Course Project

CS 4710 / 6710: *Database Systems*

Fall 2021

This project involves every step of the development of a database system for a real-world problem from the conceptual design, logical design, to the implementation, operation, and maintenance of a relational database and associated applications. The idea is to work in groups of 2 or 3 (3 maximum), with no more than one graduate or honors student per group. (Working by yourself, *i.e.*, in a group of size 1, is also possible, if you wish.)

Project timeline

While none of these checkpoints are graded (except for the final one), they are an opportunity for getting feedback, and to know whether you are on track or not.

- Week 9 (right after the midterm): groups decided / assigned
- Week 11: meet with me to discuss your E-R design, if you want some feedback
- Week 12: meet with me to discuss your relational design, if you want some feedback
- Week 13: (last chance to) meet with me to discuss any implementation choices (or anything else), if you want some feedback
- Week 14: project presentations / project due

Goal of this project

The goal of this project is to provide a realistic experience in the conceptual design, logical design, implementation, operation, and maintenance of a relational database and associated applications. First, I shall describe the application, then the categories of requirements, and then some suggestions on how deeply you need to go in each category. A real project of this sort would require a substantial development team working for several months (or more). The project can go well beyond the minimal requirements I outline at the end. I encourage such extensions. They could turn into a senior project or other independent work.

If your group completes the project before the final week, you may move on to possible extensions to your system, for up to 10% extra credit. I am open to many ideas, however, bad ideas will likely not earn too much extra credit, so please discuss your extension idea with me before you embark upon it. A good example is:

- Add an easy-to-use web interface to your system. Of course it will not have all of the features of, *e.g.*, Amazon's web interface, but it should simulate the experience, and have a decent set of features.

The description given here of the enterprise you are modeling is necessarily somewhat vague and incomplete. This is by design; in real life, your "customers" are managers in the enterprise whose degree of computer literacy is, to put it kindly, variable. You will need to fill in the holes in this specification document, creating a precise design and concrete implementation of the interfaces and applications using the database. The checkpoints specified for the project are designed to help you get some feedback along the way.

Enterprise description

The enterprise is a retailer, such as a department store, discount store, supermarket, convenience store, etc. Each of you will choose a specific retailer (either use a real one as your model or a made-up one). To keep the project within bounds, we will ignore issues of employees, corporate finance, etc., and focus on the retail sales end of the activities.

Your retailer sells a large variety of products at multiple stores. Not all products are at all stores. Pricing may be different at different stores, or be in different currencies (dollars, euros, etc.). Each store has its own inventory of products and needs to decide when to reorder and in what quantity. Stores may have to abide by state (provincial, regional or national) laws, *e.g.*, stores in dry counties of Tennessee may not sell alcohol, while Mountain Dew sold in Canada may not contain caffeine. Customers may identify themselves by joining your frequent-shopper program. Others may remain anonymous. Your retailer has a website that accepts orders. From a database perspective it is just a special store that has no physical location and has no anonymous customers.

The database tracks inventory at each store, customer purchases (by transaction and by customer, where possible), sales history by store, etc. Various user interfaces and applications access the database to record sales, initiate reorders, process new orders that arrive, etc.

- **The enterprise**: You may pick the enterprise that you will model. I would like to see a wide variety chosen, so here is a list (of US-based and international enterprises) to serve as a starting point to give you ideas, but other choices are welcome, indeed encouraged: Walmart, Sears, Costco, HEB, Best Buy, LIDL, Marshalls, Albert Heijn, American Eagle, Esselunga, Safeway, Walgreens, Hudson Bay Company, Game Stop, CVS, Carrefour, Petsmart, Radio Shack.
- **products**: Products come in a variety of sizes or means of packaging or labeling, *e.g.*, with appropriate warnings and recycling laws for California — or size and weight reported in metric and labeling *en français* in France. Each product has its own UPC code (the bar code that is scanned at the checkout).
- **brands**: A variety of products may be sold under the same brand (*e.g.*, Coca-cola and Coca-cola products). For such applications as reorder, specific products and sizes matter. For other applications, data may be aggregated by brand.
- **product types**: A particular type of product may be sold in a variety of sizes and a variety of brands. For example, cola is sold under such brands as Coca-cola and Pepsi. Product types form a specialization/generalization hierarchy. For example, cola is a type of soda, which is a type of beverage, which is a type of food. Some products fit into multiple categories. For example, baking soda is a cleaner, a food (since it is used for baking), and a drug (since it may be used as an antacid), but it is not a type of soda. Products may be grouped together, *e.g.*, Liberté Greek-style yogurt for \$3 / two for \$5 — or, 1 free bottle of Pepsi with the purchase of 3 bags of Lay's chips.
- **vendors**: Products are sold to stores by vendors. A vendor may sell many brands, *e.g.*, The Coca-cola Company sells: Coke, Fanta, Sprite, Nestea and Frutopia, etc.

