LAB 7 Notes

Working with JDBC

- Any questions on the project (Discuss)
- In the previous lab we discussed Postgres. Today we will talk al little more about how to work with postgres using JDBC and we are going to cover some topics you will need for the project.

Ch.1: Overview of Database Systems Ch.2: Introduction to Database Design Ch.3: The Relational Model Ch.4: Relational Algebra Ch.5: SQL Ch.8: Storage and Indexing	Ch.9: Storing Data: Disks and Files Ch.10: Tree-Structured Indexing Ch.11: Hash-Based Indexing Ch.12: Overview of Query Evaluation Ch.13: External Sorting Ch.14: Evaluation of Relational Operators Ch.15: A Typical Relational Query Optimizer Ch.16: Overview of Transaction Management

Outline

1) Working with JDBC

2) String manipulation in Java

3) Some Postgres features needed for the project

- **Sequences**
- **Date/Timestamp functions**

4) If time permits we will look at the project manual.

1) Working with JDBC

There are four architectural components in JDBC

- Application (initiates and terminates connections, submits SQL statements)
- Driver manager (load JDBC driver)
- Driver (connects to data source, transmits requests and returns/translates results and error codes)
- Data source (processes SQL statements)

Steps to submit a database query:

- Load the JDBC driver. For instance, if the class name is jdbc.DriverXYZ , you would load the driver with the following line of code:

```
Class.forName("jdbc.DriverXYZ");
```

- Connect to the data source. The second step in establishing a connection is to have the appropriate driver connect to the DBMS

  Connection con = DriverManager.getConnection(url, "myLogin", "myPassword");

- Create and execute a statement. A Statement object is what sends your SQL statement to the DBMS. You simply create a Statement object and then execute it, supplying the appropriate execute method with the SQL statement you want to send.

  - For a SELECT statement, the method to use is executeQuery .
  - For statements that create or modify tables, the method to use is executeUpdate .

  It takes an instance of an active connection to create a Statement object. In the following example, we use our Connection object con to create the Statement object stmt :

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

stmt.executeUpdate("CREATE TABLE COFFEES " +
      "(COF_NAME VARCHAR(32), SUP_ID INTEGER,
PRICE FLOAT, " +
   "SALES INTEGER, TOTAL INTEGER)");
```

Retrieving Values from Result Sets. We now show how you send the above SELECT statements from a program written in the Java programming language and how you get the results we showed. JDBC returns results in a ResultSet object, so we need to declare an instance of the class ResultSet to hold our results. The following code demonstrates declaring the ResultSet object rs and assigning the results of our earlier query to it:

```
ResultSet rs = stmt.executeQuery(
"SELECT COF_NAME, PRICE FROM COFFEES");
```

The variable rs , which is an instance of ResultSet , contains the rows of coffees and prices shown in the result set example above.
In order to access the names and prices, we will go to each row and retrieve the values according to their types. The method next moves what is called a cursor to the next row and makes that row the one upon which we can operate. Since the cursor is initially positioned just above the first row of a ResultSet object, the first call to the method next moves the cursor to the first row and makes it the current row. Successive invocations of the method next move the cursor down one row at a time from top to bottom

```
while (rs.next ()) {
  for (int i=1; i<=numCol; ++i)
    System.out.println (rsmd.getColumnName (i) +
      " = " + rs.getString (i));
  System.out.println ();
  ++rowCount;
}
```

2) String manipulation in Java

A **string** is simply a sequence of characters that can be manipulated as an 'entity'. The main operand you are going to use is the string concatenation operator (+). More information about string you can find in

http://www.scit.wlv.ac.uk/appdocs/java/api/java/lang/String.html

Converting String to value. To convert a string value to a number (for example, to convert the String value in a text field to an int), use these methods. Assume the following declarations: String s; int i; long l; float f; double d;

type	*Example statement*
Int	*i* = Integer.parseInt(*s*);
long	*l* = Long.parseLong(*s*);
float	*f* = Float.parseFloat(*s*);
double	*d* = Double.parseDouble(*s*);

3) Some Postgres features needed for the project

3.1) Sequences

A sequence object is usually used to generate unique identifiers for rows of a table. The sequence functions, listed in the following table, provide simple, multiuser-safe methods for obtaining successive sequence values from sequence objects.

Table Sequence Functions

Function	Return Type	Description
nextval(text)	bigint	Advance sequence and return new value
currval(text)	bigint	Return value most recently obtained with nextval
setval(text, bigint)	bigint	Set sequence's current value
setval(text, bigint, boolean)	bigint	Set sequence's current value and is_called flag

Example

```
SELECT nextval('foo') ;
```

3.2) Date/Timestamp functions.

PostgreSQL supports the full set of SQL date and time types to see them check section in Section 8.5 in the postages web documentation. The operations available on these data types are described in in Section 9.9.