Photo and Video Equipment Rental DBMS
Final Report

12/03/2020

Introduction

In recent times, photo and video equipment rentals have seen significant growth leading to an increasingly competitive market. The focus of businesses in this market is to help customers find ideal equipment for a variety of use cases and provide them with a rental service for these products. One of the biggest challenges facing this approach is the requirement of a sophisticated database system needed to properly store and retrieve data in a secure and efficient manner. To accomplish this goal, a variety of entities are needed in order to properly track the distribution in various ways. This includes keeping a database of the client-rental relationship, the employee-customer relationship, as well as the company's product-distribution relationship.

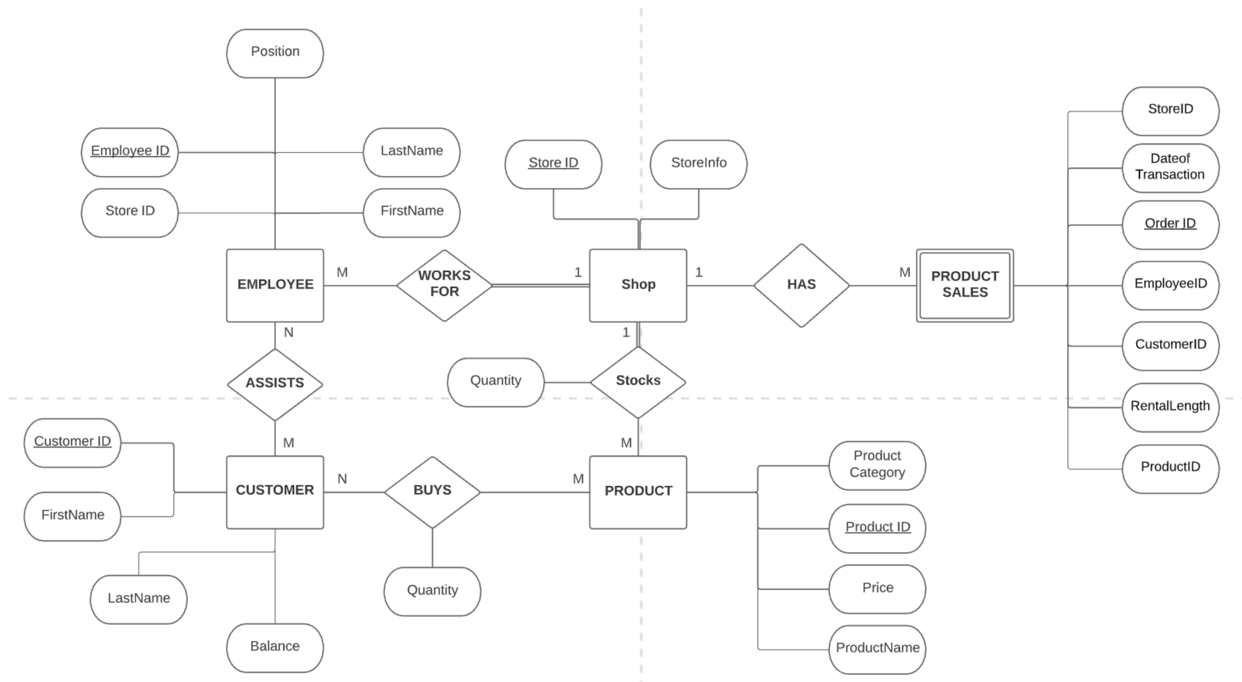
The first category that we will discuss is the side of the transaction that focuses on the customer. This category makes use of a few separate entities, mainly being the customer, product, and employee. For the customer, a unique primary key has been assigned alongside personal information, rental history, and balance owed attributes. This allows us to maintain a comprehensive understanding of each individual customer and their history. The goal of this relationship is to streamline the process for the customer. Tracking rental history allows us to easily rebook an order that was previously made. Tracking balance owed can also help build a positive relationship as a notification system can be implemented later on to help the customer keep track of their contracts and payments.

