**Andi Maroge**

**CSC 472 Database Design Project**

**1. Project description and purpose.**

The database design project is based on an online store that sells ground and beans coffee. The database has entity sets that include: orders, customers, employees, accounts, and products. The relations contain information that are important for the store’s operation online.

**2. Unnormalized Form. Show the unnormalized relations. Specify the name, attributes separated by commas and place in parenthesis for each relation.**

Unnormalized relations:

Customer(customer\_id, customer\_name, address, email, phone, account\_id, username, password)

Employee(employee\_id, employee\_name, address, email, phone, department)

Order(product\_id, order\_date, description, quantity, total)

Product(product\_id, brand, description, stock, price)

Here is an example:

customer values('1000', 'Tom Serra ', '25521 Paul Avenue, Warren, MI, 48092', '248-521-3948, 952-148-2482');

customer values('1001', 'John Oliver', '4958 Molly Blvd, Troy, MI, 48083', '248-451-3328, 429-385-2828');

customer values('1002', 'Steve Roach', '9485 Harbor Blvd, Costa Mesa, CA, 92624', '714-573-2849, 714-293-5397');

customer values('1003', ‘Rochelle Maij’, '1000 Pagi Street, Costa Mesa, CA, 92624', '714-573-2849, 714-293-5397');

employee values('5862748', 'John', '22475 Mongo Rd, Grossmont, CA, 92111', ‘john@gmail.com’, '248-527-2482');

employee values('5862749', 'Sara', '2135 Catalyst Rd, National City, CA, 92572', ‘sara@gmail.com’, '619-522-5343');

employee values('5862750', 'Laura', '2475 Bonza Street, La Jolla, CA, 92142', ‘laura@gmail.com’, '619-537-6762');

employee values('5862751', 'Austin', '2235 Katya Blvd, San Diego, CA, 92153', ‘austin@gmail.com’, '619-900-2858');

-For the customer relation, the relation is not atomic due to the “phone” attribute and the “address” attribute. The customer relation also has a customer’s user account information if the customer decided to create a user account in the online store. A separate table needs to be create for that to avoid problems that may arise from this design.

-The employee relation also does not have atomic values in the “phone” and “address” attributes. It is therefore not in 1NF. The employee relation also has a candidate key that other attributes have partial dependency on, which also violates the rule of 2NF.

-For the orders relation, the relation has a flaw of functional dependencies where the primary keys are not sufficient to find other attributes.

**3. Normalized Form. Perform normalization or decomposition. Eliminate any partial functional dependencies, transitive functional dependences, repetitive repetition, and anomalies. Show or explain how you perform the normalization or decomposition process.**

**Customer relation:**

Unnormalized:

*Customer(customer\_id, customer\_name, address, email, phone, account\_id, username, password)*

Normalized:

1. The customer relation first has to be made into 1NF. First, the address attribute has to be made atomic by not including multiple values in any row of the relation. Instead of “address”, we can normalize the relation by creating a separate attribute value for the street, city, state, zip.

2. The customer relation will look like this after step 1: *customer(customer\_id, customer\_name, street, city, state, zip, email, phone, account\_id, username, password).*

The relation is still not in 2NF. To further normalize it. We should separate the “zip”, “city”, and “state” into a separate relation as “zip” as the primary key and is used to find the city and state. Next, an account relation should be created and “account\_id”, “username”, and “password” should be removed from the customer relation.

The customer relation will look like this: *customer(customer\_id, customer\_name, street, zip, email, phone)*

and two additional relations are created: *location(zip, city, state)* *account(account\_id, username, password)*

3. Now the phone attribute should be put into its own relation also as:  *customer\_phone(customer\_id, type, phone)*

Final normalized relation for Customer:

*Customer(customer\_id, customer\_name, street, zip, email)*

*location(zip, city, state)*

*customer\_phone(customer\_id, type, phone)*

*account(account\_id, username, password)*

**Employee relation:**

Unnormalized:

Employee(employee\_id, employee\_name, address, email, phone, department)

Normalized:

1. The employee relation values also have to be made atomic. The employee relation’s address should include values of zip, city, and state just like the way customer was normalized.
2. After step 1, employee relation will be: *employee(employee\_id, employee\_name, street, city, state, zip, email, phone, department)*

From there, employee can be further normalized by separating the “zip”, “city”, and “state” from the relation. We can use the Location relation that we created from decomposing the Customer relation to look up the city and state by using the “zip” attribute as the primary key in that relation.

