**Final Project – Instructions**
**Due by Noon on 10th May 2024**

**General Info:**

- The project will be done in pairs.

- The teams will present a short demo of their working software and database during the final 2 or 3 classes of the semester.

- Each student will individually turn in a write-up due at noon on 10th May 2024 so that I can get the seniors grades in on time. A description of what to turn in can be found below. Please submit a zip file containing everything to Moodle.

- To keep you on track, there will be 2 check-ins: one the week of 4th Mar, and another one the week of 1st Apr (yes, I know spring break is in-between…).
    - These will be at most ½ hour long. Do not need to have tables fully designed by the first check-in.
    - Please schedule your check-ins as soon as you can, they will most likely be by Google Meet.
    - Check-ins are not graded, they're just to help keep you on track.

**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. Note that 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 outlined at the end, and such extensions are encouraged.

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 in create a precise design and concrete implementation of the interfaces and applications using the database.

**Enterprise description:**

The enterprise is a retailer, such as a department store, discount store, supermarket, convenience store, etc.; each team 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, ShopRite, Walgreens, Rite Aid, CVS, Tengelmann, Hankyu, Dillards, Wawa, Sheetz, Modells, Petsmart, Radio Shack, Tesco, etc. 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/generailzation 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 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. Do not worry about customer rewards.

- **Restocking:** How is this going to be done? From a central warehouse? Are there regional warehouse? Do vendors ship directly to each store?

**Data Generation:**

For simplicity, I will not require perfectly realistic data, however, you should strive for a good degree of realism in your data. Where appropriate, randomly generated data are acceptable (and a good way to avoid having lots of data entry to do).

**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 8).
   - Create the relations in Oracle database.
   - Create indices and constraints as appropriate.

- 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.
   - Functions to think about:
     - Adding inventory to a store or all stores
     - Remove inventory from a store or all stores
     - Shirt inventory between stores
     - Order more inventory

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 Workbench.
   - Markets run various OLAP queries.
   - Online customers need an elegant Web interface to order products. However, for this project, a command-line interface will suffice if your Web and/or GUI skills are not up to the challenge. (After all, this is a database course, not a Web Apps course, nor a 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

- Since this course is not 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. **Even if you decide to build a GUI (e.g. using Flask), focus on getting everything to work from the command line first.**

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 transaction mechanism is providing the needed guarantees. By running the various queries and applications in separate sessions (you can run multiple 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, 5 concurrent sessions will be sufficient.)

**Presentation** 30% of grade

Everyone in your group should be involved in the presentation. Each group has 15 minutes to do the presentation.

Use PowerPoint slides for everything other than the actual demo.

Your presentation should cover at a minimum:

- A description of the store
- Any business decisions that influenced the model development
- Any significant problems you ran into and how you solved them
- Show your E-R schema

    - Describe any business rules that influenced the database design
    - Describe any "tricky" situations you had to deal with, and how you solved them.
- Show your tables / relational model
    - Describe any "tricky" situations you had to deal with, and how you solved them.
- Describe any significant points about your test data, if there are any significant points.
- Descriptions of the queries that you are showing in the demo
    - Why these queries? What's the business reason?
    - And are there others that will be part of the final write-up but not part of the demo? And if so, why are they not being included in the demo?
- Do demonstrations of the queries

**Write-up to turn in** 70% of grade **Due noon, 10th May 2024**

The write up is based on the group work but should be done individually.

Things to include in the write-up:

1. A description of the store, products, customers, etc.
    **a.** Include any business rules that influenced the database design.

2. Include a description of any significant problems you encountered and how you solved them

3. 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.

4. 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 2. This is the point where we cut implementation effort and data-entry time to something realistic for the course time frame.

5. A set of sample queries.

**To Turn In**

1. The Write-up

2. The code to implement the various interfaces. Be sure to let me know of any packages that are needed beyond the standard installation. I will accept quite basic interfaces (command line with a modest command set), but I 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.

3. Do NOT turn in a listing of all your data. I can see them online if I find it necessary. *I, as DBA, will have access to your database online and will use that if your submission leaves me with some unresolved questions.*

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

5. **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. Upload the zip file to Moodle.**