Joseph Hartman

IST-659

2018-0105

Project 2 Deliverable

**Summary**

For my project, I will be trying to create a database that would be used in the order processing system of a subscription box service that deals with cocktails. The idea is that this database will be used in the service to allow subscribers to try different cocktails as those that aren’t well versed in cocktail culture are often hesitant to try new cocktails at bars. This service not only helps educate them on new cocktails and spirits but also does so in an affordable way so that the subscriber doesn’t have to buy an entire bottle of a spirit to try and make one cocktail.

For this subscription service to work, it needs a database for employees to track registrations for subscriptions and payment methods, what boxes to send them, what cocktails will be in which box as well as the composition in each cocktail, and a way to track if the boxes have been shipped and if they have been received. This will not only be used by employees that track subscriptions and the reoccurring payments but also be used by those that will assemble the boxes.

To clarify a possible confusion, boxes consist of a collection of ingredients to make 2-3 cocktails. When a person subscribes to the service, that person chooses a couple of boxes that will be delivered over their subscription period.

**Conceptual Model**

Business Rules

* A subscriber will get multiple boxes throughout his subscription and one box will be received by multiple subscribers.
* A subscriber can have one or many different forms of payment cards on his account. There must be at least one form of payment on his subscription.
* A subscriber will choose only one subscription plan, either a basic or premium subscription plan.
* There should be a way to track what boxes goes to what subscribers by subscription start date, end date, whether or not not payment should be approved, and whether or not the plan is cancelled
* Shipping is tracked between boxes and subscribers through Shipping Date, Tracking Number, and whether or not the package has been received.
* A subscription box has many cocktails.
* Each cocktail has many ingredients.

Unrefined Data Examples

An example of what some, not all, of the unrefined data might look like on the subscription side would be a row in a table. Greater detail in the data will be shown in the ERD below

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Subscriber | Subscription Plan | Subscription Start Date | Subscription End Date | Renewed | Payment Plan | Boxes |
| John Smith  123 Long Ave  Apt 2  Berkeley, CA, 94709  (123) 456-7890  j.smith@gmail.com | Basic | 01/08/18 | 02/08/18 | Yes | (Card Info) | -Prohibition Era  -Tiki Time |
| Angela R Weir  478 Broadway  Orlando, FL 32789  (789) 345-9083  angelaweir@yahoo.com | Premium | 04/28/18 | 05/28/18 | No | (Card Info) | -Modern Classics  -Latin Summer |

Payment info might be in the form of a simple log

|  |
| --- |
| Card Holder Name: John Smith  Billing Address: 123 Long Ave, Berkeley, CA 94709  Card Number: 5879635402153557  Security Code: 586 |

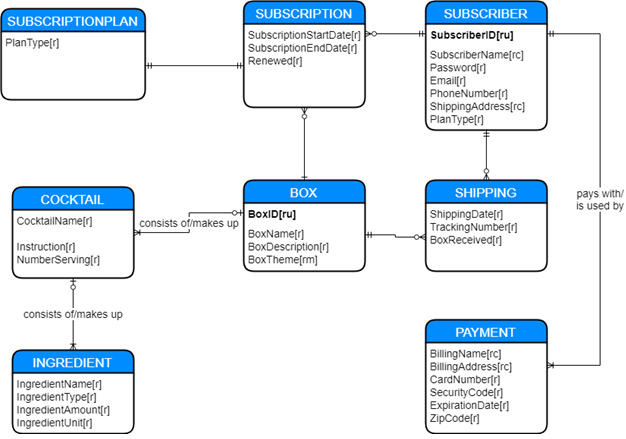
Box and cocktail data in unrefined form would be in a recipe like below

|  |  |  |
| --- | --- | --- |
| Cocktail | Box | Recipe |
| Aviation | Prohibition Era Cocktails | 2 oz Gin  ½ oz Maraschino liquer  ¼ oz Crème de violette  ¾ oz Lemon Juice |

An Example of the Shipping Data might look like a simple log

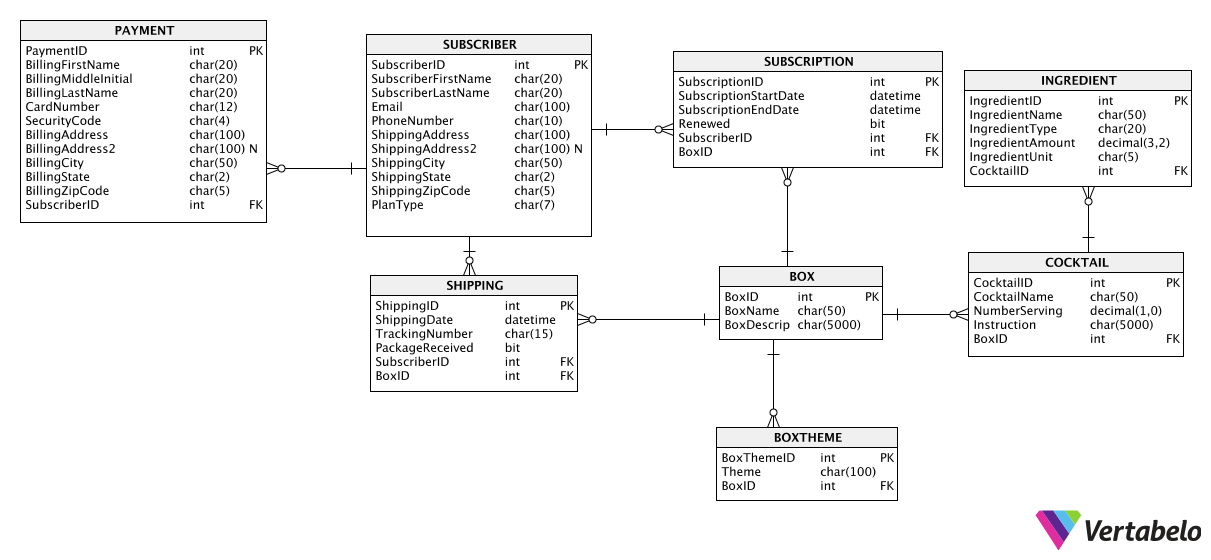
|  |
| --- |
| Name: John Smith  Shipping Address: 123 Long Ave, Berkeley, CA, 94709  Tracking Number: 158764325668871  Package Received: NO |

ERD



In this diagram SUBSCRIPTION and SHIPPING are associative identities between SUBSCRIBER and BOX. Only SUBSCRIBER and BOX have unique identifiers. Surrogate keys will be given to the other entities in the logical model. Also, SUBSCRIPTONPLAN is a weak entity that is dependent on SUBSCRIPTION.