The second category will focus on the employee and customer relationship. This section makes use of the previously three mentioned entities, and adds another in the form of a product sale entity. The focus of this entity is to keep track of a unique order ID, which will include important information like the date of transaction, the customer ID, the employee who put the transaction through, as well as things like the length of the contract and the rental rate that was paid. This relationship allows the business to build an organized database of transactions which will be useful towards keeping track of a variety of attributes like sales volume. Another benefit is that if a customer has a problem, they will be able to be referred to the same employee that they made the transaction with, streamlining the process for both the business and the customer.

The final category is the product-distribution relationship. This category has a large breadth and can allow for use of new attributes for things like separate franchises, brands or manufacturers, product categories, as well as individual products themselves. The goal of this section is to be able to closely monitor distribution across various locations. This distribution includes what products each location will stock, as well as what products are currently in stock or sold out. This data is especially useful as it allows stores to monitor and adjust stock levels to better suit the needs of the individual customers in varying demographics. Manufacturers of the product will also be able to add new products, categories, and even make changes to the rates that they charge for rentals. As for the products themselves, essential attribute information is provided such as specifications, rental rate, and which stores currently have it in stock.

This highlights the general structure that we have followed when constructing the application. Following this we will present the final ER model, which has been structured to implement the features as highlighted above.

ER DIAGRAM DESIGN



RELATIONS & TABLES WITH DUMMY DATA

Examples:

Shop

StoreID → StoreInformation

STOREID	STOREINFORMATION
1	The Dundas Bay Location
2	The Yorkdale location

Employee

EmployeeID → FirstName
EmployeeID → LastName
EmployeeID → Position
EmployeeID → StoreID

EMPLOYEEID	FIRSTNAME	LASTNAME	POSITION	STOREID
1	Samuel	Khan	Manager	1
2	Bolu	Zhang	Cashier	1
3	Carol	Sinclair	Cashier	1

Customer

CUSTOMERID	BALANCE	FIRSTNAME	LASTNAME
1	724.96	Jonathan	(null)
2	765.97	Emily	Brown
3	815.93	Jonas	Rei

CustomerID → Balance
CustomerID → FirstName
CustomerID → LastName

Product

ProductID → ProductCategory
ProductID → ProductBrand
ProductID → ProductName
ProductID → Price

PRODUCTID	PRODUCTCATEGORY	PRODUCTB...	PRODUCTNAME	PRICE
1	Camera	Gucci	500D	133
2	Camera	Canon	The capturer	500
3	Camera	Nikon	the visualizer	450
4	Audio	Blue Yeti	Midnight blue microphone	75

ProductSale

OrderID → CustomerID
OrderID → ProductID
OrderID → StoreID
OrderID → RentalLength
OrderID → EmployeeID
OrderID → DateOfTransaction

ORDERID	DATEOFTRANSACTION	RENTALLENGTH	EMPLOYEEID	CUSTOMERID	PRODUCTID	STOREID
1	20-09-15	7	3	1	4	1
2	20-09-16	5	3	2	3	1
3	20-09-15	2	1	3	1	1

Stocks

StoreID, ProductID → Quantity

STOREID	PRODUCTID	QUANTITY
1	1	5
1	2	15
1	3	12

Buys

CustomerID, ProductID → Quantity

CUSTOMERID	PRODUCTID	QUANTITY
1	2	1
1	4	3
2	1	2

CROSS-TABLE RELATIONS

Employee-Shop

EmployeeID → StoreID

ProductSale-Product

OrderID → ProductID

ProductSale-Shop

OrderID → StoreID

ProductSale-Employee

OrderID → EmployeeID

ProductSale-Customer

OrderID → CustomerID

Product&Shop-Stocks

StoreID,ProductID → Stocks.Quantity

Product&Customer-Buys

CustomerID,ProductID → Buys.Quantity

Normalization of the below relation to BCNF using Bernstein's Algorithm

Our table was already in BCNF, so we have added tables 2 and 3 as shown below to our database in order to visualize the process of normalization.

