BJ&K

Email: BJ&K@corporations.com
Website: BJ&K.com

James River Jewelry.

March 9th, 2019

Authored by: Brendan Turner, Bryan Gonzalez Moyano, John Lee, Josh Urbach, Karen Kimmel

BJ&K

CONTENTS

Executive Summary

James River Jewelry is based in the United States and is currently one of the very few retailers that specialize in importing and vending exclusive pieces of hard-to-find Asian jewelry. Therefore, James River has the edge over its local competitors due to this defining feature of their business. Their assets along with their exceeding customer service has led them to acquire a very loyal clients along the years which is why James River Jewelry believes these customers should be awarded with a special buyer's program to further enhance customer loyalty and satisfaction. Therefore, we at BJ&K believe that it is the company's duty to make sure the company can keep track of their clients and transcend their level of customer service.

Moreover, our database solutions have the ability to Moreover, our database solutions have the ability to customize the database and is willing to work with all clientele to develop a tailored database for a company's specific needs no matter what is required especially for businesses that have special specifications. Additionally, we also offer complete and thorough training of all new users of its databases for complete optimized utilization. This guarantees the software gets in integrated and homogenized at each and all levels of the company's businesses complex.

Here at BJ&K, during the years that we have been in service with installations across the country, we have implemented and supported stand-alone database solutions that provide essential information data management that is critical to a company's work flow, utility management as well as room for future planning and implementations within the company. With our database solutions, hundreds of successful companies such as James River Jewelry now have the satisfaction of having the highest standard of performance with little to no concernment of managing their information. We are committed to lending our expertise to the clientele who make the decision to become a part of the countless businesses and companies that made the right choice to let our services become a part of them.

Stakeholders

List of individuals important to the project

Name of the Stakeholder	Role Tile
James River	Co-Owner of James River Jewelry
Jane River	Co-Owner of James River Jewelry
Nate Smith	Co-Owner of BJ & K
Brendon Turner	Designer of Customer Loyalty Database
Bryan Gonzalez Moyano	Designer of Customer Loyalty Database
John Lee	Designer of Customer Loyalty Database
Josh Urbach	Designer of Customer Loyalty Database
Karen Kimmel	Designer of Customer Loyalty Database
Jimmy Pond	Artisan of Asian Jewelry
Alex Lakes	Artisan of Asian Jewelry
Sarah Ken	Front End Clerk/Sales of Manufactured Jewelry
William Bruno	Front End Clerk/Sales of Manufactured Jewelry
Dylan Steal	Front End Clerk/Sales of Manufactured Jewelry
Justin Russ	Investor
Ana Lane	Investor
Ellen Cruz	HR Director
Mary Ayer	HR Director
Jesse Soltz	Operation Manager
Steve Chan	Admin Staff

Business Rules

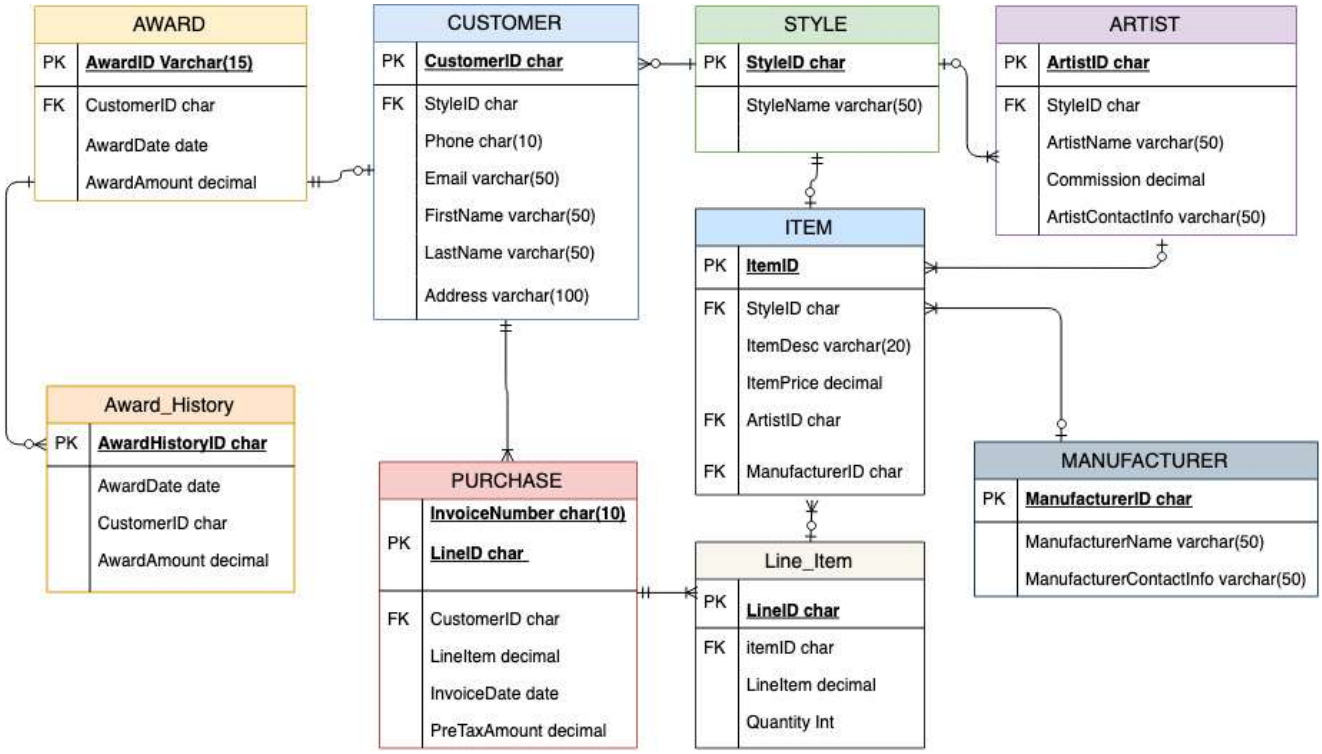
- Each artist must have at least one style
- Each customer may have anywhere from no favorite styles to many different favorite styles
- To be a customer you must have made at least one purchase
- Each manufacturer will not have an associated style
- Manufacturers do not get a commission, but artists do
- Each customer may enroll in the rewards program but do not need to in order to be considered a customer
- All sales are final, no refunds.
- Each item can only have one style associated with it
- Awards must be applied to the next purchase after the customer has received one
- Artists generally get 60% commission, but this rate may be negotiated depending on the specific piece

Assumptions

None all assumptions have become business rules based on our answered questions

Logical Model

The E-R crow's foot diagram, also known as the logical model, is what led us to the physical model shown below.



Data Dictionary

CUSTOMER

Column ID	Data Type (Length)	Key	Required	Default Value	Remark
CustomerID	Char	Primary Key	Yes	None	DBMS Supplied Initial Value = 1 Increment = 1
FirstName	Varchar (50)	No	Yes	None	
LastName	Varchar (50)	No	Yes	None	
AddressLine1	Varchar (100)	No	No	None	
AddressLine2	Varchar (100)	No	No	None	
City	Varchar (100)	No	No	None	
State	Varchar (100)	No	No	None	
Zip	Varchar (5)	No	No	None	
Phone	char (10)	No	No	None	Up to front end
Email	char (50)	No	No	None	Up to front end
StyleID	char	Primary Key Foreign Key	Yes	None	REF: STYLE

PURCHASE

Column ID	Data Type (Length)	Key	Required	Default Value	Remark
InvoiceNumber	char (10)	Primary Key	Yes	None	DBMS Supplied Initial Value = 1 Increment = 1
LineID	Char	Primary Key	Yes	None	
InvoiceDate	Date	No	Yes	None	Format: yyyy-mm-dd
PreTaxAmount	Decimal	No	Yes	None	
LineItem	Decimal	No	Yes	None	
CustomerID	char	Primary Key Foreign Key	Yes Yes	None None	Ref: CUSTOMER

