CSC 411 Course Programming Project

Option 1: Retailer Enterprise

Due date: 11: 59 PM, November 6, 2024

Checkpoint	Task	Deadline
1	Web Application Development Learning (Canvas "Web Programming Materials" Module, Chapter 9, Reference books Options 1~2)	September, 13
2	Project Website Design & Initial Implementation	September, 20
3	E-R Diagram	October, 11
4	Relation Schema	October, 18
5	DB Implementation and Queries	November, 1
6	Report	November, 6

Goal:

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). You will do this alone over several weeks. I have chosen to go with individual rather than group projects because the goal of this project is for you to gain a personal appreciation of the depth and breadth of issues that go into the design of a database application, rather than to have you specialize in just one aspect (and rely on others for the rest).

The project can go well beyond the minimal requirements I outline at the end. I encourage such extensions. They could turn into a senior design project or other independent work.

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 document to create 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'll ignore issues of employees, corporate finance, etc., and focus on the retail sales 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. Each store has its own inventory of products and needs to decide when to reorder and in what quantity. Customers may identify themselves by joining your frequent-shopper program. Others may remain anonymous. Your retailer has a Web site 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 market basket 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'd like to see a wide variety chosen, so here is a list to serve as starting point to give you ideas, but other choices are welcome, indeed encouraged: Walmart, Target, J.C. Penny, Sears, Costco, BJ's, Best Buy, American Eagle, Nordstrom, Safeway, Aldi, Albertsons, Acme, HEB, Food Lion, Piggly Wiggly, Wegmans, Walgreens, Rite Aid, CVS, Longs, Superfresh, Carrefour, Tengelmann, Hankyu, Dillards, Wawa, Sheetz, Modells, Petsmart, Radio Shack. I've included some non-US enterprises on this list and encourage you to consider them, or other non-US retailers.

- **products:** Products come in a variety of sizes or means of packaging. 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. Pepsi and diet Pepsi). 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 Pepsi and Coke. 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.

- **vendors:** Products are sold to stores by vendors. A vendor may sell many brands (e.g. Pepsico sells Pepsi, Tropicana, Aquafina, Gatorade, Lay's, Doritos, Quaker, and others).

- **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 a market basket of 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 that you don't have to ensure that, for example, if you are modeling an HEB Supermarket, 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.

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 a DBMS system, such as MySQL.

   - Create indices and constraints (optional).

   - If as you refine your design, you discover flaws in the E-R design, 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 see 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?

- 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 don't sell soda.)

- What are the top 3 types of product that customers buy in addition to milk? (Or similar question for nonfood enterprises.)

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 SQL either via the command line or SQL Developer.

   - Markets run various OLAP queries.

   - Online customers need an elegant Web interface to order products. **However, for this project, a command-line interface will still be acceptable (but will minus 10 points for this item) 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.)**

   - Your system may generate reorders 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 market-basket, updates inventory, and gathers frequent-shopper data.

   These interfaces can be built as

   - Web applications using Java applets or a scripting language. A standalone Java application using Swing to create a GUI

   - Other GUI development tools you may know (but be sure they are platform independent, see note below)

   - Since this course is not a Java course, nor a GUI course, I will accept a simple command-line interface. In fact, even if you are expert in GUI development, you may want to start with a simple command-line and then upgrade later.

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 the MySQL/Other DBMS transaction mechanism is providing the needed guarantees. By running the various queries and applications in separate sessions (you can run multiple JDBC connections at once), you can simulate the real-life operation of your enterprise. Test concurrency carefully: don't fire up several processes that submit customer market baskets until you are sure things work (and don't scale this up to hundreds of customers or the system is likely to crash or bog down - it is only a P4 with 4GB of memory!)

What to turn in:

The checkpoints are not graded. Usually, I find the fourth checkpoint requires some discussion, while checkpoint 5 can often be handled quickly. Checkpoint 6 is mainly in place to ensure that the projects are on schedule. Please talk to me about questions at any point; don't wait for a checkpoint.

The final version of the project is to be turned in as a single zip file on **Canvas**. I will accept paper or scanned PDF for the ER diagram since we are not covering drawing tools for these diagrams, but PowerPoint does work fairly well for this purpose.

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 ER design will be sufficiently extensive that we agree that only a part of the resulting relational design will actually be implemented. This is something on which we'll agree before Checkpoint 5. This is the point where we cut implementation effort and data-entry time to something realistic for the course time frame.

3. You may choose not turning in a listing of all your data which can be seen online. I will do if I find it necessary. In this case, I, as DBA, will have access to your database online and will use that if your submission leaves me with some unresolved questions. Please create a DBA account for that purpose. Otherwise, you may have to upload all of your data to Canvas as well.

4. A set of sample queries and the code you wrote for each of the listed queries, and the screenshot result from running the queries.

5. The code to implement the various interfaces. (Your code should be platform independent. I will accept quite basic interfaces (command line with a modest command set), but encourage more elegant interfaces. Depending on the degree of sophistication you plan for each interface, we can agree to fewer than the 7 interfaces requested by the client.

6. 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. Platform-specific solutions will become a nightmare in the data-integration phase.

   If you use Java, please submit your java code as .java files that I can compile and run. Please do not submit them embedded in a NetBeans project or any other IDE. For other programming language, you also need to provide related instructions to guide me to compile and run your programs.

7. A README file in the top-level folder that explains what is where, etc. Include usage instructions for the interfaces.

8. 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:

I shall use the following approximate template for grading:

1. ER design: 20 points
2. Relational design (and constraints and indices, if applicable): 20 points
3. Data creation: sufficient quantity, reasonable realism, sufficiently “interesting”: 20 points
4. User interfaces, including proper features, proper updating of the database, etc.: 30 points
5. Concurrent operation of interfaces: 10 points
6. I reserve the right to give extra points for exception solutions to parts of the project.