**Normalized Logical Model**



Model Flow

The Logical Model mostly follows the Conceptual Model with a couple of changes. As BOXTHEME is a multi-value attribute of BOX, it has been given its own table in the logical model and has a weak relation to BOX. SUBSCRIPTIONPLAN has been incorporated into SUBSCRIPTION due to its one-to-one relationship.

Additional Attributes

All tables that have been given surrogate keys (with the exception of BOX and SUBSCRIBER as they already have a primary key identifier in the conceptual model). BOXTHEME has been given a foreign key by BOX and its FK acts as the PK in this weak entities. The FKs of BOX and SUBSCRIBER also form a composite PK in both SHIPPING and SUBSCRIPTION.

Composite attributes have been broken down into individual attributes. SubscriberName is now SubscriberFirstName and SubscriberLastName. BillingName is now BillingFirstName, BillingLastName, and BillingMiddleInitial. Addresses in both SUBSCRIBER and PAYMENT have been broken down into Address, Address2, City, State, and ZipCode, each with Billing or Shipping in front of each attribute. Address2, used for apartment numbers, suites, etc., is a non-required attribute for both entities.

Data Types

The only attributes that require math be done are IngredientAmount and NumberServing, as these will be used to calculate the amount of ingredients to put into the box to make X number of servings of a certain cocktail. Numeral attributes like BillingZipCode, PhoneNumber, CardNumber, and TrackingNumber have char data types as they have no math being done on them. Any sort of text such as SubscriberName, Instruction, BoxDescrip, etc. use the char data type of various length depending on the purpose of the attribute. There are two bit data type attributes, Renewed and PackageRecieved. These attributes are binary in nature and so will show 1 if True and 0 if False.

**Physical Database Design**

This is the code that creates the tables for the database. The first nine statements drop the tables if they already exist in the database.

IF EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'Ingredient')

BEGIN

DROP TABLE Ingredient

END

IF EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'Cocktail')

BEGIN

DROP TABLE Cocktail

END

IF EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'BoxTheme')

BEGIN

DROP TABLE BoxTheme

END

IF EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'Shipping')

BEGIN

DROP TABLE Shipping

END

IF EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'Subscription')

BEGIN

DROP TABLE Subscription

END

IF EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'Box')

BEGIN

DROP TABLE Box

END

IF EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'Payment')

BEGIN

DROP TABLE Payment

END

IF EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'Subscriber')

BEGIN

DROP TABLE Subscriber

END

CREATE TABLE Subscriber (

SubscriberID int IDENTITY PRIMARY KEY

, SubscriberFirstName char(20) NOT NULL

, SubscriberMiddleInitial char(1)

, SubscriberLastName char(20) NOT NULL

, Email char(100) NOT NULL

, PhoneNumber char(10) NOT NULL

, ShippingAddress char(100) NOT NULL

, ShippingAddress2 char(100)

, ShippingCity char(50) NOT NULL

, ShippingState char(2) NOT NULL

, ShippingZipCode char(5) NOT NULL

, PlanType char(7) NOT NULL

)

GO

CREATE TABLE Payment (

PaymentID int IDENTITY PRIMARY KEY

,BillingFirstName char(20) NOT NULL

, BillingMiddleInital char(1)

, BillingLastName char(20) NOT NULL

, CardNumber char(12) NOT NULL

, SecurityCode char(4) NOT NULL

, BillingAddress char(100) NOT NULL

, BillingAddress2 char(100)

, BillingCity char(50) NOT NULL

, BillingState char(20) NOT NULL

, BillingZipCode char(5) NOT NULL

, SubscriberID int NOT NULL FOREIGN KEY REFERENCES Subscriber(SubscriberID)

)

GO

CREATE TABLE Box (

BoxID int IDENTITY PRIMARY KEY

, BoxName char(50) NOT NULL

, BoxDescription char(5000) NOT NULL

)

GO

CREATE TABLE Subscription(

SubscriptionID int IDENTITY PRIMARY KEY

, SubscriptionStartDate datetime NOT NULL

, SubscriptionEndDate datetime NOT NULL

, Renewed bit NOT NULL

, SubscriberID int NOT NULL FOREIGN KEY REFERENCES Subscriber(SubscriberID)

, BoxID int NOT NULL FOREIGN KEY REFERENCES Box(BoxID)

)

GO

CREATE TABLE Shipping(

ShippingID int IDENTITY PRIMARY KEY

, ShippingDate datetime NOT NULL

, TrackingNumber char(15) NOT NULL

, PackageRecieved bit NOT NULL

, SubscriberID int NOT NULL FOREIGN KEY REFERENCES Subscriber(SubscriberID)

, BoxID int NOT NULL FOREIGN KEY REFERENCES Box(BoxID)

)

GO

CREATE TABLE BoxTheme (

BoxThemeID int IDENTITY PRIMARY KEY

, Theme char(100) NOT NULL

, BoxID int NOT NULL FOREIGN KEY REFERENCES Box(BoxID)

)

GO

CREATE TABLE Cocktail (

CocktailID int IDENTITY PRIMARY KEY

, CocktailName char(50) NOT NULL

, NumberServing decimal(1,0) NOT NULL

, Instruction char(5000) NOT NULL

, BoxID int NOT NULL FOREIGN KEY REFERENCES Box(BoxID)

)

GO

CREATE TABLE Ingredient (

IngredientID int IDENTITY PRIMARY KEY

, IngredientName char(50) NOT NULL

, IngredientType char(20) NOT NULL

, IngredientAmount decimal(12,3) NOT NULL

, IngredientUnit char(5) NOT NULL

, CocktailID int NOT NULL FOREIGN KEY REFERENCES Cocktail(CocktailID)

)

GO

Data Creation

This code creates a procedure which uses new Subscriber data to fill the Payment table.

