**Ben Harwood**

**IST 659 Project 2nd deliverable**

**September 13, 2019**

**Introduction**

As of the moment this is being written, the Mercedes-Benz dealership I work for has 214 cars and SUVs available for sale, and another 359 in various stages of production. Now a good number of those 359 vehicles are customer orders, but the remaining are vehicles that we obviously must sell. Our future build slot allocation is based on sales, and obviously Mercedes-Benz USA gives us monthly and quarterly sales goals (which are lofty, given our volume and ongoing success).

Thankfully, our general sales manager who orders most of the cars is a, to quote the managing partner, “mad scientist” when it comes to configuring incoming vehicles. And while our managing partner often quips, “there’s a butt for every seat,” sometimes vehicles that we think will sell well (based on color combination and option configuration) don’t, and we end up with a slew of vehicles that we can’t seem to get rid of. There could be many reasons for this: color; options; final price, etc. But why is it that one black on black C300 with panoramic sunroof sells after, say 34 days on our lot and another black on black C300 with panoramic sunroof sits stagnant for 6 months? We seek to address this.

**Plan**

By designing and building a database, we will be able to quickly answer a variety of questions. Among these are some easy things like how many cars of certain model and color combination do we have, average lot life for our entire inventory and specific models, how long cars sit before being sold both by model and overall, and whether we had to facilitate a dealer trade to get a car for a client. If the data is able to be collected, we will be able to compare our dealership statistics to our district, region, zone, and the nation.

Additionally, we will be able to see trends in terms of what is selling, specifically in regard to color combination, and thus be able to adjust what we order. This will have two fairly significant financial benefits:

1. Having more cars configured similar to top moving vehicles will reduce the time cars stay on the lot, constantly being charged against floorplan cost.
2. Similarly, more cars configured in popular fashion should mean less dealer trades will need to be completed to get clients what they are looking for. Most dealer trades require two-way trucking to get the car we need to us and whatever car we give up to the other dealership, and this can cost up to $1200 each way depending on where the other dealership is in the country (which is taken directly from whatever gross is made in the deal), so as a whole the dealership would save a substantial amount of gross.

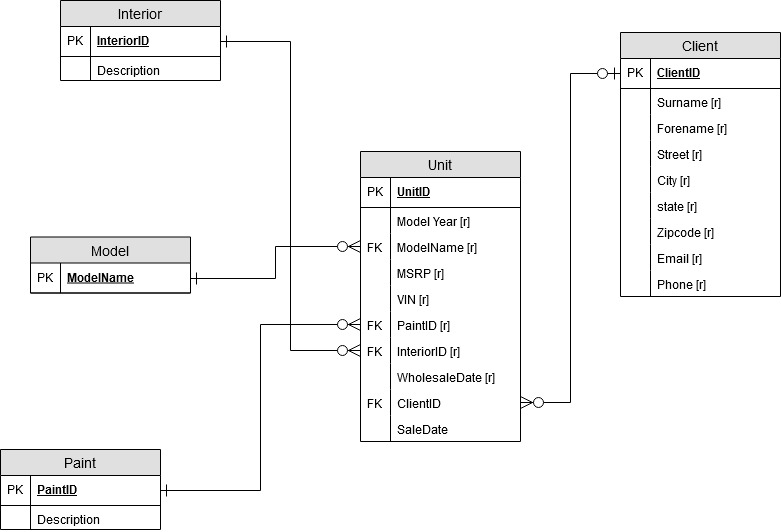
Let’s look at number 2) more closely. While I need to verify, I believe our average transport cost associated with dealer trades is around $500 each way. So, for sake of discussion (i.e. subject to verification), assume we have $1000 in essentially unnecessary expense for every sale that requires a dealer trade. While the number of deals each month that requires a dealer trade varies, I would estimate (again subject to verification) that 25-30% of our deals every month require a trade. Last month (June 2019) we sold 58 new cars and SUVs. Assuming even only 25% of those 58 required a dealer trade, that’s basically 15 cars we had to trade for, which equates to 15\*$1,000 = $15,000 of gross profit lost to transport fees, and that’s just for June. A total of 58 for June is on the low end of what we normally sell, but even if that’s the average it equates to $180,000 for a full year. Again, some of these figures need verification and do fluctuate, but this gives you an idea of the kind of change we could affect by solving this problem.

There is perhaps a wrinkle with the task we are undertaking. The vehicle ordering process is a 2 to 4-month process, which means that while we can investigate our existing inventory and compare it to sales data (probably only for July for logistical reasons) we will not be able to immediately measure the output of whatever changes we make to how we order cars until after the holiday shopping season. That being said, our goal is to identify which (and to what extent) aspects of the cars we order (color, interior, configuration, etc.) contribute to how quickly vehicles sell and develop a “best practice” ordering guide to implement. Then we will revisit after the holidays to measure the impact of the change.

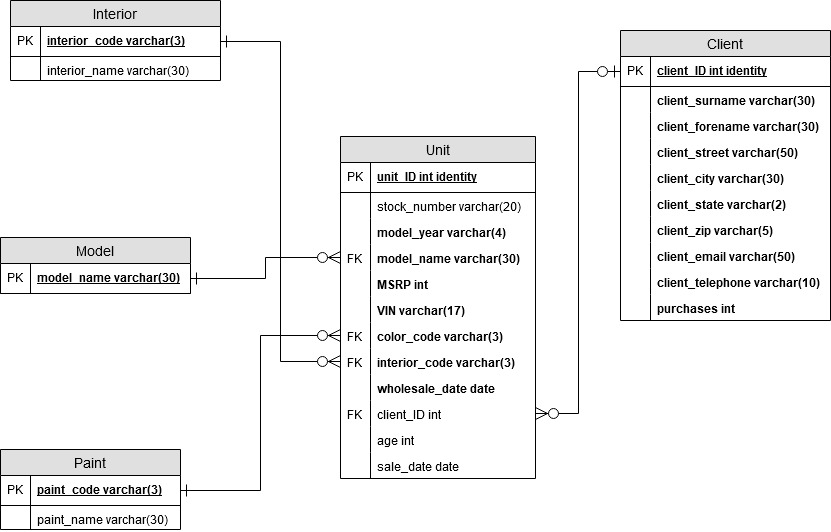
**Conceptual Model**

Our database shall be built around the following rules:

1. Each vehicle or UNIT, differentiated by a UnitID, must have an MSRP, model year and unique VIN.
2. Additionally, each UNIT must have a MODEL, COLOR, INTERIOR, and wholesale date however each ModelID, ColorID, and InteriorID can be associated to multiple UNITs.
3. Each UNIT can have a unique Stock\_number, but this is not required until time of sale.
4. Each UNIT also has a derived Age, the length of time in days that the UNIT has been in stock (determined by the difference between the current date and the WHOLESALE date).
5. When a UNIT is marked sold, it is must be assigned a sale date and CLIENT, and its age is re-calculated to be the difference between its Wholesale\_Date and Sale\_Date.
6. Each CLIENT must have a first and last name (Forename and Surname, respectively), street address (Street, City, State, and Zip), and unique email and telephone numbers.



The logical model for our database is on the following page.



Note: these diagrams differ slightly from our original design.

**DDL**

For readability purposes, full code is in Appendix A.

