Assignment #1: Relational Database Construction

Shakeeb Tahir

500837388

Dr. Christopher Daniel

SA8902 – Database Management and Spatial Technologies

1. Normalization

Normalization is the process of breaking a table or relation with more than one theme into a set of tables such that each has only one theme (Kroenke et al., 2020, p. 88). In the BracerInc excel spreadsheet, we can see a variety of themes in the table are present and that the table must be normalized in order to design an effective database. Initially, through examining the fields of the table, we can see that there are 4 major themes present. Product, customer, order, and review. Fields such as “Product ID”, “Product Name”, “List Price”, etc., indicate the need for a product table. “Customer ID”, “Customer Name”, etc., indicate the need for a customer table. “Order ID”, “Quantity”, “Order Date”, etc., indicate the need for an order table, and finally “Review ID”, “Rating”, etc., indicate the need for a review table.

It is also important when designing our database that we identify the candidate keys, the primary keys, and any functional dependencies. For a bit of background, a primary key is the field (or a combination of fields (composite key)) which is used to uniquely identify each record in a table. Candidate keys are all the unique fields within a table, and as such are eligible to become primary keys. Functional dependency refers to the relationship between attributes in which one attribute (or group of attributes) determines the value of another attribute in the same table (Kroenke et al., 2020). The attribute that determines the value of the other attribute in the table is called the determinant.

To begin normalization, we must first identify all the candidate keys. Looking through the spreadsheet, we can see that most of the fields such as “Product Name”, “Product Color”, “Rating”, “Order Date”, etc., are not unique as they contain or could contain duplicates. Typically, ID fields are generally unique and in the spreadsheet table we have 4 ID fields, “Product ID”, “Order ID”, “Customer ID”, and “Review ID”. However, upon closer examination, we can see that “Product ID” and “Customer ID” are not unique as we have multiple orders of a product, and we could also have the same customer order again. “Review ID” is also dependent on an order being placed and we cannot have a composite key of “Order ID” and “Review ID” because in some cases there might not be a “Review ID” at all if the customer chooses not to leave a review. As such, the only candidate key is “Order ID”.

Next, we must identify all functional dependencies. We can see that “Order ID” determines all the attributes in the table as a product is dependent on an order being placed, and since each order is only for one customer, it also determines all attributes about the customer. Since the reviews are dependent on the customer and the product, they are also inherently dependent on “Order ID”. We can show the functionality dependency as follows:

Order ID → (Product ID, Product Name, Product Category, Product Color, Cost, List Price, In Stock, Customer ID, Customer Name, City, Postal Code, Quantity, Total Price, Credit card, Order Date, Shipping Date, Completed, Review ID, Rating, Review, Review Date)

Furthermore, we see additional functional dependencies within the table, specifically for "Product ID", "Customer ID", and "Review ID". They can be represented as follows:

Product ID → (Product Name, Product Category, Product Color, Cost, List Price, In Stock)

Customer ID → (Customer Name, City, Postal Code, Credit Card)

Review ID → (Rating, Review, Review Date)

As such we can now split the original orders table from the spreadsheet into 4 different tables. We will now have an "Order" table, "Product" table, "Customer" table, and "Review" table. The primary keys for each table will be their ID fields as they are unique within that table. We must also now link foreign keys between the tables. The foreign key in the "Order" table will be "Customer ID" and "Product ID" since each order requires a product and a customer, so we need a way to reference them. The "Product" table and the "Customer" table do not require any foreign keys as they are independent and do not need to know anything about the order or review. The "Review" table requires foreign keys for "Product ID" and "Customer ID" since a review requires a customer and a product.

These table can now be represented as such, with an underline referencing the primary key, and italics representing a foreign key:

ORDER (Order ID, *Product ID*, *Customer ID*, Quantity, Total Price, Order Date, Shipping Date, Completed)

PRODUCT (Product ID, Product Name, Product Category, Product Color, Cost, List Price, In Stock)

CUSTOMER → (Customer ID, Customer Name, City, Postal Code, Credit Card)

REVIEW→ (Review ID, *Product ID*, *Customer ID*, Rating, Review, Review Date)

2. Modification Problems

Insertion Problems:

The insertion problems within the BracerInc spreadsheet are quite apparent. For instance, many of the same products are repeated and would need to be inserted over and over again while with our database, we would already have all the information about a specific product in our product table. This would reduce the need for manually inputting the same information repeatedly and would save lots of time while also helping to avoid mistakenly typing incorrect information.