1. After step 2, the employee relation will be: *employee(employee\_id, employee\_name ,street, zip, email, phone, department)*

Next, we can decompose the relation further by creating a new relation for the phone attribute since it is a multi-valued attribute.

The table that we create will be a table that has employee phone number, the type and the primary key to find the phone number will be the “employee\_id” attribute: *employee\_phone(employee\_id, type, phone)*

1. After step 3, the employee relation will be:

*employee(employee\_id, employee\_name, street, zip, email, department)* and *employee\_phone(employee\_id, type, phone)*

Next, the employee relation can be decomposed further because there is a partial key dependency. The employee\_id attribute alone can identify all other attributes in the relation, so we can just make the “employee\_id” attribute as the primary key and remove the “employee\_name” attribute from being part of the primary key.

Final normalized form of Employee:

*employee(employee\_id, employee\_name, street, zip, email, department)*

*employee\_phone(employee\_id, type, phone)*

The relation Location created from decomposing Customer relation can be used to lookup the city and state. Creating a new relation will just cause more memory overhead.

**Order relation:**

Unnormalized:

Order(product\_id, order\_date, description, quantity, total)

Normalized:

1. For the order relation, the attributes “product\_id” and “order\_date” cannot be the primary keys of the relation. When some customer makes an order, another customer can also another order with the same product\_id and also on the same order\_date. So if these are the primary keys of the relation, there could be issues in finding the other attributes of the relation. The problem of functional dependencies in the order relation can be corrected by keeping the “product\_id” attribute in that relation, but removing it from the primary key. An “order\_id” attribute must be added into the relation to uniquely identify the other attributes in the relation.
2. After step 1: *order(order\_id, order\_date, description, product\_id, quantity, total)*

The relation now has to be made in 2NF by removing the “order\_date” attribute from the primary key. The “order\_id” attribute is sufficient for finding other attributes in the relation. This will make the relation in 2NF.

Final normalized form of Order:

*order(order\_id, order\_date, description, product\_id, quantity, total)*

**Product relation:**

Normalized:

*Product(product\_id, brand, description, stock, price)*

The relation doesn’t have problems. There will be some redundancy in repeating brands in the product relation, but not all redundancy can be eliminated from the relational database. The product relation has a good design since each attribute can be uniquely determined by the “product\_id” attribute. There are also no non-atomic values and no partial dependencies on the candidate key.

**4. Show all the schemas of the resulting or normalized relations in your database. Specify the name, attributes separated by commas and place in parenthesis for each relation. Underline the primary key(s).**

