University of Oklahoma



Team Waterfall

“Go with the Flow”

Anna Dowhower, Bennett Friedman, Cody Glover, Jennifer Ha, Alessio Hall

MIS 3353 – Database Management

Dr. Durcikova

March 25, 2021

# **Executive Summary**

Team Waterfall has formed a partnership with Sonner Tire located in Shawnee, Oklahoma. The primary goal of the relationship was to have Team Waterfall help implement a better database system than Sonner Tire’s current one, to help accommodate their growing business needs. We accomplished our goal of creating a fluid, easy to use and proficient database that provides Sonner Tire with an interface that will be much more efficient and most importantly productive for the company. The provided database system will allow Sonner Tire to unify its business processes of sales, invoicing and inventory, while allowing them to quickly gather data for customer information, sales history, inventory and employee information to allow them to optimize their business proceedings.

Team Waterfall approached this project by breaking it into three main sections, conceptual design, logical design and physical design. We gathered information and parameters from the client and from there was able to produce a database that will be a very useful tool in further growing the Sonner Tire brand and business. Our tool will be able to allow Sonner Tire employees and management to access information regarding any part of their business such as compatible tires for specific vehicles, customer payment information, and points to reorder for tires just to name a few. Our database stores all this information in one easy to use and quick place to access.

This report contains the steps Team Waterfall took in designing our database and processes on how to access the database. We provide the ERD that was used in the conceptual stage of the process while also showing all the logical relationships that were used as well. We provide a data dictionary for understanding of our database and the results you should expect. We also show specific queries asked by Sonner Tire for their database and the results to match them. Finally, Team Waterfall members discuss lessons learned throughout the process and information about each member of the team.

The total cost of the project was $1,102. We kept track of this by creating an excel sheet that we would log our minutes worked on every portion of the project, and who worked on that portion of the project. The project took a total time of 44.08 hours and cost $25 per hour.

Contents

[Executive Summary (To be done for final submission) 2](#_Toc70364714)

[Conceptual Design 4](#_Toc70364715)

[The Client Meeting 4](#_Toc70364716)

[Q&A during the Meeting & Information We Learned 4](#_Toc70364717)

[Significant Assumptions 6](#_Toc70364718)

[What is an ERD? Why is it necessary? 7](#_Toc70364719)

[Business Cycles Used 7](#_Toc70364720)

[Data Provided by Client 7](#_Toc70364721)

[ERD Created 9](#_Toc70364722)

[Query Feasibility and Current ERD 10](#_Toc70364723)

[Logical Design 12](#_Toc70364724)

[Normalization 12](#_Toc70364725)

[Normalized Relations 13](#_Toc70364726)

[Differences between ERD and Normalized Relations 14](#_Toc70364727)

[Database Integrity 15](#_Toc70364728)

[Physical Design and Implementation 16](#_Toc70364729)

[Data Dictionary 16](#_Toc70364730)

[Denormalization 16](#_Toc70364731)

[Implemented Physical Design 16](#_Toc70364732)

[Challenges Faced/Addressed during Implementation 17](#_Toc70364733)

[Strengths and Weaknesses Encountered during Implementation 17](#_Toc70364734)

[Specific SQL Statements Requested 18](#_Toc70364735)

[Three Additional Queries 21](#_Toc70364736)

[User Documentation 22](#_Toc70364737)

[Opening the Database 22](#_Toc70364738)

[How to Enter New Data to the Existing Tables 23](#_Toc70364739)

[How to Create a New Table 24](#_Toc70364740)

[How to Pull Information from a Table 25](#_Toc70364741)

[What We Learned Throughout this Process 26](#_Toc70364742)

[Appendix 27](#_Toc70364743)

[Data Dictionary 27](#_Toc70364744)

[Project Management 30](#_Toc70364745)

[Get to Know Team Waterfall 31](#_Toc70364746)

[Team Contract 32](#_Toc70364747)

[Expectations 32](#_Toc70364748)

[Negative Behavior 32](#_Toc70364749)

# **Conceptual Design**

The following section will cover the process of conceptual design. In other words, how we created our Entity Relationship Diagram (ERD) and what information led to certain decisions. Starting out, we held a meeting with the client in order to confirm information, allowing us to make specific assumptions. That information, along with the assumptions and case details, were then used in the design of our ERD.

## The Client Meeting

Before starting on this project, it was important for us to meet with our client. This allowed us to verify certain information as well as ask questions about the specific tasks our client wanted to be able to accomplish. In our 20-minute meeting, we were able to assess our clients' needs and concerns. We also obtained answers to questions that were necessary for designing our Entity Relationship Diagram.

* Meeting Time: March 15, 2021, 11:00AM
* Location: Zoom Video Conference
* Interviewers: Anna Dowhower, Bennett Friedman, Cody Glover, Jennifer Ha, & Alessio Hall
* Interviewee: Helen Anderson

## Q&A during the Meeting & Information We Learned

**Q: What contact information do you get from customers?**

A: We have their current address, phone number, and email.

**Q: How much is their discount they give to customers (each set of four tires)?**

A: Depends on specials (seasonal specials), and construction on tires (ex: nails in tire)

**Q: What sort of problems have you had in the past with multiple invoices?**

A: Caused by someone not showing up on time, needing more training or being let go (variety of issues).

**Q: What are the different job titles an employee can have?**

A: There’s a manager (will state the job needed), whoever took on the car, and whichever employee worked on that car.

**Q: Is there a specific insurance the company accepts?**

A: Insurance is bought from us, not necessarily specific limits of what insurances we take.

**Q: At how many tires would you like us to notify you that you are low on stock?**

A: Depends on the type of cars – orders less inventory for sports car, etc. Doesn’t keep it in stock since rare?

**Q: How many missed payments or problems until a customer is considered problematic**

A: If they don’t pay within 30 days; if account is 90 days over it is sent to collections.

**Q: Are there managers and do they partake in the same jobs as regular employees?**

A: Two managers on duty at all times - don’t go in and do work on tires. If needed, they can help a new employee, but they don’t usually do the tire work. Quality checks.

**Q: Would you like us to include why a customer chose Sonner Tire? (these are because there is a section that says we should recommend 3 additional reports that are not required)**

A: If you can include it, yes.

**Q: Would you like us to Inquire about other possible vehicles in a person’s family or ownership? (these are because there is a section that says we should recommend 3 additional reports that are not required)**

A: Want to know if someone has multiple cars – only ask when they bring the car in. Need to know the owners of the car because it might be a family; multiple people can drop the car off or pick it up.

**Q: How many employees are there?**

A: 10

**Q: Other than tires are there any other materials that need to be procured?**

A: Materials for flats

**Q: Are there set rules for accepting or denying financing? (What is the interest rate)?**

A: Interest rate is flat rate of 2.5%; extend to pretty much any customer $200 they can pay. Do a credit check for lending more money- has to be a manger that does it.

**Q: (Broader Question): I am a little confused as to what data Sonner Tire is looking for to help in decision making about possible expansion and/or additional business opportunities. (Are there any specifics in which they are looking for?)**

A: Do they have enough cash inflow; can they keep more tires in their warehouse?

**Q: Loyalty Program?**

A: Don’t currently do it but would like to do it once database is set up and margins are known. (think this could be possible expansion)

# **Significant Assumptions**

Assumptions are always made whenever designing an ERD. They are primarily formed based on a client's needs and adjustments are made to the ERD accordingly. Below we list 5 of our significant assumptions.

|  |  |
| --- | --- |
| 1. One employee sells the repair, and one performs the repair (technician). |  |
| 2. The Tire Entity will not need all the detailed attributes, but they will be kept outside of the database on an information sheet. | N/A |
| 3. There are no returns so that will not be present/represented in the Service Order Line. | N/A |
| 4. The ordering of equipment may be added on later in the development of the database. | N/A |
| 5. As most companies do not want to deal with the security risks involved with storing payment data, there will be no credit card numbers or payment methods stored. | N/A |

## What is an ERD? Why is it necessary?

An Entity Relationship Diagram or ERD for short, is a database schema that shows the structure and organization of the database we are creating. It also shows the type of information that we will be storing in our database for Sonner Tire. For example, we will have an entity for a Car when one comes in, and this entity will store various attributes about the car such as Make, Model and Year, while also assigning it a unique ID to that specific vehicle. An ERD is like a blueprint for a database, the same way a blueprint would work for construction, as it will map out exactly what is needed for a database and where. It is extremely important that we use an ERD for the Sonner Tire database because it gives us a conceptual idea of what we need while also letting us visualize this complex database.

## Business Cycles Used

In our ERD, we used two business cycles: revenue and expenditure. The revenue cycle is used in order to keep track of our customers, their payments, and their vehicles. The expenditure cycle covers the purchases that Sonner Tire makes from manufacturers.

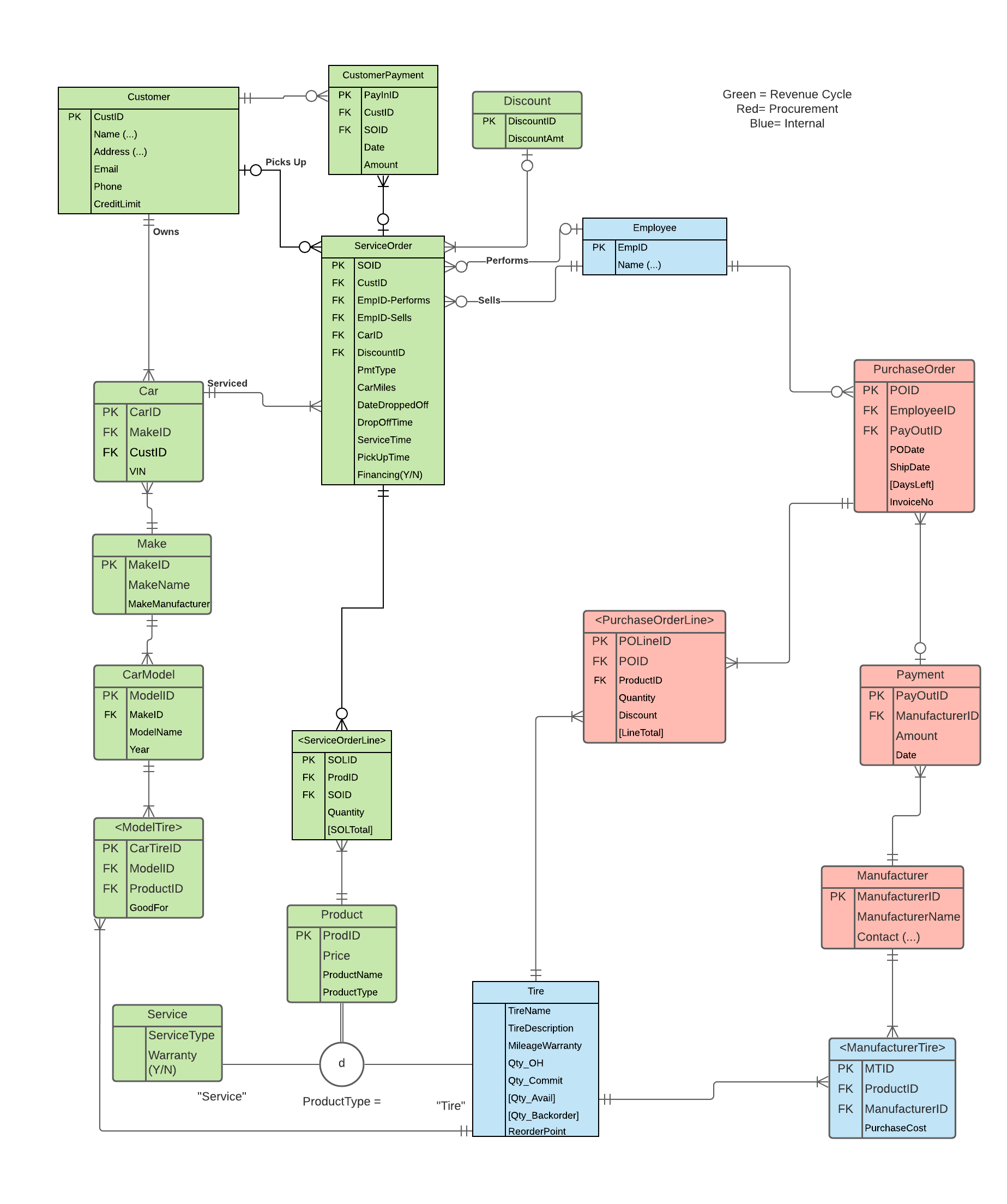
## Data Provided by Client

The client has provided us with some sample data of tires and the cars that they work for. We will be using the majority of this data in the implementation stage to make sure that we include all of the specific tires, and match them with their respective cars, but we will also be using this data in other parts of the process to make sure that we name the columns and their attributes with respect to the client’s current data. When looking at the data which we have attached below one can see that this is raw data, and it needs to be further broken down. When we look at the Model column we need to split all of the different model years into their own rows so that we can easily, efficiently, and effectively pair ever each model year to its tire. In the second snippet of data we can see a few of the more in-depth information for each tire and we will need to make sure that moving forward all of the information regarding each tire is connected. This data will also make it easier to draw a mental picture of how to set things up.