AWARD

Column ID	Data Type (Length)	Key	Required	Default Value	Remark
AwardID	char	Primary Key	Yes	None	DBMS Supplied Increment = 1
CustomerID	char	Primary Key Foreign Key	Yes	None	
AwardDate	date	No	No	None	Format: yyyy-mm-dd
AwardAmount	Decimal	No	Yes	None	

STYLE

Column ID	Data Type (Length)	Key	Required	Default Value	Remark
StyleID	Char	Primary Key	Yes	None	DBMS Supplied Initial Value = 1 Increment = 1
StyleName	Varchar (50)	No	Yes	None	

ARTIST

Column ID	Data Type (Length)	Key	Required	Default Value	Remark
ArtistID	Char	Primary Key	Yes	None	DBMS Supplied Initial Value = 1 Increment = 1
StyleID	char	Primary Key	Yes	None	Ref: STYLE
		Foreign Key	Yes	None	
ArtistName	Varchar (50)	No	Yes	None	
Commission	decimal	No	Yes	None	
ArtistPhone	Varchar (50)	No	No	None	
ArtistEmail	Varchar (50)	No	No	None	
ArtistAddressLine1	Varchar (50)	No	No	None	
ArtistAddressLine2	Varchar (50)	No	No	None	
ArtistCity	Varchar (50)	No	No	None	
ArtistState	Varchar (50)	No	No	None	
ArtistZip	Varchar (50)	No	No	None	

Line_Item

Column ID	Data Type (Length)	Key	Required	Default Value	Remark
LineID	char	Primary Key	Yes	none	
ItemID	char	Primary Key	Yes	none	Ref: ITEM
		Foreign Key	Yes	none	
LineItem	decimal	No	Yes	none	
Quantity	Integer	No	Yes	none	

ITEM

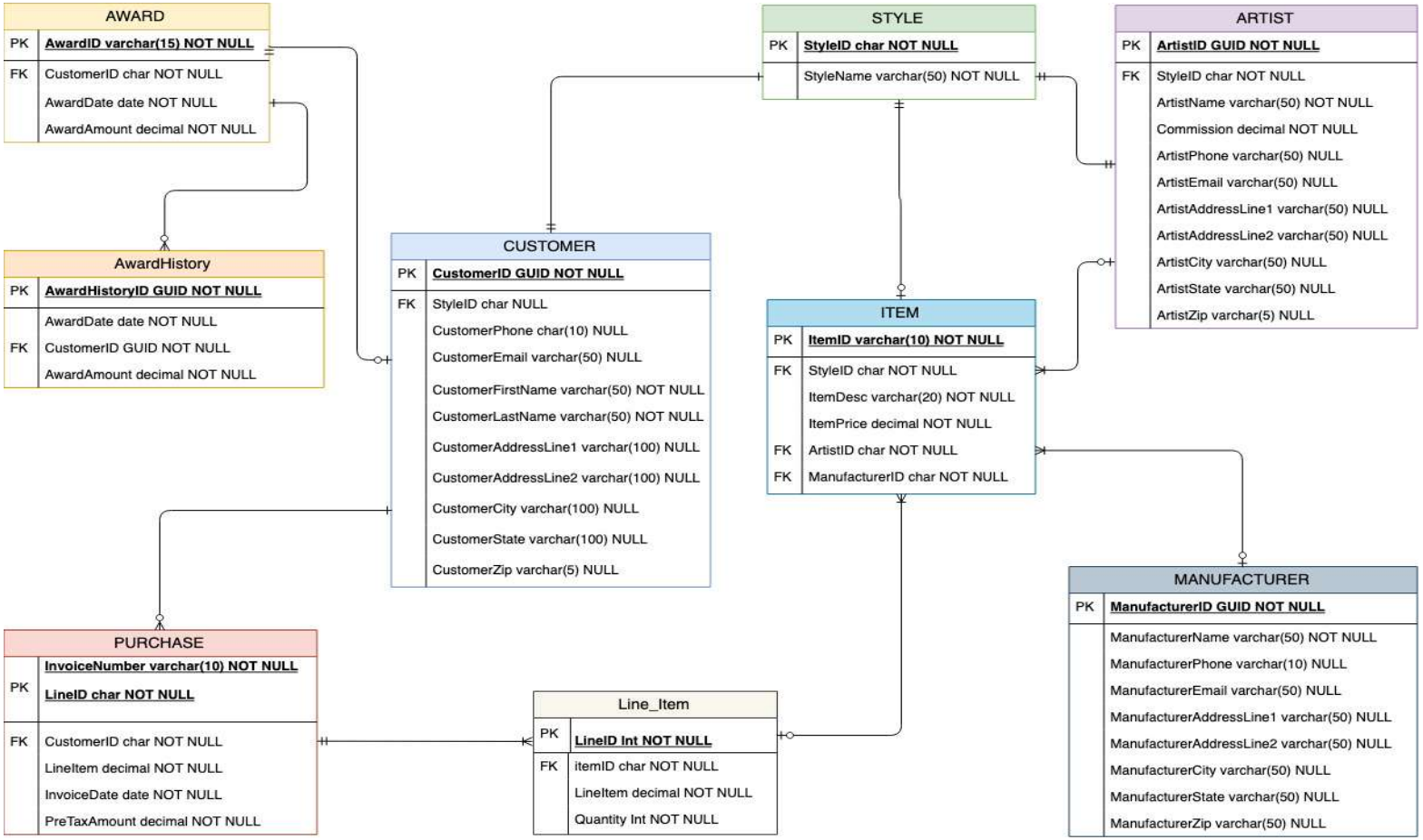
Column ID	Data Type (Length)	Key	Required	Default Value	Remark
ItemID	char	Primary Key	Yes	none	
StyleID	char	Primary Key	Yes	none	Ref: STYLE
		Foreign Key	Yes	none	
ItemDesc	Varchar (20)	No	Yes	none	
ItemPrice	decimal	No	Yes	none	
ArtistID	char	Primary Key	Yes	none	Ref: ARTIST
		Foreign Key	Yes	none	
ManufacturerID	char	Primary Key	Yes	none	Ref: MANUFACTURER
		Foreign Key	Yes	none	

MANUFACTURER

Column ID	Data Type (Length)	Key	Required	Default Value	Remark
ManufacturerID	char	Primary Key	Yes	none	
ManufacturerName	Varchar (50)	No	Yes	none	
ManufacturerPhone	Varchar (50)	No	No	none	
ManufacturerEmail	Varchar (50)	No	No	none	
ManufacturerAddressLine1	Varchar (50)	No	No	none	
ManufacturerAddressLine2	Varchar (50)	No	No	none	
ManufacturerCity	Varchar (50)	No	No	none	
ManufacturerState	Varchar (50)	No	No	none	
ManufacturerZip	Varchar (50)	No	No	none	

Physical Model

This is our physical model that will be implemented in the database design.



Design Validation

Our design is a good representation of the data model because we have accommodated for all the requests thus far. We are able to have a customer's favorite style or styles, track the rewards program for customers, and include artists/manufacturers as well. We have created join tables between ARTIST and STYLE, along with CUSTOMER in STYLE to remove the many to many relationships between them. Our design shows what is mandatory, and optional as well as the type of relationship between each table using the crow's foot E-R model. We have accounted for any referential integrity constraints within our data dictionary. This design is a great representation of the data model because we have accounted for everything that is currently necessary.

