**Transactions … Adding Orders Via Stored Procedure**

This assignment focusses on Transactions but the entire scope of each transaction is handled from within a single stored procedure. Aside from that basic requirement, this assignment is functionally identical to Assignment 9. (It will be repeated here.) The transactions commitment and roll back will also be executed within the bounds of that stored procedure.

As with Assignment 9, you will be both committing and rolling back a transaction – one each - wherein an arbitrary existing customer makes an order consisting of one or more ordered items. You will be using the classicmodels schema.

You might have noticed that both of the tables *orders* and *orderdetails* have constraints. For example:

* All orders must be of products with product IDs in the *products* table.
* All orders must be made by known customers with customer IDs in the *customers* table.
* All items in the *orderdetails* table must have orders with order IDs in the *orders* table.

With this assignment modifying – or attempting to modify – the *orders* and *orderdetails* tables, this program needs to start by determining on the fly (i.e., no user input) available products and current customers. For this assignment, a single randomly chosen customer (i.e., their customer number) will be used for both transactions and a randomly selected set of between one to four unique products - with number randomly selected - (i.e., their product codes) will be used in each of the two transactions.

To be clear, there will be two transactions, one will be allowed to complete and commit, the other to be effectively complete but then will be rolled back, rolled back as though your program had detected as problem, which it will not. If not for the subsequent roll back, both transactions must be capable of successfully completing and being committed. (Indeed, for reasons of debug, it will be prudent to initially allow both to be successfully completed.)

Within each transaction your program will be creating a record in the *orders* table which means that at least the following columns will be set:

* **orderNumber** … A unique value one more than the current maximum order number.
* **orderDate** … An SQL Date for the date of the execution of your program. (You might want to do some research on Java classes Calendar, Date, and TimeStamp, but there are other approaches.)
* **requiredDate** … An SQL Date seven days forward from the orderDate.
* **shippedDate** … Being a new order, shipped date has not yet occurred and so is unknown.
* **status** … ‘In Process’
* **comments** … None required.
* **customerNumber** … Your chosen customer number.

Execute the *orders* table insertion using a generated string.

The contents of each record being inserted into the *orderdetails* table are or may be as follows:

* **orderNumber** … The unique orderNumber value from the *order* table.
* **productCode** … For each record inserted, uniquely one of your selected product codes from the *products* table.
* **orderLineNumber** … The line number of this item in this order. (The value is 1 for the first item inserted, 2 for the second, and so on.)
* **quantityOrdered** … Set it to the same value as the orderLineNumber.
* **priceEach** … Set the dollar amount to the same value as the orderLineNumber. (This column is defined as a Decimal(10,2). You will want to look up class BigDecimal.)

Execute the *orderdetails* table insertion via a PreparedStatement.

Within the scope of each of the transactions, within the stored procedure, the following SQL operations must be included:

* The determination of the current maximum order number,
* The insertion into the orders table
* The insertion(s) into the orderdetails table.

Again, there will be two passes on this transaction; the first is committed and the second is rolled back. Programmatic validation is done after the completion of each transaction by going back and accessing the maximum order number from the orders table. In the case of the commit pass, the order number should be equal to the value of your committed insertion. In the case of the rolled back transaction, the order number returned should be one less than that of your rolled-back transaction (i.e., the *orders* table was not changed).

As a hint on the Stored Procedure, one called **AddOrder**, parameters can be as follows:

* IN **orderTime** TIMESTAMP
* IN **requiredTime** TIMESTAMP
* IN **customerNumber** INT
* IN **ordersDetails** VARCHAR(1024) … This is a comma-delimited string of product codes.
* IN **commitRollback** VARCHAR(8)
* INOUT **maximumOrderNumber** INT

As a particular note on the commitRollback parameter, that merely indicates the case you are testing, either “COMMIT” or “ROLLBACK”.

As always …

* Close all closeable JDBC objects for all possible paths out of your program.
* Submit results, Eclipse directory (DBTransactions\_SP) and text file of stored procedure (AddOrder), into your class storage.