Relational Database Construction for

Bracer Inc.

David Cardenas Ochoa

Toronto Metropolitan University

SA 8902

Table of Contents

[Project Overview 3](#_Toc147485785)

[Normalization Process 3](#_Toc147485786)

[ER Diagram 4](#_Toc147485787)

[Tables 4](#_Toc147485788)

[Relations between Entities 6](#_Toc147485789)

[Challenges 6](#_Toc147485790)

[Data Dictionary 7](#_Toc147485791)

[Query Files 11](#_Toc147485792)

[Recommendations 15](#_Toc147485793)

# Project Overview

As instructed by Bracer Inc., a personal fitness company, a SQL Server database was constructed to help organize all information collected to revitalize the system and make it more manageable. Before the final SQL Server was constructed, an Entity Relation diagram (ER diagram) was created, followed by a data dictionary to produce the final product. As requested by the company, the following report will summarize the specifications of how each item was created and the significance behind each value imputed.

# Normalization Process

When analyzing the file provided by the company, it was evident that it had to be deconstructed and normalized into various entities. It is essential to understand the definitions of the terms before using them to explain the ER diagram and how it was constructed. Before normalizing data, all candidate keys must be identified to help select the best primary key for relations. The primary key is a unique identifier of the relation (Kroenke et al., 2020). These relations create functional dependencies in which one attribute or group of attributes determines the value of another attribute, causing the second attribute to be dependent on the first (Kroenke et al., 2020). This process of going through candidate keys, selecting a primary key, establishing dependencies, and creating entities is to help normalize the data. The proper definition of normalizing data is the construction of one or more relations such that in every relation, the determinant of every functional dependency is a candidate key (Kroenke et al., 2020). Now, with a general explanation of how to normalize data this will help explain how the ER diagram was constructed.

# ER Diagram

With the data provided, four entities were created: ORDER, PRODUCT, CUSTOMER and REVIEW. All the attributes used in these entities use the same attributes as those provided by the company, with the only difference being that these tables are organized and show the relation from one to another.

## Tables

Starting with the ORDER table, there are nine attributes with one primary key and three foreign keys. The primary key is OrderID, a unique identifier for each order placed within the system. It can be considered a transaction number, ensuring that no two orders share the same identifier. This primary key is vital for quickly accessing and referencing specific orders when needed. The three foreign keys are ReviewID, CustomerID, and ProductID. These attributes act like a link that helps connect information between different tables in a database. Within the entity, the remaining entities are used to describe when the order was placed (OrderDate), when the order was completed (completed), the quantity that was purchased for an object (Quantity), the total price paid for an order (TotalPrice), and when the order was shipped or expected to be shipped (ShippingDate). All the attributes mentioned in this section share similar characteristics, hence being grouped into one entity.

In the following table, PRODUCT, the primary key is ProductID. It serves as a product identifier, like a barcode, ensuring that each product is distinguishable within the system. When looking at the attributes provided, an effortless way of creating this table is adding attributes with the word ‘product’ already in its name as it is given; they will describe the characteristics of a product. The remaining three attributes were added to this entity as they specify a particular product, such as how much it was bought for, if it is in stock, and how much it is sold for.

Like the previous tables, CUSTOMER and REVIEW primary keys are CustomerID and ReviewID. CustomerID is like a customer membership or account number and ensures that each customer's information remains distinct within the database. For ReviewID, it is a review number that distinguishes one review from another. This primary key allows for easy reference to specific reviews when needed. When looking at the remaining seven attributes, it becomes clear which attributes belong to which entity.

Table 1. ER Diagram of Bracer Inc. data

A diagram of a product

Description automatically generated

## Relations between entities

As explained earlier, foreign keys are primary keys in other tables relating entities together. The relationship can be demonstrated through the lines and symbols connecting different entities. Each relation can vary from one entity to another, hence why there are three main symbols: ⎢, O, ∈. ⎢is meant to demonstrate one or mandatory. One way to know the difference is that if the symbol has a symbol on each side, it means mandatory, and if there is a symbol on one side and an entity on the other, it means one. O and ∈ each have one meaning, with the first symbol meaning many and the latter meaning many.

The relationship between ORDER and CUSTOMER is "mandatory one to optional many." It means that each order in ORDER must be associated with one customer (mandatory), but a customer can have multiple orders or none (optional). Between ORDER and REVIEW, the relationship is described as "optional one to mandatory one." It implies that each order in ORDER can be associated with one review (optional), but every review must be connected to one order (mandatory). The final relation between ORDER and PRODUCT is "mandatory to optional many." It signifies that each order in ORDER must include at least one product (mandatory), but a product may or may not appear in many orders (optional).

