Name: Mi Zhang

Table of Contents

Project Direction Overview

I would like to develop a mobile app or website which provides an all-in-one system to help people to buy, sell, or trade-in vehicles, named "AutoBuy". This idea is inspired by my family in which my brother had a car accident in March and my mom had a car accident in August this year. Luckily, they didn't get hurt that much in those two car accidents, but both cars were totaled. And due to the worldwide automotive microchip shortage, the price of cars surged to new high records in the last two years. As my family struggled through the process of vehicle purchasing, I came up with the idea that there should be an all-in-one system help people to choose a car, buy a car, and even include auto insurance purchases.

In addition, this is not just for car dealers to sell their available cars, but also for private owners to sell their cars. One reason to include auto insurance here is that we usually need to compare multiple different insurance companies to find the best price, and we need to fill in personal information across different websites multiple times to get quotes, and usually end up receiving all kinds of advertisement emails. With AutoBuy, you only need to fill in the information once and get quotes directly shown in the app, and you can purchase auto insurance directly on the app.

Building an app or website requires tremendous programming skills, but I will focus on the database component here. I hope this app or website can help people buy, sell, or trade in cars easily and smoothly. In addition, I hope this database can make the price transparent, especially for people like me who do not how to bargain a price.

Use Cases and Fields

First, we need customers who want to buy, sell, or trade in cars.

Account Signup/Installation Use Case

1. The customer visits AutoBuy's website or app store and install the application
2. The application asks them to create either a free or paid account when its first run
3. The customer selects the type of account and enters their information and the account is created in the database, some account information includes name, email, and zip code.

Field	What it stores	Why it's Needed
AccountName	This field stores a unique user name created by the account user.	This is necessary for people's privacy in the case of people don't want to reveal their real name, or if people have multiple accounts in the app.
AccountEmail	This is the email address of the account holder	User can use this email to activate account, backup account if forget password, or receive confirmation
FirstName	First name of the account holder.	This is necessary for displaying the person's name on screens and addressing them when

		sending them emails or other communications.
LastName	Last name of the account holder.	This is necessary for displaying the person's name on screens and addressing them when sending them emails or other communications.
Zipcode	Zipcode will give the residency information of the account holder.	This is necessary because vehicle prices, insurance policy prices, or availabilities are different in different cities or states.
EncryptedPassword	This is the password for the account.	This is necessary to protect customer's account and ensure it is customer self that make the order.
RenewalDate	This is renewal date for renew paid account.	This acts as a reminder to remind customer to renew their paid account in order to get better services, including more cars, better price, or so on.
Balance	This is the account balance added by the customer for their paid account.	This can be used to purchase the premium membership or even can be as booking fee when purchase a car.

We then need car dealers or retails that can provide vehicle inventories, and their information should be public and trustworthy.

Automotive retailers/Dealerships signup use case

1. Car dealers or retail stores decide to cooperate with AutoBuy and they start to visit the website or download the app
2. The application asks them to register their certified information
3. Retailer fills in their information as a company and this information adds to the database
4. After successfully registering basic information, the retailer starts to add individual salespeople's information to the database

Field	What it stores	Why it's Needed
RetailName	This is the name of the automotive dealership.	This is necessary for the retailer's name to display on the screens and customer can checker their reliability
RetailLocation	The retail store location.	This is necessary if people want to schedule a test drive or visit the retail store, they need to know where it is.

Description	Description about business of the retail store	This is necessary for customers to know their expertise, like what service they provide, how many of experiences, and so on
SaleFirstName	Sale's first name	This is necessary because retail can employ many salespersons and this can help customer to find a dealer that he/she wants to talk to.
SaleLastName	Sale's last name	This addresses the dealer's name
RetailPhone	Phone number of retail store	This is necessary for customer to contact them for more information or helps.

When we have customers and car dealers, business starts.

People/Dealers List Vehicle use case

1. The user decides to sell his car on AutoBuy
2. The user logins into the app and start a sale page
3. The user fills in his vehicle's detailed information and this adds the data to the database

Field	What it stores	Why it's Needed
listing_price	This is the price of the car	It is needed for customer to know and they can compare the price for different cars.
Make	This is the make of the car	It is necessary to show different brands of cars.
Year	The year of the car is certified.	This can tell the age of the car and has a direct impact on the price
Model	This is the model of car	This can show what kind car it is and people can choose their like model.
BodyType	A car body type is a categorization of a vehicle based on its design, shape and space.	This is necessary because it tells the size of the car :SUV, sedan, coupe…
VIN	This is Vehicle Identification Number(VIN).	This is necessary because it is unique for each car, and people can use it to find car associated history: owner, car accident….
Mileage	This is the total mileage shown on the odometer.	It's an indication of the amount of wear and tear

		the car has sustained over time.
ExteriorColor	The exterior color of the car	This is necessary because people can choose their like color.
InteriorColor	The interior color of the car	This is necessary because people can choose their like color.
AvailableDate	This is the date that the car going to be available	This is necessary because this tells people when the car is going to be available, people can modify their appointment or purchase date.

Vehicle Purchase Order use case

1. The user decides to buy a car on AutoBuy
2. User login into the application and begin to look around
3. The user selects a retail store or dealership that is near her resident zip code and heads to the page.
4. The user browsers by categories and searches for her favorite car.
5. The user makes her/his decision and adds a car to her shopping cart and fills in her personal information to check out.
6. The user then chooses a date and time at her convenience and confirms the schedule to pick up the car.

Field	What it stores	Why it's needed
OrderNumber	This is the confirmation for the purchase.	This is necessary to let the customer know she/he has successfully made a purchase.
ScheuleDate	This is the schedule date and time to pick up the car.	This is necessary for customer and retailer to set up the appoint.
CarDetail	This can be car that the customer wants to buy.	This is necessary for the car dealer to prepare the car and make sure it is available on the date of the appointment.

I noticed that some fields are repeated from previous tables, so for later on when I have those data, I will use foreign key constraints for those repeated ones.

Loan Application Use Case

1. After a vehicle purchase order is placed, a salesperson is appointed for the order.
2. Customer is told that she/he needs to fill out the loan application in order to buy the car, and decides the amount of down payment she/he is going to pay

3. Customer enters her/his information
4. Salesperson processes the loan application and send it to the bank, then the loan has been approved for purchase, and confirmation is generated.

Field	What it stores	Why it is needed
ConfirmationNumber	This is the confirmation for the order.	This is necessary to let the customer know she/he has successfully got loan approved.
FinanceCompany	This is the name of credit unio or bank that provide loan	This is necessary for the customer to know which lender approves her/his loan and she/he needs to pay back
Annual_Rate	The cost of the customer's credit as a yearly rate.	This is linked with the customer's credit score and tells the finance interest
FinanceCharge	The dollar amounts the credit will cost.	This is necessary to let the customer know the total of interests charged for her/he.
AmountFinanced	The amount of credit provided to you or on your behalf	This is necessary to let the customer know the amount of loan she/he has applied.
TotalofPayments	The amount you will have paid after you have made all payments as scheduled	This is necessary to let the customer know the total payment with interest that she needs to pay
Downpayment	This is the cash that the buyer pays upfront in a purchase.	This is necessary because it is required by law to pay to purchase a car unless a special policy is provided.
TotalSalePrice	The total cost of your purchase on credit, including your downpayment	This is necessary to tell the customer the total sale price for customer's right to know.

Auto Insurance Order Use case

1. The user just purchased a vehicle, she now decides to purchase auto insurance for her car
2. The user opens the app and starts to fill in her new car information and select her insurance coverages.
3. The user carefully compares different insurance companies' quote in the app, and decide to pick one insurance company.
4. The user then fills in her payment information and billing address and the order is placed.
5. A confirmation number and insurance policy number are generated to confirm the order is placed successfully.

Field	What it stores	Why it is needed

ConfirmationNumber	This is the confirmation for the order, or payment.	This is necessary to let the customer know she/he has successfully purchased the insurance.
InsuranceCompany	This is the name of the insurance company	This is necessary to let the user know which auto insurance company she/he has chosen.
PolicyNumber	This is the policy number of the insurance.	This is necessary because each policy number is unique assigned by insurance company as a prove of purchase.
CarDetail	This can be car that is insured.	This is necessary to confirm the user chooses the car to be insured.
EffectiveDate	This is the effective date of the auto insurance	This is necessary because the user can know the starting date of her/his insurance.
ExpirationDate	This is the expiration date of the insurance	This is necessary because the user can know the ending date of her/his insurance.

Structural Database Rules

Account Signup/Installation Use Case

1. The customer visits AutoBuy's website or app store and install the application
2. The application asks them to create either a free or paid account when its first run
3. The customer selects the type of account and enters their information and the account is created in the database, some account information includes name, email, and zip code.

From this use case, I see one major entity – Account. As inspired by the iteration example, I decide to make create two kinds of accounts, paid or free, as mentioned in #2. By doing so, I created a new database rule here using the Account entity.

1) An account is a free account or paid account.

This specialization-generalization rule indicates that my database only has two kinds of accounts—free and paid—and that is the complete list. Notice that Account is the supertype and free and paid accounts are subtypes in the relationship. The relationship is totally complete and disjoint because the account must be either free or paid.

Automotive retailers/Dealerships signup use case

1. Car dealers or retail stores decide to cooperate with AutoBuy and they start to visit the website or download the app
2. The application asks them to register their certified information
3. Retailer fills in their information as a company and this information adds to the database
1. After successfully registering basic information, the retailer then tell each individual salesperson to add their own information

4.

From this use case, I see two significant data points as entities, Automotive Dealership and Salesman. First of all, we have automotive retails or dealerships register their accounts as big companies, and then each retail or dealership hires many salesmen that work for them to list cars on AutoBuy. I think I have enough information to create some structural database rules.

The structural database rules:

2) Each automotive dealership hires many salespeople; Each salesperson is hired by one or many automotive dealerships.

The structural rule indicates that each dealership hires many salespeople since a dealership will have many cars and many different businesses going on, hire a car salesperson which ensures the store functions properly and each salesperson has their own tasks. I make dealership to salesman mandatory and the relationship is plural. However, I made salesmen to dealership mandatory but can be singular or plural because not all salesmen work for more than one dealership.

People/Salesperson list their own vehicle use case

1. The user decides to sell his cars on AutoBuy
2. The user logins into the app and start a sale page
3. The user fills in his vehicle's detailed information and this adds the data to the database, information includes price, make, model, mileage, available date, and so on.

From this use case, I see two significant data points as entities, People/Salespeople and Car. Like what I mentioned before, a person can privately sell his/her car on AutoBuy, but most of the car inventories on AutoBuy are added by salespeople from dealerships. We need salespeople to provide car inventories and deal with orders which will be mentioned in the next use case, and we need abundant cars for customers to choose from and buy. I think I have enough information to create some structural database rules.

3) A salesperson can sell many cars; Each car is sold by one to many salespersons.

The structural rule indicates that a salesperson can sell many cars, and a car can be sold multiples times by different salespersons. I make the Salesperson to Car optional but plural because a salesperson may sell many cars or may not sell any cars. I also make the Car to Salesperson mandatory but plural which makes the participation constraint clear.

Vehicle Purchase use case

1. The user decides to buy a car on AutoBuy
2. User login into the application and begin to look around
3. The user selects a retail store or dealership that is near her resident zip code and heads to the page.
4. The user browsers by categories and searches for her favorite car.
5. The user makes her/his decision and adds a car to her shopping cart and fills in her personal information to check out.
6. The user then chooses a date and time at her convenience and confirms the schedule to pick up the car.

7. Car dealership receives the order and a salesperson is appointed to handle the order.

From this use case, I see three significant data points as entities, Car, Purchase, and Dealership. The user chooses a car and places an order to buy the car in her/his account. And recall the Account entity we discovered in the first use case, Account Signup/Installation Use Case, we can now link the Order entity to the Account entity. The ordered car is owned by the dealership, and when the dealership receives the order, a salesperson will be appointed to accept the order and process the order to sell the car. Notice that each dealership may involve in more than one order. Since we describe the Salesperon to Car in the previous use case, we are not going to mention it again here. We now have enough information to create some structural database rules.

4) An account may have many orders; Each order is created by an account.

The structural rule indicates that each order is placed by an account, however, an account can have zero or many orders placed because an account exists before any order is placed. I make the Account to Order optional but plural. And I make the Order to account mandatory and singular because an order must be created along with the creation of an order number to confirm the order is placed successfully and each order number is unique.