SQL CODE

A. Write SQL CREATE TABLE statements for each of these tables.

```
DROP TABLE AWARD
DROP TABLE LINE_ITEM
DROP TABLE ITEM
DROP TABLE ARTIST
DROP TABLE PURCHASE
DROP TABLE AWARDHISTORY
DROP TABLE CUSTOMER
DROP TABLE STYLE
DROP TABLE MANUFACTURER

CREATE TABLE STYLE (
        StyleID                 char            NOT NULL,
        StyleName               varchar(50)     NOT NULL,
        Constraint          STYLE_PK        PRIMARY KEY(StyleID)
)

CREATE TABLE CUSTOMER (
CustomerID                  varchar(50)         NOT NULL,
Customer_Phone              char(10)            NULL,
Customer_Email              varchar(50)         NULL,
Customer_FirstName          varchar(50)         NOT NULL,
Customer_LastName           varchar(50)         NOT NULL,
Customer_AddressLine1       varchar(100)        NULL,
Customer_AddressLine2       varchar(100)        NULL,
Customer_City               varchar(100)        NULL,
Customer_State              varchar(100)        NULL,
Customer_Zip                varchar(5)          NULL,

StyleID char NOT NULL FOREIGN KEY REFERENCES Style(StyleID),
constraint      CUSTOMER_PK     PRIMARY KEY(CustomerID)
)

CREATE TABLE AWARD (
AwardID                 varchar(15)             NOT NULL,
AwardDate               char(20)                NULL,
AwardAmount             decimal                 NOT NULL,

CustomerID varchar(50) NOT NULL FOREIGN KEY REFERENCES Customer(CustomerID),
constraint          AWARD_PK        PRIMARY KEY(AwardID)
)

CREATE TABLE AWARDHISTORY(
```

```
AwardHistoryID          uniqueidentifier                NOT NULL,
AwardDate               date                            NOT NULL,
AwardAmount                 decimal                     NOT NULL,

CustomerID varchar(50) NOT NULL FOREIGN KEY REFERENCES Customer(CustomerID),
constraint          AWARDHISTORY_PK     PRIMARY KEY(AwardHistoryID)
)

CREATE TABLE PURCHASE (
InvoiceNumber               varchar(50)             NOT NULL,
LineID                      varchar(50)             NOT NULL,
InvoiceDate                 date                    NOT NULL,
PreTaxAmount                decimal                 NOT NULL,

CustomerID varchar(50) NOT NULL FOREIGN KEY REFERENCES Customer(CustomerID),
constraint          PURCHASE_PK     PRIMARY KEY(InvoiceNumber, LineID)
)


CREATE TABLE MANUFACTURER (
        ManufacturerID              varchar(50)             NOT NULL,
        ManufacturerName            varchar(50)             NOT NULL,
        ManufacturerPhone           varchar(10)             NULL,
        ManufacturerEmail           varchar(50)             NULL,
        ManufacturerAddressLine1    varchar(50)             NULL,
        ManufacturerAddressLine2    varchar(50)             NULL,
        ManufacturerCity            varchar(50)             NULL,
        ManufacturerState           varchar(50)             NULL,
        ManufacturerZip             varchar(50)             NULL,

constraint              MANUFACTURER_PK     PRIMARY KEY(ManufacturerID)
)

CREATE TABLE ARTIST (
        ArtistID                    varchar(50)             NOT NULL,
        ArtistFirstName             varchar(50)             NOT NULL,
        ArtistLastName              varchar(50)             NOT NULL,
        Commission                  decimal                 NOT NULL,
        ArtistPhone                 varchar(50)             NULL,
        ArtistEmail                 varchar(50)             NULL,
        ArtistAddressLine1          varchar(50)             NULL,
        ArtistAddressLine2          varchar(50)             NULL,
        ArtistCity                  varchar(50)             NULL,
        ArtistState                 varchar(50)             NULL,
        ArtistZip                   varchar(5)              NULL,

StyleID char NOT NULL FOREIGN KEY REFERENCES Style(StyleID),
constraint              ARTIST_PK PRIMARY KEY(ArtistID)
)
```

```
CREATE TABLE ITEM (
ItemID                          varchar(10)             NOT NULL,
ItemDesc                        varchar(20)             NOT NULL,
ItemPrice                       decimal                 NOT NULL,

StyleID char NOT NULL FOREIGN KEY REFERENCES Style(StyleID),
ArtistID varchar(50)  NULL FOREIGN KEY REFERENCES Artist(ArtistID),
ManufacturerID varchar(50) NULL FOREIGN KEY REFERENCES
Manufacturer(ManufacturerID),
constraint              ITEM_PK     PRIMARY KEY(ItemID)
)


CREATE TABLE LINE_ITEM (
LineID                          int                     NOT NULL,
Quantity                        int                     NOT NULL,

ItemID varchar(10) NOT NULL FOREIGN KEY REFERENCES Item(ItemID),
constraint              LINE_ITEM_PK     PRIMARY KEY(LineID)
)
```

B. Write foreign key constraints for the relationships in each of these tables. Make your own assumptions regarding cascading deletions and justify those assumptions. (Hint: You can combine the SQL for your answers to parts A and B.)

```
CREATE TABLE CUSTOMER (
CustomerID                      varchar(50)             NOT NULL,
Customer_Phone                  char(10)                NULL,
Customer_Email                  varchar(50)             NULL,
Customer_FirstName              varchar(50)             NOT NULL,
Customer_LastName               varchar(50)             NOT NULL,
Customer_AddressLine1           varchar(100)            NULL,
Customer_AddressLine2           varchar(100)            NULL,
Customer_City                   varchar(100)            NULL,
Customer_State                  varchar(100)            NULL,
Customer_Zip                    varchar(5)              NULL,

StyleID char NOT NULL FOREIGN KEY REFERENCES Style(StyleID),
constraint      CUSTOMER_PK     PRIMARY KEY(CustomerID)
)
```

We use the following code to give the table CUSTOMER a foreign ID.

```
StyleID char NOT NULL FOREIGN KEY REFERENCES Style(StyleID),
```