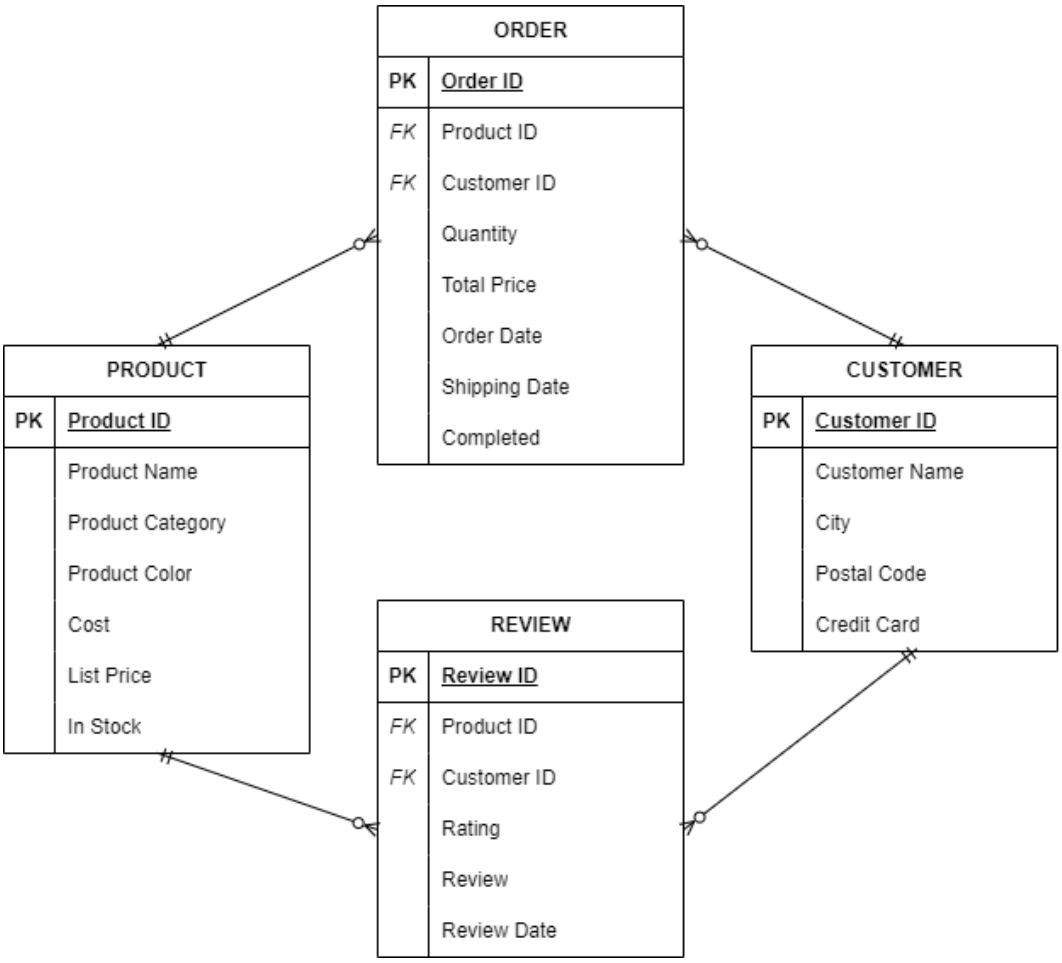
Deletion Problems:

The BracerInc spreadsheet also faces a lot of deletion problems. For example, if we wanted to delete an order such as if a customer were to cancel the order, we would scan through the spreadsheet and first find the order and then delete it from the table. This could also mess up the rows in the spreadsheet as we would now have an empty row which we would need to get rid of. This would not be a problem in our database. Another problem is that say we only have one order for a specific item and no other orders for that item, if that order were to then be deleted, we would lose all information about the product such as its color, price, category, etc. The same follows for information about a customer if we only have one instance of that customer, deleting that row would cause us to lose all information about them. Our database solves this problem such that even if information in one table is deleted, we would still keep information in the other tables. For instance, if a customer cancels an order and we delete information about the order in our order table, we can still keep their contact information in our customer table.

