MDE518: Database Management Systems
University of Athens
Deprt. of Informatics & Telecommunications

Programming Project I
Today's Date:November 7th, 2014
Due Date: December 8th, 2014

PREAMBLE

In this project, you will design, implement, demonstrate and document a database system that could automate the functions (including administrative operations) of a grocery online store along the lines of `www.freshdirect.com` (or *FD*).

PROBLEM DESCRIPTION:

Your system should include information about the (registered) users, the products they have ordered, as well as availability of all products. Below is a description of the general guidelines of the project. While you are working, keep in mind that these items make up a minimum set of requirements. Hence, you may extend the scope of your work as you see it appropriate and/or interesting.

Store data: they will help *FD* run its business on-line; the company markets a range of grocery products from fresh vegetables and meats to canned food and household goods. *FD*'s regular customers include individuals and businesses across the country. *FD* wishes to satisfy their orders as quickly as possible. To this end, the company tries to keep adequate level of stock for each product traded. Whenever the stock in hand falls below some predetermined level then another batch is procured.

User data: should include the name of every registered user, his/her login name, password, address, email, product items he/she has ordered, as well as (needed) details on all user transactions that have occurred.

There are at least *five* relations used to keep track of products, orders, and customers. Their schemas are shown below (initial layout of relations). Possible relation keys are underlined.

USERS:
cust_no login password cust_name email street town post_code cr_limit curr_bal

PRODUCTS:
prod_code name description prod_group list_price qty_on_hand procur_level procur_qty

ORDERS:
order_no order_date cust_no total_cost

ORDER_DETAILS:
order_no prod_code order_qty order_sum

SUPPLIERS:
prod_code supplier_id supplier_name email quant_sofar

Some columns, such as `cust_name` are self-explanatory; others may need some explanation.

1. `cr_limit`: the maximum that a customer is allowed to own to the *FD*.

2. `curr_bal`: the amount currently owed by the customer.

3. `list_price`: the advertised price for a single unit of a particular product.

4. `order_date`: the date on which the order was made; you may assume that the date is held in the YYYYMMDD form.

5. `order_sum`: the sum total money charged for a specific type of product purchased.

6. `prod_group`: a code that indicates whether a product is grouped as vegetable and fruits (V), meats and dairy products(M), chemical products (C), industrial products (I), canned/boxed food (B), garden (G), or household (H) in nature.

7. `procur_level`: the level at which the quantity on hand is compared; if stock falls below this level then *FD* will usually request another batch to avoid stock-out.

8. `procur_qty`: the amount usually involved in any new procurement.

In the database, we should maintain for each product sold by *FD* a unique `prod_code`, its name, its product group (M, C, I, B, G, H), its list price, the quantity in the stock of the store, the level below which procurement of new quantity in needed and the procurement quantity filed.

FD procures each of the products it markets from *one or more suppliers*. For every supplier we have to maintain her/his name, contact email, and the sum quantity bought so far.

Using a database management system of your choice, create the schema of the whole database (i.e., the five –and perhaps more– relations). Then insert (substantial) sample data tuples in your database (of your own choice). You have also the freedom to add more tables of your own choice/design, should you consider this necessary.

System Requirements:
The following type of events should be handled by the target system:

1. *Registration*: a new user has to provide the appropriate information; he/she can pick a login-name and a password. The login name should be checked for uniqueness.

2. *Searching and Ordering*: After registration, a user can order one or more products. The total amount of its order is reported to him/her. A user may order multiple items of a product based on availability. Although the actual charging of the credit card and the shipping of the orders taken are outside the scope of the project, the customer's consent and the memorization of her credit card numbers must be part of your work.

3. *User record*: upon user demand, we should display the full record of a user:
   - all his/her personal data
   - the full history of sales (product name, number of items purchased, date)

4. *New product*: The store manager records the details of every new product along with their respective quantities that have arrived at *FD*'s.

5. *Arrival of more products*: The store manager increases the appropriate counts.

6. *Product Browsing*: Users may search for products, by asking conjunctive queries on products, and/or suppliers, and/or description-words, and/or categories (M, C, I, etc.). Your system should allow the user to specify that the results are to be sorted (a) list price, (b) by product name, (c) by the name of the supplier.

7. *Querying the database*: users may want the following specific queries to be materialized:

   (a) For each product, print the product's lowest and highest ever order sum.