5) An order contains one or many cars; A car is on one order.

The structural rule indicates that each order must contain one or many cars, and I make the Order to Car mandatory and plural. I make the Car to Order mandatory and singular because a car can only be in one order at a time.

6) Each order is made from a dealership; each dealership associates with zero to many orders.

The structural rule indicates that each order is associated with a dealership, and of course, a dealership can have many purchases associated with it. I make the Order to Dealership mandatory and singular since an order can only happen at a dealership. And I make the Dealership to Order optional and plural because the database already contains data about dealership and the dealership may be associated with zero or many orders.

7) Each car has a make; Each make can manufacture one to many cars.

The structural rule indicates that each car must have a make which is the brand of the car, and each make can manufacture many cars. Therefore, I make the Car to Make mandatory and singular, while the Make to Car mandatory and plural relationship.

Loan Application Use Case

1. After a vehicle purchase order is placed, a salesperson is appointed for the order.
2. Customer is told that she/he needs to fill out the loan application in order to buy the car, and decides the amount of down payment she/he is going to pay
3. Customer enters her/his information
4. Salesperson processes the loan application and sends it to the bank, then the loan has been approved for purchase, and confirmation is generated.

From this use case, I see three significant data points as entities, Loan, Car, and Salesperson. We know that almost all people need to apply for a loan to buy a car, and a salesperson helps to gather all information to check a customer's credit score, and then send it to multiple banks for approval. I now have enough information to create some structural database rules.

8) An auto loan only finances one car; A car is financed by an auto loan.

The structural rule indicates that when buying a car, an auto loan is needed if the customer is not able to pay the total in a one-time payment. And most of the time, an auto loan is only available to finance one car. Therefore, the constraints are clear, I make the Loan to Car and Car to Loan mandatory and singular for the two entities.

9) A salesperson processes one or many auto loans; an auto loan is processed by a salesperson.

The structural rule indicates that a salesperson processes one or many auto loans because the salesperson sells one or more vehicles, and I make Salesperson to Loan mandatory and plural. This must be mandatory otherwise there would be no database stored for the auto loan. On the other hand, an auto loan is usually only processed by a salesperson, I make the Loan to Salesperson mandatory and singular.

Auto Insurance Purchase Use case

1. The user just purchased a vehicle, she now decides to purchase auto insurance for her car
2. The user opens the app and starts to fill in her new car information and select her insurance coverages.
3. The user carefully compares different insurance companies' quote in the app, and decide to pick one insurance company.
4. The user then fills in her payment information and billing address and the order is placed.
5. A confirmation number and an insurance policy number are generated to confirm the order is placed successfully.

From this use case, I see three significant data points as entities, Car, Insurance, and order. First of all, the customer has a car and wants to buy auto insurance for the car. As we know that we must purchase auto insurance for a vehicle that is in use. And like the last use case, when an order is placed, it will be linked to the account. Since we talked about the structural rule for Order and Account, we will skip here. I now have enough information to create some structural database rules.

10) A car must be insured by an insurance; Each insurance insures one to many cars.

The structural rule indicates that a car must be insured by auto insurance, and I make Car to Insurance mandatory and singular because if I don't make it mandatory, the insurance-linked policy number will not be generated and stored in my database. On the other, if the customer already has an insurance policy, he/she can add multiple cars to the policy. So I make the Insurance to Car mandatory and plural.

11) Each insurance is provided by an insurance company; Each insurance company can provide one to many insurances.

The structural rule indicates a one-to-many and mandatory relationship between insurance and the insurance company. Generally speaking, each insurance is provided by an insurance company, so I make Insurance to Insurance company mandatory and singular. On the other hand, an insurance company mostly provides more than one insurance which makes it a plural relationship.

Conceptual Entity-Relationship Diagram

Here are some structural database rules that I created:

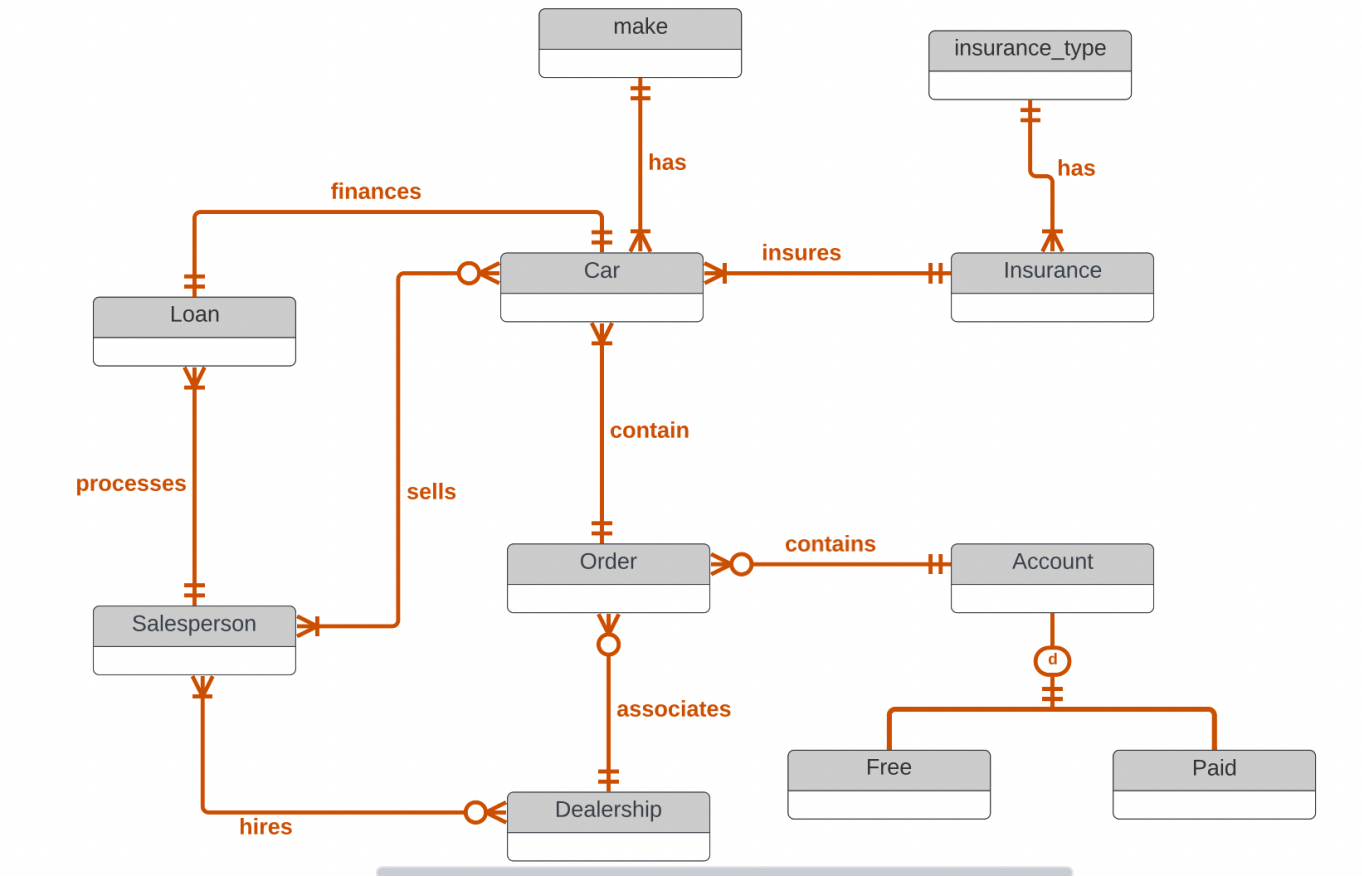
1) An account is a free account or paid account.
2) Each automotive dealership hires many salespeople; Each salesperson is hired by one or many automotive dealerships.
3) A salesperson can sell many cars; Each car is sold by one to many salespersons.
4) An account may have many orders; Each order is created by an account.
5) An order contains one or many cars; A car is on one order.
6) Each order is made from a dealership; each dealership associates with zero to many orders.
7) Each car has a make; Each make can manufacture one to many cars.
8) An auto loan only finances one car; A car is financed by an auto loan.
9) A salesperson processes one or many auto loans; an auto loan is processed by a salesperson.
10) A car must be insured by insurance; Each insurance insures one to many cars.
11) Each insurance is provided by an insurance company; Each insurance company can provide one to many insurances.

Numbers 7 and 11 are newly created domain tables. I create the insurance domain table to make each insurance order easier because they don't need to enter the insurance company's information every time when there is a purchase, they can simply key in the insurance company id to look it up. Same as the Make table which helps to simplify the process of entering cars' information in the database.

Notice that the Order entity is associated with three entities, including account, car, and dealership. The Car entity is associated with four entities, including salesperson, order, loan, and insurance. The Salesperson is associated with three entities, including dealership, car, and loan. The participation and plurality constraints reflect what is in each of the structural database rules by Crow's Foot below.

For my conceptual ERD, I am only able to identify one specialization-generalization rule for the Account entity. The supertype is the Account, and the subtypes are Free or Paid accounts. The relationship between them is totally complete and disjoint as shown using Crow's Foot.

I also added domain tables for Car-make and Insurance-insurance_type to my conceptual ERD. Below is my updated Conceptual ERD:



Full DBMS Physical ERD

Update your normalized DBMS physical ERD to include the new history table.

Table	Attribute	Datatype	Reason
All	ID or Number	DECIMAL(12)	For all ID numbers, I choose a length of 12 digits to keep all consistent and those IDs all have a primary key constraint to make them unique and not nullable.
Account	AccountName	VARCHAR(32)	Every account has a username associated with it which will be used to login to AutoBuy. I allow account names to be up to 32 characters.
Account	FirstName	VARCHAR(32)	This is the first name of the account holder, and I think most names are within 32 characters.
Account	LastName	VARCHAR(32)	This is the last name of the account holder, and I think most names are within 32 characters.

Account	AccountEmail	VARCHAR(64)	This is the email address of the account holder, Up 64 characters should be a safe upperbound.
Account	Zipcode	DECIMAL(5)	Zipcode are always 5 digits in the U.S..
Account	EncryptedPassword	VARCHAR(32)	This is the password for the account and it will consist of the string length of 32, including letters, numbers, and special characters.
Paid_Account	RenewalDate	DATE	This is the paid account and the account is purchased on the annual base. This serves to remain customer to renew their account.
Paid_Account	Balance	DECIMAL(8,2)	This is the account balance used to purchase premium membership or can use to purchase a car. I think up to $999999.99 should be good upperbound.
Retail	RetailName	VARCHAR(128)	This is the name of the retail store, 128 characters should be the safe upperbound.
Retail	RetailLocation	VARCHAR(256)	This is the address for the retail store, up to 256 characters should able to describe the address.
Retail	Description	VARCHAR(2560)	This is the description provided by the retail store and may use to describe year of their services, service types, and other information.
Retail	RetailPhone	DECIMAL(10)	Notice that I created retail phone and salesperson's phone separately which I think will be easier for customer to choose appropriate contact number.
Salesperson	SalespersonPhone	DECIMAL(10)	This is the phone number of salesperson and phone numbers are always 10 digits in the U.S..
Loan	FinanceCompany	VARCHAR(64)	This is the name of the finance company, up to 64 characters should cover the whole name.

