DM505 Databases

Spring 2006 Project

Department of Mathematics and Computer Science
University of Southern Denmark

April 19, 2006

Introduction

The purpose of this project is to try in practice the process of creating a relational database application. This process includes design in the ER-model, transfer to the relational model, normalization of relations, implementation in a DBMS, and programming of an application controlling user interaction with the database.

The project should be done in groups of two persons. PostgreSQL should be used as DBMS, and Java with JDBC should be used for programming.

System Requirements

The subject of the project is an electronic cookbook. The idea is to keep information about recipes and kitchen inventory in a system which can suggest recipes, make list of things to buy, and ensure a minimum inventory at all times.

At least the following objects should be modeled in the system:

Recipe Required features: name, list of ingredients, number of persons served, the actions for making the dish, and the preparation time (divided into the actual working time and the rest (i.e. baking, boiling, and other things where the cook is not active)).

Ingredient Required features: name, price.

Menu Required features: list of recipes.

Minimum Inventory Required features: list of ingredients and their allowed minimum amount.

Current Inventory Required features: list of ingredients and their current amount.

(Note: the above is the specification from the user—the objects may or may not be entities in the ER-diagram you develop.)

Several minimum inventories can be envisioned to exist, for use in different situations (e.g. everyday, Christmas time, guests in house), whereas there is only one current inventory (most houses have only one kitchen).

Scaling the preparation time for recipes in a realistic way is not trivial. The approach which should be taken here, is for each recipe to store the times for serving one person and for serving ten persons. This implicitly defines a linear function $f(x) = ax + b$, which should be used for calculating the time when serving other numbers of persons.[1] Such information should be stored for both actual working time and the rest of the preparation time (i.e. store two times two values). When preparing several dishes, e.g. for a menu, sum the actual working times, but take the max of the rest times (assuming these can parallelize). Sum the two resulting values to find an estimate for the total preparation time.

At least the following functionality should be provided by the application:

[1] The slope of the function will differ widely among dishes—think of cooking pasta vs. making a salad.

Find recipes containing a given set of ingredients. Showing the name is enough.

Print recipe scaled to a given number of persons. Show all relevant info, including the price.

Find menus possible to make for a given number of guests while fulfilling any subset of the following constraints (specified by the user as part of the query): doable with the current inventory, max preparation time, max price.

Write out list of things to buy to be able to make a given recipe or menu, given the current inventory and a number of persons to serve.

Write out list of things to buy to ensure a given minimum inventory.

In a real system, facilities for updates of course is necessary, e.g. updating inventories based on the use of a given recipe, or based on shopping according to a list produced by the system, as well as manual updates of amounts of single ingredients. However, to limit the amount of programming, *no* facilities for updates should be included in the application (i.e. in the project, you add recipes, ingredients, etc. to your database simply by using `psql`). Likewise, further useful functionality is easy to envision (recipy types such as dessert, cooks with different skills in the recipies they master, guest with allergies or things they do not like, supermarkets with inventories and associated partial list of things to buy, etc.). Again, such extensions have been left out in order to limit the size of the project.

Tasks

Your tasks are:

1. To develop an appropriate ER-model of the system.
2. To transfer the ER-model to the relational model.
3. To ensure that all relations are in BCNF form (if necessary, decomposing relations, which again should lead to a refinement of the ER-model).
4. To create these relations in a database in the PostgreSQL DBMS.
5. To program (using Java and JDBC) an application controlling the user interaction with the system. The application should provide the functionality described above.

Input and Output

To limit the amount of programming, only a simple, command-line based interface for user interaction should be made. For instance, choices by the user can be input by showing a numbered list of alternatives, or by prompting for IDs of recipes, menus, etc.

Test Data

For testing purposes, material from the web can provide the basic info for recipes. The rest can just be made up. A decent amount of test data must be made (at least on the order of 12 recipes and associated ingredients, 4 menus, and 2 minimum inventories). Sharing data between groups to expand the amount of data is fine.

Formalities

In the end, a printed report of 10 to 15 pages should be produced. Its main aim should be to

- Describe the design choices made during development.
- The reasoning behind these choices.[2]
- The structure of the final solution.

Specific items that must also be included in the report are: A diagram of your ER-model, the schemas of your relations (probably in an appendix), arguments showing that these are in BCNF form, central/interesting examples (with explaination) of your SQL code, and a short user manual for the application. The emphasis of this project is not on testing, so no documentation of testing is required in the report (however, your program *will* be tried out during grading).

The report should be handed produced in stages, where the report so far is handed in, commented on and handed back, and then the report is extended in the next stage. The stages and their deadlines are:

- Task 1: Deadline *Wednesday, April 26*
- Tasks 2–4: Deadline *Monday, May 8*
- Task 5: Deadline *Wednesday, May 24*

The SQL and Java code should not be given as a printed appendix, but rather be handed in using the `aflever` command on the Imada system: Move to the directory containing your code and issue the command `aflever DM505`. This will copy the contents of the directory to a place accessible by the lecturer. Repeated use of the command is possible (later uses overwrite the contents from earlier uses). In the directory, you must for identification purposes have an ASCII file named `names.txt` containing the names of the group members, with one name per line.

You should have a copy of your final database on the PostgreSQL server of the department (at the machine `dbhost`), and your code should work on this copy (i.e. it should be possible to try out your system).

[2] To prepare for the writing of the report, it is a good idea to keep a log of the discussions within the group and of the work done.