- **stores**: Stores sell certain products, each of which has a certain inventory amount at any point in time. Stores have locations (addresses), hours at which they are open, etc.
- **customers**: Customers who join a frequent-shopper program provide some personal information based on what the enterprise requests. They may refuse to provide some information. Customers come into a store (or go online) to buy goods. Not only must this data be stored, but also the system must be able to handle multiple customers buying goods at the same time.

Data generation

For simplicity, I will not require perfectly realistic data so you do not have to ensure that, for example, if you are modeling an HEB Supermarket (Texas-based chain), it really does sell Shiner Bock beer. However, you should strive for a good degree of realism in your data (and, yes, by the way, except in dry counties, HEB sells Shiner Bock). Where appropriate, randomly generated data are acceptable (and a good way to avoid having lots of data entry to do).

Note the comments on collaboration below and that sharing data with others is acceptable as long as appropriate credit is given to your source. You may even obtain your data from the website/database of an existing supermarket, *e.g.*, "data was obtained from `www.marshalls.com`"

Client requests

1. E-R Model
    - Construct an E-R diagram representing the conceptual design of the database.
    - Be sure to identify primary keys, relationship cardinalities, etc.
2. Relational Model
    - After creating an initial relational design from your E-R design, refine it based on the principles of relational design (Chapter 7).
    - Create the relations in the database system of your choice (PostgreSQL, MySQL, Oracle, etc.).
    - Create indices and constraints as appropriate.
    - If, as you refine your design, you discover flaws in the E-R design, you can go back and change it (even if the earlier design passed the checkpoint). Your final E-R design must be consistent with your relational design.
3. Populate Relations
    - Include enough data to make answers to your queries interesting and nontrivial for test purposes.
    - You may find it helpful to write a program to generate test data.
4. Queries: You should run a number of test queries to verify that you have loaded your database in the way you intended. The queries listed below are those that your clients (managers from the retail enterprise) may find of interest. They may provide further hints about database design, so think about them at the outset of your work on this project.
    - What are the 20 top-selling products at each store?
    - What are the 20 top-selling products in each state (province, region)?

- What are the 5 stores with the most sales so far this year?
- In how many stores does Coke outsell Pepsi? (Or, a similar query for enterprises that do not sell soda.)
- What are the top 3 types of product that customers buy in addition to milk? (Or similar question for non-food enterprises.)
- How often (*e.g.*, a percentage) are soda (Coke) and chips (Lay's) purchased together?
- Which month(s) do frozen turkeys (cast-iron BBQ grills) sell the best?
- Find those products that are out of stock at every store in Florida.
- Show sales trends for various products over the past 3 years, by year, month and week. Then break these data down by state and/or income range of the buyer.

5. Interfaces: There are several types of users who access the database, and several applications that run on their own.
   - The database administrator (you) may use simply SQL commands.
   - Online customers need an elegant web interface to order products. However, for this project, an SQL commands interface will suffice if your web and/or GUI skills are not up to the challenge. (After all, this is a database course, not the web apps course, nor the user interface course.) Such an elegant web interface is one of the possible extensions for extra credit, of course (see above).
   - Your system may generate reorders on items automatically using triggers. Or, you may have an application that runs periodically to scan the database looking for items to reorder.
   - Vendors periodically query the database to check for reorder requests, which they fill by entering into the database a shipment, a delivery date, and the reorder purchase order or orders that are satisfied by the shipment.
   - An application records the increase in inventories resulting from the arrival of a shipment. (For simplicity here, assume that shipments arrive at the time specified by the vendor for the shipment.)
   - Each checkout register runs an application that records the items in each transaction, updates inventory, and gathers frequent-shopper data.