**Data Creation**

Here are examples of our procedures to add colors, interiors, models, and units to the database:

Color:

CREATE PROCEDURE addColor (@code varchar(3), @name varchar(30))

AS

BEGIN TRANSACTION

BEGIN TRY -- check for redundany, i.e. if the color is already in the database

INSERT INTO color (color\_code, color\_name)

VALUES (@code, @name)

SELECT 'Color successfully added'

COMMIT TRANSACTION

END TRY

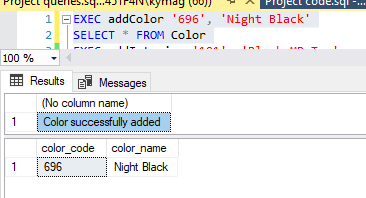
BEGIN CATCH

SELECT 'Color already exists'

ROLLBACK TRANSACTION

END CATCH

GO



Interior:

CREATE PROCEDURE addInterior (@code varchar(3), @name varchar(30))

AS

BEGIN TRANSACTION

BEGIN TRY -- similar check as for color

INSERT INTO Interior(interior\_code, interior\_name)

VALUES (@code, @name)

SELECT 'Interior successfully added'

COMMIT TRANSACTION

END TRY

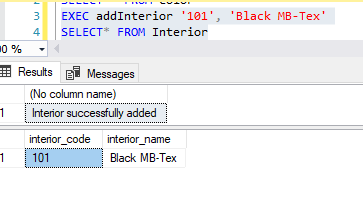
BEGIN CATCH

SELECT 'Interior already exists'

ROLLBACK TRANSACTION

END CATCH

GO



Model:

CREATE PROCEDURE addModel (@name varchar(30))

AS

BEGIN TRANSACTION

BEGIN TRY -- similar check as for color and interior

INSERT INTO Model(model\_name)

VALUES (@name)

SELECT 'Model successfully added'

COMMIT TRANSACTION

END TRY

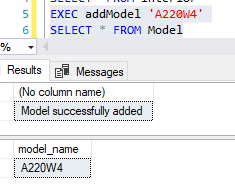
BEGIN CATCH

SELECT 'Model already exists'

ROLLBACK TRANSACTION

END CATCH

GO



Unit:

CREATE PROCEDURE addUnit (@year varchar(4), @model varchar(30), @vin varchar(17), @color varchar(3), @interior varchar(3), @msrp int, @wholesale date, @stock varchar(10))

AS

BEGIN TRANSACTION

BEGIN TRY -- check to see if the unit we are trying to add is already in our inventory or already sold. If not, complete the addition

INSERT INTO Unit(model\_year, model\_name, VIN, color\_code, interior\_code, MSRP, wholesale\_date, stock\_number)

VALUES (@year, @model, @vin, @color, @interior, @msrp, @wholesale, @stock)

SELECT CONCAT('Stock number',' ',@stock,' ','added successfully')

COMMIT TRANSACTION

END TRY

BEGIN CATCH

IF EXISTS (SELECT \* FROM Sold WHERE stock\_number = @stock) --report reason why transaction does not complete

SELECT CONCAT('Stock number ',@stock,' was sold on ', (SELECT sale\_Date FROM Sold WHERE stock\_number=@stock))

ELSE IF NOT EXISTS (SELECT \* FROM Model WHERE model\_name=@model)

SELECT CONCAT('Model ',@model,' does not exist')

ELSE IF NOT EXISTS (SELECT \* FROM Color WHERE color\_code=@color)

SELECT CONCAT('Color code ',@color,' does not exist')

ELSE IF NOT EXISTS (SELECT \* FROM Interior WHERE interior\_code=@interior)

SELECT CONCAT('Interior code ',@interior,' does not exist')

ELSE IF EXISTS (SELECT \* FROM Unit WHERE VIN=@vin)

SELECT CONCAT('VIN ',@VIN,' is already in inventory with stock number ',(SELECT stock\_number FROM Unit WHERE VIN =@vin))

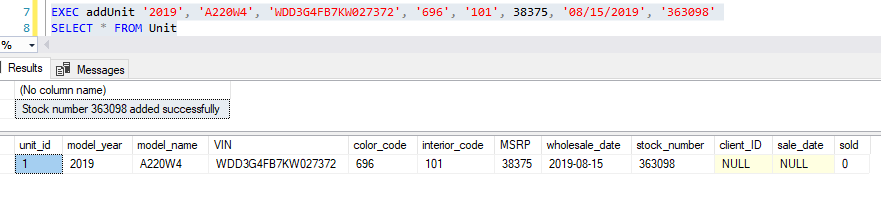
ELSE IF EXISTS (SELECT \* FROM Unit WHERE stock\_number=@stock)

SELECT CONCAT('Stock number ',@stock,' already exists')

ROLLBACK TRANSACTION

END CATCH

GO



Once a unit has been added, and we have clients in our database (seen through the following procedure):

CREATE PROCEDURE addClient (@lastname varchar(30), @firstname varchar(30), @street varchar(50), @city varchar(20), @state varchar(2), @zip varchar(5), @email varchar(50), @phone varchar(10))

AS

BEGIN TRANSACTION

BEGIN TRY -- checking for client existence

INSERT INTO Clients(client\_surname, client\_forename, client\_street, client\_city, client\_state, client\_zip, client\_email, client\_telephone)

VALUES (@lastname, @firstname, @street, @city, @state, @zip, @email, @phone)

SELECT CONCAT('Client ', @firstname,' ',@lastname,' succesfully added'), (Select max(client\_ID) FROM Clients) AS Client\_ID

COMMIT TRANSACTION

END TRY

BEGIN CATCH

IF EXISTS (SELECT \* FROM Clients WHERE client\_email=@email) --report reason why transaction does not complete

SELECT CONCAT('Email ',@email,' already exists for client') AS Result,(SELECT client\_ID FROM Clients WHERE client\_telephone=@phone) AS Client\_ID

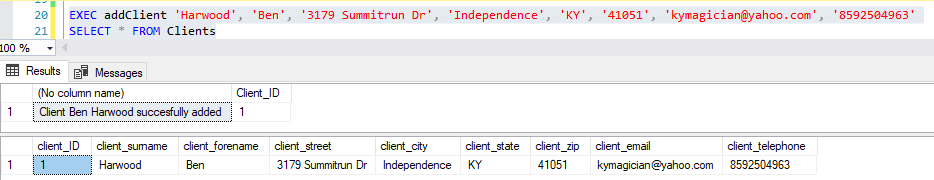
ELSE

SELECT CONCAT('Phone number ',@phone,' already exists') AS Result, (SELECT client\_ID FROM Clients WHERE client\_telephone=@phone) AS Client\_ID

ROLLBACK TRANSACTION

END CATCH

GO



**Data Manipulation**

We can use our tagUnit and sellUnit procedures to assign client names to cars and mark them sold, effectively taking them out of our active inventory:

CREATE PROCEDURE tagUnit (@stock varchar(10), @client int, @sold date)

AS

BEGIN TRANSACTION

BEGIN TRY -- only commit if stock number exists in inventory, is not sold already, and the client exists

IF EXISTS (SELECT \* FROM Unit WHERE stock\_number = @stock AND sold =0) AND EXISTS (SELECT \* FROM Clients WHERE client\_ID = @client)

UPDATE Unit SET sale\_date = @sold

WHERE stock\_number =@stock

UPDATE Unit SET client\_ID = @client

WHERE stock\_number = @stock

SELECT CONCAT('Stock number ',@stock,' successfully tagged for client') AS Result, (SELECT client\_ID FROM Clients WHERE client\_ID=@client) AS Client\_ID

COMMIT TRANSACTION

END TRY

BEGIN CATCH

IF NOT EXISTS (SELECT \* FROM Clients WHERE client\_ID = @client) --report reason why transaction does not complete

SELECT 'Client does not exist. Please add them'

ELSE IF (SELECT client\_ID FROM Unit WHERE stock\_number=@stock)=@client

SELECT 'No changes necessary'