CREATE PROCEDURE AddSubscriberWithPayment (

@FirstName char(20)

, @MiddleInitial char(1) = NULL

, @LastName char(20)

, @Email char(100)

, @Phone char(10)

, @ShippingAddress char(100)

, @ShippingAddress2 char(100) = NULL

, @ShippingCity char(50)

, @ShippingState char(2)

, @ShippingZipCode char(5)

, @Plan char(7)

, @CardNumber char(12)

, @SecurityCode char(4)

) AS

BEGIN

INSERT INTO Subscriber (

SubscriberFirstName, SubscriberMiddleInitial, SubscriberLastName,

Email, PhoneNumber, ShippingAddress, ShippingAddress2,

ShippingCity, ShippingState, ShippingZipCode, PlanType

) VALUES (

@FirstName, @MiddleInitial, @LastName, @Email, @Phone,

@ShippingAddress, @ShippingAddress2, @ShippingCity,

@ShippingState, @ShippingZipCode, @Plan

)

-- Finds the PK of the newly inserted data to use as FK for 'Payment'

DECLARE @NewSubscriber int

SELECT @NewSubscriber = SCOPE\_IDENTITY()

--Uses 'Subscriber' data to fill 'Payment'

BEGIN

INSERT INTO Payment (

BillingFirstName, BillingMiddleInital, BillingLastName,

BillingAddress, BillingAddress2, BillingCity, BillingState,

BillingZipCode, CardNumber, SecurityCode, SubscriberID

) VALUES (

@FirstName, @MiddleInitial, @LastName, @ShippingAddress,

@ShippingAddress2, @ShippingCity, @ShippingZipCode,

@ShippingState, @CardNumber, @SecurityCode, @NewSubscriber

)

END

END

An example input would be

EXEC AddSubscriberWithPayment

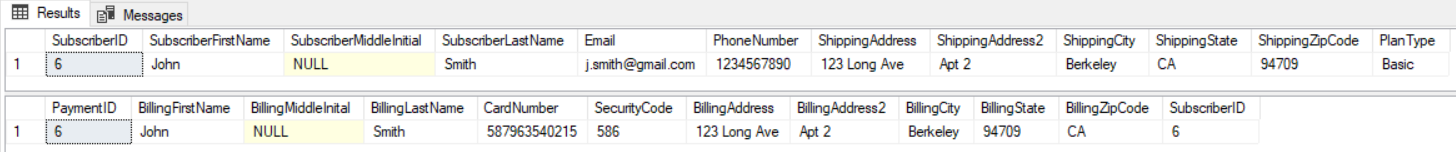
@firstname = 'John', @lastname = 'Smith', @email = 'j.smith@gmail.com',

@phone = '1234567890', @shippingaddress = '123 Long Ave', @shippingaddress2 = 'Apt 2',

@shippingcity = 'Berkeley', @shippingstate = 'CA', @shippingzipcode = '94709',

@cardnumber = '5879635402153557', @securitycode = '586', @plan = 'basic'

Which produces this output, As you can see because no middle initial was provided the stored procedure left it NULL.



A similar stored procedure is used for Box and Cocktail

CREATE PROCEDURE AddBoxWithCocktails (

@name char(50)

, @description char(5000)

, @theme char(100)

, @cocktailone char(50)

, @cocktailtwo char(50)

, @cocktailthree char(50)

, @servings decimal(1,0)

, @instructionone char(5000)

, @instructiontwo char(5000)

, @instructionthree char(5000)

) AS

BEGIN

INSERT INTO Box (BoxName, BoxDescription)

VALUES (@name, @description)

DECLARE @NewBox int

SELECT @NewBox = SCOPE\_IDENTITY()

--Inserts Theme into BoxTheme, uses @NewBox as FK

INSERT INTO BoxTheme (Theme, BoxID)

VALUES (@theme, @NewBox)

--Inserts Cocktails into Box using @NewBox as FK

INSERT INTO Cocktail (CocktailName, NumberServing, Instruction, BoxID)

VALUES (@cocktailone, @servings, @instructionone, @NewBox)

INSERT INTO Cocktail (CocktailName, NumberServing, Instruction, BoxID)

VALUES (@cocktailtwo, @servings, @instructiontwo, @NewBox)

INSERT INTO Cocktail (CocktailName, NumberServing, Instruction, BoxID)

Values (@cocktailthree, @servings, @instructionthree, @NewBox)

END

Here are some example inputs

EXEC AddBoxWithCocktails

@name = 'Tiki Time'

, @description = 'Want to feel like you are relaxing off on a distant island, we will get you halfway there'

, @theme = 'Tiki'

, @cocktailone = 'Planters Punch'

, @cocktailtwo = 'Mai Tai'

, @cocktailthree = 'Fog Cutter'

, @servings = 4

, @instructionone = '(Instructions)'

, @instructiontwo = '(Instructions)'

, @instructionthree = '(Instructions)'

EXEC AddBoxWithCocktails

@name = 'Modern Classics'

, @description = 'Discover some new contemporary classics that are popular in craft cocktail bars around the nation'

, @theme = 'Contemporary'

, @cocktailone = 'Gin Basil Smash'

, @cocktailtwo = 'Espresso Martini'

, @cocktailthree = 'Penicillin'