## Challenges

Although the data provided had all the necessary information, three specific challenges were still faced in creating the diagram. The first challenge was choosing which key would be the best primary key to unite all the data and if it would allow the database to function as needed. The second challenge was understanding the relationship between each entity and which constraints were needed when connecting them. The last issue would be any modification problems that may occur at any given time when updating the table. With the first mentioned obstacle, it was not easy to understand if the database would still work with OrderID as the primary key because one product needed an OrderID since it was out of stock. However, because no one ordered the item, it would not affect the database when retrieving data. The second challenge was more tedious than anything, as the data had to be carefully reviewed to ensure the correct symbols were used. To fix the last issue, the primary and foreign keys were used so that whenever one record was updated or deleted, the rest followed suit to avoid unnecessary and most accurate data. This issue needed fixing as with the current system, whenever a customer would change credit card numbers, only one record might be updated while the other may not, making this a necessary challenge that had to be fixed.

# Data Dictionary

Normalizing the data provided, creating various entities, and establishing relations through primary and foreign keys allows for the next step in database creation. In this step, the data dictionary is a centralized document that contains detailed information about the data used in a database. It serves as a reference guide that provides data about data. In other words, it explains what the data means, how it is structured, and how it is used within a database or system.

Table 2. Data Dictionary for the ER Diagram

ORDER (OrderId, ReviewID, CustomerID, ProductID, OrderDate, Completed, Quantity, TotalPrice, ShippingDate)

* An ORDER instance can be any order that has been placed to “Bracer Inc.” by a customer

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Column Name** | **Data Type (Length)** | **Key** | **Null Status** | **Default Value** | **Remarks** | **Description** |
| OrderID | Integer | Primary Key | Not Null | DBMS supplied | Surrogate key | Unique identifier for each order |
| ReviewID | Integer | Foreign Key | Null | DBMS supplied | REF: REVIEW | Unique identifier for each review |
| CustomerID | Integer | Foreign Key | Not Null | DBMS supplied | REF: CUSTOMER | Unique identifier for each customer |
| ProductID | Integer | Foreign Key | Not Null | DBMS supplied | REF: PRODUCT | Unique identifier for each product |
| OrderDate | Date | No | Not Null | None | Format: yyyy:mm:dd | When an order was placed |
| Completed | Char(3) | No | Not Null | None |  | Whether the order is completed or not |
| Quantity | Integer | No | Not Null | None |  | Amount ordered |
| TotalPrice | Numeric(9,2) | No | Not Null | None |  | Total amount |
| ShippingDate | Date | No | Not Null | None | Format: yyyy:mm:dd | When the order was shipped |

PRODUCT (ProductID, ProductName, ProductCategory, ProductColour, Cost, ListPrice, InStock)

* A PRODUCT instance can be any one product that is being sold by “Bracer Inc.”

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Column Name** | **Data Type (Length)** | **Key** | **Null Status** | **Default Value** | **Remarks** | **Description** |
| ProductID | Integer | Primary Key | Not Null | None | Surrogate key | Unique identifier for each product |
| ProductName | VarChar(100) | No | Not Null | None |  | The name of the product sold |
| ProductCategory | Char(25) | No | Not Null | None |  | Category of the product |
| ProductColour | Char(25) | No | Not Null | None |  | Colour of the product |
| Cost | Numeric(9,2) | No | Not Null | None |  | How much an item was bought for |
| ListPrice | Numeric(9,2) | No | Not Null | None |  | How much an item is sold for |
| InStock | Char(3) | No | Not Null | None |  | Whether or not there is stock for an item |

CUSTOMER (CustomerID, CustomerName, City, PostalCode, CreditCard)

* A CUSTOMER Instance can be any person who has purchased a product from “Bracer Inc.”

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Column Name** | **Data Type (Length)** | **Key** | **Null Status** | **Default Value** | **Remarks** | **Description** |
| CustomerID | Integer | Primary Key | Not Null | None | Surrogate key | Unique identifier for each customer |
| CustomerName | VarChar(50) | No | Not Null | None |  | Customer’s Name |
| City | Char(35) | No | Not Null | None |  | Customer’s city |
| PostalCode | Char(7) | No | Not Null | None | Format: A#A #A# | Customer’s postal code |
| CreditCard | Char(16) | No | Not Null | None |  | Customer’s credit card number |

REVIEW (ReviewID, Rating, Review, ReviewDate)

* A REVIEW instance can be any review that has been done by a “Bracer Inc.” customer

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Column Name** | **Data Type (Length)** | **Key** | **Null Status** | **Default Value** | **Remarks** | **Description** |
| ReviewID | Integer | Primary Key | Not Null | None | Surrogate key | Unique identifier for each review |
| Rating | Integer | No | Not Null | None |  | Customer’s rating for a product |
| Review | VarChar(250) | No | Not Null | None |  | Customer’s review on a product |
| ReviewDate | Date | No | Not Null | None | Format: yyyy:mm:dd | Date the customer reviews an order |

Relation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parent | Child | Referential Integrity Constraint | On Update | On Delete |
| PRODUCT | ORDER | ProductID in ORDER must exist in PRODUCT | No | No |
| CUSTOMER | ORDER | CustomerID in ORDER must exist in CUSTOMER | No | No |
| REVIEW | ORDER | ReviewID in ORDER must exist in REVIEW | No | No |