# **ERD Created**

Below is our customized ERD for Sonner Tire Company. It includes the necessary attributes for each entity and the relationship between each of the entities. It uses the revenue and expenditure cycles.



## Query Feasibility and Current ERD

In general, when creating an ERD we begin with a generic ERD with the basic cycles, but then we must make changes to the generic model in order to include the client's requirements and meet their expectations. In this section, we shed light on the changes to generic ERD as well as provide projections for the SQL statements.

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Query Question** | **Tables needed to run the query** | **Projected SQL Statement** |
| 1 | Total sales (in dollars) by region for a given tire manufacturer and car manufacturer. It would be great if we can specify the car model and year too (note that we would like to be able to input the month to be calculated). | Tire, Car Manufacturer, Customer, Service Order Line, Service Order | SELECT T.manufacturerID, SUM(T.saleprice\*SOL.quantity) Sales, make, model, year, SO.datepickedup, CR.address  FROM manufacturer M  JOIN Tire T ON M.manufacturerID=T.ManufacturerID  JOIN carTire CT ON CT.cartireID=T.tireID  JOIN carmanufacturer CM ON CT.carmanID=CM.carmanID  JOIN car C ON CM.carmanID=C.carmanID  JOIN serviceorder SO ON C.carID=SO.carID  JOIN serviceorderline SOL ON SO.soid=SOL.soid  JOIN customer CR ON SOL.custID=CR.custID  GROUP BY CR.address, T.manufacturerID, CM.make |
| 2 | Total sales (in dollars) by customer in a given year. | Customer, Service Order, Tire, Service Order Line | SELECT CR.custID, SUM(T.saleprice\*SOL.quantity) Sales  FROM customer CR  JOIN car C ON CR.custID=C.custID  JOIN serviceorder SO ON C.carID=SO.carID  JOIN serviceorderline SOL ON SO.soid=SOL.soid  JOIN employee E ON SO.empID=E.empID  JOIN delivery D ON E.empID=D.empID  JOIN purchaseorderline POL ON D.delID=POL.delID  JOIN tire T ON POL.tireID=T.tireID  Group BY CR.custID, SO.datepickedup |
| 3 | The five highest selling tires. | Tire, Service Order Line | SELCT SUM(SOL.quantity\*T.saleprice) Sales, T.tireID  FROM serviceorderline SOL  JOIN serviceorder SO ON SOL.soid=SO.soid  JOIN employee E ON SO.empID=E.empID  JOIN delivery D ON E.empID=D.empID  JOIN purchaseorderline POL ON D.delID=POL.delID  JOIN tire T ON POL.tireID=T.tireID  GROUP BY T.tireID |
| 4 | Itemized invoices for jobs for each customer that need to include tires purchased/tire rotation/tire repair/tire protection. | Customer, Service, Tire, Service Order | SELECT CR.customerID, T.saleprice, S.servicetype, SO.datepickedup  FROM customer CR  JOIN car C ON CR.custID=C.custID  JOIN serviceorder SO ON C.carID=SO.carID  JOIN serviceorderline SOL ON SO.soid=SOL.soid  JOIN product P ON SOL.prodID=P.prodID  JOIN service S ON P.ProductID=S.ProductIDService  JOIN tire T ON P.ProductID=T.ProductIDTire |
| 5 | The number and type of job performed by each of our employees. | Position, Employee | SELECT P.positionname, COUNT(P.positionname) NumberOfPositions  FROM employee E  JOIN position P ON E.positionID=P.positionID  GROUP BY E.name, P.positionname |
| 6 | Number of times a tire protection has been purchased for a particular tire and number of times free service has been applied (free tire damage repair, free replacement). | Tire, Service Order Line, Service | SELECT T.tireID, SUM(SOL.quantity) FreeRepairs, \*\*\*incomplete  FROM serviceorderline SOL  JOIN product P ON SOL.prodID=P.prodID  JOIN service S ON P.ProductID=S.ProductIDService  JOIN tire T ON P.ProductID=T.ProductIDTire  WHERE servicetype LIKE ‘maintenance’  AND warranty=’Y’  GROUP BY T.tireID |
| 7 | The following items for Purchase Orders: manufacturer name, number of POs, total cost. | Purchase Order, Purchase Order Line, Manufacturer | SELECT PO.PayOutID, manufacturername, POL.quantity, POL.linetotal  FROM TPurchaseOrderLine POL  JOIN TPurchaseOrder PO ON POL.POID=PO.POID  JOIN TPayment P ON PO.PayOutID=P.PayOutID  JOIN TManufacturer M ON P.manufacturerID=M.manufacturerID |
| 8 | Number of orders and total sales per customer in the past 2 years. This report is particularly important as it shows the number of returning customers. | Customer, Customer Payment, Service Order Line, Service Order | SELECT CFName,CLName, Count(SOID)SOID  FROM TCustomer1 CR  JOIN TCar C ON CR.CustID=C.CustID  JOIN TServiceORder SO ON C.CarID=SO.CarID  WHERE DateDroppedOff>= '2020-05-01'  GROUP BY CFName, CLName |
| 9 | List of tires that have not been purchased within the last 6 months (in order to better manage inventory). | Tire, Service Order Line, Service Order | SELECT ProductID, ProductName,ProductType  FROM TServiceOrderLine SOL LEFT JOIN TProduct P  ON P.ProductID=SOL.ProdID  WHERE SOL.ProdID IS Null AND ProductType='Tire' |
| 10 | Names of customers who took advantage of the financing option, date purchased, total amount purchased, credit limit, number of payments made, total amount paid, outstanding amount, is time less than 6 months, all displayed from latest date and then largest amount owed. | Customer, Customer Payment, Service Order, Service Order Line | SELECT CR.CFName,CR.CLName, SO.DateDroppedOff, SUM(SOL.soltotal)Total , COUNT(CP.payinID) Paid  FROM TCustomerPayment CP  JOIN TCustomer1 CR ON CP.custID=CR.custID  JOIN TCar C ON CR.custID=C.custID  JOIN TServiceOrder SO ON C.carID=SO.carID  JOIN TServiceOrderLine SOL ON SO.soid=SOL.soid  WHERE pmttype = 'financing'  GROUP BY CR.CFName,CR.CLName, SO.DateDroppedOff |
| 11 | Total profit per tire type and manufacturer type in the past 6 months. | Tire, Manufacturer, Service Order Line | SELECT T.tirename, M.manufacturername, SUM(SOL.soltotal) FROM manufacturer M  JOIN tire T ON M.manufacturerID=T.manufacturerID  JOIN product P ON T.ProductIDTire=P.ProductID  JOIN serviceorderline SOL ON P.prodID=SOL.prodID  GROUP BY T.tireID, M.manufacturerID |
| 12 | List of all customers that have not made a purchase within the last 12months from the current date. | Customer, Service Order | SELECT CR.name FROM customer CR  JOIN serviceorder SO ON C.custID=SO.custID  WHERE datepickedup <=\_\_\_\_  AND datepickedup >=\_\_\_\_  AND SO.soid IS NULL |
| 13 | List of customers whose average sales is less than the average of all sales. This will help us to find customers whom we should target to get a higher volume of sales. | Customer, Service Order Line | SELECT C.name FROM customerpayment CP  JOIN customer C ON CP.custID=CR.CustID  JOIN serviceorder SO ON C.carID=SO.carID  JOIN serviceorderline SOL ON SO.soid=SOL.soid  GROUP BY C.name  HAVING AVG(SOL.soltotal) <  (SELECT AVG(soltotal)  FROM serviceorderline) |

# **Logical Design**

Logical Design is important when creating a database because it is the abstract mapping of the relationships from the ERD. Where conceptual design works with entities, attributes, and data instances, logical design instead looks at these things as relations, columns, and rows. Part of this process involves organizing data in order to achieve atomicity and avoid data redundancy. This can be done through normalization, which is discussed below. Logical design is something that requires a great amount of detail, as this process will determine the integrity of a database later on.

## Normalization

Normalization is the process of structuring a database with regards to the normal forms (0NF, 1NF, 2NF, and 3NF) in order to reduce data redundancies, and avoid the three types of anomalies (deletion, insertion, and modification) that will create errors and/or inconsistencies when performing. It is vital to use Normalization of ERD in this project to ensure that when the database is built no information is overlooked and all the foreign key restraints are valid. Without the normalization it can be very easy to overlook important pieces of information, and overall normalization creates a vivid pathway in which to follow.

Normalization of the Data Provided by the Client

To Normalize the Data provided by the client the first thing that we did was decide on the pathway to follow in order to ensure that all foreign keys will be valid. Given the complexity of the ERD it is sometimes difficult to decipher the navigation from outside to inside. This task required an agile approach because as you move through the ERD you can see where specific attributes need to be looked at/referenced again and being able to normalize the data piece by piece rather than running through the whole thing at once allows for us to go back and make the required changes in order to make all the attributes within each table valid. Although all of the normalized relations are in the next section, we have listed below the normalization relations for the data specifically provided by the client.