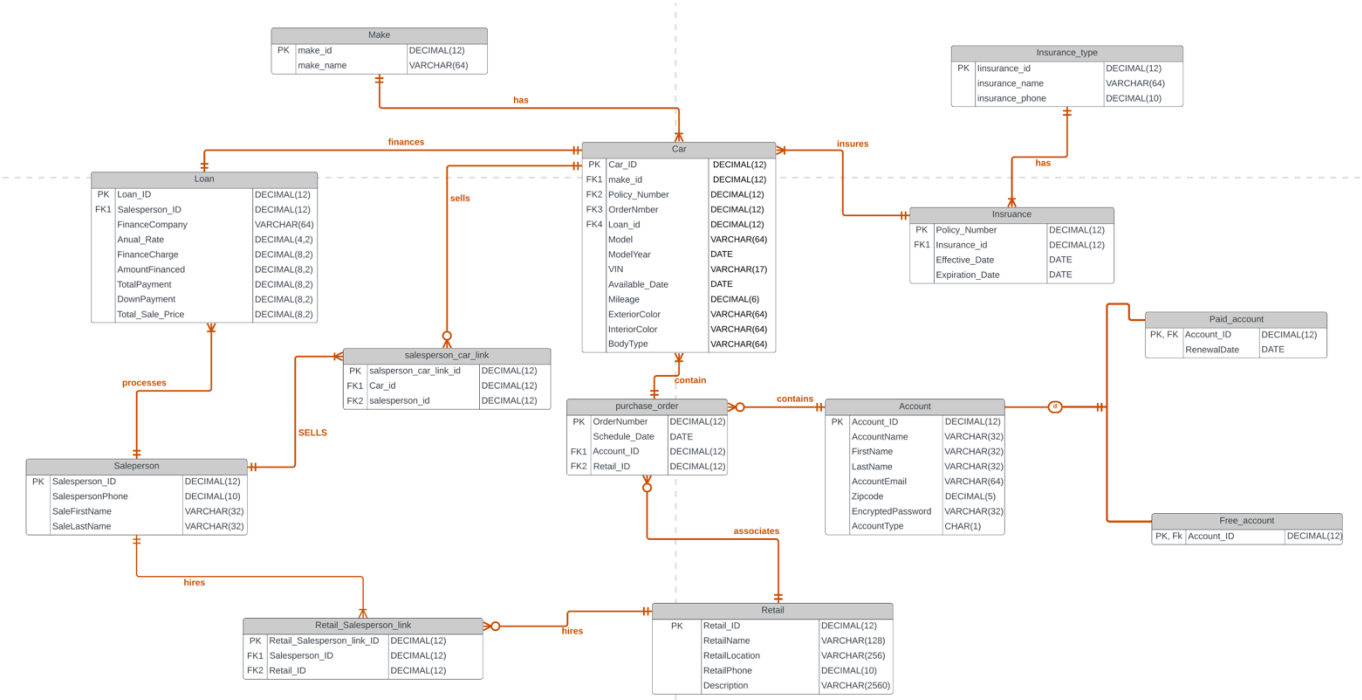
Loan	Annual_Rate	DECIMAL(4,2)	This is the annual percentage rate of the loan, and I allow 4 digits length with 2 decimals for the rate.
Loan	TotalPayment	DECIMAL(8,2)	This is going to be the sum of FinanceCharge (Interest) + Amount Financed. For the price, I think most car won't be over $999999,99, so 8digits with 2 decimals should be safe upperbound.
Loan	Total_Sale_Price	DECIMAL(8,2)	This is going to be the sum of FinanceCharge + AmountFinanced + DownPayment.
Car	VIN	VARCHAR(17)	Each vehicle identity number is unique and it always consist a string length of 17.
Car	Mileage	DECIMAL(6)	Most car wont have a mileage higher than 999999 miles, so this should be a safe upperbound.
Car	BodyType	VARCHAR(64)	This is the body type of a car, 64 characters should give a good description about the body type.
Make	Make_name	VARCHAR(64)	This is the manufacturer of the car, I allow 64 characters for the make.
Insurance_Company	Insurance_name	VARCHAR(64)	This is the name of the insurance company, and I allow 64 characters for the name of insurance company.
Insurance_Company	Insurance_phone	DECIMAL(10)	This is the phone number of insurance company and phone numbers are always 10 digits in the U.S..
Order	Schedule_Date	DATE	This is the date that customer can pick up their ordered car and sign the contract with salesperson.

After carefully checking all my tables and looking for possible normalizations in my physical ERD, I think I had done that in previous iterations along with the use case creation process. For example, I separated out the salesperson from the retail table and avoided creating redundant salesperson information on the retail table.

In addition, I notice that VIN number on the Car table covers many information about a car, such as a vehicle manufacturer, vehicle type, vehicle brand, model year, and so on. But I decide not to remove the car's make, model, and body type because that information is important for a buyer. With an abbreviated character or code hiding in the VIN, the customer won't able to get it straight. I think I had tried to normalize the car table to a lesser normal form.

Last but not the least, I think creating the domain tables for car make and the insurance company is also a way to normalize my physical ERD.

I feel like I have captured all of the necessary fundamental attributes for AutoBuy in the table above. I see that I could add more details with car information, but I think those are not core entities, I will focus on what I have for now.



Stored Procedure Execution and Explanations

Update, if necessary, your screenshots and explanations of defining and executing your stored procedures to transactionally add data to your database.

Account Signup/Installation Use Case

1. The customer visits AutoBuy's website or app store and installs the application
2. The application asks them to create either a free or paid account when its first run
3. The customer selects the type of account and enters their information and the account is created in the database, some account information includes name, email, and zip code.

For this use case, I can implement two transactions that creates a free account or paid account. I name the stored procedure 'AddFreeAccount' and give it parameters that correspond to the account and Free_account tables. Since this procedure is always for a free account, I do not use a parameter for AccountType, but hardcode the character 'F'. I name the second stored procedure 'AddPaidAccount' and

give it parameters that correspond to the Account and Paid_account tables. Since the procedure is always for a paid account, I do not use a parameter for AccountType, but hardcode the character ‘P’.

Here are screenshots of my updated stored procedure definition.

```
--FIRST USE CASE
--Free_account and account
CREATE OR REPLACE PROCEDURE addFreeAccount(Account_ID IN DECIMAL, AccountName IN VARCHAR, FirstName IN VARCHAR,
                                           LastName IN VARCHAR, AccountEmail IN VARCHAR,
                                           Zipcode IN DECIMAL,EncryptedPassword IN VARCHAR,AccountType IN CHAR)
AS
$proc$
BEGIN
INSERT INTO account
VALUES(account_id, accountname, firstname, lastname, accountemail, zipcode, encryptedpassword, accounttype);

INSERT INTO free_account
VALUES (account_id);
END;
$proc$ LANGUAGE plpgsql;

--Paid_Account and account
CREATE OR REPLACE PROCEDURE addPaidAccount(Account_ID IN DECIMAL, RenewalDate IN DATE, AccountName IN VARCHAR, FirstName IN VARCHAR,
                                           LastName IN VARCHAR, AccountEmail IN VARCHAR, Zipcode IN DECIMAL,
                                           EncryptedPassword IN VARCHAR,AccountType IN CHAR, Balance in DECIMAL)
AS
$proc$
BEGIN
    INSERT INTO account
    VALUES(account_id, accountname, firstname, lastname, accountemail, zipcode, encryptedpassword, accounttype);

    INSERT INTO paid_account
    VALUES(RenewalDate, account_id, Balance);
END;
$proc$ LANGUAGE plpgsql;
```

Below is my stored procedure execution.

```
----AddFreeAccount
START TRANSACTION;
DO
 $$BEGIN
    CALL AddFreeAccount(nextval('account_seq'), 'edwardc', 'Edward', 'Cullen', 'edwarc@gmail.com', 95828, 'edddwww34', 'F');
    CALL AddFreeAccount(nextval('account_seq'), 'bellas', 'Bella', 'Swan', 'bella@gmail.com', 02134, 'bellla21', 'F');
    CALL AddFreeAccount(nextval('account_seq'), 'jacobb', 'Jacob', 'Black', 'jblack@gmail.com', '11232', 'jablack009', 'F');
    CALL AddFreeAccount(nextval('account_seq'), 'alicec', 'Alice', 'Greene', 'aliceg@gmail.com', '95233', 'allliceggg', 'F');
    CALL AddFreeAccount(nextval('account_seq'), 'rosalieha', 'Rosalie', 'Hale', 'rhale@gmail.com', '59383', 'halerossa22', 'F');
    CALL AddFreeAccount(nextval('account_seq'), 'jasperh', 'Jasper', 'Hale', 'jasperh@gmail.com', 10012, 'jasper22111','F');
    CALL AddFreeAccount(nextval('account_seq'), 'peterfa', 'peter', 'Facinelli', 'perterf@gmail.com', 63842, 'perperper11', 'F');
    CALL AddFreeAccount(nextval('account_seq'), 'victoriall', 'Victoria', 'Lefevre', 'lefevrev@gmail.com', 38272, 'victorialef', 'F')
    CALL AddFreeAccount(nextval('account_seq'), 'billb', 'Billy', 'Burke', 'billbb@gmail.com', 87302, 'billbill919', 'F');
    CALL AddFreeAccount(nextval('account_seq'), 'emmettl', 'Emmett', 'Lutz', 'emmettl@gmail.com', 38302, 'emmiittt000', 'F');
 END$$;
COMMIT TRANSACTION;

--AddPaidAccount
START TRANSACTION;
DO
$$BEGIN
    CALL addPaidAccount(nextval('account_seq'), 'elizare', 'Elizabeth', 'Reaser', 'elreaser@gmail.com', 94753, 'elizabbbbeth', 'P', '
    CALL AddPaidAccount(nextval('account_seq'), 'jessica', 'Jessica', 'Stanley', 'Jessicas@gmail.com', 78390, 'jeesss122', 'P', '9/9/
    CALL AddPaidAccount(nextval('account_seq'), 'camgi', 'Cam', 'Gigandet', 'camgig@gmail.com', 93882, 'cammmgggi2', 'P', '11/23/2023
    CALL AddPaidAccount(nextval('account_seq'), 'mikenew', 'Mike', 'Newton', 'miken@gmail.com', 28293, 'mikemike22', 'P', '7/22/2023'
    CALL AddPaidAccount(nextval('account_seq'), 'angelaw', 'Angela', 'Weber', 'angelaw@gmail.com', 48302, 'angelawwww', 'P', '8/20/20
    CALL AddPaidAccount(nextval('account_seq'), 'samul', 'Sam', 'Uley', 'samuley@gmail.com', 92873, 'sammejbs', 'P', '12/29/2022');
    CALL AddPaidAccount(nextval('account_seq'), 'embryc', 'Embry', 'Call', 'calle@gmail.com', 10022, 'callmeemm212', 'P', '1/21/2024'
    CALL AddPaidAccount(nextval('account_seq'), 'tylerc', 'Tyler', 'Crowley', 'tylerc@gmail.com', 83721, 'tyllerller', 'P', '12/19/20
    CALL AddPaidAccount(nextval('account_seq'), 'ericy', 'Eric', 'Yorkie', 'ericy@gmail.com', 02122, 'ericycccyyy', 'P', '2/12/2024')
    CALL AddPaidAccount(nextval('account_seq'), 'mattbu', 'Matt', 'Bushell', 'mattbu@gmail.com', 10024, 'mattbububuu', 'P', '9/25/20
END$$;
COMMIT TRANSACTION;
```

Automotive retailers/Dealerships signup use case

2. Car dealers or retail stores decide to cooperate with AutoBuy and they start to visit the website or download the app
3. The application asks them to register their certified information
4. Retailer fills in their information as a company and this information adds to the database
5. After successfully registering basic information, the retailers then tell each individual salesperson to add their own information

For this use case, I can implement two transactions that create a retail table, and a salesperson table. I name the stored procedure 'AddRetail' and give it parameters that correspond to the Retail table. I name the second stored procedure 'AddSalesperson' and give it parameters that correspond to the Salesperson table. And then, I populate the Retail_Salesperson_Link using 'INSERT INTO … VALUES…'.