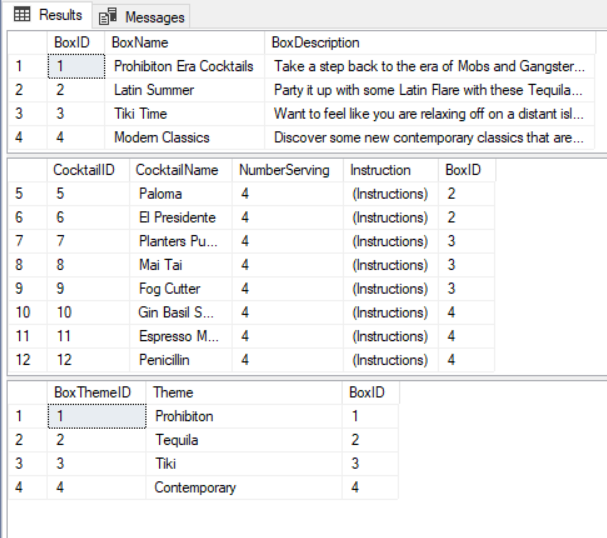
, @servings = 4

, @instructionone = '(Instructions)'

, @instructiontwo = '(Instructions)'

, @instructionthree = '(Instructions)'

Along with some other queries it produces this output. For the purposes of simply showcasing the database I’m using a placeholder for the instructions in order to not take up to much space in the report.



I had some trouble creating procedures to be used in the case where there were multiple FKs so here are some insert statements that were used to fill some of the other tables like Subscription and Shipping. Something to improve on the future is to implement a system that does not involved looking up FKs

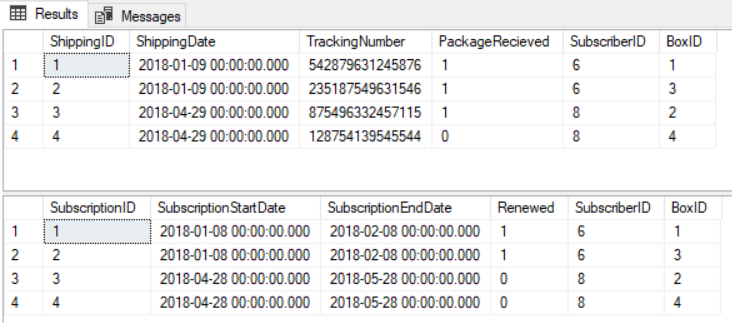
INSERT INTO Subscription (SubscriptionStartDate, SubscriptionEndDate, Renewed, SubscriberID, BoxID)

VALUES ('04/28/2018', '05/28/2018', 0, 8, 4)

INSERT INTO Shipping (ShippingDate, TrackingNumber, PackageRecieved, SubscriberID, BoxID)

VALUES ('01/09/2018', '542879631245876', 1, 6, 1)

The output along with more INSERT statements shows the following output



Data Manipulation

Three tables that will probably need constant updating are the Payment, Shipping, and Subscription tables.

For Subscription and Shipping the stored procedures are very simple in that allows the user to update Renewal Status for Subscription and whether or not a box has been received for Shipping. These procedures change the binary values on Renewed of Subscription and PackageRecieved of Shipping.

CREATE PROCEDURE IsRenewed (@SubscriberID int, @BoxID int) AS

BEGIN

UPDATE Subscription

SET Renewed = 1

WHERE SubscriberID = @SubscriberID AND BoxID = @BoxID

END

GO

CREATE PROCEDURE NotRenewed (@SubscriberID int, @BoxID int) AS

BEGIN

UPDATE Subscription

SET Renewed = 0

WHERE SubscriberID = @SubscriberID AND BoxID = @BoxID

END

GO

CREATE PROCEDURE IsRecieved (@SubscriberID int, @BoxID int) AS

BEGIN

UPDATE Shipping

SET PackageRecieved = 1

WHERE SubscriberID = @SubscriberID AND BoxID = @BoxID

END

GO

CREATE PROCEDURE NotRecieved (@SubscriberID int, @BoxID int) AS

BEGIN

UPDATE Shipping

SET PackageRecieved = 0

WHERE SubscriberID = @SubscriberID AND BoxID = @BoxID

END

GO

Since any aspect of Payment data might need to be updated, a simple update statement is better. For example to update the address of a subscriber, an UPDATE statement might look like:

UPDATE Payment

SET BillingAddress = '1 Main St', BillingAddress2 = 'Apt 4', BillingCity = 'Los Angeles', BillingState = 'CA', BillingZipCode = '91731'

WHERE PaymentID = 6

For when a subscriber chooses not to renew and needs to be removed from the database the procedure to delete the subscriber looks like this

CREATE PROCEDURE DeleteSubscriber (@SubscriberID int)

AS

BEGIN

DELETE Shipping

WHERE SubscriberID = @SubscriberID

DELETE Subscription

WHERE SubscriberID = @SubscriberID

DELETE Payment

WHERE SubscriberID = @SubscriberID

DELETE Subscriber

WHERE SubscriberID = @SubscriberID

END

This deletes the subscriber from all tables from where the entity is present.

Answering Data Questions

There are three views that are particularly important one that shows shipping information for logistical purposes, one that shows subscription information for accounting purposes, and one that shows box assembly information for operational purposes

The SQL code to create the subscription information view looks like:

CREATE VIEW Subscriptions AS

SELECT TRIM(SubscriberFirstName)+' '+TRIM(SubscriberLastName) AS SubscriberName

, SubscriptionStartDate

, SubscriptionEndDate

, Renewed

, BoxName

FROM Subscription

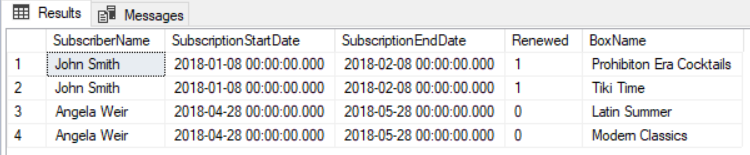
JOIN Subscriber ON

Subscription.SubscriberID = Subscriber.SubscriberID

JOIN Box ON

Subscription.BoxID = Box.BoxID

This creates a view gives all the subscription data for the user as well as which boxes are part of their subscription.