   (b) On which days of the month MM in year YYYY the total sum of the value of all orders taken was larger than 10,000 Euros?

   (c) Which product has led to the biggest volume of sales in terms of the units sold?

   (d) For each product group, print the product code, name, and list price of the most expensive product in that group.

   (e) List any products that may have never been ordered.

   (f) What are the products that were not ordered in the month MM of year YYYY?

8. *Buying suggestions*: Like `www.freshdirect.com`, when a user orders a product 'A', your system should give a list of other suggested products. Product 'B' is suggested, if there exist a user 'X' that bought both 'A' and 'B'. The suggested products should be sorted on decreasing sales count (i.e., most 'popular' first); count only sales to users like 'X'.

9. *'Six degrees of separation'*: Given any two supplier names, determine their 'degree of separation', defined as follows: Two suppliers 'A' and 'B' are 1-degree away if they provide at least one product both; they are 2-degrees away if there exists an product 'C' that is 1-degree away from each of 'A' and 'B'; and so on.

10. *Final check-out*: Any time a client decides to complete her purchase does so by providing a credit card number. Information about the credit cards is stored (so that it can be reused later on).

11. *Statistics*: every week, the manager wants:
    - the list of the m (say $m = 10$) most popular products (in terms of quantity sold in this week),
    - the list of m most popular suppliers and
    - the list of m most popular postal codes goods were sent at.

12. *User awards*: At random points in time, the store manager also wants to give awards to the 'best' clients; thus, the manager needs to know the *top-m* clients who spent the most money.

IMPLEMENTATION ASPECTS:

You have absolute freedom in selecting the database engine for your project. You may use either database systems that are either product or are freely-available with GPL on the Internet. In addition, you may use any language/framework you may want including `C++,Java,Python,PHP,Django`, etc. As a matter of fact, the use of the `http://projects.spring.io/spring-framework/` is encouraged.

All the above-mentioned queries to the database could be realized via parameterized *SQL* stored-procedures and/or views. The Web user interface (UI) will present users with input-fields (Web forms) and the data are expected to be passed on to the *SQL* stored procedures. The results have to also be displayed in some manageable manner (should they are long).

You may work in any environment you wish but at the end you should be able to demonstrate your work (by bringing in or having remote access a machine). Your work has to feature a two-tier architecture that includes the database back-end and the `Web/Spring/GUI/PHP` interface.

Last but not least, your database relation are expected to be loaded with a sufficiently "large" number of data tuple.

OVERVIEW OF PROJECT PHASES:

There are two distinct phases in this project that you will need to work on:

◇ Design and Implementation

In this phase, you are required to sketch (possibly a E/R-diagram) and generate the precise database schema you need based on the problem definition. You must also provide the SQL queries and implement the stored-procedures and views—the latter if deemed necessary. You must also be able to create at least two types of database-users with appropriate query/update permissions; they are: the *customer*s and the *manager/admin*.

It is a good idea to have one stored-procedure for each operation described in System Requirements. For instance in addressing requirement 6, you may create a stored-procedure named

```
ProductBrowsing(supplier_nane, name, description, prod_group)
```

The types of the parameters are not displayed here as they are left to your discretion.

◇ Web-Based User Interface

You will use the framework of your choice (such as *Spring*, *PHP*, *Django*, etc.) to offer required interfaces. You will integrate/extend the Web-based user interface that consumes the stored-procedures you wrote. In this context, you will create two Web-interfaces: one to be used by managers for their work and the second by users to buy goods.

The managers' interface has to handle the following requirements: 4, 5, 11 and 12. The users' interface has to handle requirements: 1, 2, 6, 7, 8 and 10. Moreover, requirements 3 and 9 are present in both interfaces.

COOPERATION:

You may either work individually or pick at most *one partner* for this project. If you pick a partner you should let me know who this person is during the week of November 10th, 2014.

REPORTING:

The final *typed* project report (brief report) must consist of:

1. A final schema design of the database used along with justification for your choices.
2. A parameterized stored-procedure for each of the requirements presented in the System Requirements section.
3. A code listing of your function implementations, with comments describing design choices.
4. Sample output from each of your functions based on input parameters chosen by you (if applicable).
5. Code listing for you interface application (if applicable).
6. Sample snapshots of the interface.

Finally as mentioned earlier, you will have to demonstrate the system.