Group 5 (Aidan Webster, Karli Young, Nop Lertsumitkul)

CS3200 Project - Phase 3

Summary and problem context

For our project, we are examining a local coffeehouse chain. The chain's executives, financial department, and HR department need records of various data in order to ensure a smooth operation of the business. Our solution is a database that keeps track of all the stores in the chain, the employees that work there, customers that shop there, and the inventory flow in and out of each store. With all this information, there are many useful outcomes of the database; we have identified three in particular below.

Benefits/objectives of the database

1. Inventory management

The company will have to record each piece of inventory that is used in production. Comparing the monthly inventory to sales can help managers to identify which products are using the most inventory. Furthermore, if some inventory or sales are not recorded, the difference between these two records will notify managers about the mistake, which can be corrected accordingly. Losses in inventory can also be attributed to human error during the production of food/drink, which can be acted upon to encourage employees to efficiently use the inventory. Keeping track of inventory depletion rates can also help management plan the next order of raw materials depending on the current usage.

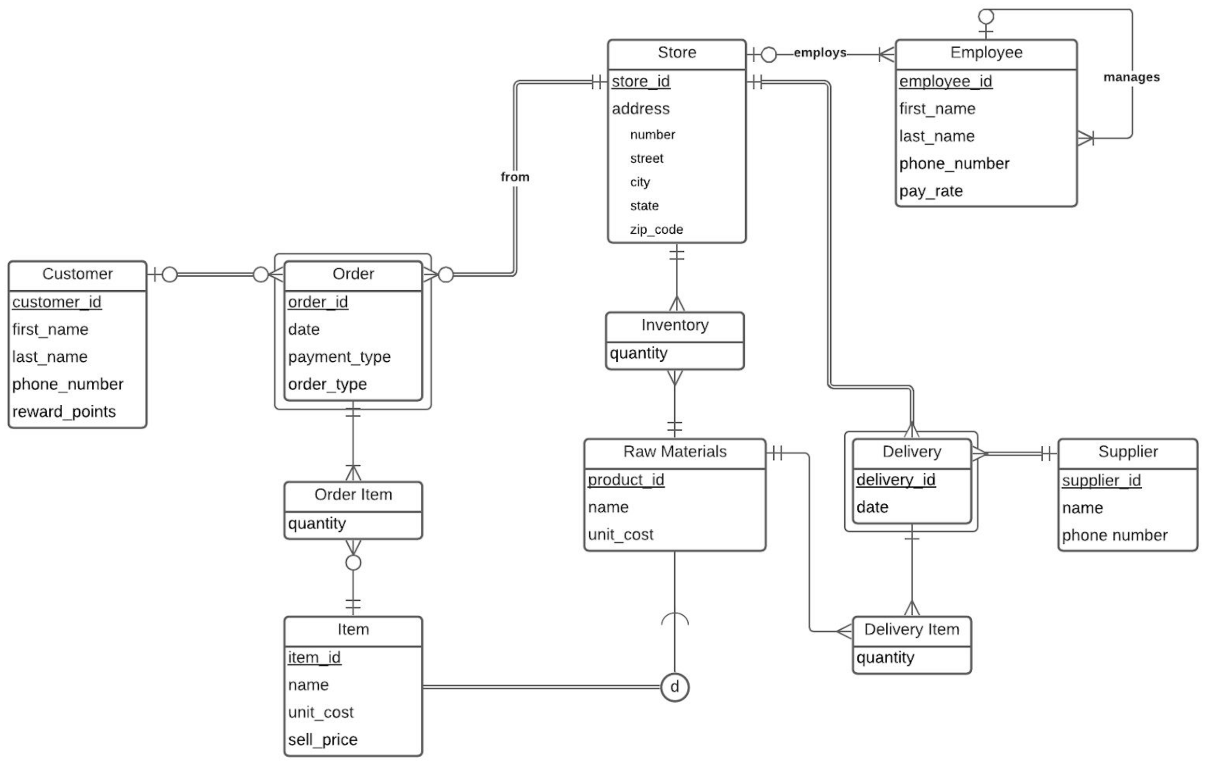
2. Comparing online vs in-person ordering

Online ordering has become immensely popular in recent years and firms are having to decide how much of the firm's resources to dedicate to further developing online capabilities vs in-person functionalities. Using this database, store executives can compare rates of in-person vs online ordering, and perform analysis on which type of order is more likely to be associated with a customer account, which type of order is likely to be larger, and which type of order is more expensive for the firm to process (credit vs cash purchases).