Table two shows that many variables are used and will be explained from the top tables to the bottom right to left with duplicated values skipped in the explanation. Each entity shown in the ER diagram gets its table, with the table name written out in all capitals with the attributes used inside the brackets. Underneath the table name is a description of each entity and the requirements for an attribute to be included in the entity. All the attributes Bracer Inc. provided in the column name section with their respective tables. The first attribute is always the primary key, as that is the unique identifier for the table. For data type, there were various types used. ‘Integer’ represents that the value for that attribute must be a whole number, ‘Date’ must be a date, and Numeric(#,#) specifies how many decimal places are wanted. ‘Char(#)’ means how many characters can be used in a field with the brackets specifying the character limit; if there are unused characters, the remaining characters will be spaces to fill the amount and always reach the maximum amount. ‘VarChar(#)’ represents variable characteristics like ‘Char(#).’ The only difference is that VarChar can have empty characters so that character amounts can vary in a field for one attribute. The ‘key’ attribute identifies which attribute is the primary key and the foreign key, if any, is in an entity. Null status helps identify if an attribute can be null in a record. The answer for the database created was no because they were all important and did not allow for any circumstances where they could be empty values. Default values were not needed. DBMS Supplied is a value the database comes up with for using surrogate keys. The relationship table shows the relationship between entities, with the parent value being the primary key of a one-to-many relation and the child the foreign key on the other side of a many-to-one (Kroenke et al., 2020).

# Query Files

As requested, seven queries were requested, with each providing a different result. In SQL, certain words have commands that will trigger actions within the server. The most used commands in the query are: select, from, order by, inner join, <, <>, and =. The select statement retrieves data from one or more database tables. It specifies which columns you want to retrieve from. The ‘from’ clause is used in SQL to specify the source table or tables from which data should be retrieved. The inner join statement combines rows from two or more tables based on a related column. It returns only the rows that have matching values in both tables.

*Query 1*

SELECT CustomerName, City, PostalCode

FROM Customer

ORDER BY CustomerName

This means the query will return a list of customer names along with their respective cities and postal codes, sorted alphabetically by customer name.

*Query 2*

SELECT Customer.CustomerName, [Order].TotalPrice

FROM Customer INNER JOIN [Order] ON Customer.CustomerID=[Order].CustomerID

WHERE ([Order].TotalPrice < 200)

ORDER BY [Order].TotalPrice DESC

This SQL query will retrieve data from the "Customer" and "Order" tables, specifically the "CustomerName" from the "Customer" table and the "TotalPrice" from the "Order" table, for orders with a total price less than $200. The results will be sorted in descending order based on the total price.

*Query 3*

SELECT CustomerName, ProductName

FROM [order] AS A INNER JOIN Customer ON A.CustomerID=Customer.CustomerID INNER JOIN Products ON A.ProductID=Products.ProductID Inner join Review ON A.ReviewID=review.ReviewID

Where Rating=4

In summary, this SQL query will retrieve data from multiple tables, including customer names ‘CustomerName’ and product names ‘ProductName’ for orders associated with reviews with a rating of 4.

*Query 4*

SELECT Select CustomerName, PostalCode, ProductName

FROM [Order] as a INNER JOIN Customer ON a.customerid=Customer.CustomerID INNER JOIN Products on a.ProductID=Products.ProductID

WHERE city<>'Toronto'

ORDER BY CustomerName

This SQL query will retrieve data from multiple tables, including ‘CustomerName’ and ‘PostalCode’ for orders associated with products. The query shows customers that live outside of Toronto and orders the results by customer name in ascending order.

*Query 5*

SELECT [Order].OrderID, Customer.CustomerName, Products.ProductID, Products.ProductName, [Order].ShippingDate, [Order].Completed

FROM Customer INNER JOIN [Order] ON Customer.CustomerID = [Order].CustomerID INNER JOIN Products ON [Order].ProductID = Products.ProductID

WHERE ([Order].Completed <> 'yes')

ORDER BY [Order].OrderID

This SQL query will retrieve data from multiple tables and show which orders are yet to be completed.

*Query 6*

SELECT InStock, Cost, ProductCategory, ProductName

FROM Products

WHERE (InStock <> 'yes')

ORDER BY ProductName

This SQL query will retrieve data from multiple tables and show which products are out of stock.

*Query 7*

SELECT Customer.City, [Order].CustomerID, Customer.CustomerName, Products.ProductID, Products.ProductName, Products.ProductCategory, Products.ProductColour, [Order].Quantity, Products.ListPrice

FROM Customer INNER JOIN [Order] ON Customer.CustomerID = [Order].CustomerID INNER JOIN Products ON [Order].ProductID = Products.ProductID

WHERE (Customer.City = 'London')

ORDER BY [Order].CustomerID

The query filters for customers in London and orders the results by the customer ID in ascending order.

# Recommendations

There are a few recommendations that can improve the database as the company continues to grow. The first would be to ensure an SQL code is typed in to prevent users from purchasing an item that is out of stock, as the company would not be able to provide it immediately. Secondly, the data dictionary used Char(3) for either yes or no variables. It would improve the database and reduce storage needed if ‘bit’ was used instead, with the only major challenge being keeping track of whether yes is 1 or 0.

References

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