Update Problems:

The update problems are also quite noticeable in the BracerInc spreadsheet. For instance, if a customer decided to change their order, we would then have to go in and change all the information about the product. The same problem would exist if the customer edited their review, we would have to then go in and change all the information about their review. This could potentially lead to errors being made when trying to change the date and is very time-consuming. This is solved by our database design since if a customer were to change their order, we would just have to change the “Product ID”, instead of all the information about the product like we would have to in the excel table.

3. E-R Diagram



4. E-R Diagram Explanation

The order table contains an “Order ID” as the primary key since it is the unique way to reference the order. It has “Product ID” and “Customer ID” as foreign keys since an order depends on a customer ordering a product so we need a way to reference those two table. The customer table and product tables only contain their unique identifiers as the primary keys and do not need foreign keys to any of the other tables since they do not need information about them as they are independent from them. The review table contains a “Review ID” as the primary key in order to uniquely identify each review and “Product ID” and “Customer ID” as foreign keys since a review depends on a customer giving feedback for a product so we need to access information about the customer and the product.

A product can have zero or many orders as the company can carry many products but it does not mean that they will get order for them. So in this case it has a minimum cardinality of zero and a maximum cardinality of many.Meanwhile an order needs to have a product and can only have one product at a time so the order has a minimum cardinality of one and a maximum cardinality of one.

A customer can also have zero or many orders as a customer can exist but does not need to have an active order so it has a minimum cardinality of zero and maximum of many. However, each order needs a customer and cannot have more than one customer so it has a minimum cardinality of one and a max of one.

A product can have many review but could also have zero review so it has a min of zero and a max of many. However a review needs to be for one product and can only be for one product so it has a min of one and a max cardinality of one.

A customer can also leave many reviews but could also have zero reviews so it has a min of zero and a max of many. The review however needs one customer and can only be from one customer so it has a min of one and a max of one.

5. Data Dictionary

PRODUCT:

Column Name	Data Type (Length)	Key	Null Status	Default Value	Remarks	Description
ProductID	Integer	Primary Key	Not Null	None		Unique ID for each product
ProductName	Char(100)	No	Not Null	None	100 characters enough for a product name	Name of the product
ProductCategory	Char(50)	No	Not Null	None	50 characters enough for a product category	Category of the product
ProductColor	Char(25)	No	Not Null	None	25 characters enough for a color	Color of the product

Cost	Numeric(7,2)	No	Not Null	None	Numeric(7,2) gives max value of 99999.99, product not likely to cost more than this	Cost of the product
ListPrice	Numeric(7,2)	No	Not Null	None	Numeric(7,2) gives max value of 99999.99, product not likely to be listed for more than this	List price of the product
InStock	Char(3)	No	Not Null	None	Yes or No input	Tells whether product is has stock or not

ORDER:

Column Name	**Data Type (Length)**	**Key**	**Null Status**	**Default Value**	**Remarks**	**Description**
OrderID	Int	Primary Key	Not Null	DBMS Supplied	Initial Value = 1; Increment = 1	Unique ID for each order
ProductID	Int	Foreign Key	Not Null	None	REF: PRODUCT	Unique ID for each product
CustomerID	Int	Foreign Key	Not Null	None	REF: CUSTOMER	Unique ID for each customer
Quantity	Int	No	Not Null	None		How much of the product has been ordered
TotalPrice	Numeric(7,2)	No	Not Null	None	Numeric(7,2) gives max value of 99999.99, order not likely to exceed this much	Total price of the order
OrderDate	Date	No	Not Null	None	Format: yyyy-mm-dd	Date order was made
ShippingDate	Date	No	Not Null	None	Format: yyyy-mm-dd	Date order was shipped
Completed	Char(3)	No	Not Null	None	Yes or No input	Tells whether order has been completed or not