C. Write SQL statements to insert the data shown in Figures D-2, D-3, D-4 and D-5 into these tables. Assume that surrogate key column values will be supplied by the DBMS.

```
-----Style Table Data----------
INSERT INTO STYLE(StyleID,StyleName)
VALUES (1, 'Rings');
INSERT INTO STYLE(StyleID,StyleName)
```

```
VALUES (2, 'Bracelet');
INSERT INTO STYLE(StyleID,StyleName)
VALUES (3, 'Neclaces');
INSERT INTO STYLE(StyleID,StyleName)
VALUES (4, 'Earings');

----------Artist Table Data----------

INSERT INTO ARTIST(ArtistID, ArtistFirstName, ArtistLastName, Commission, ArtistPhone,
ArtistEmail,
ArtistAddressline1, ArtistAddressline2, ArtistCity, ArtistState, ArtistZip, StyleID)
VALUES(1, 'Guy', 'Gouda', 30, 2152427895, 'Goodguy@gmail.com', '345 West Straight Street',
null, 'Philadelphia', 'PA', '19118', 4)

INSERT INTO ARTIST(ArtistID, ArtistFirstName, ArtistLastName, Commission, ArtistPhone,
ArtistEmail,
ArtistAddressline1, ArtistAddressline2, ArtistCity, ArtistState, ArtistZip, StyleID)
VALUES(2, 'Jenn', 'Ifher', 25, 2153271653, 'JennnotJennifer@gmail.com', '12 Here Street', null,
'Rochester', 'NY', 14628, 1)

INSERT INTO ARTIST(ArtistID, ArtistFirstName, ArtistLastName, Commission, ArtistPhone,
ArtistEmail,
ArtistAddressline1, ArtistAddressline2, ArtistCity, ArtistState, ArtistZip, StyleID)
VALUES(3, 'Lisa', 'Guisa', 30, 4806511777, 'Lgjewelry@live.com', '871 Leftright Street', null,
'Orlando', 'FL', 19452, 3)

INSERT INTO ARTIST(ArtistID, ArtistFirstName, ArtistLastName, Commission, ArtistPhone,
ArtistEmail,
ArtistAddressline1, ArtistAddressline2, ArtistCity, ArtistState, ArtistZip, StyleID)
VALUES(4, 'Alex', 'Phalanx', 35, 6517893456, 'Apearrings@gmail.com', '21 Fancypancy road',
'Apartment 120', 'Los Angeles' , 'CA', 13485, 2)

----------Manufacturer Table Data----------

INSERT INTO MANUFACTURER(ManufacturerID, ManufacturerName,
ManufacturerPhone, ManufacturerEmail, ManufacturerAddressline1,
ManufacturerCity, ManufacturerState,ManufacturerZip)
VALUES(1, 'The Jeweliest Jewelry Jawns', '2152421234', 'JJJbro@gmail.com',
'32 Radical Road', 'Philadelphia', 'PA', '19118')


----------Item Table Data----------

INSERT INTO ITEM(ItemID, ItemDesc, ItemPrice, StyleID, ArtistID, ManufacturerID)
VALUES(1, 'Bling Ring', 120, 1, 2, 1)

INSERT INTO ITEM(ItemID, ItemDesc, ItemPrice, StyleID, ArtistID, ManufacturerID)
VALUES(2, 'Braceme Bracelet', 200, 2, 1, 1)
```

```
INSERT INTO ITEM(ItemID, ItemDesc, ItemPrice, StyleID, ArtistID, ManufacturerID)
VALUES(3, 'Moonglow Earrings', 125, 4, 4, Null)

INSERT INTO ITEM(ItemID, ItemDesc, ItemPrice, StyleID, ArtistID, ManufacturerID)
VALUES(4, 'Void Ring', 25, 1, 1, Null)

INSERT INTO ITEM(ItemID, ItemDesc, ItemPrice, StyleID, ArtistID, ManufacturerID)
VALUES(5, 'Golden Ivy', 15, 2, 4, Null)

INSERT INTO ITEM(ItemID, ItemDesc, ItemPrice, StyleID, ArtistID, ManufacturerID)
VALUES(6, 'Ice Jaguar', 400, 3, Null, Null)

INSERT INTO ITEM(ItemID, ItemDesc, ItemPrice, StyleID, ArtistID, ManufacturerID)
VALUES(7, 'Silver Earrings', 130, 4, 2, Null)

INSERT INTO ITEM(ItemID, ItemDesc, ItemPrice, StyleID, ArtistID, ManufacturerID)
VALUES(8, 'Silver Chain', 225, 3, 1, Null)

INSERT INTO ITEM(ItemID, ItemDesc, ItemPrice, StyleID, ArtistID, ManufacturerID)
VALUES(9, 'Golden Chain', 300, 3, 3, Null)

INSERT INTO ITEM(ItemID, ItemDesc, ItemPrice, StyleID, ArtistID, ManufacturerID)
VALUES(10, 'Diamond Ring', 325, 1, 3, Null)

INSERT INTO ITEM(ItemID, ItemDesc, ItemPrice, StyleID, ArtistID, ManufacturerID)
VALUES(11, 'Ruby Earrings', 100, 4, 4, Null)

INSERT INTO ITEM(ItemID, ItemDesc, ItemPrice, StyleID, ArtistID, ManufacturerID)
VALUES(12, 'Orange Bracelet', 30, 2, Null, 1)

INSERT INTO ITEM(ItemID, ItemDesc, ItemPrice, StyleID, ArtistID, ManufacturerID)
VALUES(13, 'Mood Ring', 10, 1, 3, Null)

INSERT INTO ITEM(ItemID, ItemDesc, ItemPrice, StyleID, ArtistID, ManufacturerID)
VALUES(14, 'Abtruse Necklace', 260, 3, 2, 1)

INSERT INTO ITEM(ItemID, ItemDesc, ItemPrice, StyleID, ArtistID, ManufacturerID)
VALUES(15, 'Dangly Earrings', 85, 4, 2, Null)
```

----------Customer Table Data----------

```
INSERT INTO CUSTOMER(CustomerID, Customer_Phone,Customer_Email,
Customer_FirstName, Customer_LastName, Customer_AddressLine1
, Customer_AddressLine2, Customer_City, Customer_State, Customer_Zip, StyleID)
VALUES(1, 2152424698, 'Lobeme12@live.com', 'Roger', 'Dodger', '65 East Yeast Street', null,
'Philadelphia', 'PA', '19114', 1)

INSERT INTO CUSTOMER(CustomerID, Customer_Phone,Customer_Email,
Customer_FirstName, Customer_LastName, Customer_AddressLine1
```

```
, Customer_AddressLine2, Customer_City, Customer_State, Customer_Zip, StyleID)
VALUES(2, 3465107642, 'TheFairyQueen@live.com', 'Mary', 'Fairy', '348 Lala Lane', null,
'Suburbia', 'MA', '34562', 3)

INSERT INTO CUSTOMER(CustomerID, Customer_Phone,Customer_Email,
Customer_FirstName, Customer_LastName, Customer_AddressLine1
, Customer_AddressLine2, Customer_City, Customer_State, Customer_Zip, StyleID)
VALUES(3, 8941664768, 'MaleMan@gmail.com', 'Male', 'Man', '666 Masculine Street', null,
'Patriarcy', 'NC', '91324', 2)

INSERT INTO CUSTOMER(CustomerID, Customer_Phone,Customer_Email,
Customer_FirstName, Customer_LastName, Customer_AddressLine1
, Customer_AddressLine2, Customer_City, Customer_State, Customer_Zip, StyleID)
VALUES(4, 5437861345, 'PetaJensen@live.com', 'Peta', 'Jensen', '72 Vanilla Avenue', null, 'Los
Angeles', 'CA', '16789', 4)

INSERT INTO CUSTOMER(CustomerID, Customer_Phone,Customer_Email,
Customer_FirstName, Customer_LastName, Customer_AddressLine1
, Customer_AddressLine2, Customer_City, Customer_State, Customer_Zip, StyleID)
VALUES(5, 4629645555, 'JohnnyDepp@live.com', 'Johnny', 'Depp', '78 Sparrow Road', null,
'Los Angeles', 'CA', '16789', 1)

INSERT INTO CUSTOMER(CustomerID, Customer_Phone,Customer_Email,
Customer_FirstName, Customer_LastName, Customer_AddressLine1
, Customer_AddressLine2, Customer_City, Customer_State, Customer_Zip, StyleID)
VALUES(6, 2157458572, 'alex_russel@gmail.com', 'Alex', 'Russel', '16 Park Drive', null,
'Dallas', 'GA', '30132', 1)

INSERT INTO CUSTOMER(CustomerID, Customer_Phone,Customer_Email,
Customer_FirstName, Customer_LastName, Customer_AddressLine1
, Customer_AddressLine2, Customer_City, Customer_State, Customer_Zip, StyleID)
VALUES(7, 7895855124, 'tessie_leyemer@gmail.com', 'Tessie', 'Leyemer', '7 Carriage Avenue',
null, 'Lexington', 'NC', '27292', 3)

INSERT INTO CUSTOMER(CustomerID, Customer_Phone,Customer_Email,
Customer_FirstName, Customer_LastName, Customer_AddressLine1
, Customer_AddressLine2, Customer_City, Customer_State, Customer_Zip, StyleID)
VALUES(8, 4848518895, 'dylan_bates@gmail.com', 'Dylan', 'Bates', '27 Hawthorne Road', null,
'Dallas', 'OH', '45211', 3)

INSERT INTO CUSTOMER(CustomerID, Customer_Phone,Customer_Email,
Customer_FirstName, Customer_LastName, Customer_AddressLine1
, Customer_AddressLine2, Customer_City, Customer_State, Customer_Zip, StyleID)
VALUES(9, 8968518547, 'betsy_nate@gmail.com', 'Betsy', 'Nate', '76 Beech Street', null, 'Hills',
'IL', '60156', 4)

INSERT INTO CUSTOMER(CustomerID, Customer_Phone,Customer_Email,
Customer_FirstName, Customer_LastName, Customer_AddressLine1
, Customer_AddressLine2, Customer_City, Customer_State, Customer_Zip, StyleID)
```

```
VALUES(10, 8956254125, 'kaveh_forootan@gmail.com', 'Kaveh', 'Forootan', '8 Hudson Drive',
'Apartment 3', 'Muscat', 'IA', '52761', 2)

INSERT INTO CUSTOMER(CustomerID, Customer_Phone,Customer_Email,
Customer_FirstName, Customer_LastName, Customer_AddressLine1
, Customer_AddressLine2, Customer_City, Customer_State, Customer_Zip, StyleID)
VALUES(11, 7848562514, 'barry_allen@gmail.com', 'Barry', 'Allen', '224 Jennifer Road', null,
'Parkville', 'MD', '21234', 1)

INSERT INTO CUSTOMER(CustomerID, Customer_Phone,Customer_Email,
Customer_FirstName, Customer_LastName, Customer_AddressLine1
, Customer_AddressLine2, Customer_City, Customer_State, Customer_Zip, StyleID)
VALUES(12, 8245218412, 'colter_myers@gmail.com', 'Colter', 'Myers', '754 Lanel Lane', null,
'Ville', 'TN', '37072', 4)

INSERT INTO CUSTOMER(CustomerID, Customer_Phone,Customer_Email,
Customer_FirstName, Customer_LastName, Customer_AddressLine1
, Customer_AddressLine2, Customer_City, Customer_State, Customer_Zip, StyleID)
VALUES(13, 3545215232, 'linda_cilio@gmail.com', 'Linda', 'Cilio', '98 Port Street', null,
'Baster', 'AL', '35007', 4)

INSERT INTO CUSTOMER(CustomerID, Customer_Phone,Customer_Email,
Customer_FirstName, Customer_LastName, Customer_AddressLine1
, Customer_AddressLine2, Customer_City, Customer_State, Customer_Zip, StyleID)
VALUES(14, 2185328425, 'eva_luna@gmail.com', 'Eva', 'Luna', '1 Lington Avenue', null,
'Jefferson', 'NY', '11776', 3)

INSERT INTO CUSTOMER(CustomerID, Customer_Phone,Customer_Email,
Customer_FirstName, Customer_LastName, Customer_AddressLine1
, Customer_AddressLine2, Customer_City, Customer_State, Customer_Zip, StyleID)
VALUES(15, 5928426321, 'jessica_walter@gmail.com', 'Jessica', 'Walter', '27 Cherry Street',
null, 'Biloxi', 'MS', '39532', 2)
```