3. How does store performance compare across the chain?

Executives will want to compare the total costs/revenues and the performance of various products across different store locations in order to inform decisions about budget distribution, opening/closing locations, and restocking inventory (our assumption is that inventory going from suppliers to stores is planned by a subscription model where executives decide the frequency and amount of inventory going to the stores on a daily or weekly basis, and they only need to adjust their inventory order when they want to). For example, they may need to allot more of the budget to a store with much more business, in order for them to keep up or they may decide to close one location that is doing significantly worse than all the other locations.

Entity-Relationship Diagram



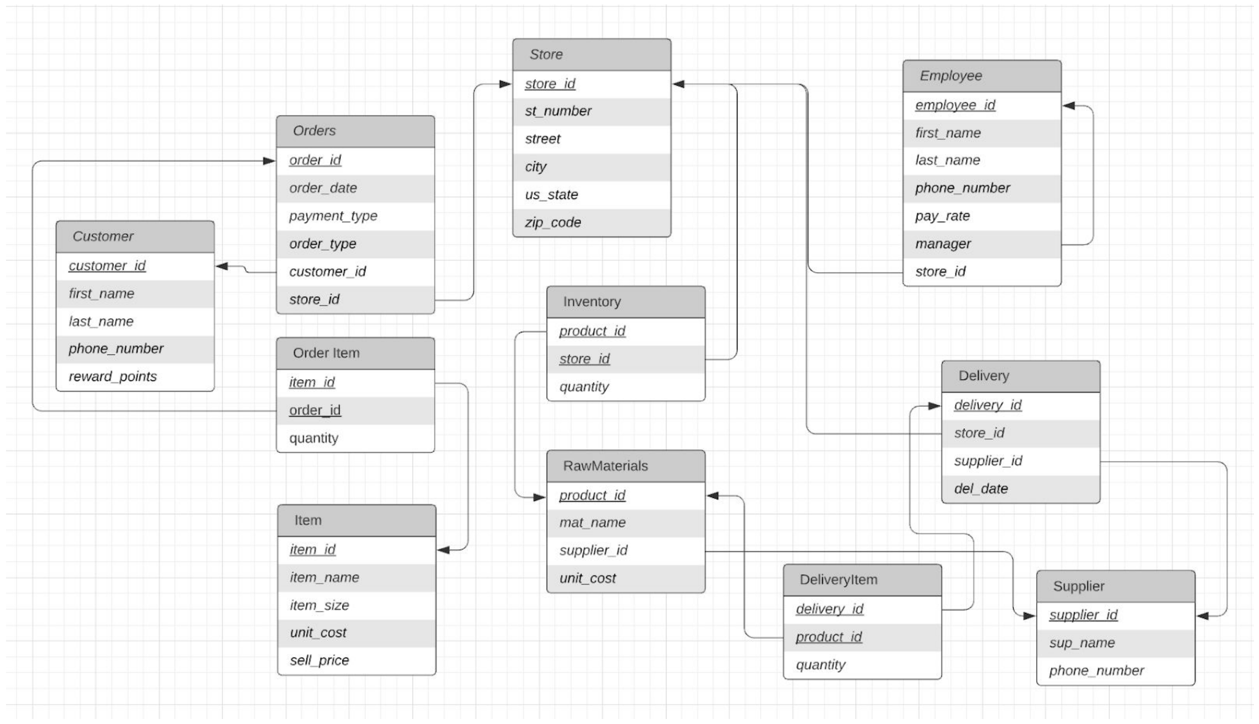
With this ER diagram, we are able to determine a wide range of relationships within a cafe chain. For example, we need to model the employee structure, customer orders, and inventory management. These relationships are centered around the store entity in this ER diagram.

