Take-Home Project 1

This homework will cover several concepts that we will use in the class. Some of these concepts should be concepts you knew before INSC 484, others may be new. For some parts you will need to turn in a simple text file, for others you will have to turn in Python code.

Part 1: Relational Algebra and SQL

Given the Great British Bakeoff (gbb) database (found in Module 4 on Canvas), write the following queries using SQL. You are encouraged to test your SQL queries on the database using DBBrowser. It may also be helpful to use DBBrowser to examine the schema of the database or to look at the SQL file used to create the database to examine the structure of the database (baking_database.sql).

1. **Return the full name of bakers who placed 2nd in at least one technical.**

SELECT bakers.fullname FROM bakers JOIN technicals ON bakers.baker = technicals.baker WHERE technicals.rank == 2;

2. **Return the full name of bakers who placed 2nd in at least one technical but never placed 1st.**

SELECT fullname FROM bakers WHERE baker IN (SELECT bake FROM technical WHERE rank = 2) AND baker NOT IN (SELECT baker FROM technical WHERE rank = 1);

3. Return the full name and hometown of the bakers who used raisins in a signature dish.

SELECT bakers.fullname, bakers.hometown FROM bakers JOIN signatures ON bakers.baker = signatures.baker WHERE signatures.make LIKE "%Raisin%";

4. 4. Return the hometown of the bakers who used Hazelnut in either a signature dish or a showstopper dish.

SELECT bakers.hometown FROM bakers JOIN signatures ON bakers.baker = signatures.baker JOIN showstoppers ON bakers.baker = showstoppers.baker WHERE (signatures.make LIKE "%Hazelnut%") OR (showstoppers.make LIKE "%Hazelnut%");

5. 5. Return the full name and episode eliminated of all bakers who were eliminated before episode 5.

SELECT bakers.fullname, episodes.title FROM bakers JOIN results ON bakers.baker = results.baker JOIN episodes ON results.episodeid = episodes.id WHERE results.episodeid < 5 AND results.result = "eliminated"

6. Return the first name and showstopper makes of each star baker for all episodes.

SELECT results.baker, showstoppers.make FROM results JOIN showstoppers ON results.baker = showstoppers.baker WHERE results.result = "star baker" AND results.episodeid = showstoppers.episodeid;

Part 2: Relational Database Design

1. Turn the following table into a normalized relational database. Justify your design choices. Turn in a text document or word document of your normalized tables with a brief paragraph on your design choices.

Item id	**Item name**	**Cost**	**Current Market Value**	**Manufacture id**	**Item Storage Box Number**
1	2015 Panini Black Gold Nikola Jokic	$400	$900	1	B1
2	1969 Topps Wes Unseld	$500	$500	2	B2
3	2019 Upper Deck Young Guns Makar Card	$130	$400	3	H1
4	1969 Topps Lew Alcindor	$450	$455	2	B2

Manufacture id	**Manufacture name**	**Manufacture location**

1	Panini	5325 FAA Blvd Suite 100, Irving, TX 75061
2	Topps	1 Whitehall Street New York, NY 10004
3	Upper Deck	33 Unknown Street Carlsbad, CA 92008

Item Storage Box Number	**Storage Box Description**
B1	Modern Basketball
B2	Vintage Basketball
H1	Modern Hockey

Looking at the table, I identified three core components of the card collection inventory: the item, the manufacturer, and the storage box. It made sense for these three components to be separate entities. For the item, I determined that cost and market value should be included in the same table, with IDs for the manufacturer and storage box to link everything together. I created a key for the manufacturer, and I used the storage box number as the primary key for the storage box table since no two boxes can share the same number. With this design, there are no partial or transitive dependencies, and the relational database is in 3NF.

2. Turn the following table into a normalized relational database. Justify your design choices. Turn in a text document or word document of your normalized tables with a brief paragraph on your design choices.

Plant name	**Plant type**	**Plant water schedule**	**Plant produces food**	**Days Alive**	**Location id**	**Contact id**
Carl	Cactus	None	No	100	1	1
Chilly	Cayenne Pepper Plant	Monday	Yes	10	2	2
Spiderman	Spider plant	Tuesday, Thursday	No	35	3	1
Plant to the Plant	Tomato plant	Monday, Friday	Yes	15	2	2

Location id	**Plant location**	**Location Sunlight Time**

1	Kitchen Counter	4 hours
2	Front porch	6 hours
3	Home Office	2 hours

Contact id	**Plant water person**	**Plant water person contact**
1	Bob	913-875-0987
2	Linda	913-876-0932

At first, I had a hard time deciding which components of the table should be separate entities, but once I determined which columns depended on one another, it became easier. I noticed that sunlight time depends on the plant's location, and the water contact information depends on the plant's water person. I also determined that the water schedule, whether or not the plant produces food, and the number of days alive all depend on the plant itself. Based on this, I created keys for the location and the contact person and added them to the plant table. With this design, I believe the database is in 3NF.

Part 3: Software Design and Python

in this part of the homework, you will build a small Python application that does the following:

Your customer wants an application that simulates coin flipping, dice rolling, and card drawing. The

application has the following requirements:

- The application must take in a user's choice of flipping a coin, rolling a dice or drawing a card.
- The outcome of their choice should be returned to them, and the program should let them make another selection.
- When selecting to roll a die, the user must have a choice between a 6-sided, 8-sided, 10-sided, or 12-sided die.
- When selecting to draw a card, the user must have a choice to remove joker cards from the deck or not.

In addition to turning in the code for this application, you will turn in a short set of text justifying your code organization and design choices. This can be in a separate file or documented as a block comment at the top of your main.py file. Feel free to use our dice

simulation example from class for inspiration. Keep in mind the principles of modularity and that code is best maintained when broken down into simple, single concept functions.

For this program, I started by establishing simple functions for each of the three components of the gaming center: flipping a coin, rolling a die, and picking a card. I initially had only one function for rolling a die, with the choice being determined inside the function itself. However, I realized that based on the principle of modularity, that choice should instead be handled in the main file. I also recognized that each type of die is its own entity and should be handled with separate functions, so I created a different function for each die size.

For picking a card, I considered whether to simply generate a random integer within the length of the deck or to explicitly establish a deck and then draw a card from it. I determined that the more organized and efficient method would be to create a function to build a deck, with or without jokers, and then a separate function to pick a card, using the deck as an argument. Similar to the dice, I handled user input for whether jokers should be included in the main file. I also created a normalization function to ensure consistency in user input.

I believe my program can still be simplified further. I'm not sure if this would follow best practices in software development, but creating functions to handle choices directly might be an option for the future. While this could lead to a large number of functions, it might also make the code more streamlined and easier to maintain.