R(ProductID,ProductCategory,ProductBrand,ProductName,Price,StoreID,Stocks,Buys,CustomerID)

FD's{ ProductID → ProductCategory,ProductBrand,ProductName,Price 1
ProductName → ProductID, Price 2
ProductID,ProductName → ProductCategory 3
StoreID,ProductID → Stocks 4
CustomerID,ProductID → Buys } 5

FD = {I → CBNP, N → IP, IN → C, SiI → S, CiI → Bu}

FD = {I → C, I → B, I → N, I → P, N → I, N → P, IN → C, Si&I → S, Ci&I → Bu}

I → C: I+ = {I,B,N,P} We do not get C, so not redundant
I → B : I+ = {I,C,N,P} We do not get B, so not redundant
I → N : I+ = {I,C,B,P} We do not get N, so not redundant
I → P : I+ = {I,C,N,B,P} We get P, transitive

N → I: N+ = {N,P,C,B}	We do not get N, so not redundant
N → P: N+ = {N,I,C,B,P}	We get P, transitive
IN → C : IN+ = {I,N,C}	We get C, so redundant
Si,I → S : Si+ = {Si,I,S}	We do not get S, so not redundant
Ci,I → Bu: Ci+ = {Ci,I,Bu}	We do not get Bu, so not redundant

2 is transitive and redundant(ProductID → ProductName , ProductName → ProductID)
3 is redundant

After removing redundancies,
FD = {I → C,I → B,I → N,I → P,Si&I → S,Ci&I → Bu}

All of the TRs are fully dependent, no partial dependencies.

ISiCi+ = {I,Si,Ci,C,B,N,P} this is the key.

R1(ProductID,ProductCategory,ProductBrand,ProductName,Price)
R2(ProductID,StoreID,Stocks)
R3(ProductID,CustomerID,Buys)

By looking at the updated FDs, we can see that the tables are in 3NF/BCNF.

3NF example from Assign 7:

Step 1: all tables are in the form where each cell contains a single value and each record is unique.

Step 2: all tables have a single column primary key in the form of a unique ID number.

Step 3: our table has no transitive functional dependencies.

For an example, we have added a transitive functional dependency to our Customer table

CUSTOMERID	SALUTATION	BALANCE	PERSONALINFORMATION
1	Mr.	724.96	Jonathan
2	Mrs	765.97	Emily Brown
3	Mr.	815.93	Jonas Rei

Here it has been split into a foreign key so it is 3NF

CUSTOMERID	SALUTATIONID	BALANCE	PERSONALINFORMATION
1	1	724.96	Jonathan
2	2	765.97	Emily Brown
3	1	815.93	Jonas Rei

SALUTATION	SALUTATIONID
Mr.	1
Mrs.	2

LIST OF RELATIONAL ALGEBRAS FOR ALL QUERIES

Relations

Shop(<u>StoreID</u>,storeInformation)
Employee(<u>EmployeeID</u>,FirstName,LastName,Position,StoreID)
Customer(<u>CustomerID</u>,Balance,FirstName,LastName)
Product(<u>ProductID</u>,ProductCategory,ProductBrand,ProductName,Price)
ProductSale(<u>OrderID</u>,DateOfTransaction,RentalLength,EmployeeID,CustomerID,ProductID,StoreID)
Buys(<u>CustomerID</u>,<u>ProductID</u>,Quantity)
Stocks(<u>StoreID</u>,<u>ProductID</u>,Quantity)

Queries and their RA

--Selects distinct dates where products were sold
SELECT
DISTINCT dateoftransaction
FROM ProductSale
ORDER BY dateoftransaction ASC

$$\Pi_{dateoftransaction} Productsale$$

--Shows most popular products sold

```
SELECT productid,COUNT(*)  AS "Times Sold"
FROM productsale
GROUP BY productid
HAVING COUNT(*) > 1
ORDER BY COUNT(*) DESC
```

$$\Pi_{productid,\ count(*)}(\sigma_{count(*)>1}\ \rho_{"Times\ Sold"/count(*)}\ Productsale$$

--Shows the orders fulfilled by an employee on specified dates

```
SELECT orderid, dateoftransaction
FROM productsale
WHERE employeeid = 003
AND (dateoftransaction = '2020-09-16' OR dateoftransaction = '2020-09-15')
```

$$\Pi_{orderid,\ dateoftransaction}(\sigma_{employeeid\ =\ 003\ \Lambda\ (dateoftransaction\ =\ 2020-09-16\ \vee\ dateoftransaction\ =\ 2020-09-15}\ Pro$$