Here are screenshots of my stored procedure definition.

```
--RETAIL
CREATE OR REPLACE PROCEDURE addretail(retail_id IN decimal, RetailName IN VARCHAR,
                                      RetailLocation IN VARCHAR, RetailPhone IN DECIMAL, Description IN VARCHAR)
AS
$proc$
BEGIN
INSERT INTO retail
VALUES(retail_id, RetailName, RetailLocation, RetailPhone, Description);
INSERT INTO Retail_Salesperson_link
VALUES (retail_id);
END;
$proc$ LANGUAGE plpgsql;

--SALESPERSON
CREATE OR REPLACE PROCEDURE addSalesperson(salesperson_id IN DECIMAL,SalespersonPhone IN DECIMAL,
                                           SalespersonFirstName IN VARCHAR, SalesperonLastName IN VARCHAR)
AS
$proc$
BEGIN
INSERT INTO salesperson
VALUES(salesperson_id, SalespersonPhone, SalespersonFirstName, SalesperonLastName);
INSERT INTO Retail_Salesperson_link
VALUES (salesperson_id);
END;
$proc$ LANGUAGE plpgsql;

```

Below is my stored procedure execution.

```
--addretail
START TRANSACTION;
DO
$$BEGIN
    CALL AddRetail(nextval('retail_seq'), 'Select Auto Dealers Corp', '1543 Bushwick Ave, Brooklyn, NY 11207' , 7186847999, 'We are t
    CALL AddRetail(nextval('retail_seq'), 'Elk Grove Kia', '8480 Laguna Grove Dr, Elk Grove, CA 95757', 9167531000, 'Enjoy The Highes
    CALL AddRetail(nextval('retail_seq'), 'Reliable Chevrolet', '9901 Coors Blvd NW, Albuquerque, NM 87114', 5052162788, 'Your first
    CALL AddRetail(nextval('retail_seq'), 'Preferred Auto Sales Pre-owned Superstore', '655 Pennsylvania Ave, Elizabeth, NJ 07201', 9
    CALL AddRetail(nextval('retail_seq'), 'Hudson Chrysler Jeep Dodge RAM', '625 NJ-440, Jersey City, NJ 07304', 5512275676, 'Amazing
    CALL AddRetail(nextval('retail_seq'), 'Key Motors of South Burlington', '1675 Shelburne Rd, South Burlington, VT 05403', 802815820
    CALL AddRetail(nextval('retail_seq'), 'AutoFair Ford in Manchester', '1475 S Willow St, Manchester, NH 03103', 8335019327, 'We kn
    CALL AddRetail(nextval('retail_seq'), 'Virginia Auto Trader Company', '2510 Lee Hwy, Arlington, VA 22201', 7035678383, 'used car
    CALL AddRetail(nextval('retail_seq'),'Blasius of Attleboro', '800 Washington St, Attleboro, MA 02703', 5086390099, 'We are commit
    CALL AddRetail(nextval('retail_seq'), 'Antwerpen Certified Pre-Owned Center', '5717 Baltimore National Pike, Catonsville, MD 21228
END$$;
COMMIT TRANSACTION;

--addSALESPERSON
START TRANSACTION;
DO
$$BEGIN
    CALL AddSalesperson(nextval('salesperson_seq'), 7186847977, 'Evan', 'Buckley');
    CALL AddSalesperson(nextval('salesperson_seq'), 9167531111, 'Athena', 'Grant');
    CALL AddSalesperson(nextval('salesperson_seq'), 5052168888, 'Howie', 'Han');
    CALL AddSalesperson(nextval('salesperson_seq'), 9082145678, 'Henrietta', 'Wilson');
    CALL AddSalesperson(nextval('salesperson_seq'), 5512276392, 'Abby', 'Clark');
    CALL AddSalesperson(nextval('salesperson_seq'), 8028158231, 'Bobby', 'Nash');
    CALL AddSalesperson(nextval('salesperson_seq'), 7035679182, 'Chris', 'Diaz');
    CALL AddSalesperson(nextval('salesperson_seq'), 5086398888, 'Maddie', 'Kendall');
    CALL AddSalesperson(nextval('salesperson_seq'), 3476569928, 'Jeffrey', 'Hudson');
    CALL AddSalesperson(nextval('salesperson_seq'), 9172022022, 'Lena', 'Bosko');
End$$;
COMMIT TRANSACTION;

--POPULATE Retail_Salesperson_link
Insert into Retail_Salesperson_link VALUES (nextval('retail_sale_link_seq'), (SELECT retail_id FROM retail WHERE RetailName='Select Auto Dealers Corp'
Insert into Retail_Salesperson_link VALUES(nextval('retail_sale_link_seq'), (SELECT retail_id FROM retail WHERE RetailName='Elk Grove Kia'), (SELECT S
Insert into Retail_Salesperson_link VALUES(nextval('retail_sale_link_seq'), (SELECT retail_id FROM retail WHERE RetailName='Reliable Chevrolet'), (SEL
Insert into Retail_Salesperson_link VALUES(nextval('retail_sale_link_seq'), (SELECT retail_id FROM retail WHERE RetailName='Preferred Auto Sales Pre-o
Insert into Retail_Salesperson_link VALUES(nextval('retail_sale_link_seq'), (SELECT retail_id FROM retail WHERE RetailName='Hudson Chrysler Jeep Dodge
Insert into Retail_Salesperson_link VALUES(nextval('retail_sale_link_seq'), (SELECT retail_id FROM retail WHERE RetailName='Key Motors of South Burlin
Insert into Retail_Salesperson_link VALUES(nextval('retail_sale_link_seq'), (SELECT retail_id FROM retail WHERE RetailName='Virginia Auto Trader Compa
Insert into Retail_Salesperson_link VALUES(nextval('retail_sale_link_seq'), (SELECT retail_id FROM retail WHERE RetailName='Preferred Auto Sales Pre-o
Insert into Retail_Salesperson_link VALUES(nextval('retail_sale_link_seq'), (SELECT retail_id FROM retail WHERE RetailName='Blasius of Attleboro'), (S
Insert into Retail_Salesperson_link VALUES(nextval('retail_sale_link_seq'), (SELECT retail_id FROM retail WHERE RetailName='Antwerpen Certified Pre-Ow
```

People/Salesperson list their own vehicle use case

1. The user decides to sell his cars on AutoBuy
2. The user logins into the app and start a sale page
3. The user fills in his vehicle's detailed information and this adds the data to the database, information includes price, make, model, mileage, available date, and so on.

For this use case, I can implement two transactions that create a Make table and a car table. I name the stored procedure 'AddMake' and give it parameters that correspond to the Make table. I name the second stored procedure 'AddCar' and give it parameters that correspond to the Car table.

Here are screenshots of my stored procedure definition.

```
--Third USE CASE
--MAKE
CREATE OR REPLACE PROCEDURE addMake(make_id IN DECIMAL, make_name IN VARCHAR)
AS
$proc$
BEGIN
INSERT INTO make
VALUES(make_id, make_name);
END;
$proc$ LANGUAGE plpgsql;
```

```
--CAR
CREATE OR REPLACE PROCEDURE AddCar(Car_id IN DECIMAL, make_id IN DECIMAL, Policy_Number IN DECIMAL,
                                   OrderNumber IN DECIMAL, Loan_ID IN DECIMAL, Model IN VARCHAR, ModelYear IN DECIMAL,
                                   VIN IN VARCHAR, Available_date IN DATE, Mileage IN DECIMAL, ExteriorColor IN VARCHAR,
                                   InteriorColor IN VARCHAR, BodyType IN VARCHAR)
AS
$proc$
BEGIN
INSERT INTO car
VALUES(Car_id, make_id, Policy_Number, OrderNumber,Loan_ID, Model,
       ModelYear, VIN, Available_date, Mileage, ExteriorColor, InteriorColor, BodyType);
END;
$proc$ LANGUAGE plpgsql;
```

Below is my stored procedure execution.

```
START TRANSACTION;
DO
$$BEGIN
    CALL AddMake(nextval('make_seq'), 'Ford');
    CALL AddMake(nextval('make_seq'), 'Toyota');
    CALL AddMake(nextval('make_seq'), 'Honda');
    CALL AddMake(nextval('make_seq'), 'Hyundai');
    CALL AddMake(nextval('make_seq'), 'Nissa');
    CALL AddMake(nextval('make_seq'), 'Mercedes-Benz');
    CALL AddMake(nextval('make_seq'), 'Bayerische Motoren Werke AG');
    CALL AddMake(nextval('make_seq'), 'Lexus');
    CALL AddMake(nextval('make_seq'), 'Audi');
    CALL AddMake(nextval('make_seq'), 'Porsche');
End$$;
COMMIT TRANSACTION;
```

```
--AddCar
START TRANSACTION;
DO
$$BEGIN
    CALL AddCar(nextval('car_seq'), 1, 3, 1, 4, 'Escape', 2022,'1FMCU9F61NUB28311', '10-03-2022', 140, 'Oxford white', 'Dark Earth Gray', 'AWD SUV');
    CALL AddCar(nextval('car_seq'), 2, 5, 2, 3, '4 Runner', 2023, 'JTEMU5JR8P6099430', '09-08-2022', 40, 'Blue Metallic', 'Sand Beige', '4WD SUV');
    CALL AddCar(nextval('car_seq'), 3, 4, 3, 2, 'CR-V', 2022, '5J6RW2H51NL003751', '10-20-2022', 39, 'Pearl White', 'Ivory', 'AWD SUV');
    CALL AddCar(nextval('car_seq'), 4, 1, 4, 1, 'Kona', 2022, 'KM8K3CAB5PU939421', '9-03-2022', 10, 'Black', 'Luna white', 'AWD Crossover');
    CALL AddCar(nextval('car_seq'), 5, 2, 5, 5, 'Maxima', 2019, '1N4AA6AV5KC379258', '08-13-2022', 24076, 'Super Black', 'Charcoal', 'FWD Sedan');
    CALL AddCar(nextval('car_seq'), 6, 7, 6, 8, 'GLB 250', 2023, 'W1N4M4HB5NW258059', '11-22-2022', 69, 'Night Black', 'Black', '4Matic SUV');
    CALL AddCar(nextval('car_seq'), 7, 8, 7, 7, '540i', 2019, 'WBAJE5C51KWW11386', '10-13-2022', 74314, 'Alphine white', 'Black', 'RWD Sedan');
    CALL AddCar(nextval('car_seq'), 8, 9, 8, 6, 'GX 460', 2018, 'JTJJM7FX7J5208531', '03-17-2022', 20832, 'Starfire Pearl', 'Black', '4WD SUV');
    CALL AddCar(nextval('car_seq'), 9, 10, 9, 10, 'F5EBAY', 2022, 'WAU2AGF5XNN007255', '12-12-2022', 10, 'Glacier White Metallic', 'Rock Gray', 'AWD 2
    CALL AddCar(nextval('car_seq'), 10, 6, 10, 9, 'Macan', 2018, 'WP1AB2A50JLB30475', '10-03-2022', 24683, 'Volcano Grey Metallic', 'Black', 'AWD SUV'
END$$;
COMMIT TRANSACTION;
--QUERIES
--Replace this with your queries.
```

Auto Insurance Purchase Use case

1. The user just purchased a vehicle, she now decides to purchase auto insurance for her car
2. The user opens the app and starts to fill in her new car information and select her insurance coverages.
3. The user carefully compares different insurance companies' quote in the app, and decide to pick one insurance company.
4. The user then fills in her payment information and billing address and the order is placed.
5. A confirmation number and an insurance policy number are generated to confirm the order is placed successfully.

For this use case, I can implement one transaction that creates both an Insurance_type table and an Insurance table. I name the stored procedure 'AddInsType' and give it parameters that correspond to the insurance_type table and Insurance table.

Below is a screenshot of my stored procedure definition.

```
--INSRUANCE_TYPE and Insurance
CREATE OR REPLACE PROCEDURE AddInsType(insurance_id IN DECIMAL, insurance_name IN VARCHAR, Insurance_phone IN DECIMAL,
                                       Policy_Number In DECIMAL, Effective_Date IN DATE, Expiration_Date IN DATE)
AS
$proc$
BEGIN
INSERT INTO insurance_type
VALUES(insurance_id , insurance_name , Insurance_phone);

INSERT INTO insurance
VALUES(insurance_id, Policy_Number, Effective_Date, Expiration_Date);
END;
$proc$ LANGUAGE plpgsql;
```

Below is my stored procedure execution.

```
--addInsType
START TRANSACTION;
DO
$$BEGIN
    CALL AddInsType(nextval('ins_type_seq'), 'Geico', 8002077847, nextval('insurance_seq'), '07-01-2021', '06-30-2022');
    CALL AddInsType(nextval('ins_type_seq'), 'StateFarm', 8007828332, nextval('insurance_seq'), '09-01-2021', '08-31-2022');
    CALL AddInsType(nextval('ins_type_seq'), 'AAA Insurance', 8006725246, nextval('insurance_seq'), '04-01-2021', '03-31-2022');
    CALL AddInsType(nextval('ins_type_seq'), 'Allstate', 8009011732, nextval('insurance_seq'), '09-01-2022', '08-31-2023');
    CALL AddInsType(nextval('ins_type_seq'), 'Kemper Auto', 8007821020, nextval('insurance_seq'), '07-01-2021', '06-30-2022');
    CALL AddInsType(nextval('ins_type_seq'), 'Progressive', 8886714405, nextval('insurance_seq'), '05-01-2021', '04-30-2022');
    CALL AddInsType(nextval('ins_type_seq'), 'Nationwide', 8776696877, nextval('insurance_seq'), '03-01-2021', '02-28-2022');
    CALL AddInsType(nextval('ins_type_seq'), 'Liberty Mutual', 8002908711, nextval('insurance_seq'), '10-01-2022', '09-30-2023');
    CALL AddInsType(nextval('ins_type_seq'), 'The Travelers Companies', 8008425075, nextval('insurance_seq'), '03-01-2021', '02-28-2022');
    CALL AddInsType(nextval('ins_type_seq'), 'USAA', 8005318722, nextval('insurance_seq'), '11-01-2021', '10-31-2022');
END$$;
COMMIT TRANSACTION;
```

Loan Application Use Case

1. After a vehicle purchase order is placed, a salesperson is appointed for the order.
2. Customer is told that she/he needs to fill out the loan application in order to buy the car, and decides the amount of down payment she/he is going to pay
3. Customer enters her/his information
4. Salesperson processes the loan application and sends it to the bank, then the loan has been approved for purchase, and confirmation is generated.