*customer(customer\_id, customer\_name, street, zip, email)*

*location(zip, city, state)*

*customer\_phone(customer\_id, type, phone)*

*account(account\_id, username, password)*

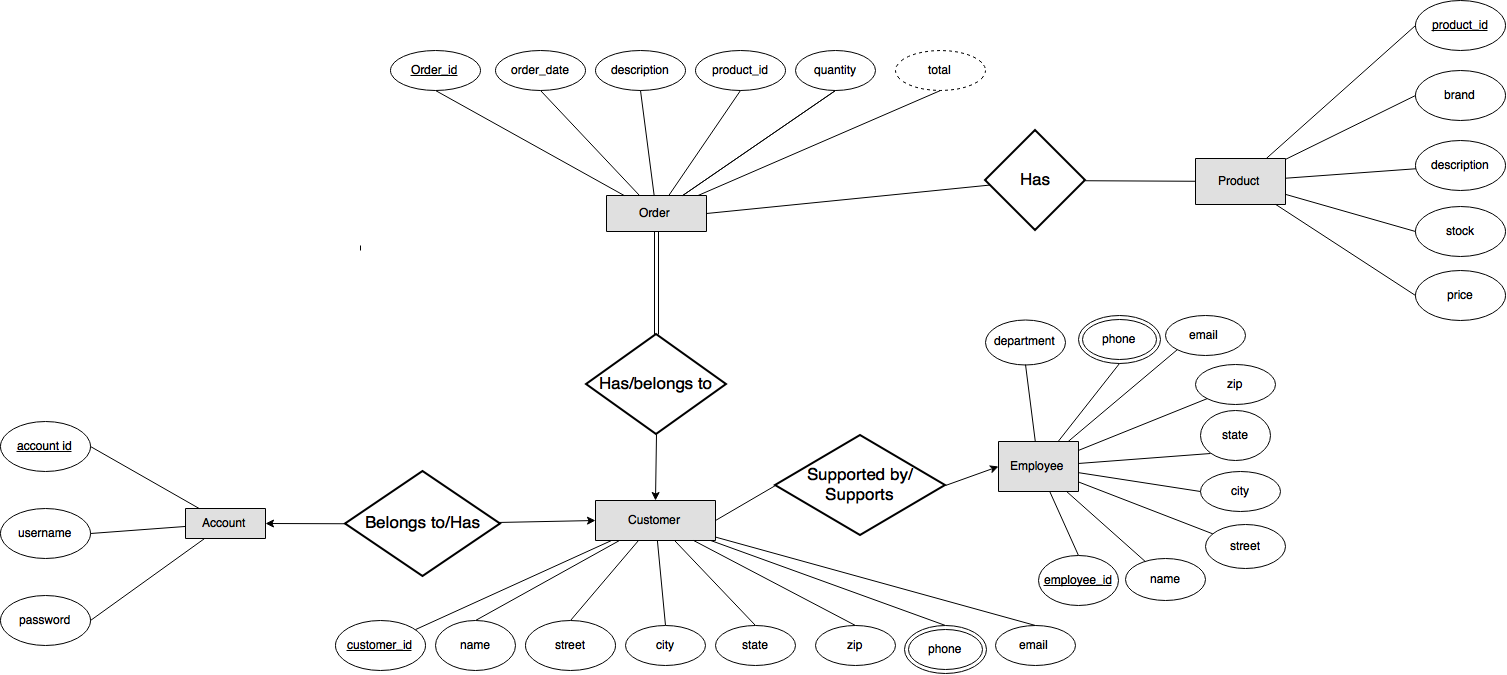
*employee(employee\_id, employee\_name, street, zip, email, department)*

*employee\_phone(employee\_id, type, phone)*

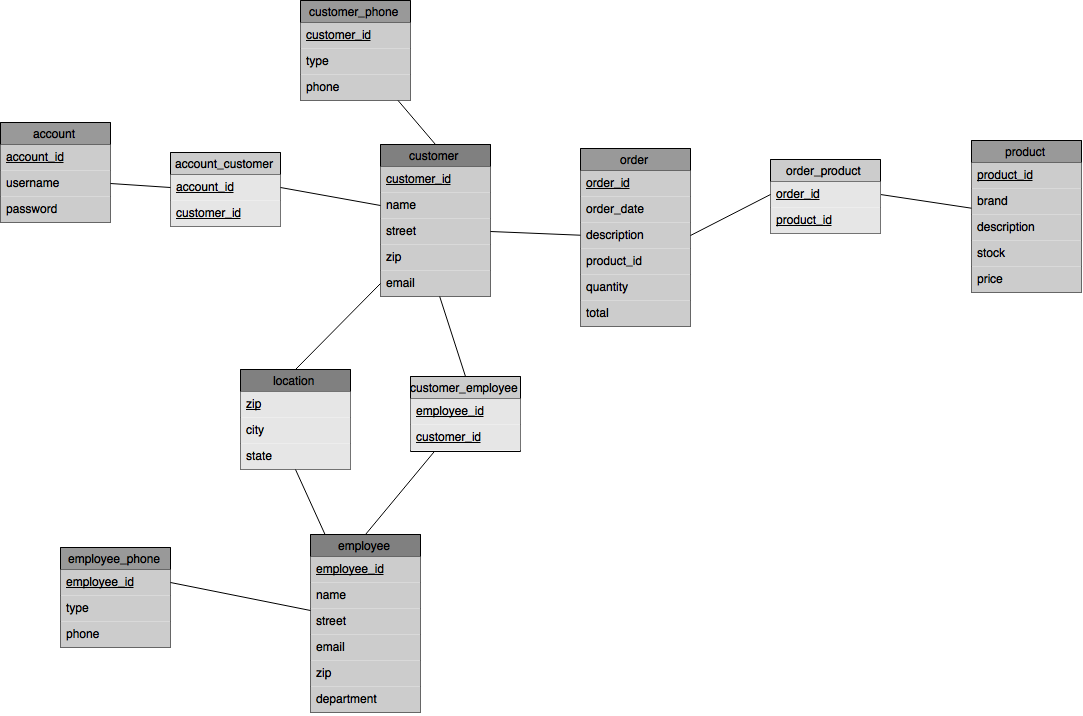
*order(order\_id, order\_date, description, product\_id, quantity, total)*