--Selects all the managers

```
SELECT DISTINCT employeeid,position,storeid, FirstName,LastName
FROM Employee
WHERE position = 'Manager'
ORDER BY storeid DESC
```

$$\Pi_{employeeid,\ position,\ storeid,\ firstname,\ lastname}\ (\sigma_{position\ =\ 'manager'}\ Empl$$

--Shows how many customers are in the database

```
SELECT 'Unique Customers', count(customerid)
FROM Customer;
```

$$\Pi_{'Unique\ customers',\ count(customerid)}Customer$$

--Selects products with a given budget and category

```
SELECT *
FROM Product
WHERE ProductCategory = 'Camera'
AND price < 250
```

$$\sigma_{productcategory\ =\ 'Camera'\ \Lambda\ price\ <\ 250}\ Product$$

--Shows store exclusive items

```
SELECT productid
FROM stocks
WHERE storeid = 2
MINUS
SELECT productid
FROM stocks
WHERE storeid = 1
ORDER BY productid
```

$$\Pi_{productid}\,(\sigma_{storeid\,=\,2}\,Stocks)\;-\;\Pi_{productid}\,(\sigma_{storeid\,=\,1}Stocks)$$

--Joins the PRODUCT table with STOCKS table to show each product and the stock at each store

```
SELECT product.productid,product.ProductName,product.productcategory,
product.productbrand,product.price,stocks.quantity,stocks.storeid
FROM product
FULL JOIN stocks
ON product.productid = stocks.productid
ORDER BY storeid
```

Includes = *product.productid,product.ProductName,product.productcategory, product.productbrand,product.price,stocks.quantity,stocks.storeid*

$$\Pi_{Includes}(\,Product\bowtie_{product.productid=\,stocks.productid}\,Stocks)$$

--Joins the CUSTOMER table with BUYS table to display products purchased by each customer

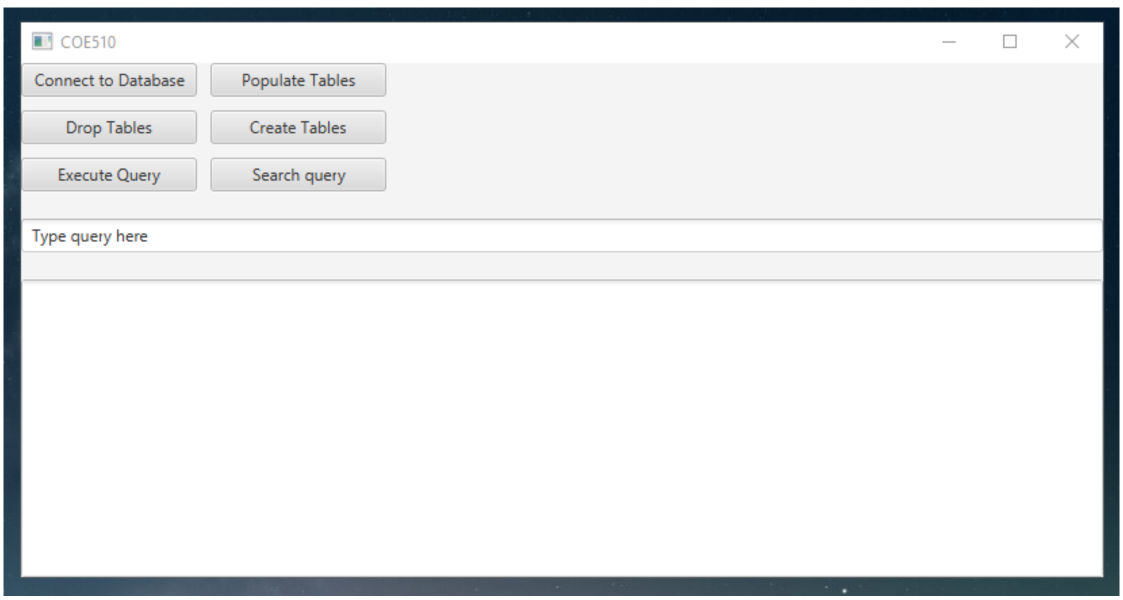
```
SELECT customer.FirstName,customer.LastName,Customer.balance,buys.productid,
buys.quantity
FROM Customer
LEFT JOIN buys
ON buys.customerid = customer.customerid
ORDER BY customer.customerid;
```