For this use case, I can implement one transaction that creates a Loan table. I name the stored procedure 'AddLoan' and give it parameters that correspond to loan table.

Below is a screenshot of my stored procedure definition.

```
--FOURTH USE CASE
--LOAN
CREATE OR REPLACE PROCEDURE AddLoan(loan_id IN DECIMAL, FinanceCompany IN VARCHAR,Annual_rate IN DECIMAL,
                                    FinanceCharge IN DECIMAL, AmountFinanced IN DECIMAL, TotalPayment IN DECIMAL,
                                    DownPayment IN DECIMAL, Total_sale_price IN DECIMAL, salesperson_id IN DECIMAL)
AS
$proc$
BEGIN
INSERT INTO loan
VALUES(loan_id, FinanceCompany, Annual_rate, FinanceCharge, AmountFinanced, TotalPayment, DownPayment, Total_sale_price, salesperson_id
END;
$proc$ LANGUAGE plpgsql;
```

Below is my stored procedure execution.

```
--addloan
START TRANSACTION;
DO
$$BEGIN
    CALL AddLoan(nextval('loan_seq'), 'Carvana', 02.99, 3029.75, 27999.00, 31028.75, 5000.00, 36028.75, 4);
    CALL AddLoan(nextval('loan_seq'), 'LendingTree', 04.79, 5903.75, 30099.00, 34002.75, 6000.00, 42002.75, 1);
    CALL AddLoan(nextval('loan_seq'), 'OneMain Financial', 03.21, 4470.32, 29870.00, 34340.32, 4000.00, 38340.32, 2);
    CALL AddLoan(nextval('loan_seq'), 'MyAutoLoan', 01.99, 4907.33, 26990.00, 31897.33, 7000.00, 38897.33, 5);
    CALL AddLoan(nextval('loan_seq'), 'AutoLoanZoom', 02.59, 4001.50, 34079.00, 38080.50, 2000.00, 40080.50, 4);
    CALL AddLoan(nextval('loan_seq'), 'Toyota Financing', 05.77, 5304.84, 28099.00, 33403.84, 3000.00, 36403.84, 6);
    CALL AddLoan(nextval('loan_seq'), 'Car.Loan.Come', 06.44, 4470.32, 22000.00, 26470.32, 8000.00, 34470.32, 9);
    CALL AddLoan(nextval('loan_seq'), 'Bank of America', 03.99, 5400.77, 31000.00, 36400.77, 7000.00, 43400.77, 8);
    CALL AddLoan(nextval('loan_seq'), 'Chase Bank', 04.51, 3078.20, 23089.00, 26167.2, 6000.00, 32167.20, 3);
    CALL AddLoan(nextval('loan_seq'), 'US Bank', 06.00, 6778.94, 30100.00, 36878.94, 6000.00, 42878.94, 5);
END$$;
COMMIT TRANSACTION;
```

For the Purchase/Order use case, I didn't use any procedure, I simply use 'INSERT INTO … VALUES' to populate my purchase_order table and use 'SELECT … FROM … WHERE' clause for foreign keys.

```
--populate purchase_order
Insert into purchase_order VALUES (nextval('order_seq'), '09-17-2022', (SELECT retail_id FROM retail WHERE RetailName='Select Auto Dealers Corp'),
                                   (SELECT account_id FROM account WHERE AccountName = 'edwardc'));
Insert into purchase_order VALUES(nextval('order_seq'), '07-19-2022', (SELECT retail_id FROM retail WHERE RetailName='Elk Grove Kia'),
                                  (SELECT account_id FROM account WHERE AccountName = 'bellas'));
Insert into purchase_order VALUES(nextval('order_seq'), '02-01-2022', (SELECT retail_id FROM retail WHERE RetailName='Reliable Chevrolet'),
                                  (SELECT account_id FROM account WHERE AccountName = 'jacobb'));
Insert into purchase_order VALUES(nextval('order_seq'), '08-12-2022', (SELECT retail_id FROM retail WHERE RetailName='Preferred Auto Sales Pre-owned S
                                  (SELECT account_id FROM account WHERE AccountName = 'alicec'));
Insert into purchase_order VALUES(nextval('order_seq'), '03-24-2023', (SELECT retail_id FROM retail WHERE RetailName='Hudson Chrysler Jeep Dodge RAM')
                                  (SELECT account_id FROM account WHERE AccountName = 'rosalieha'));
Insert into purchase_order VALUES(nextval('order_seq'), '01-01-2023', (SELECT retail_id FROM retail WHERE RetailName='Key Motors of South Burlington')
                                  (SELECT account_id FROM account WHERE AccountName = 'angelaw'));
Insert into purchase_order VALUES(nextval('order_seq'), '01-23-2022', (SELECT retail_id FROM retail WHERE RetailName='Virginia Auto Trader Company'),
                                  (SELECT account_id FROM account WHERE AccountName = 'samul'));
Insert into purchase_order VALUES(nextval('order_seq'), '09-20-2022', (SELECT retail_id FROM retail WHERE RetailName='Preferred Auto Sales Pre-owned S
                                  (SELECT account_id FROM account WHERE AccountName = 'tylerc'));
Insert into purchase_order VALUES(nextval('order_seq'), '07-18-2022', (SELECT retail_id FROM retail WHERE RetailName='Blasius of Attleboro'),
                                  (SELECT account_id FROM account WHERE AccountName = 'ericy'));
Insert into purchase_order VALUES(nextval('order_seq'), '10-23-2022', (SELECT retail_id FROM retail WHERE RetailName='Antwerpen Certified Pre-Owned Ce
                                  (SELECT account_id FROM account WHERE AccountName = 'mattbu'));

```

I also populate salesperson_car_link using the 'INSERT INTO … VALUES' clause and 'SELECT … FROM … WHERE' clause for to select foreign keys.

```
--populate salesperon_car_link
INSERT INTO salesperon_car_link VALUES(nextval('sale_car_link_seq'), (SELECT car_id FROM Car where VIN = '1FMCU9F61NUB28311'),
                                       (SELECT Salesperson_id FROM Salesperson WHERE SalespersonFirstName= 'Evan'));
INSERT INTO salesperon_car_link VALUES(nextval('sale_car_link_seq'), (SELECT car_id FROM Car where VIN = 'JTEMU5JR8P6099430'),
                                       (SELECT Salesperson_id FROM Salesperson WHERE SalespersonFirstName= 'Athena'));
INSERT INTO salesperon_car_link VALUES(nextval('sale_car_link_seq'), (SELECT car_id FROM Car where VIN = '5J6RW2H51NL003751'),
                                       (SELECT Salesperson_id FROM Salesperson WHERE SalespersonFirstName= 'Howie'));
INSERT INTO salesperon_car_link VALUES(nextval('sale_car_link_seq'), (SELECT car_id FROM Car where VIN = 'KM8K3CAB5PU939421'),
                                       (SELECT Salesperson_id FROM Salesperson WHERE SalespersonFirstName= 'Henrietta'));
INSERT INTO salesperon_car_link VALUES(nextval('sale_car_link_seq'), (SELECT car_id FROM Car where VIN = '1N4AA6AV5KC379258'),
                                       (SELECT Salesperson_id FROM Salesperson WHERE SalespersonFirstName= 'Chris'));
INSERT INTO salesperon_car_link VALUES(nextval('sale_car_link_seq'), (SELECT car_id FROM Car where VIN = 'W1N4M4HB5NW258059'),
                                       (SELECT Salesperson_id FROM Salesperson WHERE SalespersonFirstName= 'Bobby'));
INSERT INTO salesperon_car_link VALUES(nextval('sale_car_link_seq'), (SELECT car_id FROM Car where VIN = 'WBAJE5C51KWW11386'),
                                       (SELECT Salesperson_id FROM Salesperson WHERE SalespersonFirstName= 'Evan'));
INSERT INTO salesperon_car_link VALUES(nextval('sale_car_link_seq'), (SELECT car_id FROM Car where VIN = 'JTJJM7FX7J5208531'),
                                       (SELECT Salesperson_id FROM Salesperson WHERE SalespersonFirstName= 'Athena'));
INSERT INTO salesperon_car_link VALUES(nextval('sale_car_link_seq'), (SELECT car_id FROM Car where VIN = 'WAU2AGF5XNN007255'),
                                       (SELECT Salesperson_id FROM Salesperson WHERE SalespersonFirstName= 'Howie'));
INSERT INTO salesperon_car_link VALUES(nextval('sale_car_link_seq'), (SELECT car_id FROM Car where VIN = 'WP1AB2A50JLB30475'),
                                       (SELECT Salesperson_id FROM Salesperson WHERE SalespersonFirstName= 'Athena'));
```

Question Identification and Explanations

Update, if necessary, your questions useful to the organization and explanations as to why they are useful.

Query 1: Which salesperson sold the most cars on our database?

I will explain why this question is useful here. Let's say AutoBuy is going to host an award night and invite the salesperson with the highest number of cars sold on our record. This query will be useful to promote our business and encourage Salesperson and Retails to use our service more. On the other hand, with more data in our database, we can also use this query for seasonal, semi-annual, or annual awards.

Query 2: which customer has an expired or closed expired paid account? Or Which customer has a balance below $50?

Recall the specialized-generalization relationship between the Account and paid_account tables. Since our APP AutoBuy offers premium service to those paid accounts, we make good money from selling the memberships. We want to maintain our customer relationships and keep our customers in our APP rather than go to other places. On the other hand, customers can also use account balance to seek service in our app, we also want to remind customers with balances below $50 to add funds to the account. Knowing which account has recently expired or closed upon the renewal date and with a low balance will help us to remind our customers to renew paid accounts on time or use the premium service that they paid for before expiring.

QUERY 3: What is the price of SUV from low to high in the most recent two years?

I will explain why this question is useful and frequently used in the query here. First, for people who don't have a high budget, but still hope to get the most recent new car that satisfied their concern about safety, new cars are generally considered safer than used cars. In addition, new cars of the next model year usually come out in September of each year, so both 2022 and 2023 will consider a new model year. In addition, this query specified that the body type must be SUV which some people favor for more space. This query will help me to figure out the new SUV car that is within their budget.

Query Executions and Explanations

Update, if necessary, your queries answering the questions, along with screenshots and explanations.

Query 1: Which salesperson is the sales champion on our database?

For this query, I want to find the sales champion in my database and give an award not only to the salesperson but also to the retail store that hires this salesperson. I used 'JOIN … ON' to join four tables including Salesperson, Retail, Salesperson_Car_Lin, and Retail_Salesperson_Link. I found out that Athena wins the award. The 'number_cars_sold' also gives us the total number of cars sold by Athena ordered in descending order.

```sql
--Query 1: Which salesperson sold the most cars on our database?
--I used 'JOIN ... ON' to join 4 tables with associative relationships
--I order the resutl using DESC
SELECT salespersonFirstName, retailName, count(salesperon_car_link.Salesperson_id) as number_cars_sold
From salesperon_car_link
JOIN salesperson ON salesperson.salesperson_id = salesperon_car_link.salesperson_id
Join retail_salesperson_link on retail_salesperson_link.salesperson_id= Salesperson.salesperson_id
JOIN retail ON retail.retail_id = retail_salesperson_link.retail_id
Group by salespersonFirstName, retailName
Order by number_cars_sold DESC;
```

Data output | Messages | Notifications

	salespersonfirstname character varying (32)	retailname character varying (128)	number_cars_sold bigint
1	Athena	Elk Grove Kia	3
2	Evan	Select Auto Dealers Co...	2
3	Howie	Reliable Chevrolet	2
4	Bobby	Key Motors of South B...	1
5	Henrietta	Virginia Auto Trader Co...	1
6	Henrietta	Preferred Auto Sales P...	1

Query 2: which customer has an expired or closed expired paid account? Or Which customer has a balance below $50?