ELSE IF EXISTS (SELECT \* FROM Sold WHERE stock\_number = @stock)

SELECT CONCAT('Stock number ',@stock,' is already sold')

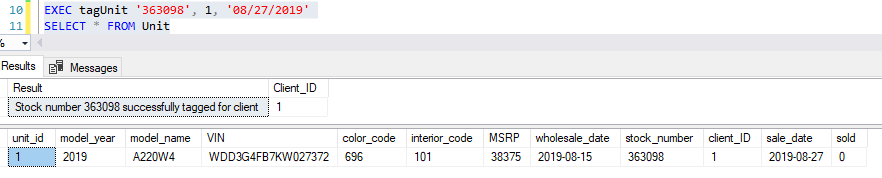
ELSE

SELECT CONCAT('Stock number ',@stock,' does not exist')

ROLLBACK TRANSACTION

END CATCH

GO



CREATE PROCEDURE sellUnit (@stock varchar(10), @saledate date, @client\_ID int)

AS

BEGIN TRANSACTION

BEGIN TRY -- "Punch" a car (sell it to a client) only if stock number exists and is not already sold

IF EXISTS (SELECT \* FROM Unit WHERE stock\_number = @stock AND sold = 0)

UPDATE Unit SET sold = 1 WHERE stock\_number=@stock

UPDATE Unit SET sale\_date =@saledate WHERE stock\_number=@stock

UPDATE Unit SET client\_ID = @client\_ID WHERE stock\_number=@stock

SELECT CONCAT('Sale of stock number ',@stock,' complete.')

COMMIT TRANSACTION

END TRY

BEGIN CATCH

IF EXISTS (SELECT \* FROM Unit WHERE stock\_number = @stock AND sold=1) --report reason why transaction does not complete

SELECT CONCAT('Stock number ',@stock,' is already sold')

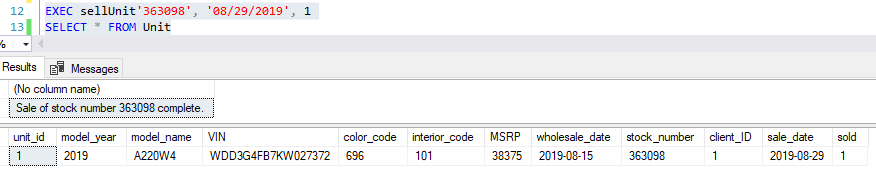
ELSE

SELECT CONCAT('Stock numer ',@stock,' does not exist')

ROLLBACK TRANSACTION

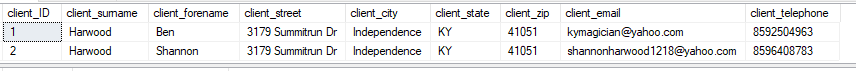
END CATCH

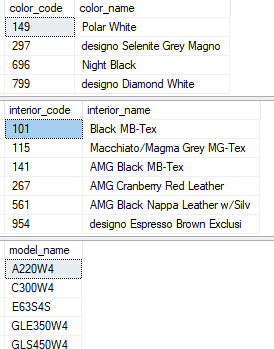
GO

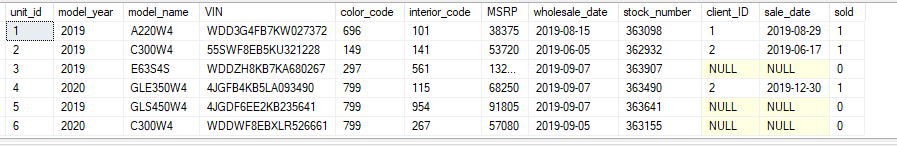


Notice that the tagUnit procedure only put the client ID and sale date (in this case when we put the client’s name on the car) in place, but the sellUnit procedure finalized the date of sale and marked the car sold.

For continued demonstration purposes, we have added sufficient additional information to our base tables to have a larger inventory, as well as some additional sales information (see Appendix B).







We also have the ability to look back at our sales for a month, quarter, or year of choice, based on input from the user:

CREATE PROCEDURE Month\_Sales (@start\_date date)

AS

IF EXISTS (SELECT \* FROM Unit WHERE MONTH(sale\_date) = month(@start\_date) AND YEAR(sale\_date) = YEAR(@start\_date) AND sold=1)

SELECT stock\_number, model\_year, model\_name, color\_code, interior\_code, VIN, MSRP, DATEDIFF(d, wholesale\_date, sale\_date) AS age FROM Unit

WHERE MONTH(sale\_date) = month(@start\_date) AND YEAR(sale\_date) = YEAR(@start\_date)

ELSE

SELECT CONCAT('No data to display for ',(SELECT DateName(month, DateAdd(month, MONTH(@start\_date),-1))),' ',YEAR(@start\_date))

GO

CREATE PROCEDURE Quarter\_Sales (@start\_date date)

AS

IF EXISTS (SELECT \* FROM Unit WHERE DATEPART(Q,sale\_date) = DATEPART(Q,@start\_date) AND YEAR(sale\_date) = YEAR(@start\_date) AND sold = 1)

SELECT stock\_number, model\_year, model\_name, color\_code, interior\_code, VIN, MSRP, DATEDIFF(d, wholesale\_date, sale\_date) AS age FROM Unit

WHERE DATEPART(Q,sale\_date) = DATEPART(Q,@start\_date) AND YEAR(sale\_date) = YEAR(@start\_date)

ELSE

SELECT CONCAT('No data to display for Q',MONTH(@start\_date)%4,' ',YEAR(@start\_date))

GO

CREATE PROCEDURE Annual\_Sales (@year int)

AS

IF EXISTS (SELECT \* FROM Unit WHERE YEAR(sale\_date) = @year AND sold = 1)

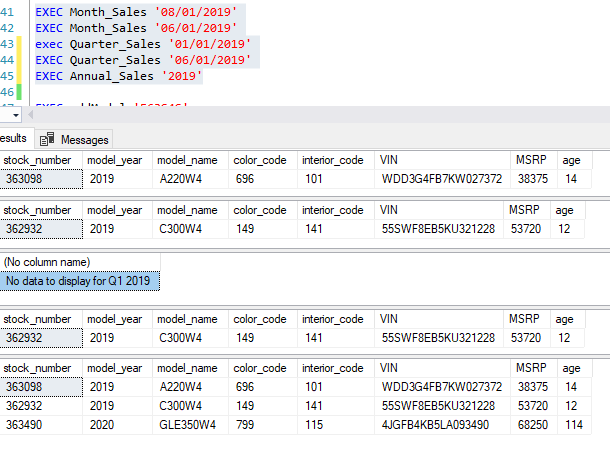
SELECT stock\_number, model\_year, model\_name, color\_code, interior\_code, VIN, MSRP, DATEDIFF(d, wholesale\_date, sale\_date) AS age FROM Unit

WHERE YEAR(sale\_date) = @year

ELSE

SELECT CONCAT('No data to display for ',YEAR(@year))

GO



The ability to check the sold status of a vehicle quickly is very helpful, as is being able to check how many units of a certain model we have available:

CREATE PROCEDURE Is\_Sold (@stock varchar(30))

AS

IF EXISTS (SELECT \* FROM Unit WHERE stock\_number=@stock and sold=1)

SELECT CONCAT('Stock number ',@stock,' has been sold.')