D. Write SQL statements to list all columns for all tables.

```
SELECT *
FROM CUSTOMER;
SELECT *
FROM AWARD;
SELECT *
FROM AwardHistory;
SELECT *
FROM STYLE;
SELECT *
FROM ARTIST;
SELECT *
FROM ITEM;
SELECT *
FROM MANUFACTURER;
SELECT *
FROM Line_Item;
```

```
SELECT *
FROM PURCHASE;
```

E. Write an SQL statement to list ItemNumber and Description for all items that cost more than $100.

```
SELECT          ItemID, ItemDesc
FROM            ITEM
WHERE           ItemPrice > 100;
```

F. Write an SQL statement to list ItemNumber and Description for all items that cost more than $100 and were produced by an artist with a last name ending with the letter's "son".

```
SELECT          ItemID, ItemDesc
FROM            ITEM
WHERE           ItemPrice > 100
        AND ArtistLastName IN
        (SELECT ArtistLastName
        FROM ARTIST
        WHERE ArtistLastName = '%son');
```

G. Write an SQL statement to list LastName and FirstName of customers who have made at least one purchase with PreTaxAmount greater than $200. Use a subquery

```
SELECT          CustomerLastName, CustomerFirstName
FROM            CUSTOMER
WHERE           PreTaxAmount IN
        (SELECT         PreTaxAmount
        FROM            PURCHASE
        WHERE           PreTaxAmount > 200);
```

H. Answer part G but use a join using JOIN...ON syntax.

```
SELECT          LastName, FirstName

FROM            CUSTOMER JOIN PURCHASE
ON (c.CustomerID = p.CustomerID)

WHERE           PreTaxAmount > 200;
```

I. Write an SQL statement to list LastName and FirstName of customers who have purchased an item that costs more than $50. Use a subquery.

```
SELECT          CustomerLastName, CustomerFirstName
FROM            CUSTOMER
WHERE           ItemPrice IN
        (SELECT         Itemprice
        FROM            ITEM
        WHERE           ItemPrice > 50);
```

J. Answer part I but use a join using JOIN ON syntax.

```
SELECT      LastName, FirstName

FROM        (CUSTOMER C JOIN PURCHASE P ON C.CustomerID = P.CustomerID)
```

```
JOIN        PURCHASE_ITEM PI ON (P.InvoiceNumber = PI.InvoiceNumber)
JOIN        ITEM I ON (PI.ItemNumber = I.ItemNumber)

WHERE Cost>50;
```

K. Write an SQL statement to list the LastName and FirstName of customers who have purchased an item that was created by an artist with a last name that begins with the letter J. Use a subquery

```
SELECT          CustomerLastName, CustomerFirstName
FROM            CUSTOMER
WHERE           ArtistLastName IN
            (SELECT         ArtistLastName
            FROM            ARTIST
            WHERE           ArtistLastName = 'J%');
```

L. Answer part K but use a join using JOIN…ON syntax

```
SELECT      LastName, FirstName

FROM        (CUSTOMER C JOIN PURCHASE P ON c.CustomerID = p.CustomerID)

JOIN        PURCHASE_ITEM PI ON (P.InvoiceNumber = PI.InvoiceNumber)

JOIN        ITEM I ON (PI.ItemNumber = I.ItemNumber)

WHERE       ArtistLastName LIKE 'J%';
```

M. Write an SQL statement to show the Name and sum of PreTaxAmount for each customer. Use a join using JOIN ON syntax.

```
SELECT      CONCAT (LastName, FirstName) AS Name,
SUM(PreTaxAmount)

FROM        CUSTOMER C

JOIN        PURCHASE P ON C.CustomerID = P.CustomerID

GROUP BY C.CustomerID;
```

N. Write an SQL statement to show the sum of PreTaxAmount for each artist (hint: the result will have only one line per each artist). Use a join using JOIN ON syntax and sort the results by ArtistLastName then ArtistFirstName in ascending order. Note this should include the full PreTaxAmount for any purchase in which the artist had an item.

```
SELECT      CONCAT (ArtistFirstName, ArtistLastName) AS Name,
SUM(PreTaxAmount)

FROM        (PURCHASE P JOIN PURCHASE_ITEM PI ON P.InvoiceNumber =
PI.InvoiceNumber)
```

```
JOIN        ITEM I ON (PI.ItemNumber = I.ItemNumber)

GROUP BY ArtistFirstName, ArtistLastName

ORDER BY ArtistLastName, ArtistFirstName;
```

O. Write an SQL statement to show the sum of PreTaxAmount for each artist but exclude any items that were part of purchases with PreTaxAmount over $25. Use a join using JOIN ON syntax, and sort the results by ArtistLastName and ArtistFirstName in descending order.

```
SELECT      CONCAT (ArtistFirstName, '' ,ArtistLastName) AS Name,
SUM(PreTaxAmount)

FROM        (PURCHASE P JOIN PURCHASE_ITEM PI ON P.InvoiceNumber =
PI.InvoiceNumber)

JOIN        ITEM I ON (PI.ItemNumber = I.ItemNumber)

WHERE       PreTaxAmount > 25

GROUP BY ArtistFirstName, ArtistLastName

ORDER BY ArtistLastName, ArtistFirstName;
```

P. Write an SQL statement to show which customers bought which items and include any items that have not been sold. Include CUSTOMER.LastName, CUSTOMER.FirstName, InvoiceNumber, InvoiceDate, ItemNumber, ItemDescription, ArtistLastName, and ArtistFirstName. Use a join using JOIN ON syntax and sort the results by ArtistLastName and ArtistFirstName in ascending order.

```
SELECT      C.LastName, C.FirstName, InvoiceNumber, InvoiceDate, ItemNumber,
ItemDescription, ArtistLastName, ArtistFirstName

FROM        (CUSTOMER C JOIN PURCHASE P ON C.CustomerID = P.CustomerID)

JOIN        PURCHASE_ITEM PI ON (P.InvoiceNumber = PI.InvoiceNumber)

JOIN        ITEM I ON (PI.ItemNumber = I.ItemNumber)

ORDER BY ArtistLastName, ArtistFirstName;
```