A high-level description of the model: There are multiple stores in the cafe chain. Stores employ employees, some of which are managers. Each store has an inventory of raw materials (including both *ingredients* for menu items like milk, coffee beans, etc. and *prepackaged items* like bottled water). Raw materials are delivered in various quantities to each store by suppliers. We keep track of all the orders of each store, and the items sold through the order. These items might be something off the menu, like a cup of coffee, or it may be a prepackaged item (which is a raw material since it is delivered from the supplier as is, as was mentioned earlier). Some customers are kept in the system for loyalty points, and we match them to their orders if they are represented in the system.

With the way we have modelled this structure, the entities for raw materials, items, and customers are not tied to a particular store. Therefore, the entries for these tables only need to exist once, and can be referred to through relationships like being in an order at a particular store, or inventory. This reduces the amount of unnecessary data stored.

Relational Schema

Normalized 3NF:



This relational schema will help to illustrate the practical usage of the database to obtain certain information. For example, if a customer was served spoiled milk, we would be able to join tables from Customer all the way to Supplier to determine which supplier and delivery was responsible. This would allow management to alert all stores with deliveries from the same supplier on the same date of the issue, and have their inventories checked.

SQL queries that implement previous business processes

1. Inventory management
   a. Scenario 1a: Executive in charge of managing inventory levels wants to check how much of each material each store has at a given point in time. (The query checks inventory levels for store #3, but a user could theoretically input whichever store number they want to view).
   ```
   SELECT rm.product_id, i.quantity, rm.mat_name, rm.unit_cost
   FROM Inventory i
   JOIN RawMaterials rm ON rm.product_id = i.product_id
   WHERE store_id = 3
   ```
   b. Scenario 2b: Executives want to know which product has the highest in-stock inventory across all stores, as an indicator of underperformance. *(Witness query, Having clause)*

```
SELECT rm.product_id, rm.mat_name, sum(quantity)
FROM RawMaterials rm
JOIN Inventory i on i.product_id = rm.product_id
GROUP BY rm.product_id
HAVING sum(quantity) >= ALL ( SELECT sum(quantity)
                              FROM RawMaterials rm
                              JOIN Inventory i
                              on i.product_id = rm.product_id
                              GROUP BY rm.product_id )
```

2. <u>Comparing online vs in-person ordering</u>
   a. Online ordering has become immensely popular in recent years and firms are having to decide how much of the firm's resources to dedicate to further developing online capabilities vs in-person functionalities. Using this database, store executives can compare rates of in-person vs online ordering, and perform analysis on which type of order is more likely to be associated with a customer account. Here marketing teams can find how many users have chosen not to provide personal information in-person and online. *(Having clause)*

```
--Of the sample data, which order type is associated with anonymous
--customers (no customer data is maintained for reward_points)
SELECT o.order_type, count(c.customer_id)
Number_of_Anonymous_Customers
FROM Orders o
JOIN Customer c on o.customer_id = c.customer_id
JOIN OrderItem oi on o.order_id = oi.order_id
GROUP BY c.customer_id, o.order_type
HAVING c.customer_id in
       (SELECT c.customer_id as non_null
       FROM Orders o
       JOIN Customer c on o.customer_id = c.customer_id
       JOIN OrderItem oi on o.order_id = oi.order_id
       WHERE c.reward_points is Null)
```

   b. Scenario 2b: Executives want to understand which order type (in-person vs online) accounts for higher revenue and COGS.

```
--Of the sample data, how much revenue was generated by in person
--sales and online sales, and what was the cost of goods sold for each
SELECT o.order_type, sum(i.unit_cost) cogs, sum(i.sell_price) revenue
FROM Orders o
JOIN Customer c on o.customer_id = c.customer_id
JOIN OrderItem oi on o.order_id = oi.order_id
JOIN Item i on oi.item_id = i.item_id
```

```
GROUP BY o.order_type
```