ELSE IF NOT EXISTS (SELECT \* FROM Unit WHERE stock\_number=@stock)

SELECT CONCAT('Stock number ',@stock,' does not exist')

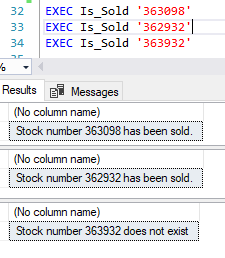
ELSE IF (SELECT client\_ID FROM Unit WHERE stock\_number=@stock) IS NOT NULL

SELECT CONCAT('Stock number ',@stock,' is currently tagged for'), (SELECT client\_ID FROM Unit WHERE stock\_number=@stock) AS Client\_ID

ELSE

SELECT CONCAT('Stock number ',@stock,' is available')

GO



CREATE PROCEDURE Available(@model varchar(30))

AS

IF EXISTS (SELECT model\_name FROM model WHERE model\_name=@model) AND EXISTS (SELECT model\_name FROM Unit WHERE model\_name=@model) AND EXISTS (SELECT model\_name FROM Unit WHERE sold =0)

SELECT TOP 100

model\_year,

model\_name,

color\_code,

interior\_code,

VIN,

MSRP,

stock\_number,

DATEDIFF(d, wholesale\_date, GETDATE()) AS age

FROM Unit

WHERE model\_name = @model

ORDER BY model\_year

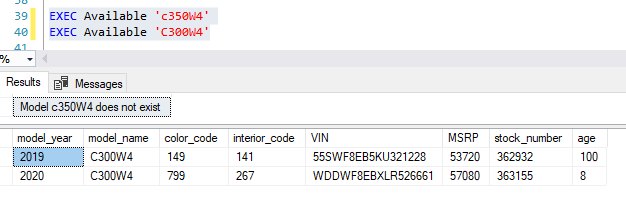
ELSE IF NOT EXISTS (SELECT model\_name FROM model WHERE model\_name=@model)

SELECT CONCAT('Model ',@model,' does not exist')

ELSE

SELECT CONCAT('No available ',@model,' units')

GO



**Data Questions**

Numerous views are available to see average inventory lot life and operational defects for both active and sold vehicles. We also have functions that provide the same but for specific, user-requested models. As these views and procedures are incredibly similar, few examples are provided and we allow the reader the option to repeat.

CREATE VIEW Inv\_AverageLotLife AS

SELECT

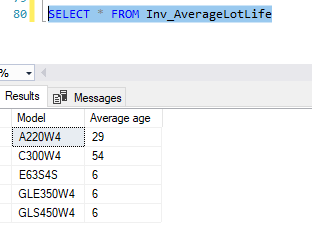
model\_name AS Model,

AVG(DATEDIFF(d, wholesale\_date, GETDATE())) AS "Average age"

FROM Unit

GROUP BY model\_name

GO



CREATE VIEW Inv\_DefectCounts AS

SELECT

model\_name AS Model,

COUNT(model\_name) AS Total,

SUM(

CASE

WHEN DATEDIFF(d, wholesale\_date, GETDATE()) > 90 then 1

ELSE 0

END)

AS Defects,

ROUND(CAST(SUM(

CASE

WHEN DATEDIFF(d, wholesale\_date, GETDATE()) > 90 then 1

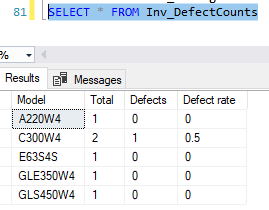
ELSE 0

END) AS FLOAT)/CAST(COUNT(model\_name) AS FLOAT),2) AS "Defect rate"

FROM Unit

GROUP BY model\_name

GO



CREATE FUNCTION Inv\_ModelLotLife (@model varchar(30))

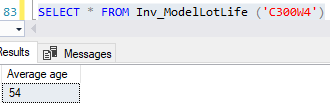
RETURNS TABLE

AS

RETURN

SELECT AVG(DATEDIFF(d, wholesale\_date, GETDATE())) AS "Average age" FROM Unit WHERE model\_name = @model

GO



CREATE FUNCTION Sold\_ModelDefects (@model varchar(30))

RETURNS TABLE

AS

RETURN

SELECT SUM(

CASE

WHEN DATEDIFF(d, wholesale\_date, sale\_date) > 90 then 1

ELSE 0

END)

AS Defects,

ROUND(CAST(SUM(

CASE

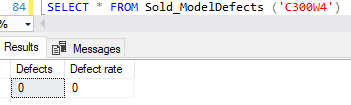
WHEN DATEDIFF(d, wholesale\_date, sale\_date) > 90 then 1

ELSE 0

END) AS FLOAT)/CAST(COUNT(model\_name) AS FLOAT),2) AS "Defect rate"

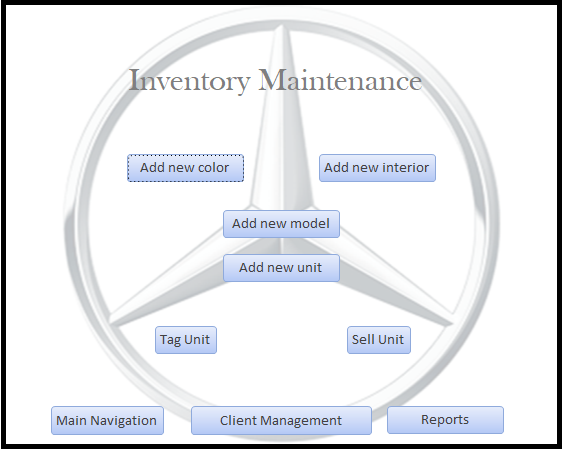
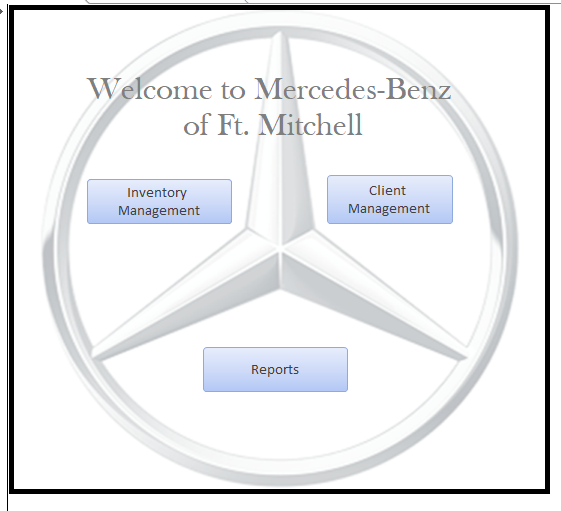
FROM Unit WHERE model\_name = @model AND sold = 1

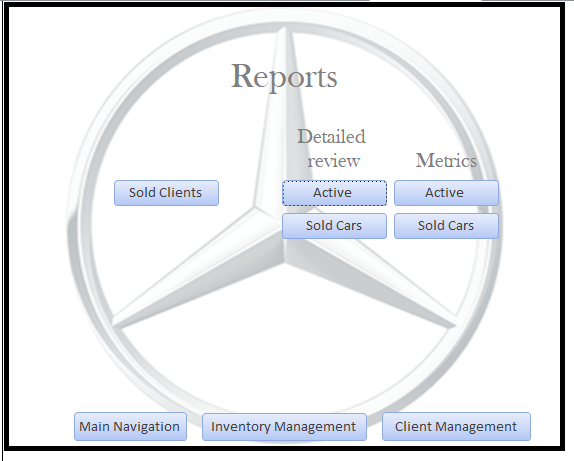
GO



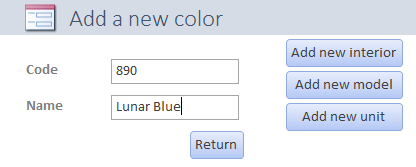
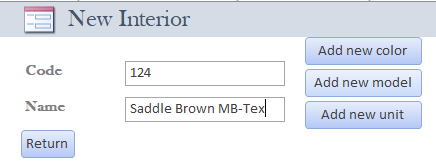
**Implementation**