Q. Write an SQL statement to modify all ITEM rows with an artist last name of Baxter to an artist first name of Rex.

```
UPDATE      ARTIST
SET         ArtistFirstName = 'Rex'
WHERE             ArtistLastName = 'Baxter';
//To see this update
SELECT            *
```

```
FROM            ARTIST
WHERE           ArtistLastName = 'Baxter';
```

R. Write SQL statements to switch the values of ArtistLastName so that all rows currently having the value Baker will have the value Baxter and all rows currently having the value Baxter will have the value Baker.

```
UPDATE      ARTIST
SET         ArtistLastName = 'Baxter'
WHERE           ArtistLastName = 'Baker';
UPDATE      ARTIST
SET         ArtistLastName = 'Baker'
WHERE           ArtistLastName = 'Baxter';
```

S. Given your assumptions about cascading deletions in your answer to part B, write the fewest number of DELETE statements possible to remove all the data in your database but leave the table structures intact. Do not run these statements if you are using an actual database!

```
//We did not cascade any deletions and used ALTER
DELETE
FROM            CUSTOMER;
DELETE
FROM             AWARD;
DELETE
FROM            AwardHistory;
DELETE
FROM            STYLE;
DELETE
FROM            ARTIST;
DELETE
FROM            ITEM;
DELETE
FROM            MANUFACTURER;
DELETE
FROM            Line_Item;
DELETE
FROM            PURCHASE;
```

A. Write a user-defined function named LastNameFirst that concatenates the customer's LastName and FirstName into a single value named FullName, and displays, in order, the LastName, a comma, a space, and the FirstName (hint: Stanley and Elizabeth would be combined to read Stanley, Elizabeth).

```
--Concatenate Last and First Name--
SELECT CONCAT (Customer_LastName, ', ', Customer_FirstName) As LastNameFirst FROM
CUSTOMER
```

B. Create the following SQL view:

1. Create an SQL view named CustomerPurchaseView that shows CustomerID, LastName, FirstName, InvoiceNumber, Date, and PreTaxAmount.

```
CREATE VIEW [CustomerPurchaseView] //Creates a view with a name of the view
AS SELECT CustomerID,
LastName, FirstName, InvoiceNumber,
Date PreTaxAmount //Defines the exact SELECT statement provides the data of the view.
FROM CustomerPurchaseView
```

2. Create an SQL view named CustomerLastNameFirstPurchaseView that shows CustomerID, then LastName and FirstName concatenated using the LastNameFirst userdefined function and displayed as CustomerName, InvoiceNumber, Date, and PreTaxAmount.

```
CREATE VIEW [CustomerLastNameFirstPurchaseView] //Creates a view with a name of the
view
AS SELECT CustomerID, LastName, FirstName
```

Appendix

Below you will find all the work that led up to what you have seen above.

Section A

Sample list of customers information and purchases

Customer Name	Phone	E-mail	Address	Purchase Date	Invoice#	Amount	Tax	Total
Alex Russel	215-745-8572	alex_russel@gmail.com	16 Park. Dallas, GA 30132	1/31/2018	215484545	$ 3,500.00	6%	$ 3,710.00
Alex Russel	215-745-8572	alex_russel@gmail.com	16 Park. Dallas, GA 30132	1/28/2018	215484545	$ 85.00	6%	$ 90.10
Alex Russel	215-745-8572	alex_russel@gmail.com	16 Park. Dallas, GA 30132	1/25/2018	215484545	$ 150.00	6%	$ 159.00
Tessie Lemeyer	789-585-5124	tessie_lemeyer@gmail.com	7 Carriage. Lex, NC 27292	1/31/2018	215484546	$ 149.00	6%	$ 157.94
Dylan Bates	484-851-8895	dylan_bates@gmail.com	27 Cherry. Dallas, OH 45211	1/31/2018	215484546	$ 85.00	6%	$ 90.10
Besty Nate	896-851-8547	besty_nate@gmail.com	6 Hawthorne. Hills, IL 60156	1/31/2018	215484547	$ 95.00	6%	$ 100.70
Kaveh Tan	895-625-4125	kaveh_tan@gmail.com	76 Beech. Muscat, IA 52761	1/31/2018	215484548	$ 250.00	6%	$ 265.00
Barry Allen	784-856-2514	barry_allen@gmail.com	08 Hud. Parkville, MD 21234	1/1/2018	215484549	$ 200.00	6%	$ 212.00
Barry Allen	784-856-2514	barry_allen@gmail.com	08 Hud. Parkville, MD 21234	1/2/2018	215484565	$ 45.00	6%	$ 47.70
Barry Allen	784-856-2514	barry_allen@gmail.com	08 Hud. Parkville, MD 21234	1/3/2018	215484598	$ 350.00	6%	$ 371.00
Barry Allen	784-856-2514	barry_allen@gmail.com	08 Hud. Parkville, MD 21234	1/4/2018	215484548	$ 900.00	6%	$ 954.00
Barry Allen	784-856-2514	barry_allen@gmail.com	08 Hud. Parkville, MD 21234	1/5/2018	215484575	$ 2,000.00	6%	$ 2,120.00
Barry Allen	784-856-2514	barry_allen@gmail.com	08 Hud. Parkville, MD 21234	1/6/2018	215484569	$ 220.00	6%	$ 233.20
Barry Allen	784-856-2514	barry_allen@gmail.com	08 Hud. Parkville, MD 21234	1/7/2018	215484514	$ 420.00	6%	$ 445.20
Barry Allen	784-856-2514	barry_allen@gmail.com	08 Hud. Parkville, MD 21234	1/8/2018	215484575	$ 875.00	6%	$ 927.50
Barry Allen	784-856-2514	barry_allen@gmail.com	08 Hud. Parkville, MD 21234	1/9/2018	215484536	$ 300.00	6%	$ 318.00
Barry Allen	784-856-2514	barry_allen@gmail.com	08 Hud. Parkville, MD 21234	1/10/2018	215484549	$ 360.00	6%	$ 381.60
Colter Myers	824-521-8412	colter_myers@gmail.com	224 Jenni. Ville, TN 37072	1/31/2018	215484550	$ 155.00	6%	$ 164.30
Linda Cilio	354-521-5232	linda_cilio@gmail.com	754 Lane. Baster, AL 35007	1/31/2018	215484551	$ 6,900.00	6%	$ 7,314.00
Eva Luna	218-532-8425	eva_luna@gmail.com	98 Port. Jefferson, NY 11776	1/31/2018	215484552	$ 85.00	6%	$ 90.10
Jessica Walter	592-842-6321	jessica_walter@gmail.com	1 Lington. Biloxi, MS 39532	1/31/2018	215484553	$ 1,500.00	6%	$ 1,590.00

Section A

Part 2

This table displays the customers total purchases, date of credit earned and total available credit.