CUSTOMER:

Column Name	Data Type (Length)	Key	Null Status	Default Value	Remarks	Description
CustomerID	Integer	Primary Key	Not Null	None		Unique ID for each customer
CustomerName	Char(75)	No	Not Null	None	75 characters enough for a name	Name of the customer
City	Char(40)	No	Not Null	None	40 characters enough for a city	City of the customer
PostalCode	Char(7)	No	Not Null	None	7 characters for a postal code including space	Postal code of the customer
CreditCard	Char(16)	No	Not Null	None	16 characters in a credit card	Credit card number of the customer

REVIEW:

Column Name	Data Type (Length)	Key	Null Status	Default Value	Remarks	Description
ReviewID	Int	Primary Key	Not Null	DBMS Supplied	Initial Value = 1; Increment = 1	Unique ID for each review
ProductID	Int	Foreign Key	Not Null	None	REF: PRODUCT	Unique ID of the product
CustomerID	Int	Foreign Key	Not Null	None	REF: CUSTOMER	Unique ID of the customer
Rating	Int	No	Not Null	None		Rating given by customer
Review	Nvarchar(MAX)	No	Not Null	None	Nvarchar(MAX) since review can be any length long	Review description
ReviewDate	Date	No	Not Null	None	Format: yyyy-mm-dd	Date review was given

6. SQL Server Query Files

1)

```
SELECT CustomerName, City, PostalCode
FROM CUSTOMER
ORDER BY CustomerName;
```

2)

```
SELECT CustomerName, TotalPrice
FROM CUSTOMER INNER JOIN dbo.[dbo.ORDERS_]
ON CUSTOMER.CustomerID = dbo.[dbo.ORDERS_].CustomerID
WHERE TotalPrice < 200
ORDER BY TotalPrice DESC;
```

3)

```
SELECT CustomerName, ProductName
FROM CUSTOMER
INNER JOIN REVIEW
ON Customer.CustomerID = Review.CustomerID
INNER JOIN PRODUCT
ON REVIEW.ProductID = Product.ProductID
WHERE REVIEW.Rating = 4;
```

4)

```
SELECT CustomerName, PostalCode, ProductName
FROM dbo.[dbo.ORDERS_]
INNER JOIN CUSTOMER
ON dbo.[dbo.ORDERS_].CustomerID = CUSTOMER.CustomerID
```

```
INNER JOIN PRODUCT
ON dbo.[dbo.ORDERS_].ProductID = PRODUCT.ProductID
WHERE CUSTOMER.City <> 'Toronto';
```

5)

```
SELECT OrderID, CustomerName, Product.ProductID, ProductName, ShippingDate
FROM dbo.[dbo.ORDERS_]
INNER JOIN CUSTOMER
ON dbo.[dbo.ORDERS_].CustomerID = CUSTOMER.CustomerID
INNER JOIN PRODUCT
ON dbo.[dbo.ORDERS_].ProductID = PRODUCT.ProductID
WHERE dbo.[dbo.ORDERS_].Completed = 'No';
```

6)

```
SELECT ProductName, ProductCategory, Cost
FROM PRODUCT
WHERE InStock = 'No';
```

7)

```
SELECT Customer.CustomerID, CustomerName, ProductName, Quantity, ListPrice
FROM PRODUCT
INNER JOIN dbo.[dbo.ORDERS_]
ON PRODUCT.ProductID = dbo.[dbo.ORDERS_].ProductID
INNER JOIN CUSTOMER
ON CUSTOMER.CustomerID = dbo.[dbo.ORDERS_].CustomerID
WHERE CUSTOMER.City = 'London';
```

References

Kroenke, D. M., Auer, D. J., Vandenberg, S. L., & Yoder, R. C. (2020). *Database concepts* (9th ed.). Pearson Education, Inc.