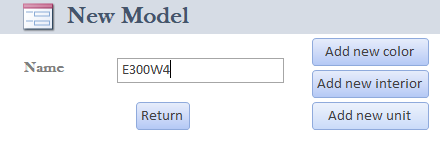
Navigation screens

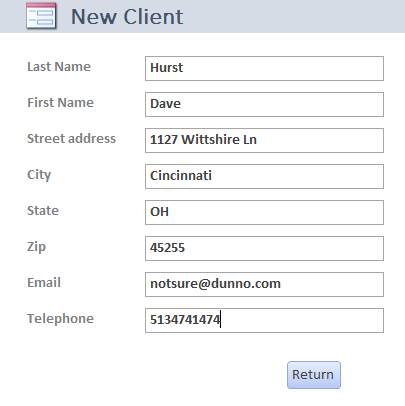


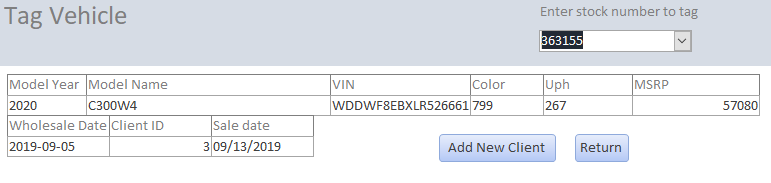
Entry screens

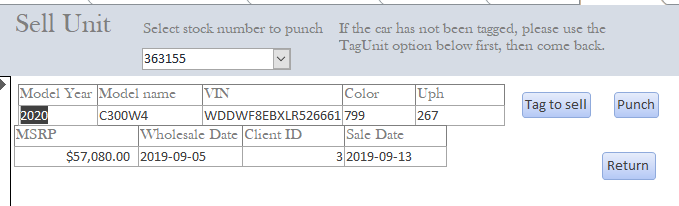
 



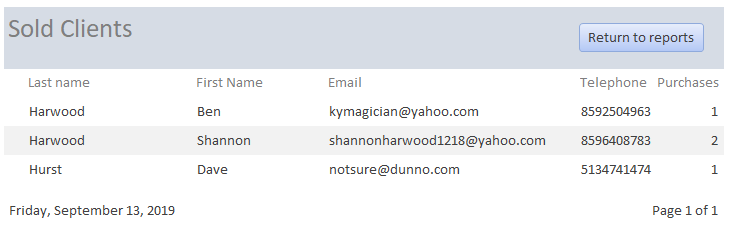
Sale processing screens

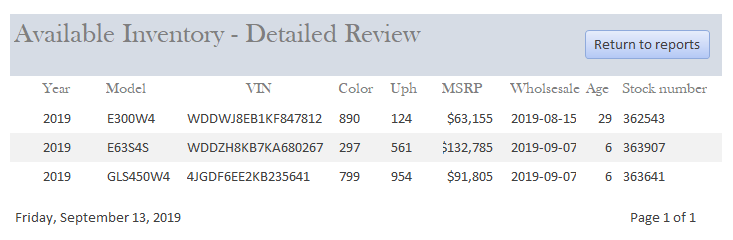


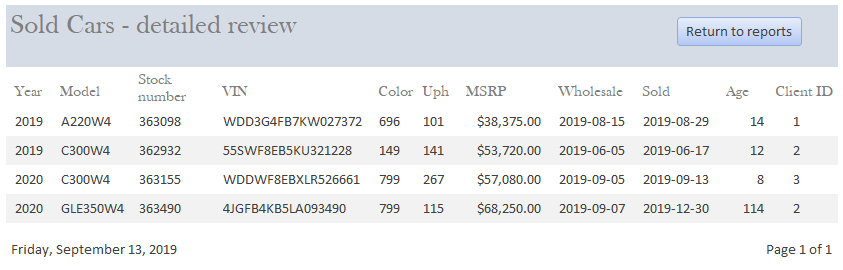


When a vehicle is sold or “punched” it is removed from the drop down menus on these screens.

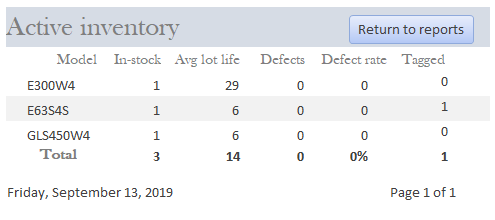
Reports:

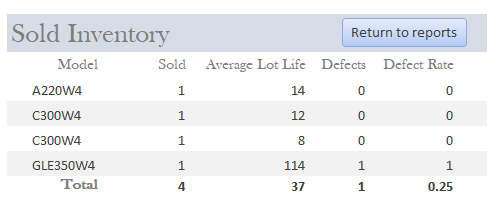






In the next screen, we have tagged stock number 363907 to client 1.





**Reflection**

At first, I just focused on building the tables. Then about week 8 I realized that all those tables were useless without the ability to actually do something with them. That’s when the magnitude of what I had undertaken set in. It was initially mind-numbing working through whether certain functions should be implemented with functions, views, or procedures, as well as navigating through syntax and trying to determine the best way to implement the updates that were inherit in what I was doing.

Once that part was done (and continued to be refined), developing the interface was very engaging… at first. Then nothing worked right, but when I realized I could basically SQL code my way through the Access queries (with the caveat that overly complex SQL could not be implemented with very deep VBA) I was able to get through that part and put together an interface that works quite well. If I had my druthers, I’d add some additional VBA code that cleaned up some of the execution of different screens, but for a database and interface that will ultimately not be implemented anywhere outside my basement, I am happy with the result.

**Appendix A – DDL**

/\* Table, view, function, and procedure drops for re-executability \*/

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

BEGIN

DROP TABLE Unit

END

GO

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

BEGIN

DROP TABLE Model

END

GO

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

BEGIN

DROP TABLE Color

END

GO

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

BEGIN

DROP TABLE Interior

END

GO

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

BEGIN

DROP TABLE Clients

END

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE type = 'p' AND name = 'addColor')

BEGIN

DROP PROCEDURE addColor

END

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE type = 'p' AND name = 'addInterior')

BEGIN

DROP PROCEDURE addInterior

END

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE type = 'p' AND name = 'addModel')

BEGIN

DROP PROCEDURE addModel

END

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE type = 'p' AND name = 'addUnit')

BEGIN

DROP PROCEDURE addUnit

END

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE type = 'p' AND name = 'sellUnit')

BEGIN

DROP PROCEDURE sellUnit

END

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE type = 'p' AND name = 'addClient')

BEGIN

DROP PROCEDURE addClient

END

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE type = 'p' AND name = 'tagUnit')

BEGIN

DROP PROCEDURE tagUnit

END

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE type = 'p' AND name = 'Month\_Sales')

BEGIN

DROP PROCEDURE Month\_Sales

END

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE type = 'p' AND name = 'Quarter\_Sales')

BEGIN

DROP PROCEDURE Quarter\_Sales

END

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE type = 'p' AND name ='Annual\_Sales')

BEGIN

DROP PROCEDURE Annual\_Sales

END

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE type = 'p' AND name = 'Is\_Sold')

BEGIN

DROP PROCEDURE Is\_Sold

END

GO

IF EXISTS (SELECT \* FROM sys.objects WHERE type = 'p' AND name = 'Available')

BEGIN

DROP PROCEDURE Available

END

GO

DROP VIEW IF EXISTS dbo.Inv\_AverageLotLife

DROP VIEW IF EXISTS dbo.Inv\_DefectCounts

DROP VIEW IF EXISTS dbo.Sold\_AverageLotLife

DROP VIEW IF EXISTS dbo.Sold\_DefectCounts

DROP VIEW IF EXISTS dbo.Sold\_Clients\_forMarketing

DROP VIEW IF EXISTS dbo.Active\_Inventory

GO

DROP FUNCTION IF EXISTS dbo.Inv\_ModelLotLife

DROP FUNCTION IF EXISTS dbo.Inv\_ModelDefects

DROP FUNCTION IF EXISTS dbo.Sold\_ModelLotLife

DROP FUNCTION IF EXISTS dbo.Sold\_Modeldefects

/\*

Table creation follows. Several are needed for various purposes, including dependency, normalization, and reporting.