Customer Name	Number of Purchases	Date of Credit	Credit
Alex Russel	3	Null	$ -
Tessie Lemeyer	1	Null	$ -
Dylan Bates	1	Null	$ -
Besty Nate	1	Null	$ -
Kaveh Forootan	1	Null	$ -
Barry Allen	10	1/10/018	$ 283.50
Colter Myers	1	Null	$ -
Linda Cilio	1	Null	$ -
Eva Luna	1	Null	$ -
Jessica Walter	1	Null	$ -

Modification Problems

A spreadsheet is fine to use when the data being stored is independent of themselves, such as if, you were to only store purchases and customer names alone. When you begin to add things such as the credit system, where ten purchases allow for a credit to the customer, using a simple list can lead to modification problems. Suppose, for example, that you wanted to remove a purchase for a customer due to a return, but that purchase was already listed as one of the ten in the credit portion of the spreadsheet. You would go into the spreadsheet and delete the returned item out of this customer's purchases but may forget to remove it from the credit portion. This would allow the customer a free credit for whichever item was returned. Moreover, if you were to change a purchase date in the Customer/Purchase's spreadsheet. Let's just say, it happened to be the tenth purchase for this customer, meaning that it would be the date of credit as well; therefore, there would be data inconsistencies showing two different dates that should match. These are two examples of modification problems that may occur from maintaining lists in a spreadsheet, but many of these may occur. In conclusion, the list would be about two different entities, and modification problem will always result whenever a list has data from more than one.

Section B

Table

This table joins the unique customer ID and Credit ID and displays the total number or purchases, date of credit and amount of credit available.

Credit_ID	Customer _ID	Number of Purchases	Date of Credit	Credit
1	1	3	NULL	NULL
2	2	1	NULL	NULL
3	3	1	NULL	NULL
4	4	1	NULL	NULL
5	5	1	NULL	NULL
6	6	10	1/10/2018	$283.50
7	7	1	NULL	NULL
8	8	1	NULL	NULL
9	9	1	NULL	NULL
10	10	1	NULL	NULL

Section B

Table

This table shows detailed information on singular purchases and links them to a Customer ID and Purchase ID.

Customer_ID	Purchase_ID	Purchase Date	Invoice#	Amount	Tax	Total
1	1	1/31/2018	215484545	$3,500.00	6%	$3,710.00
1	2	1/28/2018	215484545	$85.00	6%	$90.10
1	3	1/25/2018	215484545	$150.00	6%	$159.00
2	4	1/31/2018	215484546	$149.00	6%	$157.94
3	5	1/31/2018	215484546	$85.00	6%	$90.10
4	6	1/31/2018	215484547	$95.00	6%	$100.70
5	7	1/31/2018	215484548	$250.00	6%	$265.00
6	8	1/1/2018	215484549	$200.00	6%	$212.00
6	9	1/2/2018	215484565	$45.00	6%	$47.70
6	10	1/3/2018	215484598	$350.00	6%	$371.00
6	11	1/4/2018	215484548	$900.00	6%	$954.00
6	12	1/5/2018	215484575	$2,000.00	6%	$2,120.00
6	13	1/6/2018	215484569	$220.00	6%	$233.20
6	14	1/7/2018	215484514	$420.00	6%	$445.20
6	15	1/8/2018	215484575	$875.00	6%	$927.50
6	16	1/9/2018	215484536	$300.00	6%	$318.00
6	17	1/10/2018	215484549	$360.00	6%	$381.60
7	18	1/31/2018	215484550	$155.00	6%	$164.30
8	19	1/31/2018	215484551	$6,900.00	6%	$7,314.00
9	20	1/31/2018	215484552	$85.00	6%	$90.10
10	21	1/31/2018	215484553	$1,500.00	6%	$1,590.00

Section B

Table

This table displays customer information and gives each customer a unique ID number.

Customer_ID	Customer Name	Phone	E-mail	Address
1	Alex Russel	215-745-8572	alex_russel@gmail.com	16 Park Dr. Dallas, GA 30132
2	Tessie Lemeyer	789-585-5124	tessie_lemeyer@gmail.com	7 Carriage. Lexing, NC 27292
3	Dylan Bates	484-851-8895	dylan_bates@gmail.com	27 Cherry. Dallas, OH 45211
4	Besty Nate	896-851-8547	besty_nate@gmail.com	6 Hawthorne. Hills, IL 60156
5	Kaveh Tan	895-625-4125	kaveh_forootan@gmail.com	76 Beech. Muscat, IA 52761
6	Barry Allen	784-856-2514	barry_allen@gmail.com	08 Hud. Parkville, MD 21234
7	Colter Myers	824-521-8412	colter_myers@gmail.com	224 Jenni. Ville, TN 37072
8	Linda Cilio	354-521-5232	linda_cilio@gmail.com	754 Lane. Baster, AL 35007
9	Eva Luna	218-532-8425	eva_luna@gmail.com	98 Port. Jefferson, NY 11776
10	Jessica Walter	592-842-6321	jessica_walter@gmail.com	1 Lington. Biloxi, MS 39532

Multiple list issue

When combining the two lists, multiple issues arise. The customer information is present in both tables causing the information to appear twice in the merged table. This takes up space, can lead to confusion in data interpretation, and can make it difficult to alter information in the system. Some of the problems this redundant data can lead to happen when changing information. When trying to change information altering a single instance of redundant data will not alter the other instance. This can cause information to not be properly portrayed and can make it so the system cannot grab the right information. Also, the number of attributes in this single table exceeds a reasonable amount. With such a large amount of data, it would be difficult to keep track and extract the necessary information. Some of the values in these tables are dependent on cumulative information from other cells. The need for a cumulative amount of purchases and amount spent by a single customer cannot be calculated and represented intuitively in this merged table.

Section C

Table

This table links the Customer ID, Credit ID, and purchase ID while detailed information about singular purchases.

Customer_ID	Credit_ID	Purchase_ID	Purchase Date	Invoice#	Amount	Tax	Total
1	1	1	1/31/2018	215484545	$3,500.00	6%	$3,710.00
1	1	2	1/28/2018	215484545	$ 85.00	6%	$ 90.10
1	1	3	1/25/2018	215484545	$ 150.00	6%	$ 159.00
2	2	4	1/31/2018	215484546	$ 149.00	6%	$ 157.94
3	3	5	1/31/2018	215484546	$ 85.00	6%	$ 90.10
4	4	6	1/31/2018	215484547	$ 95.00	6%	$ 100.70
5	5	7	1/31/2018	215484548	$ 250.00	6%	$ 265.00
6	6	8	1/1/2018	215484549	$ 200.00	6%	$ 212.00
6	6	9	1/2/2018	215484565	$ 45.00	6%	$ 47.70
6	6	10	1/3/2018	215484598	$ 350.00	6%	$ 371.00
6	6	11	1/4/2018	215484548	$ 900.00	6%	$ 954.00
6	6	12	1/5/2018	215484575	$2,000.00	6%	$2,120.00
6	6	13	1/6/2018	215484569	$ 220.00	6%	$ 233.20
6	6	14	1/7/2018	215484514	$ 420.00	6%	$ 445.20
6	6	15	1/8/2018	215484575	$ 875.00	6%	$ 927.50
6	6	16	1/9/2018	215484536	$ 300.00	6%	$ 318.00
6	6	17	1/10/2018	215484549	$ 360.00	6%	$ 381.60
7	7	18	1/31/2018	215484550	$ 155.00	6%	$ 164.30
8	8	19	1/31/2018	215484551	$6,900.00	6%	$7,314.00
9	9	20	1/31/2018	215484552	$ 85.00	6%	$ 90.10
10	10	21	1/31/2018	215484553	$1,500.00	6%	$1,590.00

Section A

Based on the figure D-1: Sample Data for James River Jewelry, we can make assumptions relating to the functional dependencies of the columns of data. We assume that Name, Phone, and Email will be functionally dependent upon some unique identifier that we will call Customer_ID for now. The other three attributes are functionally dependent upon some unique identifier that we will call Purchase_ID.

The reason that we assume this is because Name, Phone, and Email are all essential to the Customer while InvoiceNumber, Date, and PreTaxAmount are not; however, when it comes to Purchase_ID the roles reverse where InvoiceNumber, Date, and PreTaxAmount are essential to the purchase.

This set of data should actually be in two different tables due to this being a CUSTOMER table, and a PURCHASE table. Based on general knowledge of retail sales, it shows that the customer is dependent on things that describe the customer, which in this case would be the Name, Phone, and Email attributes. The same can be said for purchase, that an InvoiceNumber, Date, and Pretax amount provide crucial information about the purchase.