c. Scenario 2c: The marketing department wants to know which customers have only utilized online ordering. *(Outer join)*

```
SELECT c.customer_id, c.first_name, c.last_name
FROM Orders o
JOIN Customer c on o.customer_id = c.customer_id
LEFT JOIN
       (SELECT c.customer_id cust_id
       FROM Orders o
       JOIN Customer c on o.customer_id = c.customer_id
       WHERE o.order_type = 'in-person') as X
       ON c.customer_id = X.cust_id
WHERE X.cust_id is Null
GROUP BY c.customer_id
```

3. <u>How does store performance compare across the chain?</u>
   a. Scenario 3a: Executives want to know how many sales were made and how many customers shop at each store. They want to be able to view the stores in order of their performance with regard to the number of sales made.

```
SELECT s.store_id, count(distinct o.order_id) numsales,
       count (distinct o.customer_id) as numcustomers
FROM Store s
JOIN Orders o on s.store_id = o.store_id
JOIN OrderItem oi on oi.order_id = o.order_id
JOIN Item i on oi.item_id = i.item_id
GROUP BY s.store_id
ORDER BY numsales desc
```

   b. Scenario 3b: Executives want to know what each store's gross revenue, cost of goods sold and net revenue was for the month of January.

```
SELECT s.store_id,
       sum(oi.quantity*i.sell_price) as gross_rev,
       sum(oi.quantity*i.unit_cost) as cogs,
       sum(oi.quantity*i.sell_price - oi.quantity*i.unit_cost) as
net_rev
FROM Store s
JOIN Orders o on s.store_id = o.store_id
JOIN OrderItem oi on oi.order_id = o.order_id
JOIN Item i on oi.item_id = i.item_id
WHERE o.order_date > '2019-12-31'
AND o.order_date < '2020-02-01'
```

```
GROUP BY s.store_id
ORDER BY s.store_id
```

Routine Updates:

1) For each point-of-sale transaction, a new tuple must be added to the database to reflect the sale. Below is an example tuple where order # is 1 + the last order number used, order data is Dec 4 of 2020, the payment type is debit, the order type in in-person, the customer who made the purchase is stored in the customer table using the id 00046313, and the transaction was at store 001.

```
INSERT INTO Orders (order_id, order_date, payment_type, order_type,
customer_id, store_id)
VALUES
(00046313, '2020-12-04', 'debit', 'in-person', 002354, 001);
```

2) Inventory quantity needs to be updated regularly at the end of every day to reflect actual stock levels for each product at each store.

```
UPDATE Inventory
SET quantity = 50
WHERE product_id = 021 and store_id = 003;
```

What we learned

This project turned out to be a lot more nuanced than we initially thought. We found that having well thought-out real world requirements from the beginning was vitally important. These requirements set the model for the entire project, so having clear requirements led to a clearer conceptual model, while ambiguous guidelines led to ambiguity in the model. Plus, finding an inconsistency or mistake in the conceptual model as a result of ambiguity later on makes the whole thing much harder to fix.

We were also challenged throughout this project to consider the practical uses/benefits and usage of a database. This helped us to make certain design decisions, like which attributes were necessary or trivial, which information is relevant to other information (to determine foreign keys), cardinalities of relationships, etc. We sometimes had to make certain assumptions, refer back to the scope often, and have discussions about benefits vs. drawbacks to inform these decisions.

Project work distribution

We each took a little bit of a lead on different aspects of the project based on what we felt comfortable with (Karli on ERD, Nop on relational schema reduction and normalization, Aidan on SQL implementation and business usages/requirements), but ultimately worked together on everything. We met with each other often and kept in constant communication to make sure we were all kept updated about the progress of the project.