* **T\_Make**(MakeID, MakeName)
* **T\_CarModel**(ModelID, *MakeID*, ModelName, Year)
  + Foreign Key MakeID references T\_Make not null on delete restrict
* **T\_State**(StateID, SName)
* **T\_Zip**(ZipCode, City, *ZStateID*)
  + Foreign key ZStateID references T\_State not null on delete restrict
* **T\_Manufacturer**(ManufacturerID, ManufacturerName, MFName, MLName)
* **T\_Tire**(*ProductID-Tire*,TireName, MileageWarranty, ReorderPoint)
  + ProductID-Tire references T\_Product null allowed on delete restrict
* **T\_ModelTire**(CarTireID, *ModelID*, *TireID*)
  + Foreign key CarManID references T\_CarManufacturer not null on delete restrict
  + Foreign key TireID references T\_Tire not null on delete restrict

## Normalized Relations

(Note: Document would not allow dotted underline so foreign keys are *italicized*)

* **T\_State**(StateID, SName)
* **T\_Service(**ServiceType, SWarranty)
* **T\_Product**(ProductID, ProductName, Price, ProductType)
* **T\_Service(***ProductID-Service*, ServiceType, SWarranty)
  + ProductID-Service references T\_Product null allowed on delete set null
* **T\_Manufacturer**(ManufacturerID, ManufacturerName, MFName, MLName)
* **T\_ManufacturerTire**(MTID, *ProductID*, *ManufacturerID*, PurchaseCost)
  + Foreign key ProductID references T\_Product not null on delete restrict
  + Foreign key ManufacturerID references T\_Manufacturer not null on delete restrict
* **T\_Zip**(ZipCode, City, *ZStateID*)
  + Foreign key ZStateID references T\_State not null on delete restrict
* **T\_Customer(**CustID, CFName, CLName CAddress, CState, CCity, *CZipCode*, CEmail, CPhone, CCreditLimit)
  + Foreign key CZipCode references T\_Zip not null on delete restrict
* **T\_CustomerPayment(**PayInID, *CustID*, CPDate, CPAmount)
  + Foreign key CustID references T\_Customer not null on delete restrict
* **T\_Payment**(PayOutID, *ManufacturerID*, PAmount, PDate)
  + Foreign key ManufacturerID references T\_Manufacturer not null on delete restrict
* **T\_Tire**(*ProductID-Tire*,TireName, MileageWarranty, ReorderPoint)
  + ProductID-Tire references T\_Product null allowed on delete restrict
* **T\_Make(**MakeID, MakeName)
* **T\_CarModel**(ModelID, *MakeID*, ModelName, Year)
  + Foreign key MakeID reference T\_Make not null on delete restrict
* **T\_ModelTire**(CarTireID, *ModelID*, *TireID*)
  + Foreign key CarManID references T\_CarManufacturer not null on delete restrict
  + Foreign key TireID references T\_Tire not null on delete restrict
* **T\_Car**(CarID, *CarManID, CustID,* Make)
  + Foreign key CarManID references T\_CarManufacturer not null on delete restrict
  + Foreign key CustID references T\_Customer not null on delete restrict
* **T\_Employee**(EmpID, EFName, ELName)
* **T\_PurchaseOrder**(POID, *POEmpID*, *PayOutID*, *ManufacturerID*, PODate, POShipDate, InvoiceNo)
  + Foreign Key POEmpID references T\_Employee not null on delete restrict
  + Foreign key PayOutID references T\_Payment null allowed on delete set null
  + Foreign key ManufacturerID references T\_Manufacturer not null on delete restrict
* **T\_PurchaseOrderLine**(POLineID, *POID*, *TireID*, Quantity, Discount)
  + Foreign key POID references T\_PurchaseOrder not null on delete restrict
  + Foreign key TireID references T\_Tire not null on delete restrict
* **T\_ServiceOrder**(SOID, *PayInID*, *SOCustID* , *EmpID-Sells*, *EmpID-Performs*, *CarID*, *DiscountID,* SOPmtType, , CarMiles, DateDroppedOff, DropOffTime, ServiceTime, PickUpTime, IsFinancing)
  + Foreign key PayInID references T\_CustomerPayment null allowed on delete set null
  + Foreign key SOCustID references T\_Customer null allowed on delete set null
  + Foreign key EmpID-Performs references T\_Employee null allowed on delete set null
  + Foreign key EmpID-Sells references T\_Employee null allowed on delete set null
  + Foreign key CarID references T\_Car not null on delete restrict
  + Foreign key DiscountID references T\_Discount null allowed on delete set null
* **T\_ServiceOrderLine(**SOLID, *SOLProdID*, *SOID*, Quantity, SOLStatus)
  + Foreign key SOLProdID references T\_Product not null on delete restrict
  + Foreign key SOID references T\_ServiceOrder not null on delete restrict

## Differences between ERD and Normalized Relations

The difference between our ERD and normalized relations is that our ERD shows the internal and external of the company. Things that were listed in the ERD such as customers and the services the company provides and how the payment would be processed. Along with how the company keep track of their own purchases from which manufacturers. This also involves the employee and their positions within the company. As for normalized relations, as stated previously it shows that it’s trying to eliminate the redundancy and create atomicity. In Sonner Tire case, the normalization layout the ideas or guide for the company.

* The first big change that was made was creating T\_State and T\_Zip. These changes were made since an address can have multiple attributes. This way in the database Address will be atomic.
* The second big difference that was added was changing “Contact (…)” to MFName and MLName. This is because the contact at the manufacturer will have a first name and last name which needs to be addressed so that they are correctly separated in the database
* The final big difference in the normalized relation is the same as the prior difference except in the employee table. “Name(…)” is a multi-value attribute that needs to be treated as such. Because of this we included EFName and ELName.

## Database Integrity

Database integrity is used to make sure that when moving through the entire design of the database everything is consistent and accurate. What database integrity insures is that all of the associations between entities is consistent and correct, all of the entities in the database have the correct primary keys associated with it, and that all of the values in the database are in their proper spots.

* Referential Integrity is important as it will address the association amongst our entities in our databases. Referential constraints in our databases ensure that foreign keys in our entities properly relate to primary keys in the appropriate corresponding entity. Data in an entity with a foreign key can be affected by changing the data in the entity with the corresponding primary key. Referential constraints are needed for example, if we delete an entry in the primary key table, then we would have to plan on what to do with the data in the foreign table.
  + In conclusion, referential integrity is an important factor in defining the relationships amongst entities in our database. It also sets constraints that help protect against mistakes being made in the database when manipulation data in primary keys and their corresponding foreign keys.
* Entity Integrity is important because it will ensure that every single entity in our database will have a primary key property associated with it. This primary key cannot be null and also cannot change overtime. A primary key is important because it allows to uniquely identify every single piece of data in a row. We also assign these primary keys artificial identifiers rather than semantic identifiers in order to ensure that every single entry will be eligible for the primary key.
* Domain Integrity is important because it ensures that all values set in a column will be from the same “domain”. In other words, it makes sure that every piece of data in a column if it is “state” will only be states, or if it is “color” will only be colors. These constraints will force us to split any attributes to its proper components and to also ensure that we create atomic attributes or attributes that can no longer be split again. We have ensured to follow these constraints by making every one of our attributes atomic.

# **Physical Design and Implementation**

In this stage, we are physically designing the database in SQL and implementing the data from the data dictionary that is being created. This step is done after all of the prior steps have been completed. The reason for this is because at this point we need to be confident that all of the data is in the correct spots and that that all of the foreign key constraints are being met. Once we have ensured that our ERD is finished, and all of the normalization is complete, we are able to begin assigning values to the attributes in the different tables from the ERD and implementing those values into the SQL server to create the final product. In this stage, we will be including the data dictionary, denormalization of the previously normalized data, and the physical ERD within the server. For implementation we will be using Microsoft SQL Server. In Microsoft SQL Server we will be creating the tables and their columns by using code to implement the tables as well as the values that pertain to each table and column. An example of this would be with the TMake table that you can see in the ERD and in the Normalized Relations. We would create the table called TMake and then add the values, MakeID and MakeName through coding “Insert Into TMake...” The MakeID will be an auto-increment and automatically created with each new row, but the MakeName will be handmade. For a more detailed understanding of this please refer to the Data dictionary at the end of the document.

## Data Dictionary

A data dictionary is the bones of creating the database. This dictionary presents a baseline for all of the information that is within the database. This includes the Tables, their attributes, the datatype, whether or not it is allowed to be null, which tables the foreign keys reference, as well as a sample of the how the data shows up in the database. Below we have provided two examples of how we created the data dictionary for Make and TCarModel. As you can see the name of the table is on the far left followed by the name of the columns. In the 3rd column we have the data type. Int (auto-increment) is what we make each primary key value as this allows for the primary key to automatically increment with regards to each respective row. We used Varchar and the respective lengths for columns that will have data that is in words or most everything other than numbers, including dates. In the 4th column we have indicated whether the data can be null or not null in the database. What this means is whether or not the column is required to have data or not. In the 5th column we have the table the foreign keys reference. As you can see for TCarModel the MakeID references the Make table. In the final column on the right we have provided a sample of what the data will look like in the database.

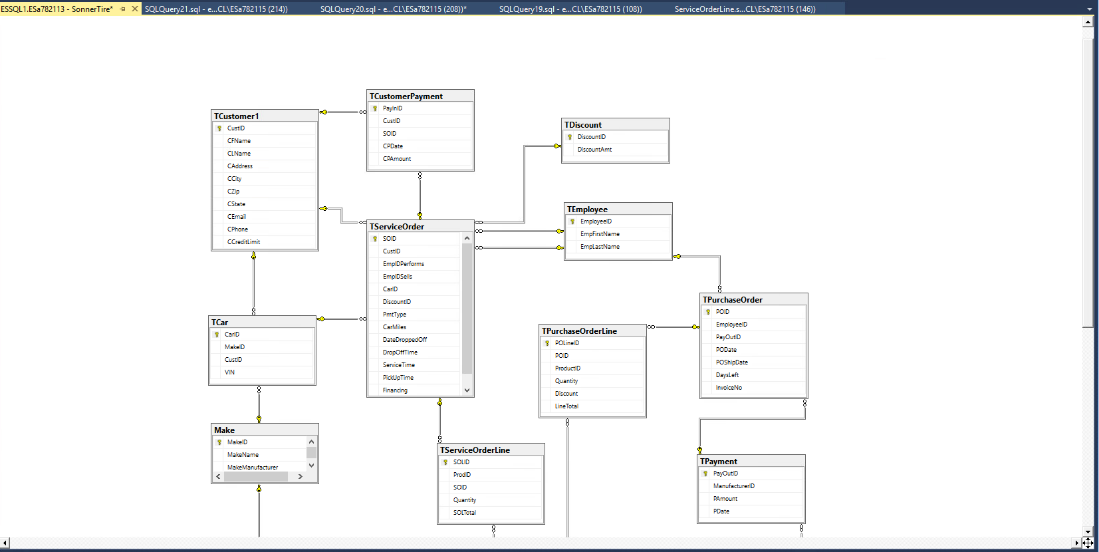
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Make | MakeID | Int(auto-increment) | Not Null |  | 1 |
|  | MakeName | Varchar(20) | Not Null |  | Ford |
|  | MakeManufacturer | Varchar(50) | Not Null |  | Ford Motor Company |
| TCarModel | ModelID | Int (auto-increment) | Not Null |  | 1000 |
|  | MakeID | int | Not Null | Make | 1 |
|  | MakeName | Varchar(20) | Not Null |  | Ford |
|  | Model | Varchar(20) | Not Null |  | 2019 |