The SQL code to create the shipping information view looks like:

CREATE VIEW ShippingInfo AS

SELECT TRIM(SubscriberFirstName)+' '+TRIM(SubscriberLastName) AS SubscriberName

, ShippingAddress

, ShippingAddress2

, ShippingCity

, ShippingState

, ShippingZipCode

, ShippingDate

, TrackingNumber

, PackageRecieved

, BoxName

FROM Shipping

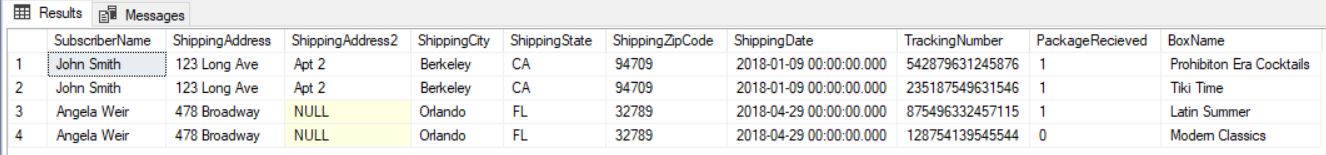
JOIN Subscriber ON

Shipping.SubscriberID = Subscriber.SubscriberID

JOIN Box ON

Shipping.BoxID = Box.BoxID

This adds the Shipping address information to the view in order to clarify where the package is going.



This code creates a view that can be used when workers assemble boxes. Ideally there would be a front-end that would present this better. As seen below the view multiples the numbers of servings in the box with the amount used in one cocktail to give the total amount of what ingredient should be put into a box.