Section B

1. This is stating that the most appropriate design would be having name determine Phone, Email, InvoiceNumber, invoiceData, and PreTaxAmount. This would not be the best approach as these attributes could be broken down further into small tables, and that shows that this design would lead to modification problems based on what was stated in that section of this document.

2. Similar to number one above we still have all the attributes lumped into one relation, except instead of name determining the rest, it is InvoiceNumber that would determine the other five attributes. Again, this would lead to modification problems later on based on that section in this document. While you can see the InvoiceNumber and are able to know the other five attributes uniquely based on this, there are still better ways to go about the design.

3. Since this still only shows the customer relation, and the attributes have not been broken up, the above answers to design 1, and design 2 work with this design. There are better approaches than having the email determine the other five attributes.

4. This one is similar to the first three designs but introduces the CustomerID attribute which makes it a little better than these, but still is not the best design out of the seven choices. Using CustomerID to determine the rest makes sense because it is unique, and a single attribute so it would be easier to follow. Still, just like the first three designs, all these attributes and lumped into the customer relation, and should be split into two different entities to prevent modification problems.

5. Out of the seven designs this is the second most appropriate in our opinion. They have split the attributes into two relations at this point which was our biggest complaint of the first four designs due to modification problems later. When it comes to the customer relation, having the contact info attributes within it make the most sense; furthermore, using the Name to determine the Phone and Email is not what we would recommend. Using the Email may prove to be more useful because when creating an email, it forces you to use on that is not already created allowing it to be unique by design. The underlined word can also be considered a primary key of the customer relation so for that to be true it has to be unique to one row. Name may be unique for now, but that cannot be said indefinitely due to changes in the future which is why we would recommend the email being the one that determines the other attributes.

When it comes to the purchase relation, we think this design was done well because there can only be one invoiceNumber so it is good to use this to determine InvoiceDate and PreTaxAmount. As stated, this is the second most appropriate of the seven designs.

6. This is the best design and everything we said about design 5 can be applied here with the caveat being that what we recommended above was done in this design. When you have email dictate Name, and Phone instead of Name it makes for a better design. On top of this Email was then placed into the purchase relation as a foreign key to create a relationship which is a far better design. Our only gripe with this one would be that having a CustomerID in the customer relation and having that as the primary key instead of Email may make this cleaner. Email still works because like we said while discussing design 5, email is automatically unique by design; however, if we added a CustomerID it would make it easier on us. This is the best design and could be made a bit better by adding a CustomerID to the customer relation.

7. This design is extremely similar to 5 and 6, but I would not choose this one because having phone in the purchase relation instead of the customer relation does not make sense when you think about it. Phone is only needed as a method of contact, and the customer relation deals mainly with contact information for the customer so that is where phone should be placed. Aside from that one change this is identical to design 6 so we can refer to the comments on design 6 above to end this.

Section C

The purpose of this column is to keep a balance of the customers' purchases for award purposes. Assume that returns will be recorded, with invoices having a negative PreTaxAmount.
The design we liked best was design 6 from the 7 choices that were provided. As stated in the section about those designs, the only alteration we would like to make is to add CustomerID to be the primary key for the customer relation instead of having it be email. With this alteration in mind we would end up with this:

CUSTOMER (CustomerID, Name, Phone, Email)

AND

PURCHASE (InvoiceNumber, InvoiceDate, PreTaxAmount, *CustomerID*)

The above is now exactly what we consider to be the best design, and now we will go ahead and add a column called AwardPurchaseAmount. James River Jewelry has stated that they do not accept returns as all sales are final, so we can disregard the thought that returns would be recorded as having a negative PreTaxAmount since they are no longer a possibility. The best place to put AwardPurchaseAmount would have to be in relation the PURCHASE within the given scenario; however, there are better options that we need to explore later on. Our new design will look like the following:

CUSTOMER (CustomerID, Name, Phone, Email)

AND

PURCHASE (InvoiceNumber, InvoiceDate, PreTaxAmount, AwardPurchaseAmount, *CustomerID*)

This works because we still have the CustomerID in the PURCHASE relation as a foreign key, and this can be what continues to keep the AwardPurchaseAmount straight because we can just query the CustomerID in the PURCHASE relation to pull up all the different InvoiceNumber's from this customer, allowing us to double check PreTaxAmounts. While this design is not terrible and could end up working, there is still a better way to do this, and that will be explored within section D.

Section D

Add a new AWARD table to your answer to part C. Assume that the new table will hold data concerning the date and amount of an award that is given after a customer has purchased 10 items. Ensure that your new table has appropriate primary and foreign keys. As per section C, our current table looks like the following:

CUSTOMER (CustomerID, Name, Phone, Email)

AND

PURCHASE (InvoiceNumber, InvoiceDate, PreTaxAmount, AwardPurchaseAmount, CustomerID)

We will be adding a new table called AWARD. This table will have a primary key of AwardID and it will also have the information for AwardDate and AwardAmount. We need to have information from PreTaxAmount from the PURCHASE table, so we can do the calculations for the AwardAmount. We will also need the information from InvoiceDate so we can calculate the past 10 purchases. InvoiceNumber will need to be a foreign key in our new AWARD table so we may access this information. Therefore, our new tables are the following:

CUSTOMER (CustomerID, Name, Phone, Email)
PURCHASE (InvoiceNumber, InvoiceDate, PreTaxAmount, AwardPurchaseAmount, CustomerID)
AWARD (AwardID, AwardDate, AwardAmount, InvoiceNumber)

These tables will allow us to do all the calculations we need for the customer's award.

Memo

Memorandum
To: James River
From: BJ&K
Date: 02/07/2019
Subject: Project Proposal.

Introduction

The purpose of the memo is to help James River Company in creating a database that will allow them to optimize their profitability and productivity. The following memo will include an explanation on how databases work and how database solutions will help them in the long-term picture.

Product

At BJ&K, we endorse database solution due to the advantages and implication of the approach. As you may already know, a database is a structure used to hold or store data. At its core, a database is used to keep track of data, and it allows for less modification problems. A database is used in everyday life, for example, when you search for an item on a popular site like Amazon, you are using a database. Here are some advantages that our company can bring to you through our database solutions:

- Reduces redundancy in the data.
- Reduces inconsistencies in modification.
- Increases potential income through efficiency.

Our company wants to help you create a database, rather than you modifying a spreadsheet. We at BJ&K appreciate the time you spent reviewing our proposal on why our database solution will be great for your company.

Long term solutions

At BJ&K, we endorse database solution due to the advantages and implication of the approach. As you may already know. Here are some advantages that our company can bring to you through our database solutions:

- Reduces redundancy in the data.
- Reduces inconsistencies in modification.
- Increases potential income through efficiency.

We can reduce redundancy in the data by creating a system that will be able to search if a person walking in the door is an existing or a prospect customer by checking a Customer ID. If a company uses a filing system, human error can come into play, when a customer file is not physically found. People can often misplace things. Then, a new file will be created for the customer and now there are two customer files. This is redundant, and we want to avoid this kind of issues by using a database.

If a company uses a spreadsheet system where everything is stored locally, one department can update certain customer information and other departments may not know of this change. For example, a customer can change their billing address with one department and may forget to do so with the billing department. We now have an inconsistency in the data that can lead to major problems in the future.

A database can also increase revenue in the long run by increasing efficiency with the employees that use it. Going back to the filing example, employees will spend time looking through cabinets filled with customer files, when they can simply find it with a simple search. Less employees may be needed to run a store or may spend more time with valued customers. The data that is stored can also be used to track things like spending habits, sales revenue during different times of the year, which items sell the most during a particular sale and even which employee has the best sales record. These are just a few examples, but databases can be modified to the best needs of the company.

In conclusion

BJ&K wants to help you create a database that will provide efficiency and reduce data errors while at the same time giving you the opportunity to increase revenue. We want to thank you and we appreciate the time you spent reviewing our proposal on why our database solution is the best investment for your company.