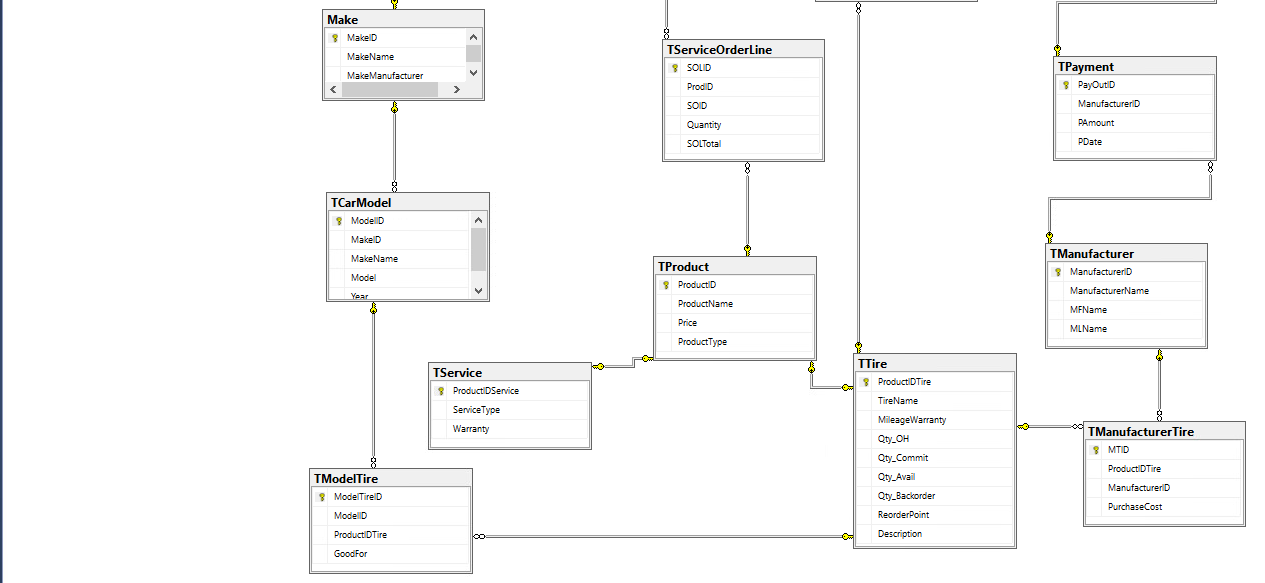
Denormalization

Denormalization is a process that can be used to increase performance as well as the overall efficiency of the database. What is done with denormalization is deleting redundant data that has already been previously normalized. In our database, we denormalized the ServiceOrder Table by deleting the OutOfServiceTime. The reason for this is because it can be implied from the DropOffTime, ServiceTime, and PickUpTime what the OutOfServiceTime is. Deleting this redundency will allow better efficiency with the database by deleting the easily implied time. On top of this being taken out we also had some aspects of our ERD that we realized at this point were actually derived attributes which we in turn returned to our ERD and removed. Examples of this are with the Qty\_Committed, and Qty\_Avail. During our initial analysis we believed that these were data points that were physical and not calculated but through further analysis we realized these were derived attributes that needed to be removed from our ERD and normalization to allow for them to be calculated.

# **Implemented Physical Design**

Below is a picture of what the ERD inside the database looks like. As you can see we stayed on track from our original ERD and the implemented design looks nearly identical. This is not an ERD that was randomly created but was the end result of all of our implementation of all of the data into the database. Unfortunately, the image is too big in the database, so we had to split it up into two images. To allow for you to see where the image is split and the relationships between the entities in the first and second pictures we included Make, TServiceOrderLine, and TPayment in both images.





## Challenges Faced/Addressed during Implementation

* The biggest challenge that we faced when creating the database was editing the formula in excel in order to transfer the data into the SQL database. The reason for this is because each table has different data and layouts which requires a bit of trial and error in order to make sure the formula accurately collects and transfers the data. In order to overcome this, we did just that, trial and error. It was at times frustrating, but in the end, it made transferring the data into the database much easier and cleaner.
* Another challenge that we faced was with the sample data provided by the client. After pairing all of the primary and foreign key values together we recognized that there was a tire made for 2014 Honda Civics but there was no 2014 Honda Civic included as a model. What we decided to do was drop this tire from the data set, so that going forward this does not cause errors in the database. If a customer does come into Sonner Tire with a 2014 Honda Civic it can still be added in though.

## Strengths and Weaknesses Encountered during Implementation

The implementation stage was by far the most difficult and time-consuming aspect of the project, and provided Team Waterfall with many difficulties, but we were able to work through these difficulties strategically to make them strengths later on in the project. The number one weakness for our team was getting all of the tables to work on the first try. The reason for this is with regards to the normalization that we did earlier on in the project. Each table that is created in the database needs to relatively follow the same path as the normalization in terms of which are created first. We tried at first to just divide and conquer, but what we realized is that we needed to strategically divide and conquer so that the members of the team who were responsible for tables that had foreign key constraints were creating their tables at a point when all the foreign keys had already been implemented in their respective tables. Our greatest strength was having our earlier stages of work fairly solidified in advance so that we were able to move into the implementation all at once without having to go back and continuously fix different aspects as we went along. This was the greatest strength because it allowed us to use our time more efficiently.

## Specific SQL Statements Requested

Below are 6 of the SQL codes as well as the answers in the output to the questions that were asked by Sonner Tire. In this portion, we inputted the SQL statement that meets the criteria of the question in order to present the output to the questions.

|  |  |  |  |
| --- | --- | --- | --- |
| **Query #** | **Question** | **SQL** | **Partial Output** |
| 1 | Total sales (in dollars) for a given tire manufacturer and car manufacturer. It would be great if we can specify the car model and year too(note that we would like to be able to input the month to be calculated).  \*Used month October(10) as example | SELECT ManufacturerName, MakeManufacturer, Model, Year, SUM(Price\*Quantity) AS TotalSales  FROM Make MA JOIN TCarModel CM ON MA.MakeID = CM.MakeID JOIN TModelTire MOT ON CM.ModelID = MOT.ModelID JOIN TProduct P ON MOT.ProductIDTire = P.ProductID JOIN TManufacturerTire MT ON P.ProductID = MT.ProductIDTire JOIN TManufacturer M ON MT.ManufacturerID = M.ManufacturerID JOIN TPayment PM ON M.ManufacturerID = PM.ManufacturerID JOIN TPurchaseOrder PO ON PM.PayOutID = PO.PayOutID JOIN TPurchaseOrderLine POL ON PO.POID = POL.POID WHERE Month(PM.PDate) = 10  GROUP BY ManufacturerName, MakeManufacturer, Model, Year | Inserting image... |
| 2 | Total sales (in dollars) by customer in a given year.  \*Used year 2020 as example | SELECT CFName, CLName, Sum (quantity\* price) TotalSales, CP.CPDate  FROM TCustomerPayment CP JOIN TCustomer1 CR ON CP.CustID=CR.CustID  JOIN TCar C ON CR.CustID=C.CustID  JOIN TServiceOrder SO ON C.CustID=SO.CustID  JOIN TServiceOrderLine SOL ON SO.SOID=SOL.SOID  JOIN TProduct P ON SOL.ProdID=P.ProductID  WHERE YEAR(CPDate) = 2020  GROUP BY CFName, CLName, CP.CPDate |  |
| 3 | The five highest selling tires. | SELECT TOP 5 (Count(SOLID)) TimesSold, ProductID, ProductName  FROM TProduct P  JOIN TServiceOrderLine SOL ON P.ProductID=SOL.ProdID  GROUP BY ProductID, ProductName  ORDER BY TimesSold desc |  |
| 4 | Itemized invoices for jobs for each customer that need to include tires purchased/tire rotation/tire repair/tire protection.  \*Used CustID=1040  CarID=141 & DateDroppedOff = ‘2020-05-24' as example | SELECT CR.CustID, SOL.SOLID, C.CarID, DateDroppedOff, Quantity, ProdID, ProductType, ProductName, P.Price, (Quantity\*P.Price) LineTotal  FROM TCustomer1 CR  JOIN TCar C ON CR.CustID=C.CustID  JOIN TServiceOrder SO ON C.CarID=SO.CarID  JOIN TServiceOrderLine SOL ON SO.SOID=SOL.SOID  JOIN TProduct P ON SOL.ProdID=P.ProductID  WHERE CR.CustID=1040  AND C.CarID=141  AND Date=’2020-05-24' |  |
| 5 | The number and type of job performed by each of our employees. | SELECT SOID, (A.EmpFirstName + ‘ ‘ + A.EmpLastName) AS PerformedBy, (B.EmpFirstName + ‘ ‘ + B.EmpLastName) AS SoldBy  FROM TServiceOrder SO JOIN TEmployee A ON SO.EmpIDPerforms = A.EmployeeID JOIN TEmployee B ON SO.EmpIDSells = B.EmployeeID |  |
| 6 | Number of times a tire protection has been purchased for a particular tire. Only report the number of times tire protection has been purchased for a particular tire. | SELECT ProductID, ProductName, Count(ProductID) NoPurchases  FROM TServiceOrderLine SOL  JOIN TProduct P ON SOL.ProdID=P.ProductID  JOIN TService S ON P.ProductID=S.ProductIDService  WHERE ServiceType = 'Protection'  GROUP BY P.ProductName, P.ProductID |  |
| 7 | The following items for Purchase Orders: manufacturer name, number of POs, total cost. | SELECT M.ManufacturerID, manufacturername, COUNT(POL.POID) NumberOfPurchaseOrders, ((Quantity \* PAmount) + Discount) TotalCost  FROM TPurchaseOrderLine POL JOIN TPurchaseOrder PO ON POL.POID=PO.POID  JOIN TPayment P ON PO.PayOutID=P.PayOutID  JOIN TManufacturer M ON P.manufacturerID=M.manufacturerID  GROUP BY M.ManufacturerID, ManufacturerName, Quantity, PAmount, Discount |  |
| 8 | Number of orders and total sales per customer in the past 2 years. This report is particularly important as it shows the number of returning customers. | SELECT CFName,CLName, Count(SO.SOID) SOID, SUM(CP.CPAmount) TotalSales  FROM TCustomerPayment CP  JOIN TCustomer1 CR ON CP.CustID=CR.CustID  JOIN TCar C ON CR.CustID=C.CustID  JOIN TServiceOrder SO ON C.CarID=SO.CarID  WHERE DateDroppedOff>= '2019-05-01'  GROUP BY CFName, CLName |  |
| 9 | List of tires that have not been purchased within the last 6 months (in order to better manage inventory). | SELECT ProductID, ProductName  FROM TServiceOrderLine SOL LEFT JOIN TProduct P ON P.ProductID=SOL.ProdID  WHERE ProductType='Tire'  AND SOL.ProdID IS NULL | \*Column results were empty, every tire has been purchased in the past 6 months |
| 10 | Names of customers who took advantage of the financing option, date purchased, total amount purchased, credit limit, total amount paid, outstanding amount, is time less than 6 months, all displayed from latest date and then largest amount owed.  \*Saved as a procedure  named “Query#10” | SELECT CFName, CLName, Total, Paid, (Total -Paid) Remaining, CPDate  FROM  (SELECT C.CustID, SO.SOID, Sum(quantity\*price) Total  FROM TCar C  JOIN TServiceOrder SO ON C.carID=SO.CarID  JOIN TServiceOrderLine SOL ON SO.SOID=SOL.SOID  JOIN TProduct P ON SOL.ProdID=P.ProductID  GROUP BY C.CustID, SO.SOID) S1  JOIN  (SELECT CP.CustID, CFName, CLName, sum(CPAmount) Paid, SO.SOID  FROM TCustomer1 CR  JOIN TCar C ON CR.CustID=C.CustID  JOIN TServiceOrder SO ON C.CarID=SO.CarID  JOIN TCustomerPayment CP ON SO.SOID=CP.SOID  GROUP BY CP.CustID, CFName, CLName, SO.SOID) S2 ON S1.SOID=S2.SOID  JOIN TServiceOrder SO ON S1.SOID=SO.SOID  WHERE SO.Financing = 'Yes'  ORDER BY CP.CPDate desc, CPAmount desc |  |
| 11 | Total profit per tire type and manufacturer type in the past 6 months.  \*We changed this question to the last year instead because the last 6 months produced no results, meaning there were no tires sold during that period. | SELECT ProductID, TireName, ManufacturerName, SUM((Price-PurchaseCost)\*Quantity) Profit  FROM TServiceOrder SO  JOIN TServiceOrderLine SOL ON SO.SOID=SOL.SOID  JOIN TProduct P ON SOL.ProdID=P.ProductID  JOIN TTire T ON P.ProductID=T.ProductIDTire  JOIN TManufacturerTire MT ON T.ProductIDTire=MT.ProductIDTire  JOIN TManufacturer M ON MT.ManufacturerID=M.ManufacturerID  WHERE DateDroppedOff >= '2020-05-01'  GROUP BY ProductID, ManufacturerName, TireName |  |
| 12 | List of all customers that have not made a purchase within the last 12months from the current date. | SELECT CR.CustID, CFName, CLName  FROM TCustomer1 CR JOIN TCar C ON CR.CustID=C.CustID  LEFT JOIN TServiceOrder SO ON C.CarID=SO.CarID |  |
| 13 | List of customers whose average sales is less than the average of all sales. This will help us to find customers whom we should target to get a higher volume of sales. | SELECT C.FName, C.CLName FROM Tcustomerpayment CP  JOIN TCustomer1 C ON CP.custID=C.CustID  JOIN Tserviceorder SO ON C.custID=SO.custID  JOIN serviceorderline SOL ON SOL.SOID=SOL.SOID  GROUP BY C.FName, C.CLName  HAVING AVG(SOL.soltotal) <  (SELECT AVG(soltotal)  FROM serviceorderline) | Table  Description automatically generated |