CREATE VIEW BoxAssembly AS

SELECT

BoxName

, CocktailName

, NumberServing

, IngredientName

, IngredientType

, NumberServing\*IngredientAmount AS BoxAmount

, IngredientUnit

FROM Cocktail

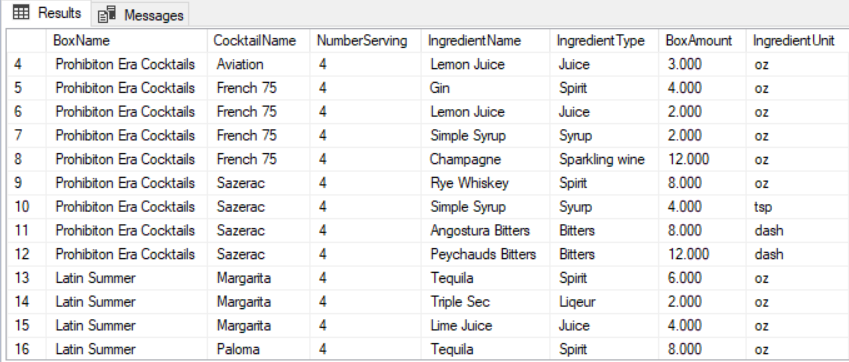
JOIN Box ON

Cocktail.BoxID = Box.BoxID

LEFT JOIN Ingredient ON

Cocktail.CocktailID = Ingredient.CocktailID

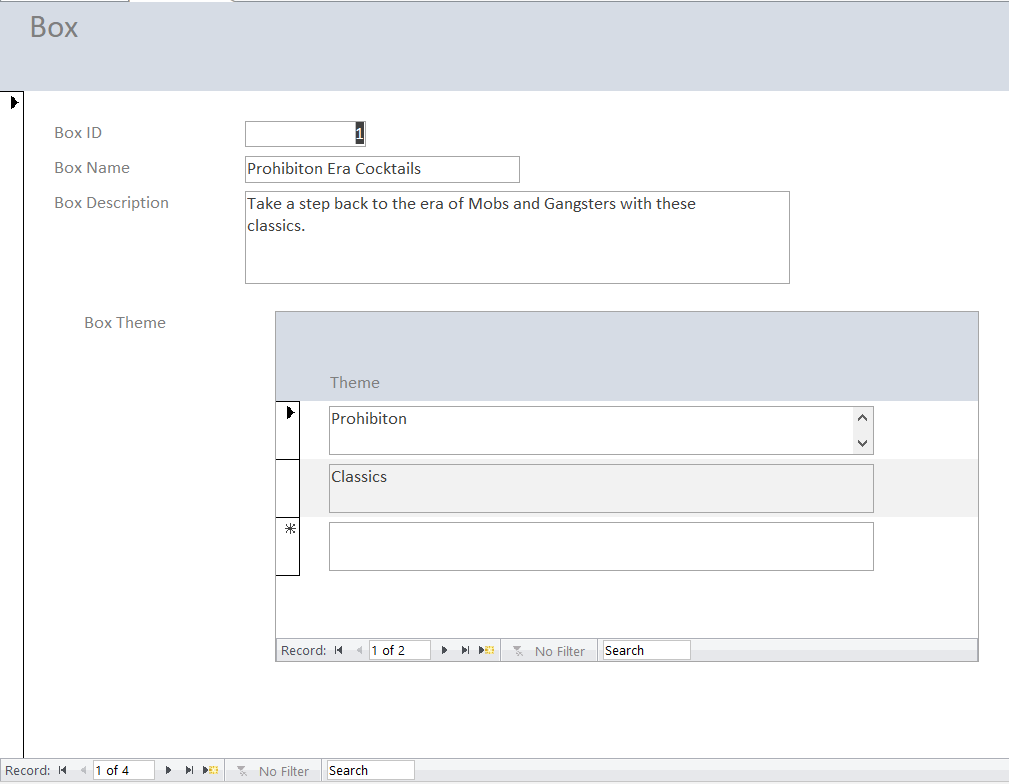
The output is below



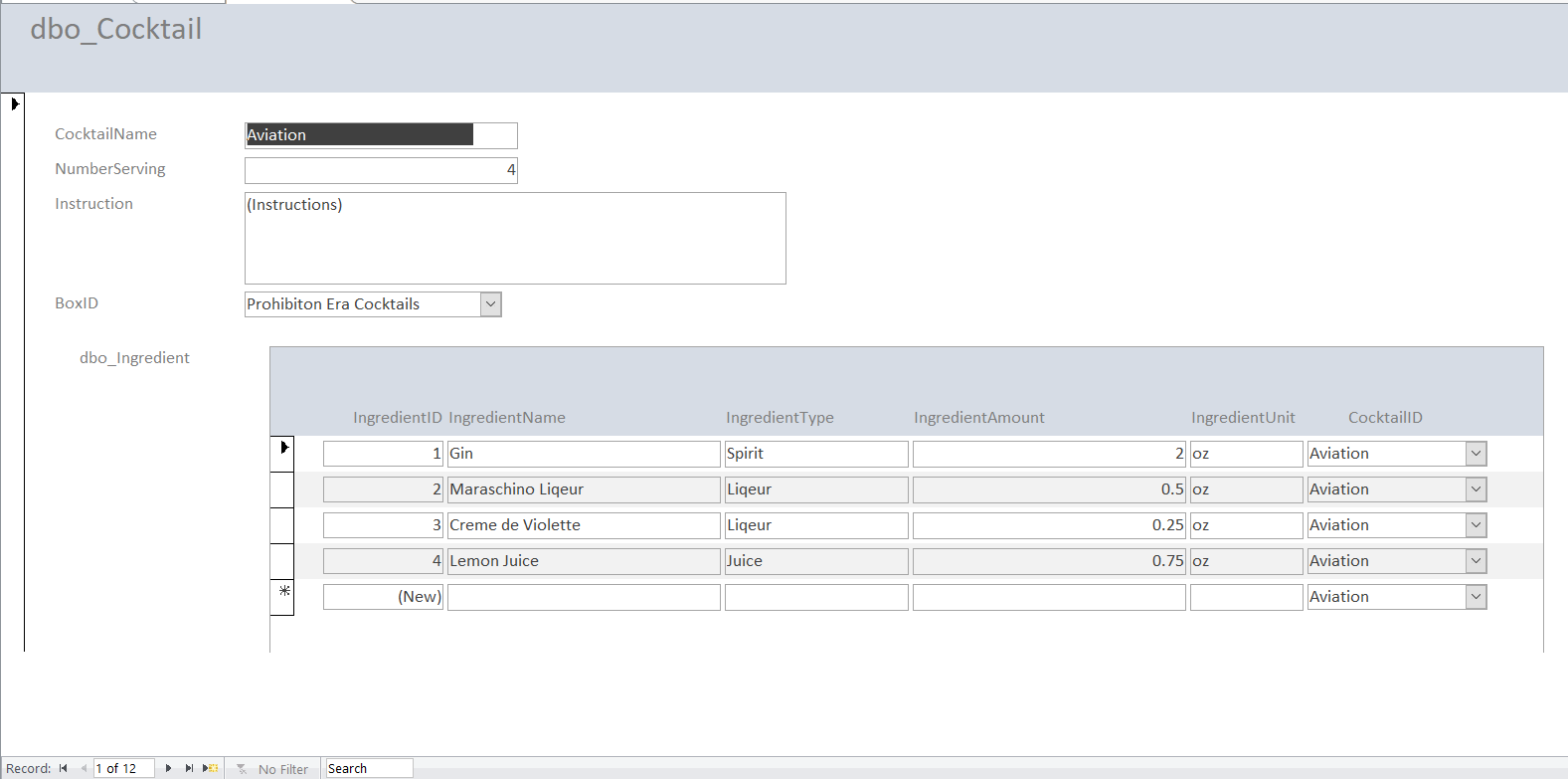
Implementation

For the Rudimentary front end we’re using Microsoft Access as I’m not familiar with PHP or other front end design tools. The forms that were developed are as follows.

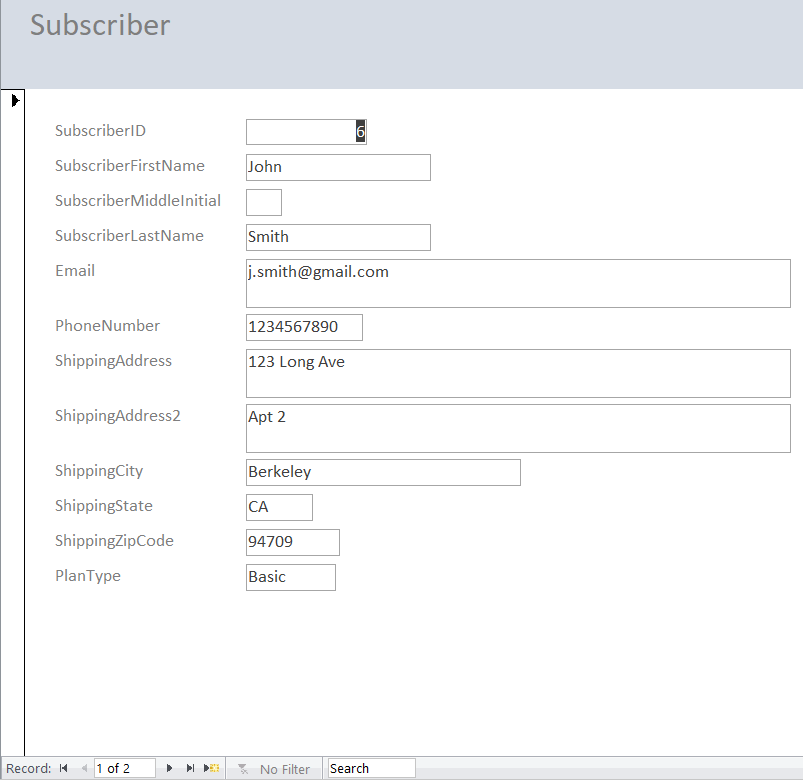
This is the form for Box with a sub form to add multiple Box themes.