Includes = customer.FirstName,customer.LastName,Customer.balance,buys.productid, buys.quantity

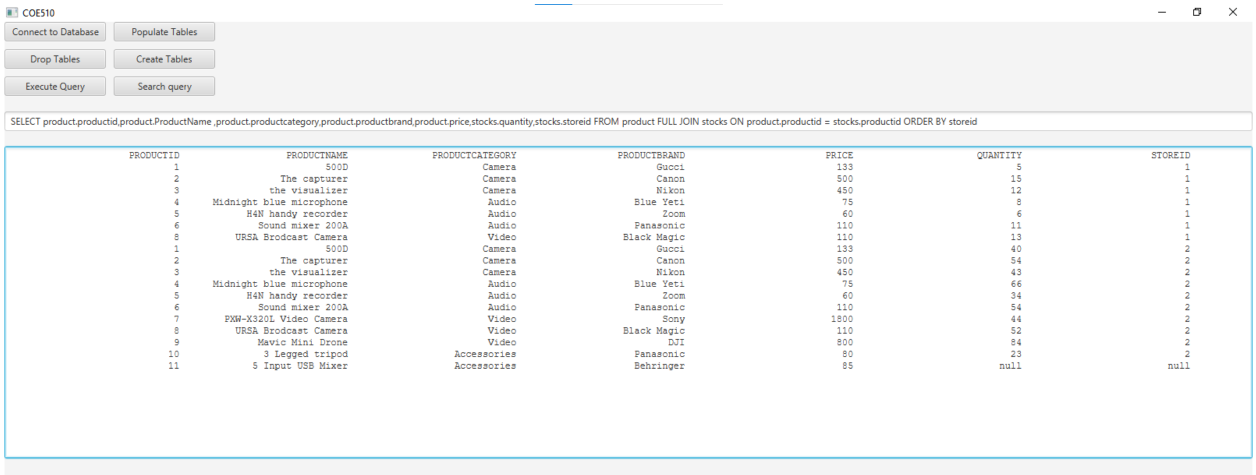
$$\Pi_{Includes}(Customer \bowtie_{buys.customerid\,=\,customer.customerid} Buys)$$

GUI PROJECT

We made a java gui application that can connect to the application, drop tables, create tables, populate tables, and execute queries. A screenshot of the gui is shown below. First, the user must click “Connect to Database” to open the connection to the server. If no data is stored, the user can click “Drop Tables” followed by “Create Tables” followed by “Populate Tables” to initialize dummy data. From there, the user can type a search query or an execution query as desired in the text bar. The result will be displayed in the text field below.