## Three Additional Queries

Below are three additional queries beyond the six questions that were asked above. These are queries that we believe are important because they can be used to retrieve important information about Sonner Tire’s business. These queries allow for better insight as to what is included in the database and how to retrieve this data.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Query # | Question | Why is this important | SQL | Partial Output | Recap of Findings |
| 1 | What is the average sale from all customers. | This is important because it allows them to know how much they expect from each customer can help Sonner Tire with financial planning. | SELECT AVG(CPAmount) AvgAmt  FROM TCustomerPayment |  | The average sale amount from all customers is $303.64 |
| 2 | Are there any tires that are at the reorder point? | This is important because it can allow Sonner tire to know what tires are running low on stock | Select ProductIDTire  From TTire  Where Qty\_Avail <= 16 |  | 5 tires, tireID 1,10,24,51,60 all have 16 or less tires available currently which means they should be reordered |
| 3 | Which customers have made a purchase at least 3 times in the past year? | This tells Sonner Tire who their loyal customers are so that they can encourage them to return. | SELECT distinct CFName, CLName, COUNT(SOLID) NoPurchases  FROM TCustomer1 CR  JOIN TCustomerPayment CP ON CR.CustID=CP.CustID  JOIN TServiceOrder SO ON CR.CustID=SO.CustID  JOIN TServiceOrderLine SOL ON SO.SOID=SOL.SOID  WHERE CP.CPDate <= '2020-04-27'  GROUP BY CFName, CLName, SOLID  HAVING COUNT(SOLID)>=3 |  | There are 6 loyal customers that Sonner could reach out to in order to show their appreciation for their continued business. If they wanted to, they could start a loyalty program for these customers. |

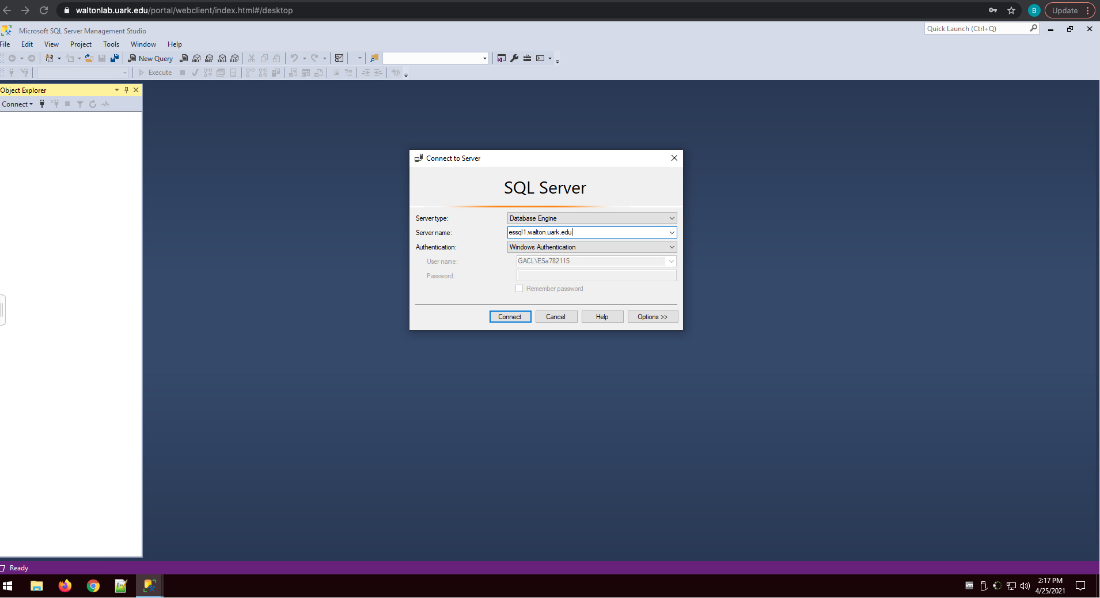
# **User Documentation**

Below is a detailed step by step guide for how to access and use the database. This will be the go-to guide moving forward for how to use the database on a daily basis.

## Opening the Database

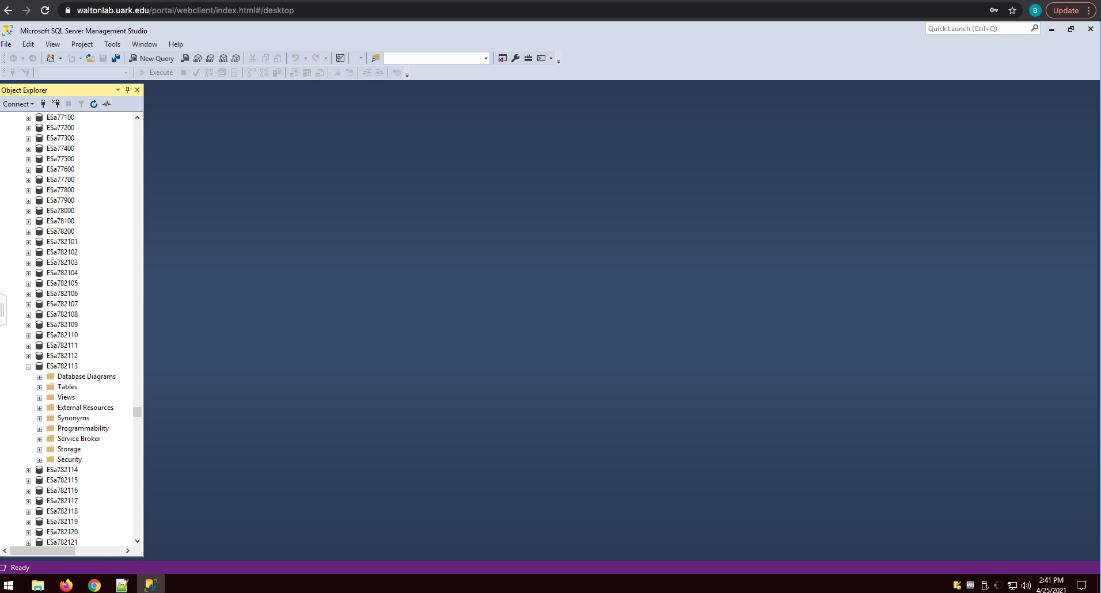
Step 1:

* Open the Microsoft SQL Server Management Studio
* Type the Server Name in (essql1.walton.uark.edu)
* Click on Connect



Step 2:

* In the object Explorer click on the + sign next databases
* Scroll down to the specific database (ours is ESa782113)
* Click on the + sign next to the specific database in to open up the contents



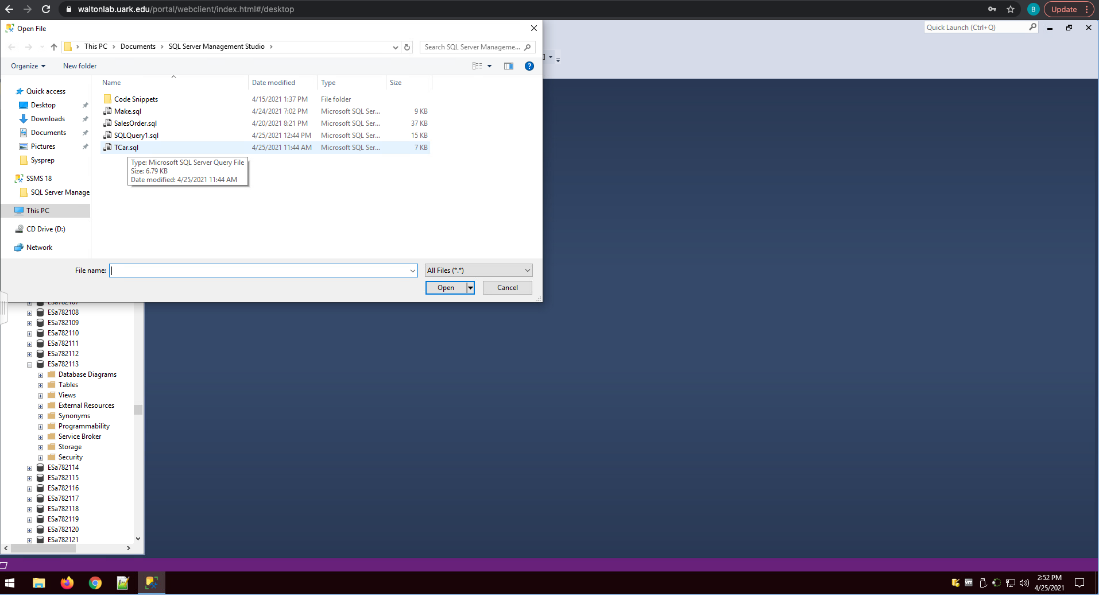
Step 3:

* From here you can click on the + next to the tables to see all of the different tables within the database

## How to Enter New Data to the Existing Tables

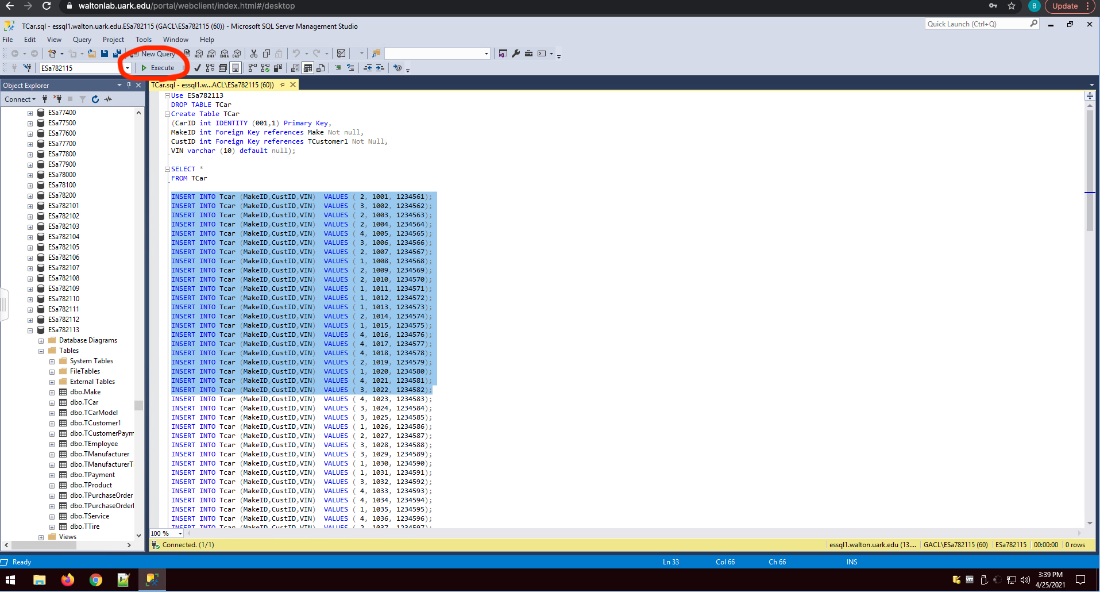
Step 1:

* First, you need to click on open file (ctrl+O)
* Next you click on the file with the table that you want to edit.