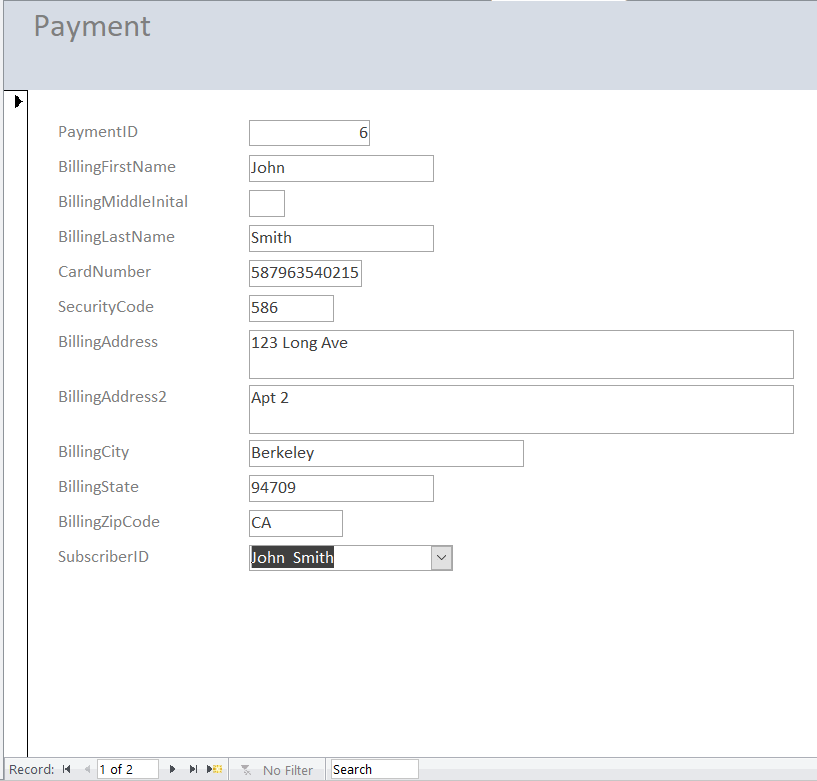
This is the form for cocktails with a subform for ingredients. This form also lets you put cocktails in certain boxes.



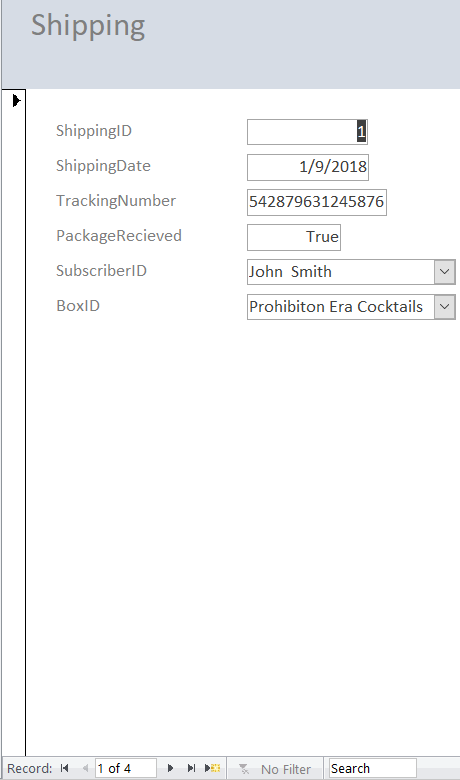
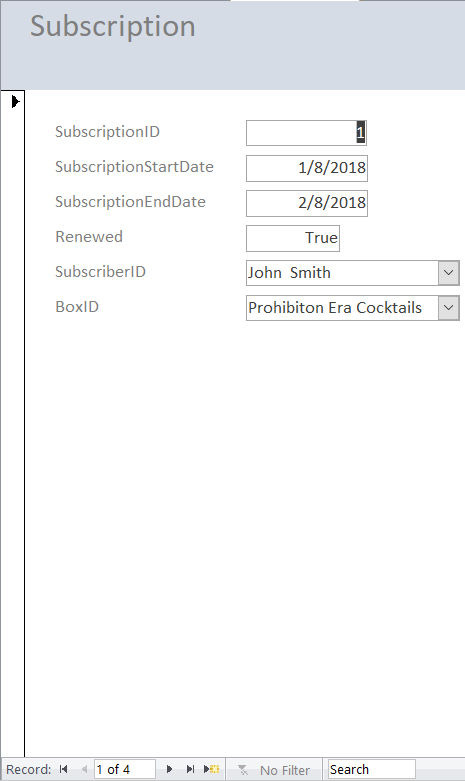
This is a simple form to put subscriber information.



This is a form to update payment information with a combo box to switch between different subscribers.

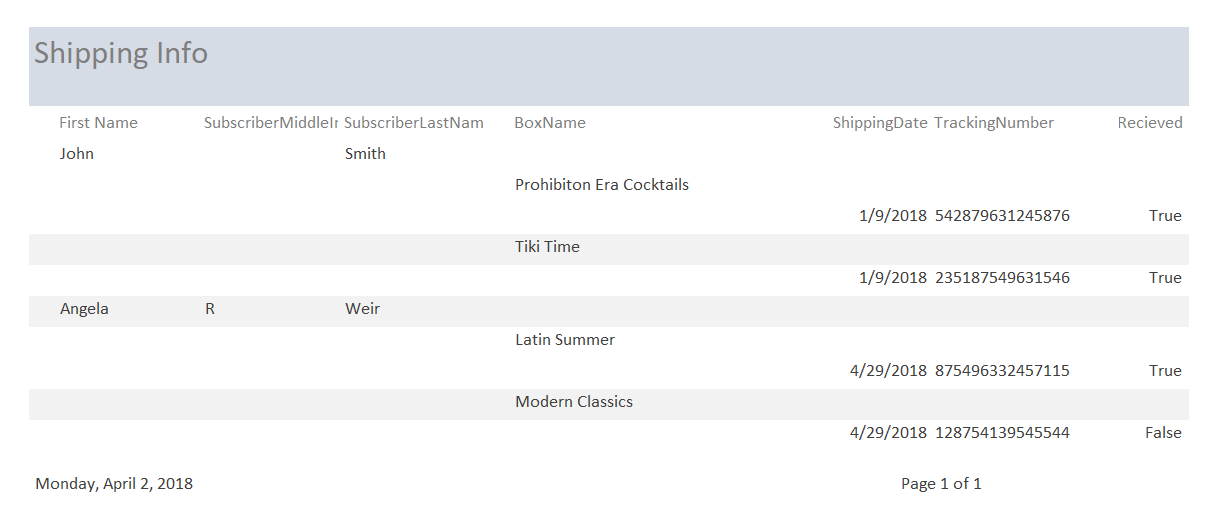


Here are two forms that show shipping and subscription information with combo boxes to select the box and subscriber.