\*/

CREATE TABLE Model

(

model\_name VARCHAR(30) not null

CONSTRAINT PK\_model PRIMARY KEY (model\_name)

-- Model name is used as primary key for ease of entry and instance creation

)

CREATE TABLE Color

(

color\_code varchar(3) NOT NULL,

color\_name varchar(30) NOT NULL,

-- Color code is used as primary key for the same reason as in the model table

CONSTRAINT PK\_color PRIMARY KEY (color\_code),

)

GO

CREATE TABLE Interior

(

interior\_code varchar(3) NOT NULL,

interior\_name varchar(30) NOT NULL,

-- Same consistent reason for using interior code as PK

CONSTRAINT PK\_interior PRIMARY KEY (interior\_code),

)

GO

CREATE TABLE Clients -- It would be helpful to have all of our clients' information for reference and marketing purposes

(

client\_ID int IDENTITY,

client\_surname varchar(30) NOT NULL,

client\_forename varchar(30) NOT NULL,

client\_street varchar(50) NOT NULL,

client\_city varchar(20) NOT NULL,

client\_state varchar(2) NOT NULL,

client\_zip varchar(5) NOT NULL,

client\_email varchar(50) NOT NULL UNIQUE,

client\_telephone varchar(10) NOT NULL UNIQUE,

CONSTRAINT PK\_client PRIMARY KEY (client\_ID),

)

GO

CREATE TABLE Unit -- this is our main table for inventory

(

unit\_id int IDENTITY,

model\_year varchar(4) NOT NULL,

model\_name varchar(30) NOT NULL,

VIN varchar(17) NOT NULL,

color\_code varchar(3) NOT NULL,

interior\_code varchar(3) NOT NULL,

MSRP int NOT NULL,

wholesale\_date date NOT NULL,

stock\_number varchar(20), -- Note that this can be null at first. This is because we may need to enter vehicles into inventory before stock number assignment, however the stock number is required to process a sale

client\_ID int, -- Only created when the vehicle is in the sale process (tagged)

sale\_date date, -- This is entered when a vehicle is entered into the sale process (tagged) and subsequently moved to the sold table

sold bit DEFAULT 0, -- Indicates if vehicle sale has been finalized

CONSTRAINT PK\_unit PRIMARY KEY (unit\_id),

CONSTRAINT FK\_model FOREIGN KEY (model\_name) REFERENCES Model(model\_name),

CONSTRAINT FK\_color FOREIGN KEY (color\_code) REFERENCES Color(color\_code),

CONSTRAINT FK\_interior FOREIGN KEY (interior\_code) REFERENCES Interior(interior\_code),

CONSTRAINT FK\_unit\_client FOREIGN KEY (client\_ID) REFERENCES Clients(client\_ID),

CONSTRAINT U1\_unit UNIQUE (VIN),

CONSTRAINT U2\_unit UNIQUE (stock\_number)

)

GO

/\*

With tables created, we move to functionality.

\*/

-- We need ways to quickly add colors, interiors, and models. The following three procedures provide this for us. Each checks to see if the color, interior, or model are already present.

CREATE PROCEDURE addColor (@code varchar(3), @name varchar(30))

AS

BEGIN TRANSACTION

BEGIN TRY -- check for redundany, i.e. if the color is already in the database

INSERT INTO color (color\_code, color\_name)

VALUES (@code, @name)

SELECT 'Color successfully added'

COMMIT TRANSACTION

END TRY

BEGIN CATCH

SELECT 'Color already exists'

ROLLBACK TRANSACTION

END CATCH

GO

CREATE PROCEDURE addInterior (@code varchar(3), @name varchar(30))

AS

BEGIN TRANSACTION

BEGIN TRY -- similar check as for color

INSERT INTO Interior(interior\_code, interior\_name)

VALUES (@code, @name)

SELECT 'Interior successfully added'

COMMIT TRANSACTION

END TRY

BEGIN CATCH

SELECT 'Interior already exists'

ROLLBACK TRANSACTION

END CATCH

GO

CREATE PROCEDURE addModel (@name varchar(30))

AS

BEGIN TRANSACTION

BEGIN TRY -- similar check as for color and interior

INSERT INTO Model(model\_name)

VALUES (@name)

SELECT 'Model successfully added'

COMMIT TRANSACTION

END TRY

BEGIN CATCH

SELECT 'Model already exists'

ROLLBACK TRANSACTION

END CATCH

GO

-- This procedure allows us to add individuals to our client base.

CREATE PROCEDURE addClient (@lastname varchar(30), @firstname varchar(30), @street varchar(50), @city varchar(20), @state varchar(2), @zip varchar(5), @email varchar(50), @phone varchar(10))

AS

BEGIN TRANSACTION

BEGIN TRY -- checking for client existence

INSERT INTO Clients(client\_surname, client\_forename, client\_street, client\_city, client\_state, client\_zip, client\_email, client\_telephone)

VALUES (@lastname, @firstname, @street, @city, @state, @zip, @email, @phone)

SELECT CONCAT('Client ', @firstname,' ',@lastname,' succesfully added'), (Select max(client\_ID) FROM Clients) AS Client\_ID

COMMIT TRANSACTION

END TRY

BEGIN CATCH

IF EXISTS (SELECT \* FROM Clients WHERE client\_email=@email) --report reason why transaction does not complete

SELECT CONCAT('Email ',@email,' already exists for client') AS Result,(SELECT client\_ID FROM Clients WHERE client\_telephone=@phone) AS Client\_ID

ELSE

SELECT CONCAT('Phone number ',@phone,' already exists') AS Result, (SELECT client\_ID FROM Clients WHERE client\_telephone=@phone) AS Client\_ID

ROLLBACK TRANSACTION

END CATCH

GO

-- Obviously we need a way to enter new vehicles into inventory. This procedure takes the requisite information from the user and inserts it into a new line in the Unit table.

CREATE PROCEDURE addUnit (@year varchar(4), @model varchar(30), @vin varchar(17), @color varchar(3), @interior varchar(3), @msrp int, @wholesale date, @stock varchar(10))

AS

BEGIN TRANSACTION

BEGIN TRY -- check to see if the unit we are trying to add is already in our inventory or already sold. If not, complete the addition

INSERT INTO Unit(model\_year, model\_name, VIN, color\_code, interior\_code, MSRP, wholesale\_date, stock\_number)

VALUES (@year, @model, @vin, @color, @interior, @msrp, @wholesale, @stock)

SELECT CONCAT('Stock number',' ',@stock,' ','added successfully')

COMMIT TRANSACTION

END TRY

BEGIN CATCH

IF EXISTS (SELECT \* FROM Sold WHERE stock\_number = @stock) --report reason why transaction does not complete

SELECT CONCAT('Stock number ',@stock,' was sold on ', (SELECT sale\_Date FROM Sold WHERE stock\_number=@stock))

ELSE IF NOT EXISTS (SELECT \* FROM Model WHERE model\_name=@model)

SELECT CONCAT('Model ',@model,' does not exist')

ELSE IF NOT EXISTS (SELECT \* FROM Color WHERE color\_code=@color)

SELECT CONCAT('Color code ',@color,' does not exist')

ELSE IF NOT EXISTS (SELECT \* FROM Interior WHERE interior\_code=@interior)

SELECT CONCAT('Interior code ',@interior,' does not exist')

ELSE IF EXISTS (SELECT \* FROM Unit WHERE VIN=@vin)

SELECT CONCAT('VIN ',@VIN,' is already in inventory with stock number ',(SELECT stock\_number FROM Unit WHERE VIN =@vin))

ELSE IF EXISTS (SELECT \* FROM Unit WHERE stock\_number=@stock)

SELECT CONCAT('Stock number ',@stock,' already exists')

ROLLBACK TRANSACTION

END CATCH

GO

-- "Tagging" a car is industry-speak for putting a client's name on the car, indicating a deal has been agreed to but not yet finalized.