Step 2:

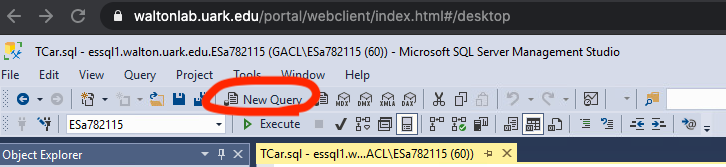
* Now that you have your table open you are ready to begin adding/editing.
* To add new data:
  + Type “Insert into” Then type the name of the table you wish to add to
  + Next type an open parenthesis and enter the column names
  + Close the parenthesis
  + Now type “VALUES” and add an open parenthesis after
  + After the open parenthesis you can type the values you wish to add (numbers do not require apostrophes before and after the number, but words do).
  + Once you are finished adding the values finish by closing the parentheses and adding “;”.
  + Now that you have finished typing in the data you must highlight with your cursor the new statement and then press execute.



## How to Create a New Table

Step 1:

* There are multiple ways to do this, but the way that we are going to do this is by first clicking on New Query



Step 2:

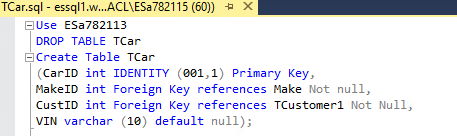
* Now that the new query is open, to be safe and make sure that you are using your specific database, type in “Use Esa782113”.

Step 3:

* Next, what you need to do is type in “Create Table” and then type the table name that you wish to create and type an open parenthesis.
* From there, you type in the primary key as “int Identity (starting number, increment) Primary Key”.
* After this, you begin entering your column names and their types (i.e. int, varchar, etc)
  + Int is for non-decimal numbers, and varchar is for words.
  + You must also include the maximum characters. (i.e. (10), (50), etc.)
* If the column is a foreign key, then you must state this after the type (Refer to image).
* After the column names are entered you must state whether they are null or not.
* Finally, once all the column names are inputted correctly, you must close the parentheses and add “;”

Step 4:

* Highlight the new table and information you have typed, and press execute.



## How to Pull Information from a Table

Step 1:

* In any of the queries that you have opened you first need to start your SELECT statement by typing “SELECT”
  + In this SELECT statement you are typing in either the columns that you wish to pull, or you can type “\*” to pull all of the information from the table.

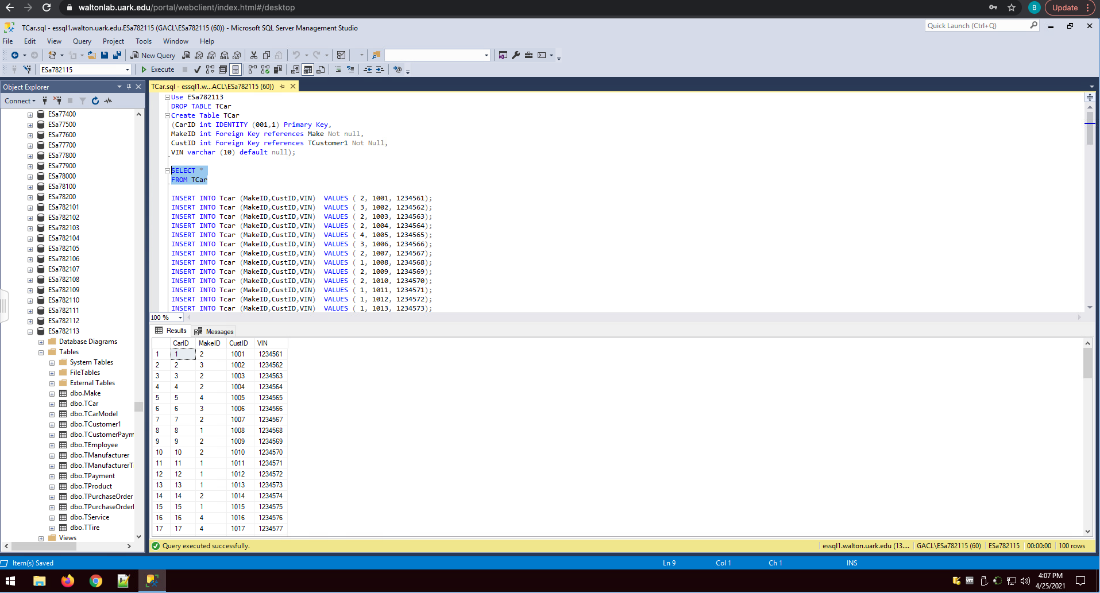
Step 2:

* Next you must type your FROM statement, and again, to do this you type “FROM” below the SELECT
  + In this FROM statement you are typing in the name of the table that you wish to pull your information from.



Step 3:

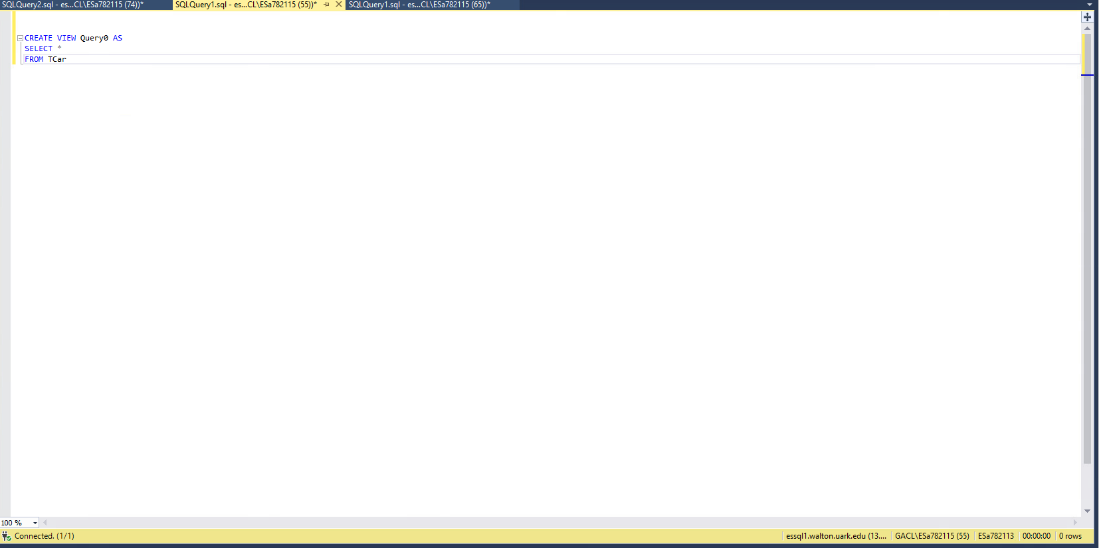
* Highlight the statement with your cursor and press execute. This will then show all the information from your table. (This in particular will show all of the data from the TCar table)



How to Run Views

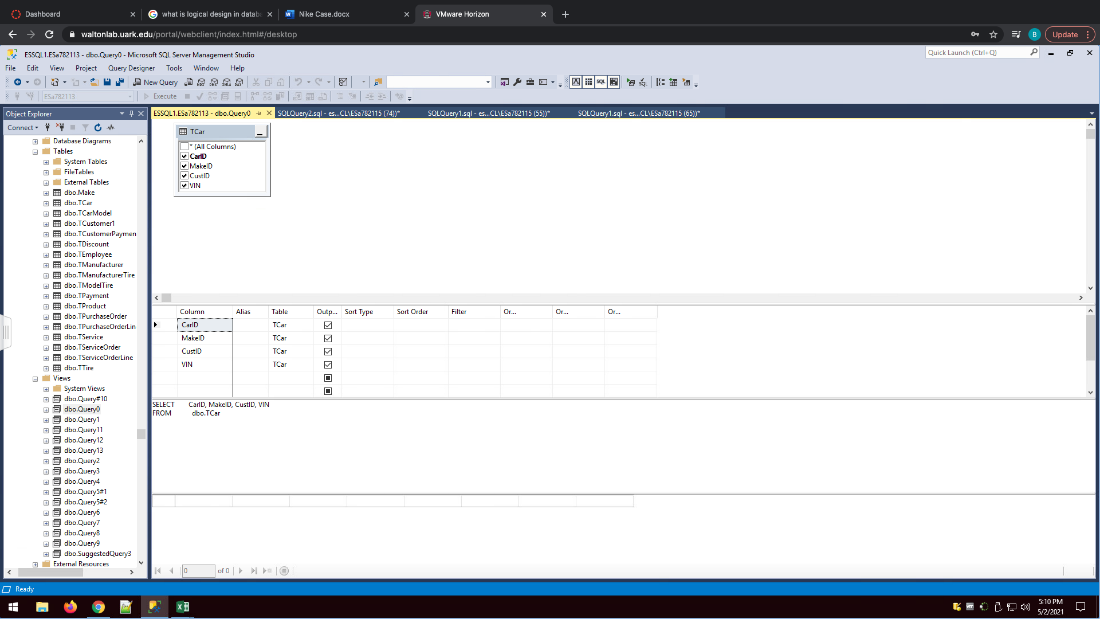
Step 1:

* The first thing that we must do to run views and stored procedures is to create a new query and then type the statement. For simplicity, the example that we have used for creating a view is “SELECT \* FROM TCar” This will give us all of the data from the car table.
* Once we have typed out our statement we need to type at the top “CREATE VIEW query# AS”
* Once we have done this we can highlight what we have typed and press execute.



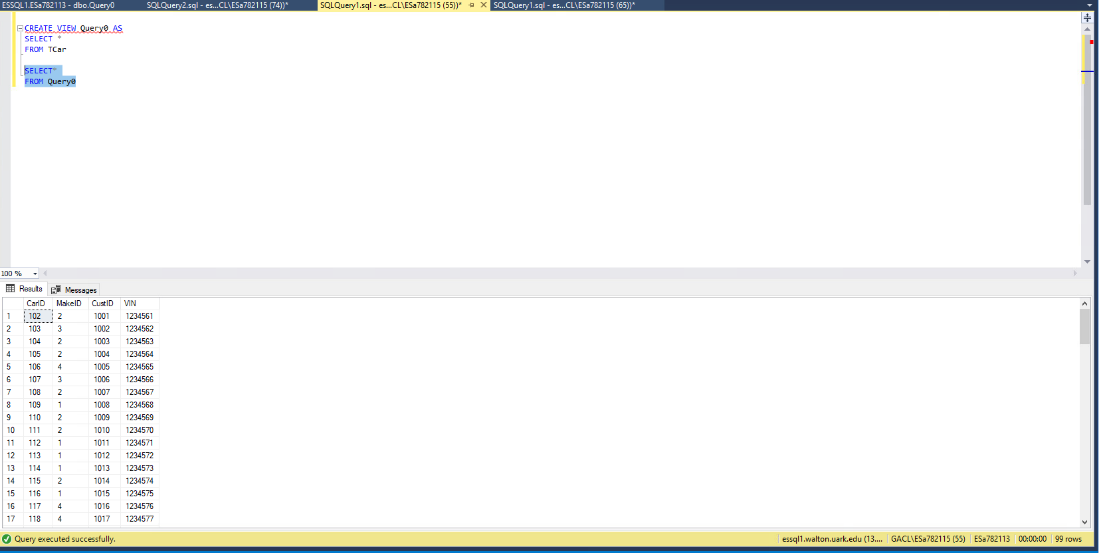
Step 2:

* Now that we have created our view we can see the results by navigating over to the object explorer, right clicking on views and pressing refresh.
* We press on the + next to views to open all of our views.
* From here we can right click on the view that we want to see and click on design to open and see the tables associated with the query as well as the query itself. (We can also edit the query in this spot)



Step 3:

* Now that we have our view created it is simple to run. All that we need to do is write “SELECT \* FROM Query0” highlight it with our cursor and press execute.