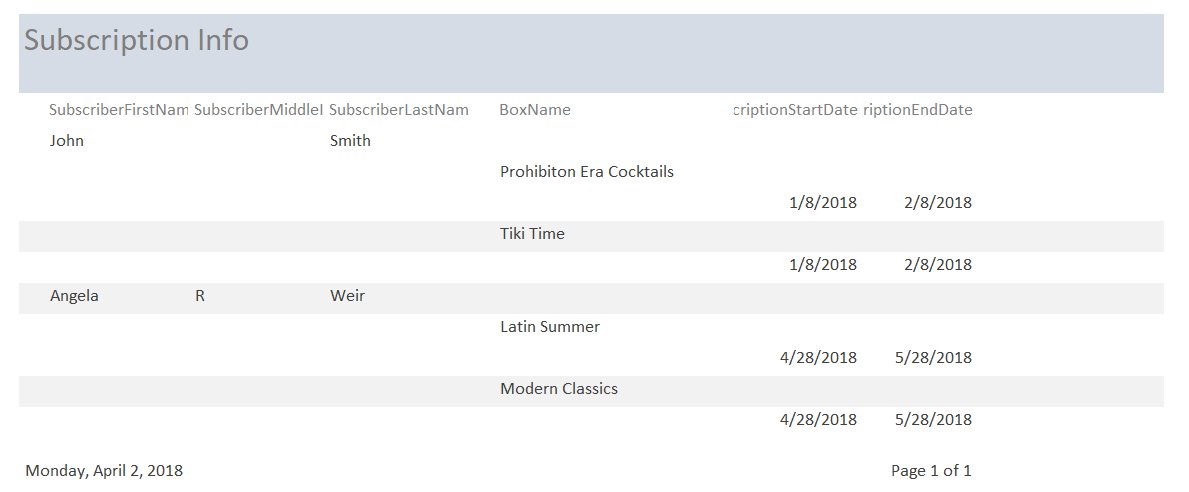
 

For reports, we’re fulfilling the same purposes that were fulfilled by the views. There should be reports on shipping, subscriptions, and box contents.

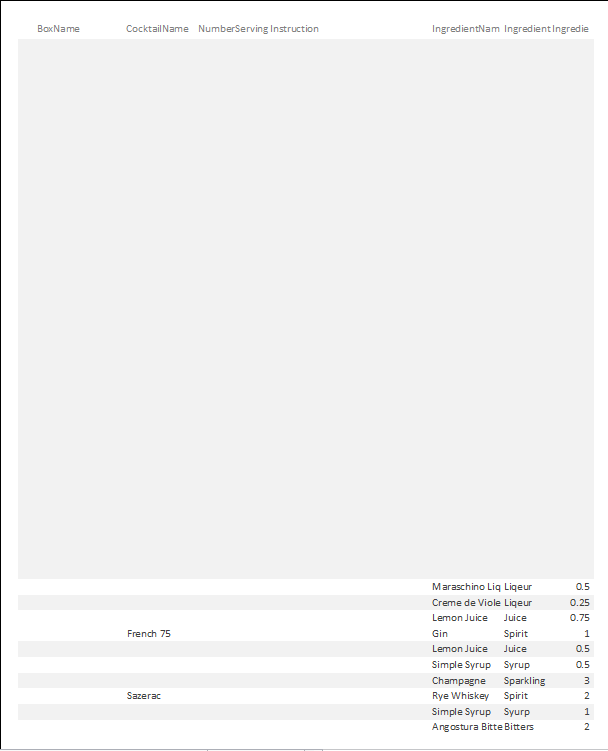
Here’s the report for shipping



Here’s the one for Subscriptions



The Box Assembly Report didn’t come out elegantly due to the huge char value of Cocktail.Instruction. More refining will be needed.



Reflection

Each step had its own set of challenges. The modelling stage was hard as it forced me to think how the different tables interacted with one another given the business rules I set for myself. Setting relationships between tables were especially hard and required serious thought. As I progressed through the actual physical database design, I didn’t realize the sheer amount of SQL code that would go into building what I thought would be a very rudimentary ordering platform. At first I thought that the physical database design would be the simplest and most expedient part, but I quickly got bogged down in the minute details of what I wanted to report in my views, what I wanted my stored procedures to do, and the proper SQL code that would give me that result. Also, as I created information and stored procedures in order abstract data, I found myself wishing that I could change the logical model that I created, specifically the creation of more associative entities between Box and BoxTheme, Box and Cocktails, and Cocktails and Ingredients. I wish I had more time to refine my front-end design, especially with my reports. Though I’m somewhat satisfied with how some of my forms came out.

Though the process of creating my database and implementing it has been long and grueling. The way the course and project have been laid out really were pivotal in giving me a top down approach to designing a database. This process has shown me that the key step to database design is in the logical model. If that step is perfected, then there is less likely of a chance to have to redesign the database as it’s being coded which can lead to some slowdowns. Another item I hope to improve is my grasp on Database Programming with functions, views, and stored procedures. I see how useful they are, and I hope to be able to create much more elegant database programming in the future, though I know that comes with practice.

I would like to give a big thank you to Chad Harper for putting this course together. I came in to IST-659 without any knowledge of database design, SQL, or database administration and I’m leaving this course with a treasure trove of knowledge that I’m sure to implement in the future. Thank you, Chad, for putting together such an amazing course.