CREATE PROCEDURE tagUnit (@stock varchar(10), @client int, @sold date)

AS

BEGIN TRANSACTION

BEGIN TRY -- only commit if stock number exists in inventory, is not sold already, and the client exists

IF EXISTS (SELECT \* FROM Unit WHERE stock\_number = @stock AND sold =0) AND EXISTS (SELECT \* FROM Clients WHERE client\_ID = @client)

UPDATE Unit SET sale\_date = @sold

WHERE stock\_number =@stock

UPDATE Unit SET client\_ID = @client

WHERE stock\_number = @stock

SELECT CONCAT('Stock number ',@stock,' successfully tagged for client') AS Result, (SELECT client\_ID FROM Clients WHERE client\_ID=@client) AS Client\_ID

COMMIT TRANSACTION

END TRY

BEGIN CATCH

IF NOT EXISTS (SELECT \* FROM Clients WHERE client\_ID = @client) --report reason why transaction does not complete

SELECT 'Client does not exist. Please add them'

ELSE IF (SELECT client\_ID FROM Unit WHERE stock\_number=@stock)=@client

SELECT 'No changes necessary'

ELSE IF EXISTS (SELECT \* FROM Sold WHERE stock\_number = @stock)

SELECT CONCAT('Stock number ',@stock,' is already sold')

ELSE

SELECT CONCAT('Stock number ',@stock,' does not exist')

ROLLBACK TRANSACTION

END CATCH

GO

-- The following procedure allows us to process a sale: logging the date of sale and re-calculating the age to be how long the car sat before it sold.

CREATE PROCEDURE sellUnit (@stock varchar(10), @saledate date, @client\_ID int)

AS

BEGIN TRANSACTION

BEGIN TRY -- "Punch" a car (sell it to a client) only if stock number exists and is not already sold

IF EXISTS (SELECT \* FROM Unit WHERE stock\_number = @stock AND sold = 0)

UPDATE Unit SET sold = 1 WHERE stock\_number=@stock

UPDATE Unit SET sale\_date =@saledate WHERE stock\_number=@stock

UPDATE Unit SET client\_ID = @client\_ID WHERE stock\_number=@stock

SELECT CONCAT('Sale of stock number ',@stock,' complete.')

COMMIT TRANSACTION

END TRY

BEGIN CATCH

IF EXISTS (SELECT \* FROM Unit WHERE stock\_number = @stock AND sold=1) --report reason why transaction does not complete

SELECT CONCAT('Stock number ',@stock,' is already sold')

ELSE

SELECT CONCAT('Stock numer ',@stock,' does not exist')

ROLLBACK TRANSACTION

END CATCH

GO

/\*

All of this data is great, but we'd like to be able to quickly generate reports or use it for something. The next few procedures will show sales data for month, quarter,

and year based on a date entered by the user. There is also a view for looking at our entire active inventory, a procedure to allow for checking if a certain stock number is sold,

and a procedure to quickly see available inventory for specific models.

\*/

CREATE PROCEDURE Month\_Sales (@start\_date date)

AS

IF EXISTS (SELECT \* FROM Unit WHERE MONTH(sale\_date) = month(@start\_date) AND YEAR(sale\_date) = YEAR(@start\_date) AND sold=1)

SELECT stock\_number, model\_year, model\_name, color\_code, interior\_code, VIN, MSRP, DATEDIFF(d, wholesale\_date, sale\_date) AS age FROM Unit

WHERE MONTH(sale\_date) = month(@start\_date) AND YEAR(sale\_date) = YEAR(@start\_date)

ELSE

SELECT CONCAT('No data to display for ',(SELECT DateName(month, DateAdd(month, MONTH(@start\_date),-1))),' ',YEAR(@start\_date))

GO

CREATE PROCEDURE Quarter\_Sales (@start\_date date)

AS

IF EXISTS (SELECT \* FROM Unit WHERE DATEPART(Q,sale\_date) = DATEPART(Q,@start\_date) AND YEAR(sale\_date) = YEAR(@start\_date) AND sold = 1)

SELECT stock\_number, model\_year, model\_name, color\_code, interior\_code, VIN, MSRP, DATEDIFF(d, wholesale\_date, sale\_date) AS age FROM Unit

WHERE DATEPART(Q,sale\_date) = DATEPART(Q,@start\_date) AND YEAR(sale\_date) = YEAR(@start\_date)

ELSE

SELECT CONCAT('No data to display for Q',MONTH(@start\_date)%4,' ',YEAR(@start\_date))

GO

CREATE PROCEDURE Annual\_Sales (@year int)

AS

IF EXISTS (SELECT \* FROM Unit WHERE YEAR(sale\_date) = @year AND sold = 1)

SELECT stock\_number, model\_year, model\_name, color\_code, interior\_code, VIN, MSRP, DATEDIFF(d, wholesale\_date, sale\_date) AS age FROM Unit

WHERE YEAR(sale\_date) = @year

ELSE

SELECT CONCAT('No data to display for ',YEAR(@year))

GO

CREATE VIEW Active\_Inventory AS

SELECT TOP 100

model\_year,

model\_name,

color\_code,

interior\_code,

VIN,

MSRP,

stock\_number,

DATEDIFF(d, wholesale\_date, GETDATE()) AS age

FROM Unit

WHERE sold = 0

ORDER BY model\_name, model\_year

GO

CREATE PROCEDURE Is\_Sold (@stock varchar(30))

AS

IF EXISTS (SELECT \* FROM Unit WHERE stock\_number=@stock and sold=1)

SELECT CONCAT('Stock number ',@stock,' has been sold.')

ELSE IF NOT EXISTS (SELECT \* FROM Unit WHERE stock\_number=@stock)

SELECT CONCAT('Stock number ',@stock,' does not exist')

ELSE IF (SELECT client\_ID FROM Unit WHERE stock\_number=@stock) IS NOT NULL

SELECT CONCAT('Stock number ',@stock,' is currently tagged for'), (SELECT client\_ID FROM Unit WHERE stock\_number=@stock) AS Client\_ID

ELSE

SELECT CONCAT('Stock number ',@stock,' is available')

GO

CREATE PROCEDURE Available(@model varchar(30))

AS

IF EXISTS (SELECT model\_name FROM model WHERE model\_name=@model) AND EXISTS (SELECT model\_name FROM Unit WHERE model\_name=@model) AND EXISTS (SELECT model\_name FROM Unit WHERE sold =0)

SELECT TOP 100

model\_year,

model\_name,

color\_code,

interior\_code,

VIN,

MSRP,

stock\_number,

DATEDIFF(d, wholesale\_date, GETDATE()) AS age

FROM Unit

WHERE model\_name = @model

ORDER BY model\_year

ELSE IF NOT EXISTS (SELECT model\_name FROM model WHERE model\_name=@model)

SELECT CONCAT('Model ',@model,' does not exist')

ELSE

SELECT CONCAT('No available ',@model,' units')

GO

/\*

Lot life is how long a car sits on the lot before it is sold. With an expressed goal of minimizing this for revenue purposes, this is important to be able to track.

