tables in Access (Create a new table and use the "design view" to define the columns with the proper types and attributes). Set `isbn` as the primary key for table `books`, and `authorID` as primary key for table `authors`. (Access does not seem to support foreign key , which ensures integrity and improves the efficiency of the query?)

```
CREATE DATABASE IF NOT EXISTS BigEShop;

USE BigEShop;

CREATE TABLE books (
   isbn INTEGER NOT NULL PRIMARY KEY,
   title VARCHAR(50) NOT NULL,
   price FLOAT,
   qty INTEGER);

CREATE TABLE authors (
   authorID INTEGER NOT NULL AUTO_INCREMENT,
   authorName VARCHAR(30) NOT NULL,
   email VARCHAR(30),
   PRIMARY KEY(authorID));

CREATE TABLE booksAuthors (
   isbn INTEGER NOT NULL,
   authorID INTEGER NOT NULL,
   FOREIGN KEY (isbn) REFERENCES books (isbn),
   FOREIGN KEY (authorID) REFERENCES authors (authorID),
   INDEX (isbn),
   INDEX (authorID));
```

```
INSERT INTO books VALUES (1001, 'Java for Dummies', 11.11, 11);
INSERT INTO books VALUES (1002, 'Only Java', 22.22, 22);
INSERT INTO books VALUES (1003, 'Java ABC', 33.33, 33);
INSERT INTO books VALUES (1004, 'Java 123', 44.44, 44);

SELECT * FROM books;
+------+------------------+-------+------+
| isbn | title            | price | qty  |
+------+------------------+-------+------+
| 1001 | Java for Dummies | 11.11 |   11 |
| 1002 | Only Java        | 22.22 |   22 |
| 1003 | Java ABC         | 33.33 |   33 |
| 1004 | Java 123         | 44.44 |   44 |
+------+------------------+-------+------+
```

```
INSERT INTO authors (authorName) VALUE ('Tan Ah Teck');
INSERT INTO authors (authorName) VALUE ('Mohamed Ali');
INSERT INTO authors (authorName) VALUE ('Kumar');
INSERT INTO authors (authorName) VALUE ('Kelvin Jones');

SELECT * FROM authors;
+----------+-------------+-------+
| authorID | authorName  | email |
+----------+-------------+-------+
|        1 | Tan Ah Teck | NULL  |
|        2 | Mohamed Ali | NULL  |
|        3 | Kumar       | NULL  |
|        4 | Kelvin Jones | NULL  |
+----------+-------------+-------+
```

```
INSERT INTO booksAuthors VALUES (1001, 1);
INSERT INTO booksAuthors VALUES (1001, 2);
INSERT INTO booksAuthors VALUES (1001, 3);
INSERT INTO booksAuthors VALUES (1002, 1);
INSERT INTO booksAuthors VALUES (1002, 3);
INSERT INTO booksAuthors VALUES (1003, 2);
INSERT INTO booksAuthors VALUES (2001, 2);

SELECT * FROM booksAuthors;
+------+----------+
| isbn | authorID |
+------+----------+
| 1001 |        1 |
| 1001 |        2 |
| 1001 |        3 |
| 1002 |        1 |
| 1002 |        3 |
| 1003 |        2 |
| 2001 |        2 |
+------+----------+
```

Joining Tables in SELECT Query

Tables can be connected via a JOIN during a SELECT query. A *join* allows a query to include references to multiple tables and to restrict the output based on the relations between those tables.

The basic join is an *equi-join*, where data in two tables is linked based on a shared value. For example,

```
SELECT * FROM books, booksAuthors
  WHERE books.isbn = booksAuthors.isbn;
+------+------------------+-------+------+------+----------+
| isbn | title            | price | qty  | isbn | authorID |
+------+------------------+-------+------+------+----------+
| 1001 | Java for Dummies | 11.11 |   11 | 1001 |        1 |
| 1001 | Java for Dummies | 11.11 |   11 | 1001 |        2 |
| 1001 | Java for Dummies | 11.11 |   11 | 1001 |        3 |
| 1002 | Only Java        | 22.22 |   22 | 1002 |        1 |
| 1002 | Only Java        | 22.22 |   22 | 1002 |        3 |
| 1003 | Java ABC         | 33.33 |   33 | 1003 |        2 |
+------+------------------+-------+------+------+----------+
```

Note that the books' record of isbn = 1004 does not produce any match and is left out of the query result. Similarly, booksAuthors's record of isbn = 2001 does not produce any match.

```
SELECT authorID, title, price FROM booksAuthors, books
  WHERE booksAuthors.isbn = books.isbn AND title LIKE 'Java%';
+----------+------------------+-------+
| authorID | title            | price |
+----------+------------------+-------+
|        1 | Java for Dummies | 11.11 |
|        2 | Java for Dummies | 11.11 |
|        3 | Java for Dummies | 11.11 |
|        2 | Java ABC         | 33.33 |
+----------+------------------+-------+
```

```
SELECT books.title, books.price, authors.authorName
   FROM books, booksAuthors, authors
   WHERE books.isbn = booksAuthors.isbn
      AND booksAuthors.authorID = authors.authorID;
+------------------+-------+-------------+
| title            | price | authorName  |
+------------------+-------+-------------+
| Java for Dummies | 11.11 | Tan Ah Teck |
| Java for Dummies | 11.11 | Mohamed Ali |
| Java for Dummies | 11.11 | Kumar       |
| Only Java        | 22.22 | Tan Ah Teck |
| Only Java        | 22.22 | Kumar       |
| Java ABC         | 33.33 | Mohamed Ali |
+------------------+-------+-------------+
```

```
SELECT books.title, books.price, authors.authorName
   FROM books, booksAuthors, authors
   WHERE books.isbn = booksAuthors.isbn
      AND booksAuthors.authorID = authors.authorID
      AND authors.authorName = 'Tan Ah Teck';
+------------------+-------+-------------+
| title            | price | authorName  |
+------------------+-------+-------------+
| Java for Dummies | 11.11 | Tan Ah Teck |
| Only Java        | 22.22 | Tan Ah Teck |
+------------------+-------+-------------+
```

8. References

[1] Java Database Connectivity (JDBC) Home Page @ http://java.sun.com/products/jdbc/overview.html.
[2] JDBC API Specifications 1.2, 2.1, 3.0, and 4.0 @ http://java.sun.com/products/jdbc.
[3] White Fisher, et al., "JDBC API Tutorial and Reference", 3rd eds, Addison Wesley, 2003.
[4] MySQL Home Page @ http://www.mysql.org.
[5] SQL.org @ http://www.sql.org.

Java Database Programming, v2009a (Last Modified: September 10, 2009)
Feedback, comments, and corrections can be sent to Chua Hock Chuan at ehchua@ntu.edu.sg