Noa Ben David (300950565)

Yonatan Greenshpan (204266191)

HW 3 – PRACTICAL DATABASES

Question 1 – Database intro [10 points]:

The scenario we will refer to is that of a particularly innovative flower shop in Tel Aviv that decided to use the services of drones. The store delivers flowers that are in its inventory to the entire city. In addition, over time, the store decided to use the "dead time" of the drones it bought to make food deliveries. Unlike the flower deliveries in the store, in food deliveries, the drone must collect the food from a business that has agreed to process it with the store. We assume that the food business can always supply the delivery. The store has a relatively small number of drones, so it is crucial to perform their tasks efficiently and quickly. For the benefit of this purpose, the store owner decided to test two different navigation algorithms: NavGar and NavFree. The store owner wants to expand her business and is looking for new opportunities. Recently, to attract new customers, the store owner decided to give a discount to the partner of all her customers.

We will define the entities and relations in our model as follow:

Entities:

1. Drone
2. Navigation Algorithm
3. Costumer
4. Product
    a. Food
    b. Flowers
5. Flowers inventory
6. Food Business

Relations:

1. Costumer is **married** to another customer - **Unary**
2. Costumer **made an order** of food or flowers - **Binary**
3. Flowers **available** at flowers inventory - **Binary**
4. Food is **sold in** the food business - **Binary**
5. Drone **operated by** navigation algorithms - **Binary**
6. Drone made a **delivery** of the product to a customer – **Trinary**
7. Drone **visited** a customer – Binary
8. Drone **carried** a product - Binary

We defined more relations than entities, as required.

Based on the entities and the relations we defined, we will create this scheme:

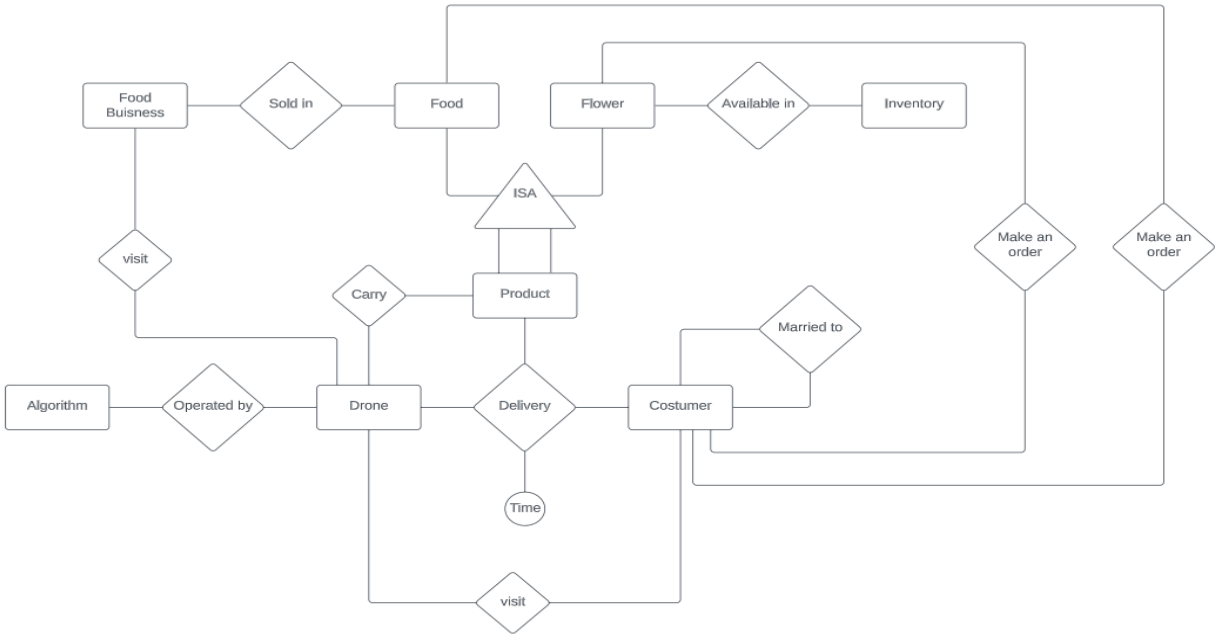
1. Drones (DroneID, Manufacturer, Model, PayloadCapacity, AlgoUses(FK))
2. Algorithms (AlgoName, License, NeedInternet, BasedGPS)
3. Costumers (ID, FullName, Gender, Address, DateOfBirth, PartnerID(FK))
4. Products (CatalogNumber, Type, Price, Weight)
   (The *type* attribute determines rather it is subclass *Food* or *Flowers* – exclusive generalization)
   a. Foods (FoodID, CatalogNumber(FK), Calories , Vegan, Cosher)
   b. Flowers (FlowerID, CatalogNumber(FK), Name, Color, LightNeed, WaterNeed, Inventory)
5. Orders (CostumerID, TimeDate, CatalogNumber, DroneID (FK, TotalPrice)
6. Food Business (BuisnessName, Address, Phone)