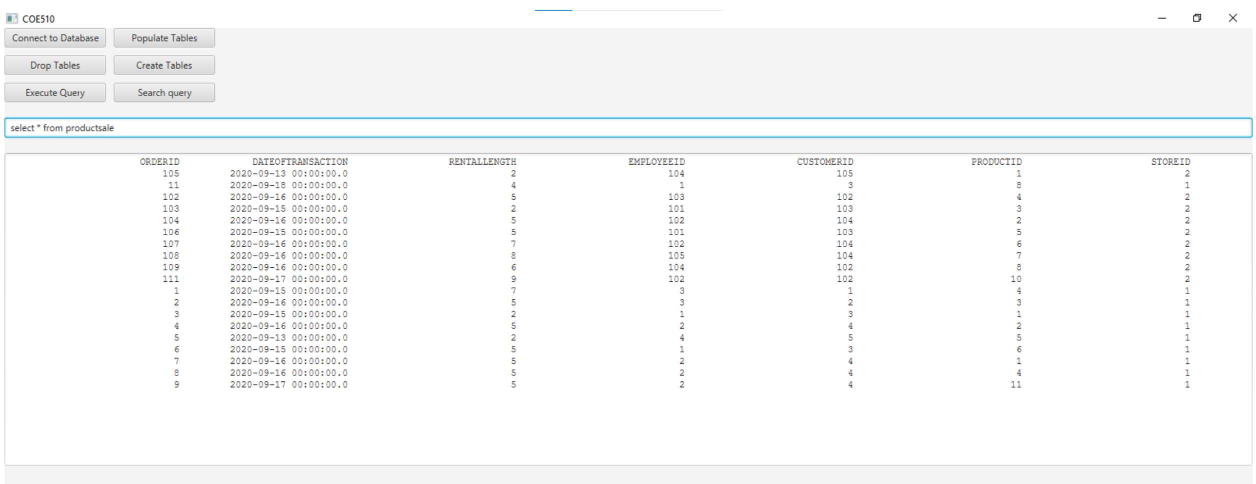


The application is submitted as a jar file so it can be run on Windows OS, along with a zip file of the Netbeans project. Below is a screenshot of it executing an advanced query and displaying results in table form.



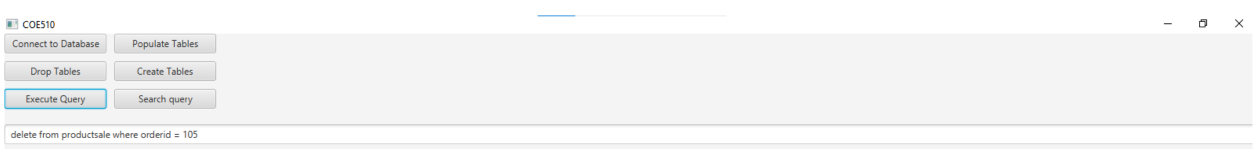
PRODUCTID	PRODUCTNAME	PRODUCTCATEGORY	PRODUCTBRAND	PRICE	QUANTITY	STOREID
1	500D	Camera	Gucci	133	5	1
2	The capturer	Camera	Canon	500	15	1
3	the visualizer	Camera	Nikon	450	12	1
4	Midnight blue microphone	Audio	Blue Yeti	75	8	1
5	H4N handy recorder	Audio	Zoom	60	6	1
6	Sound mixer 200A	Audio	Panasonic	110	11	1
8	URSA Brodcast Camera	Video	Black Magic	110	13	1
1	500D	Camera	Gucci	133	40	2
2	The capturer	Camera	Canon	500	54	2
3	the visualizer	Camera	Nikon	450	43	2
4	Midnight blue microphone	Audio	Blue Yeti	75	66	2
5	H4N handy recorder	Audio	Zoom	60	34	2
6	Sound mixer 200A	Audio	Panasonic	110	54	2
7	PXW-X320L Video Camera	Video	Sony	1800	44	2
8	URSA Brodcast Camera	Video	Black Magic	110	52	2
9	Mavic Mini Drone	Video	DJI	800	84	2
10	3 Legged tripod	Accessories	Panasonic	80	23	2
11	5 Input USB Mixer	Accessories	Behringer	85	null	null

Product sale before deletion:

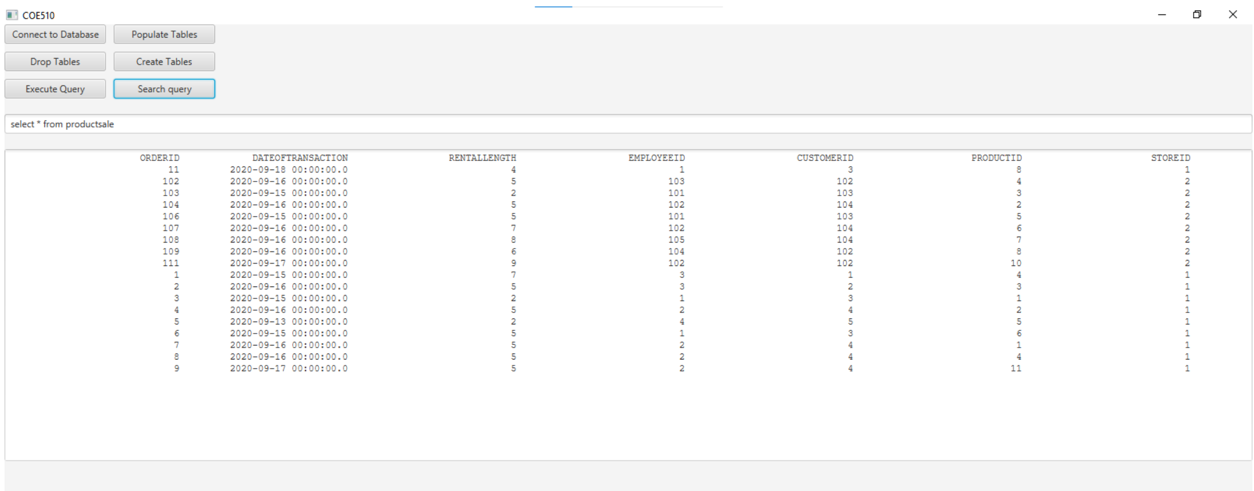


ORDERID	DATEOFTRANSACTION	RENTALLENGTH	EMPLOYEEID	CUSTOMERID	PRODUCTID	STOREID
105	2020-09-13 00:00:00.0	2	104	105	1	2
11	2020-09-18 00:00:00.0	4	1	3	8	1
102	2020-09-16 00:00:00.0	5	103	102	4	2
103	2020-09-15 00:00:00.0	2	101	103	3	2
104	2020-09-16 00:00:00.0	5	102	104	2	2
106	2020-09-15 00:00:00.0	5	101	103	5	2
107	2020-09-16 00:00:00.0	7	102	104	6	2
108	2020-09-16 00:00:00.0	8	105	104	7	2
109	2020-09-16 00:00:00.0	6	104	102	8	2
111	2020-09-17 00:00:00.0	9	102	102	10	2
1	2020-09-15 00:00:00.0	7	3	1	4	1
2	2020-09-16 00:00:00.0	5	3	2	3	1
3	2020-09-15 00:00:00.0	2	1	3	1	1
4	2020-09-16 00:00:00.0	5	2	4	2	1
5	2020-09-13 00:00:00.0	2	4	5	5	1
6	2020-09-15 00:00:00.0	5	1	3	6	1
7	2020-09-16 00:00:00.0	5	2	4	1	1
8	2020-09-16 00:00:00.0	5	2	4	4	1
9	2020-09-17 00:00:00.0	5	2	4	11	1

Deletion Query:



Product sale after deletion:



ORDERID	DATEOFTRANSACTION	RENTALLENGTH	EMPLOYEEID	CUSTOMERID	PRODUCTID	STOREID
11	2020-09-18 00:00:00.0	4	1	3	8	1
102	2020-09-16 00:00:00.0	5	103	102	4	2
103	2020-09-15 00:00:00.0	2	101	103	3	2
104	2020-09-16 00:00:00.0	5	102	104	2	2
106	2020-09-15 00:00:00.0	5	101	103	5	2
107	2020-09-16 00:00:00.0	7	102	104	6	2
108	2020-09-16 00:00:00.0	8	105	104	7	2
109	2020-09-16 00:00:00.0	6	104	102	8	2
111	2020-09-17 00:00:00.0	9	102	102	10	2
1	2020-09-15 00:00:00.0	7	3	1	4	1
2	2020-09-16 00:00:00.0	5	3	2	3	1
3	2020-09-15 00:00:00.0	2	1	3	1	1
4	2020-09-16 00:00:00.0	5	2	4	2	1
5	2020-09-13 00:00:00.0	2	4	5	5	1
6	2020-09-15 00:00:00.0	5	1	3	6	1
7	2020-09-16 00:00:00.0	5	2	4	1	1
8	2020-09-16 00:00:00.0	5	2	4	4	1
9	2020-09-17 00:00:00.0	5	2	4	11	1

Closing Thoughts

In conclusion, the database achieved all of our design goals. Looking towards the future, more advanced systems could be layered on top of the database to provide increased functionality. For example, an email notification server could be useful to help customers keep on top of their rental due dates. Integration with a website could also be useful in order for customers to be able to search stock before coming to the store. Finally, more prepared statements and text fields could be useful for database functionality in order to streamline use for less experienced users.