Similarly, we consider any car with lot life greater than 90 days to be an operational "defect". We have views coded to see both for active inventory and sold vehicles from high level,

as well as functions that accept a model name and return the average lot life and defectfor both active inventory and sold units.

\*/

CREATE VIEW Inv\_AverageLotLife AS

SELECT

model\_name AS Model,

AVG(DATEDIFF(d, wholesale\_date, GETDATE())) AS "Average age"

FROM Unit

GROUP BY model\_name

GO

CREATE VIEW Inv\_DefectCounts AS

SELECT

model\_name AS Model,

COUNT(model\_name) AS Total,

SUM(

CASE

WHEN DATEDIFF(d, wholesale\_date, GETDATE()) > 90 then 1

ELSE 0

END)

AS Defects,

ROUND(CAST(SUM(

CASE

WHEN DATEDIFF(d, wholesale\_date, GETDATE()) > 90 then 1

ELSE 0

END) AS FLOAT)/CAST(COUNT(model\_name) AS FLOAT),2) AS "Defect rate"

FROM Unit

GROUP BY model\_name

GO

CREATE VIEW Sold\_AverageLotLife AS

SELECT

model\_name AS Model,

AVG(DATEDIFF(d, wholesale\_date, sale\_date)) AS "Average age"

FROM Unit WHERE sold =1

GROUP BY model\_name

GO

CREATE VIEW Sold\_DefectCounts AS

SELECT

model\_name AS Model,

SUM(

CASE

WHEN DATEDIFF(d, wholesale\_date, sale\_date) > 90 then 1

ELSE 0

END)

AS Defects,

ROUND(CAST(SUM(

CASE

WHEN DATEDIFF(d, wholesale\_date, sale\_date) > 90 then 1

ELSE 0

END) AS FLOAT)/CAST(COUNT(model\_name) AS FLOAT),2) AS "Defect rate"

FROM Unit where sold =1

GROUP BY model\_name

GO

CREATE FUNCTION Inv\_ModelLotLife (@model varchar(30))

RETURNS TABLE

AS

RETURN

SELECT AVG(DATEDIFF(d, wholesale\_date, GETDATE())) AS "Average age" FROM Unit WHERE model\_name = @model

GO

CREATE FUNCTION Inv\_ModelDefects (@model varchar(30))

RETURNS TABLE

AS

RETURN

SELECT SUM(

CASE

WHEN DATEDIFF(d, wholesale\_date, GETDATE()) > 90 then 1

ELSE 0

END)

AS Defects,

ROUND(CAST(SUM(

CASE

WHEN DATEDIFF(d, wholesale\_date, GETDATE()) > 90 then 1

ELSE 0

END) AS FLOAT)/CAST(COUNT(model\_name) AS FLOAT),2) AS "Defect rate"

FROM Unit WHERE model\_name = @model

GO

CREATE FUNCTION Sold\_ModelLotLife (@model varchar(30))

RETURNS TABLE

AS

RETURN

SELECT AVG(DATEDIFF(d, wholesale\_date, sale\_date)) AS "Average age" FROM Unit WHERE model\_name = @model AND sold =1

GO

CREATE FUNCTION Sold\_ModelDefects (@model varchar(30))

RETURNS TABLE

AS

RETURN

SELECT SUM(

CASE

WHEN DATEDIFF(d, wholesale\_date, sale\_date) > 90 then 1

ELSE 0

END)

AS Defects,

ROUND(CAST(SUM(

CASE

WHEN DATEDIFF(d, wholesale\_date, sale\_date) > 90 then 1

ELSE 0

END) AS FLOAT)/CAST(COUNT(model\_name) AS FLOAT),2) AS "Defect rate"

FROM Unit WHERE model\_name = @model AND sold = 1

GO

/\*

This view provides a concise list of our clients who purchased vehicles along with their email addresses, for digital marketing purposes. This did not make it into the implementation.

\*/

CREATE VIEW Sold\_Clients\_forMarketing AS

SELECT DISTINCT TOP 100

client\_surname AS Last\_Name,

client\_forename AS First\_Name,

client\_email AS Email,

COUNT(Unit.client\_ID) AS Purchases

FROM Clients JOIN Unit ON Clients.client\_ID=Unit.client\_ID

WHERE Unit.sold =1

GROUP BY client\_surname, client\_forename, client\_email

ORDER BY client\_surname, client\_forename

GO

**Appendix B – Code to bring database to status of data manipulation section**

EXEC addColor '696', 'Night Black'

SELECT \* FROM Color

EXEC addInterior '101', 'Black MB-Tex'

SELECT\* FROM Interior

EXEC addModel 'A220W4'

SELECT \* FROM Model

EXEC addUnit '2019', 'A220W4', 'WDD3G4FB7KW027372', '696', '101', 38375, '08/15/2019', '363098'

SELECT \* FROM Unit

EXEC addClient 'Harwood', 'Ben', '3179 Summitrun Dr', 'Independence', 'KY', '41051', 'kymagician@yahoo.com', '8592504963'

EXEC addClient 'Harwood', 'Shannon', '3179 Summitrun Dr', 'Independence', 'KY', 41051, 'shannonharwood1218@yahoo.com', '8596408783'

SELECT \* FROM Clients

EXEC tagUnit '363098', 1, '08/27/2019'

SELECT \* FROM Unit

EXEC sellUnit'363098', '08/29/2019', 1

SELECT \* FROM Unit

SELECT \* FROM Active\_Inventory

EXEC addColor '149', 'Polar White'

EXEC addInterior'141', 'AMG Black MB-Tex'

EXEC addModel 'C300W4'

EXEC addUnit '2019', 'C300W4', '55SWF8EB5KU321228', '149', '141', 53720, '6/5/2019', '362932'

EXEC tagUnit '362932', 2, '09/06/2019'

EXEC sellUnit 362932, '6/17/2019', 2

EXEC addInterior '201', 'Black Leather'

EXEC addColor '992', 'Selenite Grey'

EXEC addInterior '205', 'Beige Leather'

EXEC Available 'c350W4'

EXEC Available 'C300W4'

EXEC addModel 'E63S4S'

EXEC addColor '297', 'designo Selenite Grey Magno'

EXEC addInterior '561', 'AMG Black Nappa Leather w/Silver Stitching'

EXEC addUnit 2019, 'E63S4S', 'WDDZH8KB7KA680267', '297', '561', 132785, '09/07/2019', 363907

EXEC addModel 'GLE350W4'

EXEC addColor '799', 'designo Diamond White'

EXEC addInterior '115', 'Macchiato/Magma Grey MG-Tex'

EXEC addUnit '2020', 'GLE350W4', '4JGFB4KB5LA093490', 799, 115, 68250, '09/07/2019', 363490

EXEC addModel 'GLS450W4'

EXEC addInterior 954, 'designo Espresso Brown Exclusive Nappa Leather'

EXEC addUnit '2019', 'GLS450W4', '4JGDF6EE2KB235641', 799, 954, 91805, '09/07/2019', 363641

EXEC addInterior 267, 'AMG Cranberry Red Leather'

exec addUnit '2020', 'C300W4', 'WDDWF8EBXLR526661', 799, 267, 57080, '09/05/2019', 363155

exec tagUnit 363490, 2, '9/8/2019'

EXEC sellUnit 363490, '12/30/2019', 2