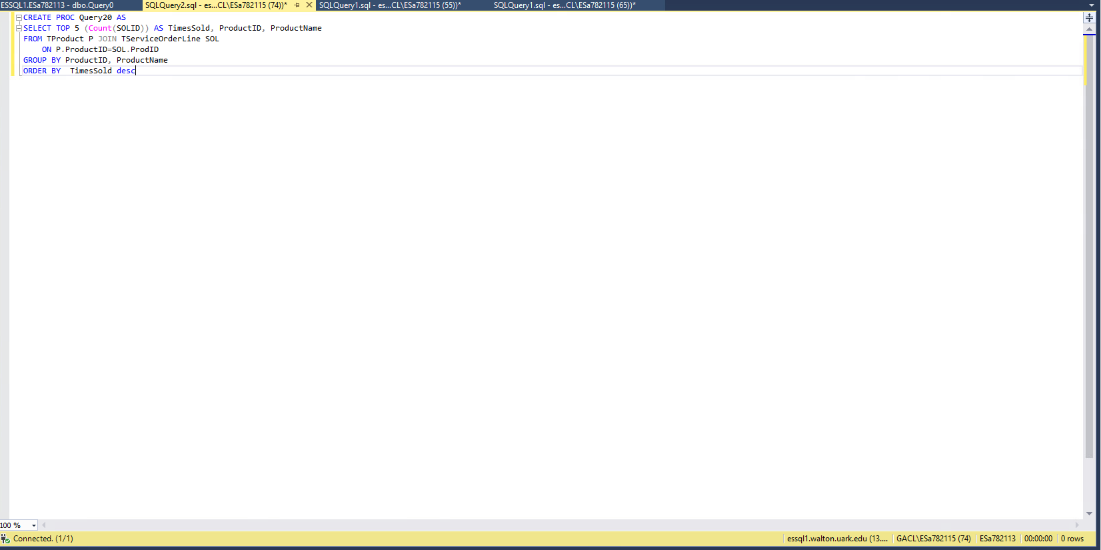
How to Store Procedures

Step 1:

* First, we must understand the difference between procedures and views. We use views for simple queries, but we use procedures for more complex queries that have an ORDER BY clause.
* To start we do the same thing from above: open a new query

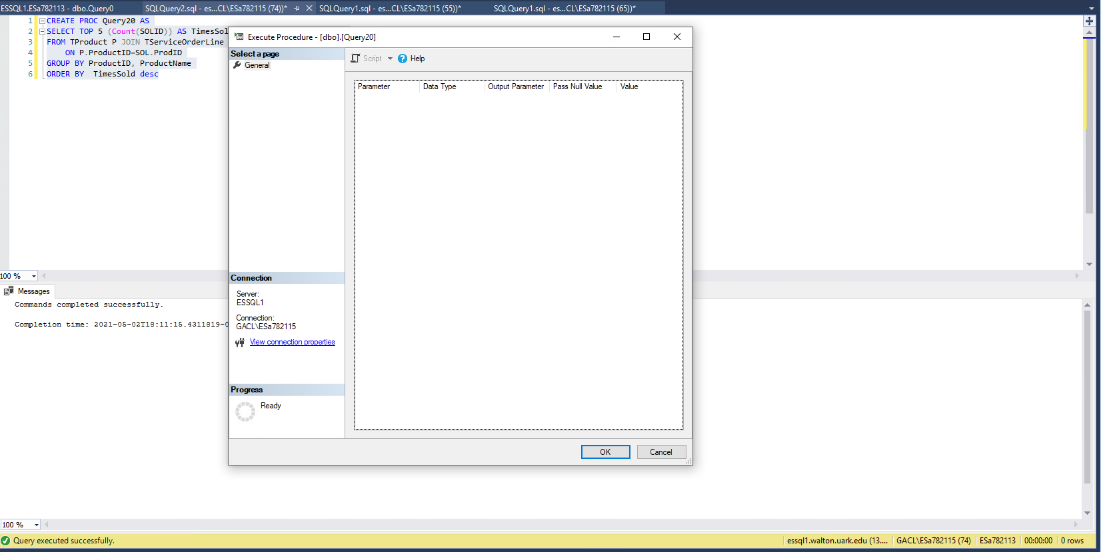
Step 2:

* The second step is to write our query and then at the top type “CREATE PROC Query# AS”



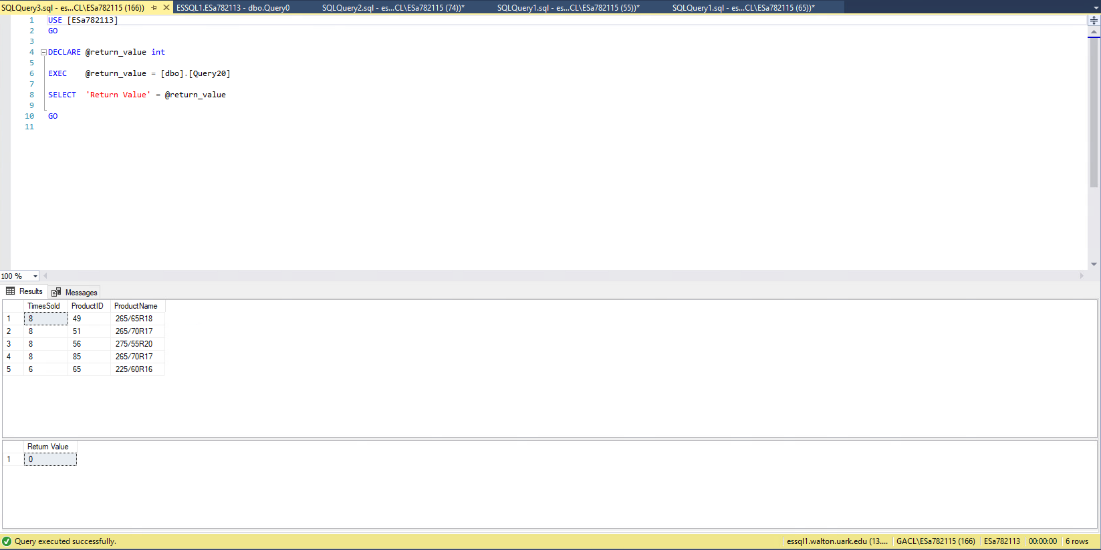
STEP 3:

* Highlight the query with your cursor and press execute.
* Now go over to object explorer click on the + next to Programmability click on the + next to stored procedures and you will see the query that has been created.
* Once you have found it right click on the stored procedure and press execute stored procedure.



STEP 4:

* Press OK and the results will then be displayed.



## What We Learned Throughout this Process

|  |  |
| --- | --- |
| **Member Name:** | **What you learned:** |
| Ben Friedman | I learned many valuable lessons throughout this entire process. First, I learned that you cannot just jump into creating a database no matter how bad you want to you must follow the process from start to finish to ensure that your database meets all of the foreign constraints and does exactly what it is supposed to do. It was quite intriguing to see how a database like this would actually work in the real world, and it sheds light on the real-world purpose. Besides the knowledge, I learned how to work with a team. Although there were times when it was quite tough, we were able to get through obstacles by bouncing ideas off each other. In the grand scheme of things, I learned that to get through abstract projects sometimes it takes thinking outside of the box to get it done. |
| Jennifer Ha | I learned that we don’t know much about the behind the scenes when it comes to organizing for a company. Each step was connected to the other and without that certain step the plan could fall apart. Along with how one small piece as we continue with the project it was only going to become a bigger piece of the puzzle. Another lesson I learn with is time management. I felt like at times we struggle a lot of keeping track of time and being able to have everyone do their part. It was an environment that involved with a lot of growth in what a team can truly be. Especially with how we come together to management our time. Overall, I learned that in database has a lot of steps and each step is valuable best to be handle well and not overlook them. |
| Cody Glover | I learned through this process just how deep creating a database can be for a client. The experience showed me the importance of all three steps of the process from conceptual to logical to physical and how one cannot be skipped over for timesake. It gave us experience and knowledge of what would truly need to be known when creating a database and the purpose of the asking important questions during the interview phase. Ultimately, I learned that a database cannot be created overnight, and steps cannot be skipped in order to make a database that will truly suit the client’s needs. |
| Anna Dowhower | This project has taught me a lot about just how complicated databases are. You really don't realize how important each little step is until you run into problems down the road and have to go back and fix things. From this, I have learned that it is important to be diligent and rethink every decision you make and how it might matter in the future. It is also important to have all of your data organized so that you know what you will be working with and what attributes you may or may not need. Additionally, completing this project through an online setting was especially difficult, as communication with the team is crucial and is harder to accomplish when not face-to-face. Overall, my appreciation for databases and those who create them has grown. It's not a simple task. |
| Alessio Hall | Throughout this entire process, I learned just how incredibly tedious it is to create a real-life database successfully and thoroughly for an actual business. I look back at how much time was spent on each section by each group member and am astounded. I cannot imagine what kind of effort and manpower it would take to create a database like this for a bigger, more well-known company. It was eye-opening to see how actual businesses set up their databases and retrieve information through the queries. The biggest aspect of databases that I learned, though, was just how much time needs to be spent going through the details with a fine-tooth comb so that there are no problems when implementing the database or trying to retrieve information. |

**Team Function and Insight**

We as a team would coordinate our time by splitting our tasks amongst the group. Each individual is in charge of a section of the project, and we would communicate through messaging via iMessage. We would discuss about possible questions we have about our part through text and along with notifying the group on which section was finished. When sections were finished other members of the group would review and make necessary edits. What we learned as a team is that it can be difficult to get everyone to be at the same pace with everyone having different schedules and that is best to have a designated date to work on the project. This would allow us as a team to respond to each other quickly and efficiently.