This query is used to filter two conditions, one is with an expired or closed-to-expired account, and the other one is with an account balance below $50. I used the 'OR' clause to ensure at least one of those two conditions is tenable. I decide to query customers' account renewal date, balance, account name, and email for this query. Emails are important because we can send reminder emails to our customers if they change their minds and decide to renew their recently expired account or are closed to expire.

```sql
--Query 2: which customer has an expired or closed expired paid account? Which customer has a balance below $50?
--I define the expiration window for three months from October to December.
--using the interval '1 day' in those three month to find account that meet my query.
--also include customers with balance below $50

SELECT RenewalDate, Balance, AccountName, accountemail
from paid_account
Join account ON account.account_id = paid_account.account_id
Where RenewalDate>= '10-01-2022'::date
    AND RenewalDate < ('12-30-2022'::date + '1 day'::interval)
    OR Balance <= 50
GROUP BY RenewalDate, balance, AccountName, accountemail
Order by RenewalDate;
```

Data output | Messages | Notifications

	renewaldate date	balance numeric (5,2)	accountname character varying (32)	accountemail character varying (64)
1	2022-10-02	30.55	elizare	elreaser@gmail.com
2	2022-10-29	244.29	samul	samuley@gmail.com
3	2022-11-09	76.22	jessica	Jessicas@gmail.com
4	2022-11-19	33.62	tylerc	tylerc@gmail.com
5	2023-01-21	19.82	embryc	calle@gmail.com
6	2023-03-09	43.78	camgi	camgig@gmail.com
7	2023-10-25	24.29	mattbu	mattbu@gmail.com

Total rows: 7 of 7 | Query complete 00:00:00.490

QUERY 3: What is the price of an SUV from low to high in the most recent two years?

I create a view called new_suv_car for this query. I used the 'Join … on' clause to join the make and car table. I also use 'Having' along with 'Like' clauses to specify that the car body type must be SUV. Finally, this view is ordered by the price of cars from low to high.

```
523 --QUERY 3: What is the price of an SUV from low to high in the most recent two years?
524 --the most recent two years are defined to be 2022 and 2023, using OR to show both required years.
525 --the car.bodyType must contain the word 'SUV'
526 --Used the ORDER BY ... ASC to show the price from low to high.
527 CREATE OR REPLACE VIEW new_suv_car AS
528 SELECT make, model, modelyear, bodyType, listing_price as price_low_to_high
529 From make
530 JOIN Car on Car.make_id = make.make_id
531 WHERE modelyear = 2023 OR modelyear=2022
532 Group by make, model, modelyear, bodytype, price_low_to_high
533 HAVING car.bodytype LIKE '%SUV'
534 ORDER BY price_low_to_high ASC;
535
```

Data output | Messages | Notifications

	make make	model character varying (64)	modelyear numeric (4)	bodytype character varying (64)	price_low_to_high numeric (8,2)
1	(1,Ford)	Escape	2022	AWD SUV	31065.00
2	(3,Honda)	CR-V	2022	AWD SUV	32110.00
3	(6,Merced…	GLB 250	2023	4Matic SUV	42150.00
4	(2,Toyota)	4 Runner	2023	4WD SUV	45048.00

Index Identification and Creations

Update, if necessary, the indexes identifications useful to your database, explanations as to why they help, and screenshots of their creations.
Following Primary keys are already indexed and they are unique:
Account.Account_id
Retail.retail_id
Salesperson.salesperson_id
Retail_salesperson_link. retail_salesperson_link_id
Loan.loan_id
Make.make_id
Purchase_order.OrderNumber
Insurance_type.insurance_id
Insurance.Policy_Number
Car.car_id
salesperon_car_link. salesperon_car_link_id

Foreign keys:

Foreign key column	Uniqueness?	Description
Paid_account.account_id	Unique	This foreign key in Paid_account table references the account table. The index is unique since each account number is unique.

Free_account.account_id	Unique	This foreign key in Free_account table references the account table. The index is unique since each account number is unique.
retail_salesperson_link.salesperson_id	Non-Unique	This foreign key in retail_salesperson_link table references the salesperson table. The index is non-unique since a salesperson can work on different retails.
retail_salesperson_link.retail_id	Non-Unique	This foreign key in retail_salesperson_link table references the retail table. The index is non-unique since a retail can hire many salesperson.
loan.salesperosn_id	Non-Unique	This foreign key in loan table references the salesperson table. The index is non-unique since a salesperson can work on different loan applications.
Purchase_order.retail_id	Non-Unique	This foreign key in purchase_order table references the retail table. The index is non-unique since a retail can appear in different purchase orders.
Insurance. insurance_id	Non-Unique	This foreign key in insurance table references the insurance_type table. The index is non-unique since different people can chose same insurance companies for their auto insurance.
Car.make_id	Non-Unique	This foreign key in make table references the car table. The index is non-unique because each make will have so many inventories available.
Car.loan_id	Unique	This foreign key in make car references the loan table. The index is unique because each loan is designed specifically for one car.
Car.policy_number	Unique	This foreign key in make car references the insurance table. The index is unique because each policy only insures one car.
Car.OrderNumber	Non-Unique	This foreign key in make car references the purchase_order table. The index is non-unique

		because an order can contain more than one car.
salesperon_car_link.car_id	Non-Unique	This foreign key in make salesperon_car_link references the car table. The index is non-unique because a car can be sold by different salesperson over the time.
salesperon_car_link.salesperson_id	Non-Unique	This foreign key in make salesperon_car_link references the salesperson table. The index is non-unique because a salesperson can sell many cars.

Below is the screenshot of indexes creations for foreign keys.

```
148  --INDEXES
149  CREATE UNIQUE INDEX paidacctIdx ON Paid_account(account_id);
150
151  CREATE UNIQUE INDEX freeacctIdx on Free_account(account_id);
152
153  CREATE INDEX RSLSIdx on retail_salesperson_link(salesperson_id);
154
155  CREATE INDEX RSLRIdx on retail_salesperson_link(retail_id);
156
157  CREATE INDEX loansaleIdx on loan(salesperson_id);
158
159  CREATE INDEX orderretailIdx on Purchase_order(retail_id);
160
161  CREATE INDEX insIdx on Insurance(insurance_id);
162
163  CREATE INDEX carmakeIdx on Car(make_id);
164
165  CREATE UNIQUE INDEX carloanIdx on Car(loan_id);
166
167  CREATE UNIQUE INDEX carpolicyIdx on Car(policy_number);
168
169  CREATE INDEX carorderIdx on Car(OrderNumber);
170
171  CREATE INDEX SCLCIdx ON salesperon_car_link(car_id);
172
173  CREATE INDEX SCLSIdx ON salesperon_car_link(salesperson_id);
174
```

Besides foreign keys and primary keys, we also need to add query-driven indexes in the 'WHERE' and 'Join' clauses. So far, I only found three query-driven indexes need for my database. If there are growing numbers of queries, I will add more indexes needed to my index creation list.

Query-Driven	Uniqueness	Description
Car.modelyear	Non-Unique	This column is used in my third query using the 'WHERE' clause. This index should be non-unique since many car has the same model year.
Paid_account.RenewalDate	Non-Unique	This column is used in my second query using the 'WHERE' clause. This index should be non-unique since many account can have same renewal date.
BALANCE	Non-unique	This column is used in my second query using the 'WHERE' clause. This index should be non-unique since many accounts can have the same amount of balance.

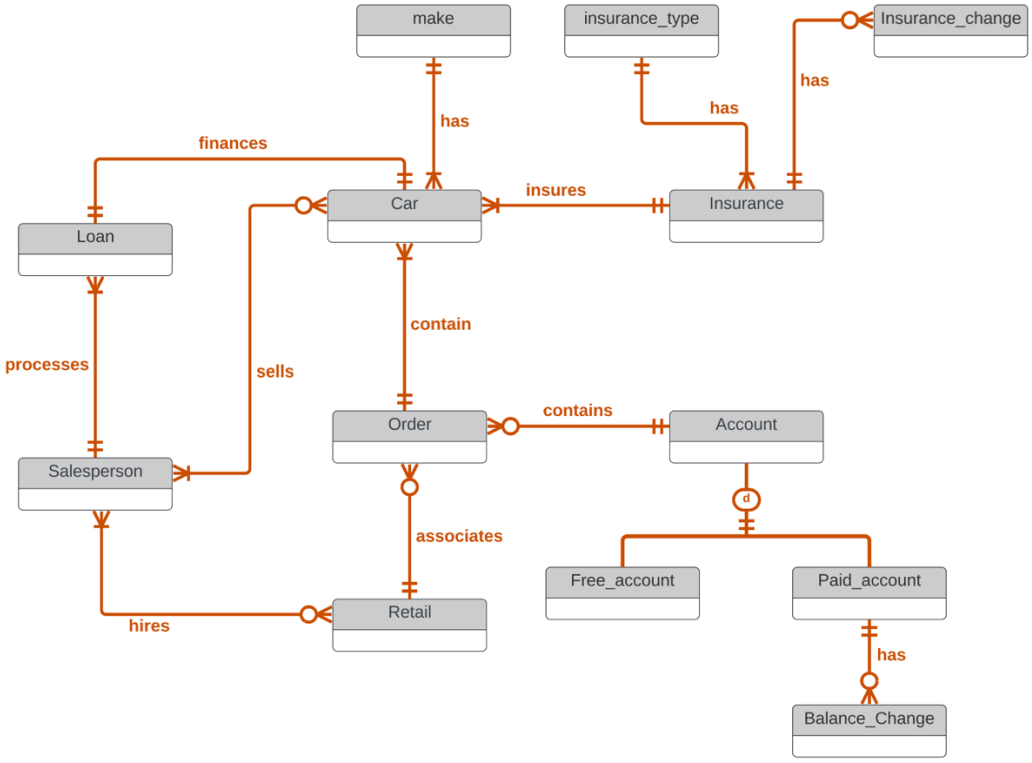
Below is the screenshot for the query-driven index.

```
179  --Query driven index
180
181  CREATE INDEX caryearIdx ON car(modelyear);
182
183  CREATE INDEX PaidrenewalIdx on Paid_account(Renewaldate);
184
185  CREATE INDEX balanceIdx on paid_account(Balance);
186
```