Regarding the primary key in all the entities, it is straightforward, except for order, food business, and product:

1. An **Order** can't be represented by the customer ID and the catalog number, because it is possible for the same person to make some different orders for the same catalog number. Therefore, we need the date too as part of the primary key.

2. To implement the exclusive generalization, we made the primary key of Products a foreign key in both Foods and Flowers. It allows them to share attributes like prices and weight but have unique attributes like "Is kosher?" and "water needed?".
3. In the **food business**, we added the address to the key to be able to distinguish between two branches of the same chain store.
4. In addition, even though the **flower inventory** is a distinct entity, it is included as a column in the Flowers table due to our business needs. It is unnecessary to store a separate record for each flower, as it suffices to keep track of the total number of flowers in our inventory within the Flowers table.

Question 2 – ERD [30 points]:



We may note some considerations regarding this ERD:

1. The core relation here is the trinary delivery relation. We had some debates about how this relation should be defined: what are the entities that take place in it, and what should be the primary key of the corresponding table? We decided that while the product, customer, and drone are the entities that are part of the relation, the time is an attribute of this relation, and it allows us to define the primary key over customer, time, and product.
2. Although it may look like the separation of Flower and Food into two different entities is artificial, it has a business logic regarding the supply chain. While flowers should be managed in an inner inventory of the owner of the shop, in the case of food, we don't need to manage the inventory because the drone goes to the available business.
3. The unary relation of being married is important to determine discount. For the sake of simplicity, we ignored divorce.
4. For the sake of better visualization, we didn’t put attributes of entities on the chart.

Executives Summary

Here we will summarize our work and the main conclusions from the process. We had three main stages:

1. Defining the scheme according to the scenario
2. Implementing the scheme and generating the data
3. Exploring the data

In this part, we will focus on stages 2 and 3.

Scheme and Data Generation

First, we defined some global variables that we will be able to draw from the relevant data (last_names, first_names and street_names). After that, we made some additional functions that will help us to actually generate the data (generate_unique_ids, check_above_18, generate_random_full_name, generate_address). In addition, we wrote the function delete_table_if_exists to make sure we can rerun cells of creating and inserting data without resetting the kernel.

After that preparation, we create a server named conn and a corresponding cursor object c to be able to make SQL commands. Following that, we created all the tables we needed in addition to inserting relevant data. In the Orders and Costumers tables, we used additional loops. In customers, we first inserted all the relevant records with the PartnerID as null. After that, we made an additional loop that uses UPDATE to add the PartnerID in a manner that part of the IDs in PartnerID will be actually a real ID from the customer table. In addition, we had to verify by the dates that only customers above 18 have Partner IDs. The next complex table to create was Orders, in which we needed to make sure that every customer had made some orders of existing items with valid Catalog numbers on a date that makes sense. In this stage, we made some assumptions about the data for simplicity, mainly in the way we generated most of the numeric variables from uniform distribution and chose arbitrary rates for the percentage of married.