# **Appendix**

## Data Dictionary

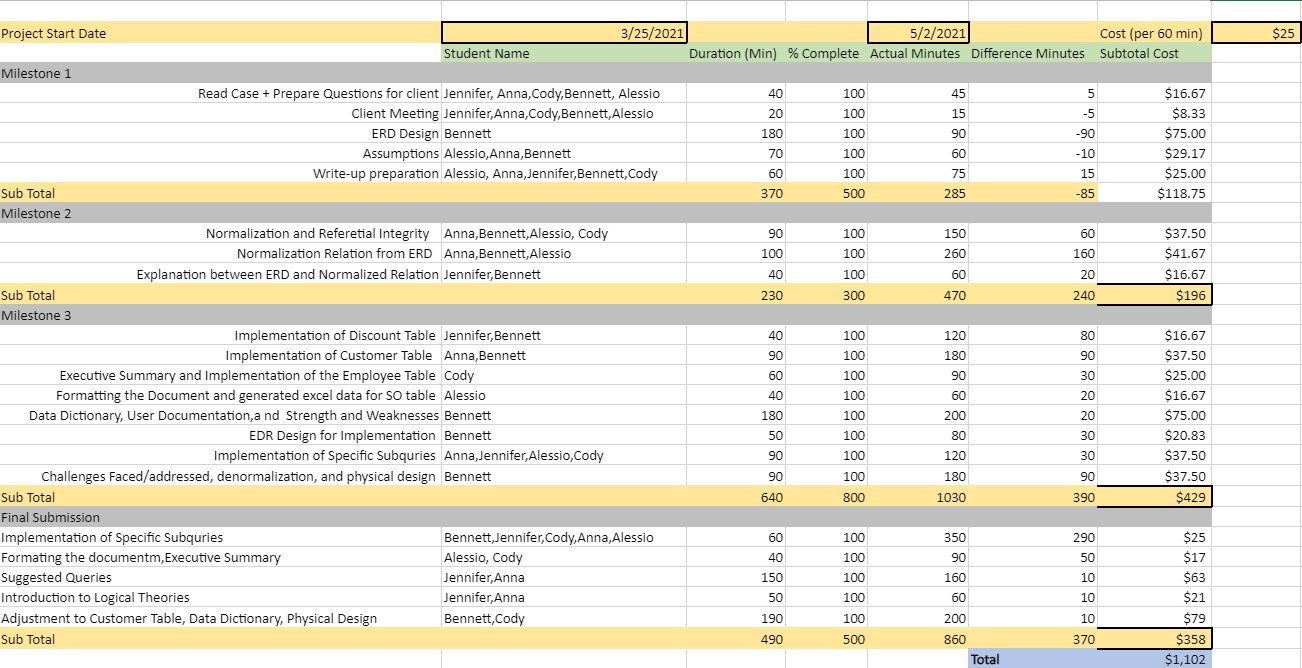
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Table** | **FieldName** | **DataType** | **Null/Not Null** | **References** | **Sample** |
| TCustomer1 | CustID | Int (auto-increment) | Not Null |  | 1000 |
|  | CFName | Varchar(50) | Not Null |  | Ursula |
|  | CLName | Varchar(50) | Not Null |  | Shepherd |
|  | CAddress | Varchar(100) | Not Null |  | Vel Street |
|  | CCity | Varchar(50) | Not Null |  | RockVille |
|  | CZip | Varchar(10) | Not Null |  | 48949 |
|  | CState | Varchar(2) | Not Null |  | MD |
|  | CEmail | Varchar(50) | Not Null |  | Semper@metus.edu |
|  | CPhone | Varchar(14) | Not Null |  | (751)-624-2482 |
|  | CCreditLimit | Varchar(10) | Not Null |  | $1,118 |
| Make | MakeID | Int(auto-increment) | Not Null |  | 1 |
|  | MakeName | Varchar(20) | Not Null |  | Ford |
|  | MakeManufacturer | Varchar(50) | Not Null |  | Ford Motor Company |
| TCarModel | ModelID | Int (auto-increment) | Not Null |  | 1000 |
|  | MakeID | int | Not Null | Make | 1 |
|  | MakeName | Varchar(20) | Not Null |  | Ford |
|  | Model | Varchar(20) | Not Null |  | 2019 |
| TCar | CarID | Int(auto-increment) | Not Null |  | 1 |
|  | MakeID | int | Not Null | Make | 1 |
|  | CustID | int | Not Null | TCustomer1 | 1000 |
|  | Vin | Varchar(10) | Not Null |  | 1234561 |
| **Table** | **FieldName** | **DataType** | **Null/Not Null** | **References** | **Sample** |
| TEmployee | EmployeeID | Int(auto-increment) | Not Null |  | 1 |
|  | EmpFirstName | Varchar(20) | NotNull |  | Cody |
|  | EmpLastName | VarChar(20) | Not Null |  | Glover |
| TDiscount | DiscountID | Int(auto-increment) | Not Null |  | 1001 |
|  | DiscountAmt | Varchar(10) | Null |  | 15.00 |
| TServiceOrder | SOID | Int(auto-Increment) | Not Null |  | 1231 |
|  | CustID | int | Not Null | TCustomer1 | 1000 |
|  | EmpIDPerforms | int | Not Null | TEmployee | 1 |
|  | EmpIDSells | int | Not Null | TEmployee | 1 |
|  | CarID | int | Not Null | TCar | 100 |
|  | DiscountID | Int | Not Null | TDiscount | 1000 |
|  | PmtType | Varchar(10) | Not Null |  | Card |
|  | CarMiles | int | Not Null |  | 120000 |
|  | DateDroppedOff | Varchar(20) | Not Null |  | 2020-05-24 |
|  | DropOffTime | Varchar(20) | Not Null |  | 08:30:32 AM |
|  | ServiceTime | Decimal(3,2) | Not Null |  | 0.37 |
|  | PickUpTime | Varchar(20) | Null |  | 10:45:25 AM |
|  | Financing | Varchar(5) | Null |  | Yes |
| TModelTire | ModelTireID | Int(auto-increment) | Not Null |  | 12341 |
|  | ModelID | int | Not Null | TCarModel | 1000 |
|  | ProductIDTire | int | Not Null | TTire | 1 |
|  | GoodFor | Varchar(100) | Not Null |  | Chevrolet Impala 2014 |
| TCustomerPayment | PayInID | Int(auto-increment) | Not Null |  | 1231 |
|  | CustID | int | Not Null | TCustomer1 | 1000 |
|  | SOID | int | Not Null | TServiceOrder | 1231 |
|  | CPDate | Varchar(20) | Null |  | 2020-05-24 |
|  | CPAmount | Money | Null |  | $100.65 |
| **Table** | **FieldName** | **DataType** | **Null/NotNull** | **References** | **Sample** |
| TManufacturer | ManufacturerID | Int(auto-increment) | Not Null |  | 1 |
|  | ManufacturerName | Varchar(50) | Not Null |  | Premier A/S |
|  | MFName | Varchar(50) | Not Null |  | Dorothy |
|  | MFName | Varchar(50) | Not Null |  | Stout |
| TService | ProductIDService | int | Not Null | TProduct | 1 |
|  | ServiceType | Varchar(50) | Not Null |  | Rotate |
|  | Warranty | Varchar(4) | Not Null |  | Yes |
| TProduct | ProductID | Int(auto-increment) | Not Null |  | 1 |
|  | ProductName | Varchar(50) | Not Null |  | 265/40R21 |
|  | Price | Money | Not Null |  | 174.99 |
|  | ProductType | Varchar(10) | Not Null |  | Tire |
| TTire | ProductIDTire | Int(auto-increment) | Not Null |  | 1 |
|  | TireName | Varchar(50) | Not Null |  | 265/40R21 |
|  | MileageWarranty | Varchar(5) | Not Null |  | 60000 |
|  | Qty\_OH | int | Not Null |  | 17 |
|  | Qty\_Commit | int | Not Null |  | 4 |
|  | Qty\_Avail | int | Not Null |  | 16 |
|  | Qty\_Backorder | int | Not Null |  | 4 |
|  | ReorderPoint | int | Not Null |  | 16 |
|  | Description | Varchar(300) | Not Null |  | Powerful Grip... |
| TServiceOrderLine | SOLID | Int(auto-increment) | Not Null |  | 1234561 |
|  | ProdID | int | Null | TProduct | 1 |
|  | SOID | int | Null | TServiceOrder | 1231 |
|  | Quantity | int | Not Null |  | 4 |
|  | SOLTotal | money | Not Null |  | 607.96 |
| TManufacturerTire | MTID | Int(auto-increment) | Not Null |  | 207 |
|  | ProductIDTire | int | Not Null | TTire | 1 |
|  | ManufacturerID | int | Not Null | TManufacturer | 1 |
|  | PurchaseCost | Money | Not Null |  | 174.99 |
| **Table** | **FieldName** | **DataType** | **Null/NotNull** | **References** | **Sample** |
| TPayment | PayOutID | Int(auto-increment) | Not Null |  | 1 |
|  | ManufacturerID | int | Not Null | TManufacturer | 1 |
|  | PAmount | money | Not Null |  | 96.90 |
|  | PDate | Date | Not Null |  | 2020-10-06 |
| TPurchaseOrderLine | POLineID | Int(auto-increment) | Not Null |  | 108 |
|  | POID | int | Not Null | TPurchaseOrder | 196 |
|  | ProductID | int | Not Null | TProduct | 1 |
|  | Quantity | int | Not Null |  | 89 |
|  | Discount | Decimal(18,0) | Null |  | -17 |
|  | LineTotal | money | Not Null |  | 14934.31 |
| TPurchaseOrder | POID | Int(auto-increment) | Not Null |  | 196 |
|  | EmployeeID | int | Not Null | TEmployee | 1 |
|  | PayOutID | int | Null | TPayment | 1 |
|  | PODate | int | Not Null |  | 2021-02-23 |
|  | POShipDate | date | Null |  | 2021-03-14 |
|  | DaysLeft | int | Null |  | 19 |
|  | InvoiceNo | int | Null |  | 3353 |

This is the data Dictionary for our provided Database for Sonner Tire. In this data dictionary it is first sorted by table name for each specific table and then it will be sorted by FieldName (ColumnName) for each corresponding table. In each row we have provided the datatype used for each specific FieldName and if the column allows nulls. We also provided what references a specific column has in relation to another table (ForeignKeys). Finally, we give a sample of what results you can expect from each FieldName when pulling data from it, whether it be money, names, phone number, etc.

We chose each DataType for a column according to what was necessary and being used for that specific column. For example if we are using a state we would use varchar(2) this will return various characters and up to 2 of them since each states initials will only be two characters, something like address however can be up to 100 since an address can be very long and take up many characters. This is the theme throughout, we use int typically when we are providing a solid number and money when we are dealing with money related data.

## 

## Project Management



# **Get to Know Team Waterfall**

|  |  |
| --- | --- |
| Anna Dowhower | Anna is currently a senior at the University of Oklahoma studying MIS. She was raised in Shawnee, Oklahoma and is part of the MIS Student Association. |
| Jennifer Ha | Jennifer is born and raised in Oklahoma City, Oklahoma and is currently a sophomore at the University of Oklahoma majoring in MIS and minoring in marketing. |
| Alessio Hall | Alessio is born and raised in Oklahoma City, Oklahoma and is an undergraduate Senior at the University of Oklahoma pursuing a degree in Business Economics and an MIS minor. |
| Bennett Friedman | Bennett is from Woodlands, Texas and is currently a senior at the University of Oklahoma set to graduate this May with a major in Finance and a minor in MIS. |
| Cody Glover | Cody is from Moore, Oklahoma and is a junior at the University of Oklahoma pursuing an MIS degree |

# **Team Contract**

The following is a list of criteria that the team will use in post-project evaluation of each team member, including expectations for behavior of each team member as well as behavior that will result in point deductions. Additionally, we have attached our original Team Contract and provided our signatures electronically.

## Expectations

For each member, the Team expects Participation within the group, communication with group members, accomplishing assigned work, and contributing as well as being present within team meetings.

## Negative Behavior

The behavior for which points will be deducted on the confidential peer evaluation are as follows: lack of contribution to the group or group ideas, lack of communicated potential issues or necessary to each other, lack of finishing work in a timely manner or at all, missing team meetings unapproved or simply not contributing to the meetings, any form of disrespectful or general negative behavior toward another group member will result in points being deducted.

Alessio Hall

Alessio Hall

Bennett Friedman

Bennett Friedman

Jennifer Ha

Jennifer Ha

Anna Dowhower

Anna Dowhower

Cody Glover

Cody Glover