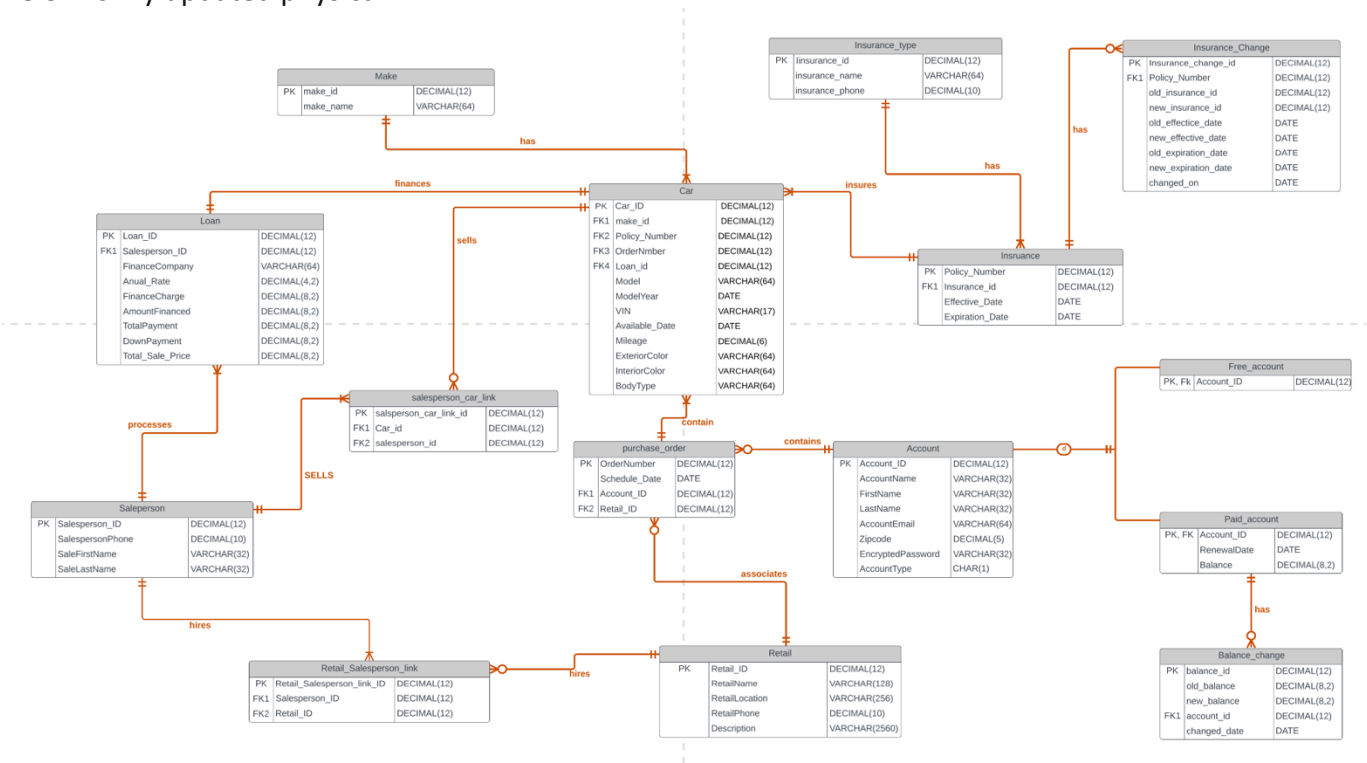
History Table Demonstration

Explain the specifics of your history table, including how the trigger works, and demonstrate that the history table captures changes.
I made two history tables here, one for balance_change and one for insurance_change.

Below is my new conceptual ERD:



Below is my updated physical ERD:



History table 1:

The new structural database rule would be: Each paid account can have many balance changes; Each balance change is for one paid account.

The structural rule indicates that each paid account can have many balance changes, but each balance change is for only one paid account which makes sense since the account balance is specifically related to one account only. I make the paid_account to balance_change mandatory and the relationship is singular. However, I made balance_change to paid_account optional and plural because the balance can change zero to many times for an account.

The attributes are described in the table below.

Attribute	Description
Balance_id	This is the primary key of the history table Balance_Change. It is allowed to contain DECIMAL(12) for values.
Old_balance	This is the balance of the account before a change. The datatype mirrors the balance datatype in the Paid_account table as DECIMAL(8,2).
New_balance	This is the balance of the account after a change. The datatype mirrors the balance datatype in the Paid_account table as DECIMAL(8,2).
Account_id	This is a foreign key to the paid_account table, a reference to the account that had the balance change.
Changed_date	This is the date the balance change occurred, with a DATE datatype.

To create the Balance_change entity and sequence, we would use the following CREATE TABLE and CREATE SEQUENCE statements.

```
136  --history table
137  CREATE TABLE Balance_Change(
138  balance_id DECIMAL(12) NOT NULL PRIMARY KEY,
139  old_balance DECIMAL(8,2) NOT NULL,
140  new_balance DECIMAL(8,2) NOT NULL,
141  account_id DECIMAL(12) NOT NULL,
142  changed_date DATE,
143  FOREIGN KEY (account_id) REFERENCES paid_account(account_id));
144
145  CREATE SEQUENCE balance_seq START WITH 1;
```

Below is my balance change trigger function.

```
--BALANCE_CHANGE
CREATE OR REPLACE FUNCTION BalanceChangeFunc()
RETURNS TRIGGER LANGUAGE plpgsql
AS $trigfunc$
BEGIN
    INSERT INTO balance_change(balance_id, old_balance, new_balance, account_id,changed_date)
    VALUES(nextval('balance_seq'), OLD.balance, NEW.balance, NEW.Account_id, NEW.RenewalDate);
RETURN NEW;
END;
$trigfunc$;

CREATE TRIGGER BalanceChangeTrigger
BEFORE UPDATE OF balance ON Paid_account
FOR EACH ROW
EXECUTE PROCEDURE BalanceChangeFunc();
```

To ensure the trigger function works properly, I randomly chose a paid_account with account_id 15 which has a balance of $104.99.

```
SELECT * FROM paid_account where account_id = 15;
```

Data output | Messages | Notifications

	renewaldate date	account_id numeric (12)	balance numeric (8,2)
1	2023-08-20	15	104.99

I decide to update the balance to $79.33, and the new renewal date is 07/20/2024. Then, I select the balance_change table to prove that the balance has been updated successfully for account_id 15.

```
UPDATE paid_account
SET balance= 79.33, renewaldate = '07-20-2024'
WHERE account_id = 15;

SELECT * FROM balance_change;
```

Data output | Messages | Notifications

	balance_id [PK] numeric (12)	old_balance numeric (8,2)	new_balance numeric (8,2)	account_id numeric (12)	changed_date date
1	1	104.99	79.33	15	2024-07-20

History table 2:

I also create an insurance_change table that follows the structural rule: Each insurance policy can have many insurance changes; Each insurance change is for one insurance policy.

The structural rule indicates that each insurance policy can have many times of insurance changes, but each insurance change is related to one single insurance policy. I make the policy to insurance change optional and plural. However, I made insurance_change to insurance policy mandatory and singular.

Attribute	Description
Insurance_change_id	This is the primary key of the history table insurance_Change. It is allowed to contain DECIMAL(12) for values.
Old_insurance_id	This is the chosen insurance of the policy before a change. The datatype mirrors the insurance_id datatype in the insurance table as DECIMAL(12)
New_insurance_id	This is the chosen insurance chosen of the policy after a change. The datatype mirrors the insurance_id datatype in the insurance table as DECIMAL(12)
Policy_number	This is a foreign key to the Insurance table, a reference to the policy that has an insurance change.
Old_effective date	This is the old effective date of the insurance before a change. The datatype mirrors the effective_date datatype in the insurance table as DATE.
New_effective_date	This is the new effective date of the insurance after a change. The datatype mirrors the effective_date datatype in the insurance table as DATE.
Old_expiration_date	This is the old expiration date of the insurance before a change. The datatype mirrors the expiration_date datatype in the insurance table as DATE.
New_expiration_date	This is the new expiration date of the insurance after a change. The datatype mirrors the expiration_date datatype in the insurance table as DATE.
Changed_date	This is the date the insurance change occurred using current_date with a DATE datatype.

To create the insurance_change entity and sequence, we would use the following CREATE TABLE and CREATE SEQUENCE statements.

```
148 CREATE TABLE Insurance_Change(
149 Insurance_Change_id DECIMAL(12) NOT NULL PRIMARY KEY,
150 policy_Number DECIMAL(12) NOT NULL,
151 old_insurance_id DECIMAL(12) NOT NULL,
152 new_insurance_id DECIMAL(12) NOT NULL,
153 Old_effective_Date DATE,
154 new_effective_date DATE,
155 old_expiration_date DATE,
156 new_expiration_date DATE,
157 changed_on Date,
158 FOREIGN KEY (Policy_Number) References insurance(Policy_number));
159
160 CREATE SEQUENCE ins_change_seq;
```

Below is my balance change trigger function.

```
543 --Insurance change
544
545 CREATE OR REPLACE FUNCTION InsuranceChangeFunc()
546 RETURNS TRIGGER LANGUAGE plpgsql
547 AS $trigfunc$
548 BEGIN
549     INSERT INTO insurance_change
550     VALUES(nextval('ins_change_seq'), NEW.Policy_number, OLD.insurance_id, NEW.insurance_id, OLD.effective_date,
551            NEW.effective_date, OLD.expiration_date, NEW.expiration_date, CURRENT_DATE);
552 RETURN NEW;
553 END;
554 $trigfunc$;
555
556 CREATE TRIGGER InsuranceChangeTrigger
557 BEFORE UPDATE OF insurance_id, effective_date, expiration_date ON Insurance
558 FOR EACH ROW
559 EXECUTE PROCEDURE BalanceChangeFunc();
```

To ensure the trigger function works properly, I randomly chose a policy_number = 5 and saw that current insurance_id is 5 with Kemper Auto.

```
561 select * from insurance
562 Join insurance_type on insurance.insurance_id = insurance_type.insurance_id
563 where Policy_number=5;
564
```

Data output | Messages | Notifications

	policy_number numeric (12)	insurance_id numeric (12)	effective_date date	expiration_date date	insurance_id numeric (12)	insurance_name character varying (64)	insurance_phone numeric (10)
1	5	5	2021-07-01	2022-06-30	5	Kemper Auto	8007821020

Since the insurance expired on 06/30/2022, I decide to purchase new insurance with a new insurance_id = 8, effective on 07/01/2022 and expiring on 06/30/2023.

```
565
566 UPDATE Insurance
567 SET insurance_id = 8, effective_date = '07-01-2022', expiration_date='06-30-2023'
568 WHERE Policy_number = 5;
569
570 SELECT * FROM insurance_change;
571
```

Data output | Messages | Notifications

	insurance_change_id [PK] numeric (12)	policy_number numeric (12)	old_insurance_id numeric (12)	new_insurance_id numeric (12)	old_effective_date date	new_effective_date date	old_expiration_date date	new_expiration_date date	changed_on date
1	1	5	5	8	2021-07-01	2022-07-01	2022-06-30	2023-06-30	2022-10-17

Then, I want to know what is the name of the current insurance company, so I did another query below which shows that policy_Number 5 now has Liberty Mutual.

```
571
572 select insurance_name, insurance.insurance_id, policy_number,effective_date, expiration_date
573 from insurance
574 Join insurance_type on insurance.insurance_id = insurance_type.insurance_id
575 where Policy_number=5;
576
577 --QUERIES
```

Data output | Messages | Notifications

	insurance_name character varying (64)	insurance_id numeric (12)	policy_number numeric (12)	effective_date date	expiration_date date
1	Liberty Mutual	8	5	2022-07-01	2023-06-30

Data Visualizations

Include and explain data visualizations and stories that tell effective stories about the information in a way that people quickly and accurately understand it.

Visualization 1:

To be effective, any business should understand the demographics of its customers. The more they know about their customers, the more effective they can retain them, provide better service, and reach new customers. AutoBuy starts its business near the coast and is now trying to expand its business. In order to do so, we need to have a better understanding of customers' demographic information, and that can only rely on the analysis of the zip code in the Account table. A question arises, "Which areas do most AutoBuy's customers reside in based on their zip codes?" The answer will give me a better understanding of where the business is made and help me to make better business decisions if I want to expand AutoBuy.