*product(product\_id, brand, description, stock, price)*

**5. Basic ER diagram. Produce a basic ER diagram of your database which reflects entity sets, attributes, and relationships among them. Underline the primary key on the ER diagram. An example is shown in below.**



1. **Relational Database Schema. Convert the ER diagram into a relational database schema. Underline primary keys.**



**7. Algebraic Statements. Formulate any two algebraic queries that are relevant to the application you have chosen.**

*1. Select all products that are of brand Starbucks*

**σ**brand = “Starbucks” (product)

*2. Select all customers who live in the city of Warren.*

**Π**customer\_name (**σ** city = “Warren”) (customer ∞ location)

*3. Select the brands where the price is greater than 5.99.*

**π**brand(**σ**price > 5.99 (product))

**8. SQL Statements. Formulate any two queries in SQL statements that are relevant to the application you have chosen.**

*1. Find the brand and description of an item where the stock of the product is greater than 10.*

Select brand, description FROM product WHERE stock > 10;

*2. Find the customers that are being supported by an employee.*

SELECT customer.name FROM customer, customer\_employee WHERE customer.customer\_id = customer\_employee.customer\_id;

9. **XPath Expression. Convert three algebraic queries into XPath expressions.**

*1. Select all products that are of brand Starbucks*

//product[brand = ‘Starbucks’]/

*2. Select the brands where the price is greater than 5.99.*

//product[price > 5.99]/brand

*3. Find the brand and description of an item where the stock of the product is greater than 10.*

//product/[stock > 10]/brand | //product/[stock > 10]/description

**10. Create Relations. Create two relations using SQL statements including constraints, i.e. show the SQL commands how to create all the relations.**

DDL for the customer and order relations:

CREATE TABLE customer (

customer\_id NUMBER NOT NULL,

name VARCHAR(15) NOT NULL,

street VARCHAR(35) NOT NULL,

zip CHAR(5) NOT NULL,

email VARCHAR(30) NOT NULL,

primary key(customer\_id));

CREATE TABLE order (

order\_id NUMBER NOT NULL,

order\_date DATE,

description VARCHAR(35),

product\_id NUMBER NOT NULL,

quantity NUMBER,

total NUMBER,

primary key(order\_id));

**11. Populate Relations. Insert three records in at least two relations, i.e. show SQL commands on how to populate the relations.**

*Customer*:

insert into customer values( '1000', 'Tom Serra', '25521 Paul Avenue’, ‘48092', ‘serrat@gmail.com’);

insert into customer values( '1001', 'John Oliver', '4958 Molly Blvd’, ‘48083', ‘oliverj@gmail.com’);

insert into customer values( '1002', 'Steve Roach', '9485 Harbor Blvd’, ‘92624', ‘roachs@gmail.com’);

insert into customer values( '1003', 'Rochelle Maij', '1000 Pagi Street’, ‘92624', ‘maijr@gmail.com’);

*Order*:

insert into order( ‘1512’, ‘08/18/2015’, ‘Ground Decaf Coffee’, ‘11843’, ‘2’, ’15.20’);

insert into order( ‘1513’, ‘08/27/2015’, ‘Whole Bean Columbia Coffee’, ‘12948’, ‘1’, ‘5.99’);

insert into order( ‘1514’, ‘09/02/2015’, ‘Ground Regular Coffee’, ‘12323’, ‘1’, ‘8.99’);

insert into order( ‘1515’, ‘10/20/2015’, ‘Ground Mocha Coffee’, ‘49184’, ‘3’, ’17.34’);

**12. Briefly describe what were the least and most challenging parts of the above and explain why. Give any recommendation about the project.**

The least challenging part of the project was creating SQL queries for the relations. It is probably because a lot of practice was done on the previous assignments on SQL queries. The most challenging part of the project was creating diagrams and normalizing them to design the database. Normalizing was the toughest part of the project because the purpose of normalization is to reduce redundancy and make access to values easier, and when there is unnecessary decomposition, memory can be wasted. Overall, the project was a good learning experience. It enforced what I read from the book and forced me to apply the concepts. I don’t have any recommendations. It was a good project to complete for learning.