Exploring the Data

In this analysis, we thoroughly examined our dataset and derived key insights from it, with the aim of identifying potential business implications. We conducted 7 straightforward queries that resulted in the following observations:

1. We tried to check which specific drones did many orders, and we found it to be drone 5, the Mavic Mini of DJI Operated by the NavGar experimental algorithm with 22 orders.

2. Conversely, we found that drone 1, the Mavic Air 2 of DJI operated by the NavFree algorithm, did the worse with ten orders only. It may be related to the new algorithm but need further investigation.
3. We also analyzed the revenue generated by each algorithm and discovered that the new algorithm significantly outperformed the old algorithm with a revenue of $1149 compared to $446.
4. To gain insight into customer preferences, we analyzed the data to determine if there were any trends or tendencies among men and women when it came to ordering flowers or food. Our analysis revealed that both men and women tend to order more flowers. However, we acknowledge that this may be due that we have only three food shops.
5. We conducted a self-join on the Customers table to determine the number of customers eligible for a discount, which involved having a partner who is also a customer. We found that five customers met this criterion.
6. Using the LIKE operator and the wildcard %, we found that the number of orders made to Ben Yehuda Street in Tel Aviv is 50.
7. Lastly, we analyzed the data to determine the customers' favorite food, which turned out to be pizza.

Finally, we made two more complicated queries for the bonus section (queries themselves and outputs attached in the appendix):

1. **COALESCE and merging Columns** – we used COALESCE and merging columns joined from different tables. We needed to merge columns where null values existed in order to find the names of the products with the highest revenue. This was necessary because the product catalog number is a foreign key in two different tables, and a traditional join would result in two columns with null values (FoodName and FlowerName). By using COALESCE, we were able to determine that the products with the highest revenue were Orchids, Sunflowers, and Pizza.
2. **LAG and Window Function** – we used the LAG function and a window function in order to determine if when people make more orders then the mean gap between orders is decreased. We retrieved the previous order's timestamp for each customer using the LAG function and calculated the time difference in days. After grouping the results by the number of orders per customer, it became apparent that as the number of orders increases, the average time between orders decreases. This outcome is logical given the way we randomly drew the data uniformly in a bounded range of dates.

Appendix – Bonus Queries and Outputs

```
c.execute('''
SELECT O.CatalogNumber, COALESCE(F.Name, FD.Name) AS ProductName, P.Price * COUNT(*) AS TotalRevenue
FROM Orders O
JOIN Products P ON O.CatalogNumber = P.CatalogNumber
LEFT JOIN Flowers F ON O.CatalogNumber = F.CatalogNumber
LEFT JOIN Foods FD ON O.CatalogNumber = FD.CatalogNumber
GROUP BY O.CatalogNumber
ORDER BY TotalRevenue DESC
LIMIT 3
''')

pd.DataFrame(c.fetchall(), columns=['Catalog Number','Product Name', 'TotalRevenue'])
```

	Catalog Number	Product Name	TotalRevenue
0	FL006	Orchids	325.00
1	FL004	Sunflowers	246.81
2	FO002	Pizza	161.82

```

c.execute('''

SELECT OrderCount, AVG(julianday(TimeDate) - julianday(LagTime)) AS MeanOrderGapInDays
FROM (
    SELECT CustomerID, TimeDate, LAG(TimeDate) OVER (PARTITION BY CustomerID ORDER BY TimeDate) AS LagTime,
           COUNT(*) OVER (PARTITION BY CustomerID) AS OrderCount
    FROM Orders
) AS Subquery
GROUP BY OrderCount;
''')

pd.DataFrame(c.fetchall(), columns=['Number of Orders','Mean Order Gap In Days'])
```

	Number of Orders	Mean Order Gap In Days
0	1	NaN
1	2	559.027778
2	3	221.547778
3	4	119.857824
4	5	132.670313
5	6	101.477847
6	8	78.479623
7	9	78.797888
8	10	58.091898