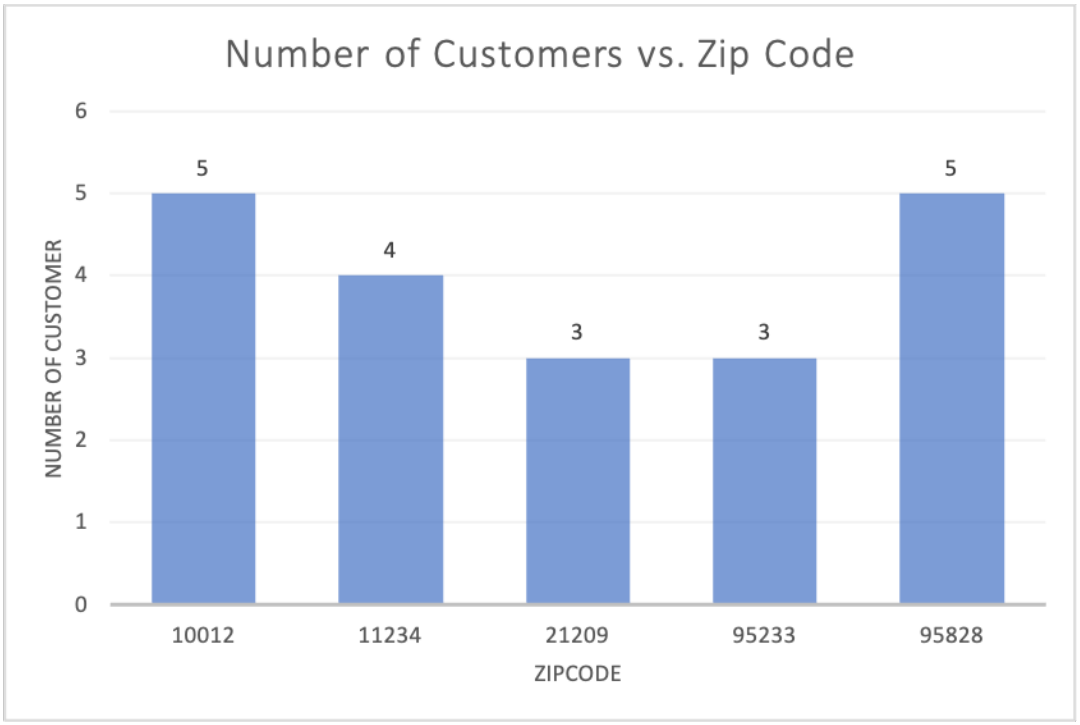
To answer this question, I develop and execute a query that counts the number of orders residing in each postal code. The result is ordered by the zip code. The query and result are shown below.

```
623
624  SELECT zipcode, count(account_id) as number_of_customer
625  FROM account
626  GROUP BY zipcode
627  ORDER BY  zipcode;
628
```

Data output | Messages | Notifications

	zipcode numeric (5)	number_of_customer bigint
1	10012	5
2	11234	4
3	21209	3
4	95233	3
5	95828	5

Since these results only have one measure, I plan to use a simple bar chart to visualize the results. After exporting to a CSV file and creating the bar chart in EXCEL, I see the following result.



What story does this visualization tell us? First, there are five zip codes shown above, 10012, 11234, 21209, 95233, and 95828. Both 10012 and 11234 are zip codes for New York and they have a total of 9 customers. While 21209 is the zip code for Maryland which has 3 customers. Last but not the least, 95233 and 95828 are the zip codes for California.

Since I only have 20 registered customers in my database, it is easy to do the math that New York has a total of 9 customers which ranks at the top, and California has a total of 8 customers which ranks

second. However, considering the growing size of the database of Autobuy, I ran a query to sum the total number of registered customers according to the first digit of the zip code, and we can see the result below.

```
629  SELECT cast(substring(cast(zipcode AS VARCHAR(5)), 1, 1) as integer) AS matchzip,
630         count(account_id) AS number_of_customer
631  from account
632  group by matchzip
633  order by number_of_customer DESC;
634
```

Data output | Messages | Notifications

	matchzip integer	number_of_customer bigint
1	1	9
2	9	8
3	2	3

This information could be used in a variety of ways. First, we would definitely need to secure those businesses in New York and California since we already have a nice volume of customers in those two places. Second, we consider our business still in the initial stage in Maryland, so we may consider investing more in Maryland using our experiences from New York and California, like more advertisements on TV or in newspapers. After all, we will try to expand our business to other states.

Visualization 2:

In many businesses, people would like to know how the cash flow in each business. The best way to understand the change in cash flow in AutoBuy can be found in the Balance_Change table. Since AutoBuy has a growing number of customers, we consider the average balance change in our database. So here is my question, “What is the average balance change in AutoBuy database including past, current, and future?” This question will help me to understand how the cash flow in Autobuy and may give some business insights for using available cash wisely.

In order to provide more data for balance change, I created an update procedure called “UpdateBalance’ to help ease the process of populating the balance_change table as shown below.

```
637  CREATE OR REPLACE PROCEDURE UpdateBalance(new_account_id IN DECIMAL, balance_new IN DECIMAL,
638                                            new_date IN DATE)
639  AS
640  $proc$
641  BEGIN
642      UPDATE paid_account
643      SET balance = balance_new, renewaldate = new_date
644      WHERE account_id = new_account_id;
645  END;
646  $proc$ LANGUAGE plpgsql;
```

```
--INSERT DATA
START TRANSACTION;
DO
$$BEGIN
    CALL UpdateBalance(11, 20.88, '11-03-2022');
    CALL UpdateBalance(11, 50.77, '12-30-2022');
    CALL UpdateBalance(12, 65.33, '12-03-2022');
    CALL UpdateBalance(12, 47.50, '02-01-2023');
    CALL UpdateBalance(13, 79.00, '08-21-2023');
    CALL UpdateBalance(13, 103.44, '11-03-2023');
    CALL UpdateBalance(14, 20.88, '06-20-2023');
    CALL UpdateBalance(14, 88.88, '12-13-2023');
    CALL UpdateBalance(14, 66.27, '12-24-2023');
    CALL UpdateBalance(15, 50.77, '10-08-2023');
    CALL UpdateBalance(16, 179.32, '12-19-2022');
    CALL UpdateBalance(16, 280.38, '01-29-2023');
    CALL UpdateBalance(16, 250.77, '11-03-2023');
    CALL UpdateBalance(17, 40.77, '12-24-2022');
    CALL UpdateBalance(18, 67.33, '12-30-2022');
    CALL UpdateBalance(19, 108.77, '11-03-2023');
    CALL UpdateBalance(19, 58.45, '01-23-2024');
    CALL UpdateBalance(20, 11.28, '11-24-2023');
END$$;
COMMIT TRANSACTION;

SELECT * FROM Balance_Change;
```

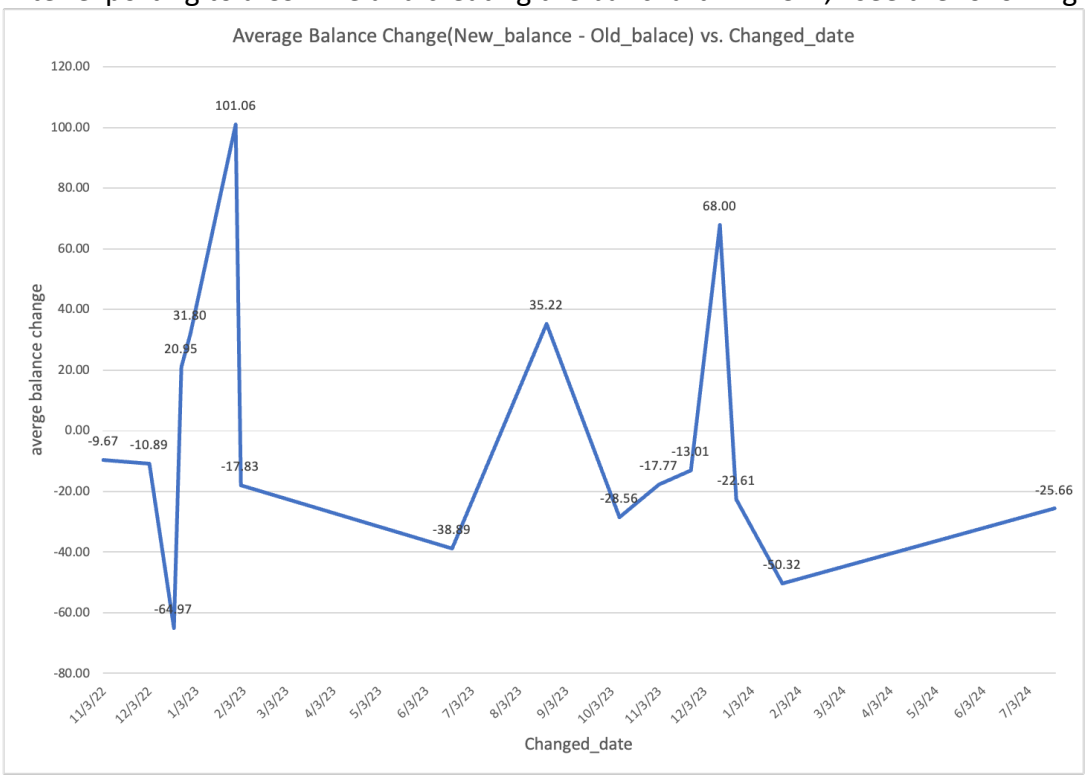
	balance_id [PK] numeric (12)	old_balance numeric (8,2)	new_balance numeric (8,2)	account_id numeric (12)	changed_date date
1	1	104.99	79.33	15	2024-07-20
2	2	30.55	20.88	11	2022-11-03
3	3	20.88	50.77	11	2022-12-30
4	4	76.22	65.33	12	2022-12-03
5	5	65.33	47.50	12	2023-02-01
6	6	43.78	79.00	13	2023-08-21
7	7	79.00	103.44	13	2023-11-03
8	8	59.77	20.88	14	2023-06-20
9	9	20.88	88.88	14	2023-12-13
10	10	88.88	66.27	14	2023-12-24
11	11	79.33	50.77	15	2023-10-08
12	12	244.29	179.32	16	2022-12-19
13	13	179.32	280.38	16	2023-01-29
14	14	280.38	250.77	16	2023-11-03
15	15	19.82	40.77	17	2022-12-24
16	16	33.62	67.33	18	2022-12-30
17	17	156.92	108.77	19	2023-11-03
18	18	108.77	58.45	19	2024-01-23
19	19	24.29	11.28	20	2023-11-24

To answer this question, I developed and execute a query that calculates the average balance change in every day's data. The results and the query are shown below.

	avg_balchange numeric	changed_date date
1	-9.6700000000000	2022-11-03
2	-10.890000000000	2022-12-03
3	-64.970000000000	2022-12-19
4	20.950000000000	2022-12-24
5	31.800000000000	2022-12-30
6	101.06000000000	2023-01-29
7	-17.830000000000	2023-02-01
8	-38.890000000000	2023-06-20
9	35.220000000000	2023-08-21
10	-28.560000000000	2023-10-08
11	-17.773333333333	2023-11-03
12	-13.010000000000	2023-11-24
13	68.000000000000	2023-12-13
14	-22.610000000000	2023-12-24
15	-50.320000000000	2024-01-23
16	-25.660000000000	2024-07-20

```
675  SELECT AVG(new_Balance - old_balance) as avg_balchange,
676         changed_date
677  From balance_change
678  Group by changed_date
679  ORDER BY changed_date;
```

Since these results only have one measure, I plan to use a simple scatter plot to visualize the results. After exporting to a CSV file and creating the bar chart in EXCEL, I see the following result.



What is the story behind this visualization? First of all, for each bottom of the average balance change on each day, we can say that customers use their balance to renew their membership or place purchase order. For each peak of the average balance change on that day, we can see that customers add more funds to their account. For example, the lowest bottom in our graph is on 12/19/2022 which means there may be many customers using their balances to renew the account or there are more orders placed on that day with an average of $64.97 decrease in their accounts. On the other hand, the highest peak in our plot is on 01/29/2023 with an average of 101.06 added to their accounts.

Currently, I didn't see a certain pattern in my visualization which may be due to the small sample size. In addition, this visualization also inspires us to give some promotions on certain dates in which people will get an extra 5% for the amount they reload into the accounts. I believe such a promotion will help to increase the cash available for AutoBuy.

Summary and Reflection

Update the concise summary of your project and the work you have completed thus far, and additionally update your questions, concerns, and observations, so that you and your facilitator or instructor are aware of them and can communicate about them.

My database is for an App called AutoBuy which records data about automobile vehicle information, auto loan, auto insurance, and customer's vehicle purchase order. AutoBuy provides a platform for customers to enjoy the all-in-one service when they want to buy, sell, or trade-in vehicles. We aim to build AutoBuy into a convenient and price transparent system for people which obviously needs multiple data.

For this project, I learned how to build a database from zero to now with many details. I learned how to figure out the associative and plural relations between each entity table, and create conceptual ERD and physical ERD to build a solid foundation for my database. In addition, I also learned how to apply my SQL skill learned from labs to my project, including creating tables, sequences, indexes, procedures, triggers, and writing useful queries for my database and many other techniques. Most importantly, I am able to use the database to provide visualizations and tell stories behind those visualizations.

As I reflect on the database and my accomplishments, I can say it is been a long but rewarding road. Before taking this course, I only know how use given database to do visualizations or analysis, but have no idea about create a database can be this complex and cost a lot of time and efforts. I understand there must be more works to be done to make AutoBuy a durable and useful database into real-world uses, but a good portion of the database is already implemented. I will keep updating Autobuy and hope it can be hooked up to the application in Android or iPhone.