   These interfaces can be built as
   - Web applications using Java applets or a scripting language (extra credit option).
   - A stand-alone Java application using Swing to create a GUI (extra credit option).
   - Other GUI development tools you may know (extra credit option).
   - Since this course is not a Java course, nor a GUI course, I will accept a simple SQL commands interface. In fact, even if you are an expert in GUI development, you may want to start with a simple SQL interface and then upgrade later, for extra credit. Often a good GUI design has a solid SQL interface running in the background anyway.

6. Concurrency: The company stock will go down rapidly if the database cannot support more than one customer at a time making a purchase, or if it cannot deal with more than one item being reordered. Be sure that the transaction mechanism is providing the needed guarantees. By running the various queries and applications in separate sessions (*e.g.*, you can run multiple JBDC connections at once),

you can simulate the real-life operation of your enterprise. Test concurrency carefully: do not fire up several processes that submit customer transactions until you are sure things work (and do not scale this up to hundreds of customers or the system is likely to crash or bog down; after all, it should be able to run in a modern laptop computer).

Logistics

I find it very important to discuss your E-R design with you. During Week 11 of the course schedule, I'll be available to meet with each group to discuss your E-R diagram (should take 10 or 15 minutes).

The choice of doing an extra credit extension option is up to you. I realize that some of you are taking several project courses and may need to scale back a bit, but for those who can participate, an extra credit option will show you the challenges of a more realistic database scenario.

What to turn in

The checkpoints are not graded; they are just there to ensure that you are on-track. Usually, I find the first checkpoint requires some discussion, while the remaining checkpoints can be handled more quickly (via email, for example). Please talk to me about questions at any point, even before the relevant checkpoint.

The final version of the project is to be turned in as a single zip file on iCollege before the last day of classes. I will accept paper for the E-R diagram since we are not covering drawing tools for these diagrams, but many such tools exist.

1. E-R diagram, plus any explanatory notes. At minimum you must include all the entity and relationship sets implied by this handout. You may go beyond the minimum. Remember that the manager who defined the specifications is not computer literate so the specifications should not be viewed as necessarily being precise and complete.
2. Relational schema. It is likely for many of you that your E-R design will be sufficiently extensive that we agree that only a part of the resulting relational design will actually be implemented. This is the point where we cut implementation effort and data-entry time to something realistic for the course time frame.
3. A set of sample queries.
4. The code to implement the various interfaces. I will accept quite basic interfaces (SQL), but encourage more elegant interfaces (see extra credit option). Depending on the degree of sophistication you plan for each interface, we can agree to fewer than 7 of the interfaces requested by the client.
5. Please avoid platform-specific solutions. It is a bit hard for me to debug custom installations in the time-frame I have to grade the projects. Please check with me before making any design decisions that bind your application to a specific hardware or software platform.
6. A README file in the top-level folder that explains what is where, etc. Include usage instructions for the interfaces.
7. Everything should be in a single zip file so that when I unzip it, I can read the README file, follow the directions, and run your project.

Grading scheme

I shall use the following rough scheme for grading:

1. E-R design: 25 points
2. Relational design, including constraints and indices: 25 points
3. Data creation: sufficient quantity, reasonable realism, sufficiently "interesting": 10 points
4. User interfaces, including proper features, proper updating of the database, etc.: 30 points
5. Concurrent operation of interfaces: 10 points
6. In cases where the group comes up with an exceptional solution to a part or parts of the project, extra points may be possible
7. Extra credit option (discussed with me beforehand): up to 10 points.

The extra credit option has low point value for the amount of work involved, but that reflects the fact that it is optional.

Collaboration

Your project design and any extensions for extra credit (if you choose to go on to do one of these) is to be your own team's work. *Different teams may share data to load into your database. You may also share code that generates those data.* Please credit your source in each such case.

Your team will receive a single grade on this work on the assumption that everyone contributed equally to the project, or extra credit extension. Your team may submit a request for a different allocation of credit. Any issues in effective team collaboration should be resolved within the team if possible (that is the best "real-world" first step), and referred to me if attempted resolution fails.