Kali Curtis, Drake Detar, Dominick Demas, Jackson Dedrick, Jonathan Domingo
University of Oklahoma | Price College of Business

TLDR;

As a part of a group project in a MIS class at OU, we developed a fully functioning database for a theoretical tire company called Sonner Tires. Below is the ERD that we created to give you an idea of how our physical database is structured.

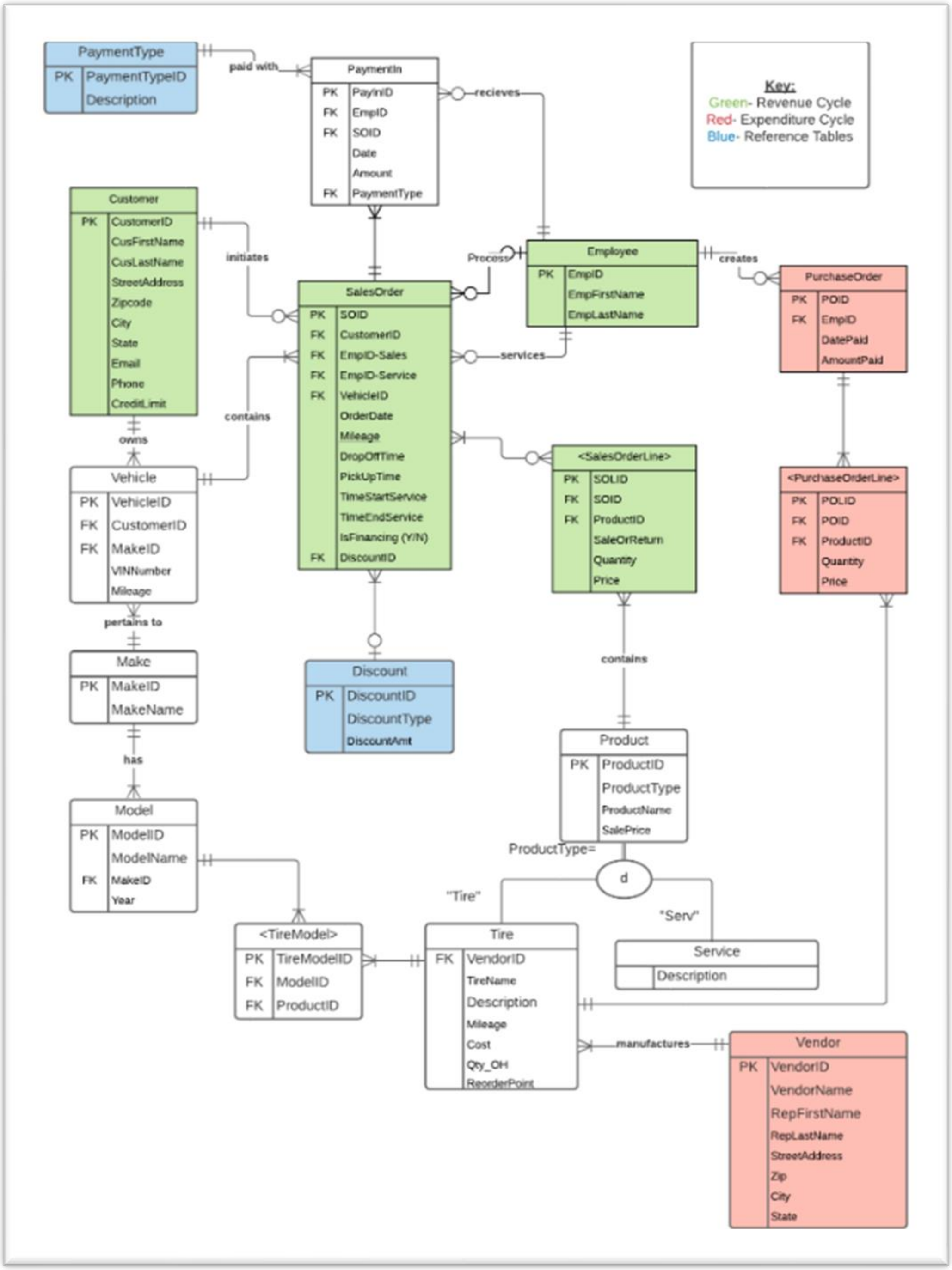


Table of Contents

What is an ERD? Why is it necessary?

An *Entity Relationship Diagram* (ERD) maps out the relationships between various entities. An *entity* is a thing or process that will be used in the database. In our case, "Employee" and "SalesOrder" are entities that will be mapped in the ERD. The ERD visually describes the relationships between entities by including verbs and cardinalities. For example, an employee might *process* a sales order. In this case, the verb "process" describes the relationship between Employee and SalesOrder. The cardinality is a number that describes the possible outputs or inputs for that grouping. For example, an Employee can process *one to many* SalesOrders, but a SalesOrder can only be processed be *one and only one* employee. We would indicate this cardinality on the ERD by using a symbol that looks like a crowsfoot on the many side, and two vertical parallel lines on the one-and-only-one side (also known as the mandatory side). It is important to use ERDs when creating databases so that we can plan out all of the necessary entities and relationship that Sonner Tire uses in their business operations. After completing the ERD, it should be an accurate representation of all the different business operations that need to be tracked in the database.

Business Cycles Used

We used the revenue and expenditures cycles because Sonner Tire only needs to track the revenue they receive from selling tires and minimize the cost of inventory and supplies and other services. The business revenue cycle occurs anytime a company sells products or services. Within the revenue cycle, the company's revenue activities will be recorded. The expenditure cycle occurs with the purchase of and payment for goods and services. Through the expenditure cycle, Sonner Tires will be able to better track and purchase inventory. Sooner Tire only sells products and services, and buys products such as the tires they work with, thus the company does not need to track productions. Sonner Tire needs to record: sales made to customers, information about the customer, employee, product, the vendor, payment, inventory, and goods purchased, which is why the revenue and expenditure cycle are appropriate to use for the company's needs.

Data Provided by Client

Based on the data provided by the client, there are three tables necessary: make, model, and tire. The make name will be associated with the make table. For the model table, name and year will be associated with it as attributes. Lastly, in the tire table, the attributes associated with it are name, mileage, description, cost, and sale price. This data tells us that we will need to have relationships between make, model, and tire in our ERD.

ERD Created

On the following page, we have provided a picture of the ERD that we created for Sonner Tire based off of the case information given to us and the feedback from the client. Our ERD is color coded according to the revenue cycle, expenditure cycle, and reference tables. The ERD was created using LucidChart. Many of the entities in our diagram were taken from the generic revenue and business cycles, but we had to make some changes based on what Sonner Tires needed. We will go into further details about the changes we had to make in the following section.

Query Feasibility and Current ERD

This chart contains each of the query questions requrested by Sonner Tires. We determined which tables would be needed to run the query, and provided a projected SQL statement. Additionally, we added comments about what changes we needed to make to the generic ERD in order to make these queries work. This process is a vital precursor to the logical design process so that we can ensure that our database will be able to run the queries that Sonner Tire needs.

#	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).	Vendor, Make, Model, SalesOrder, <SalesOrderLine>, Customer We added the entities Make, Model, and Year. These entities have a relationship with vehicle.	Select OrderDate, Mod.ModelName, M.MakeName, Year, VendorName, SUM(Quantity*Price) as TotalSales, State From Model Mod Join Make M On Mod.MakeID = M.MakeID Join Vehicle V On V.MakeID = M.MakeID Join Customer C On V.CustomerID = C.CustomerID Join SalesOrder SO On C.CustomerID = SO.CustomerID Join SalesOrderLine SOL On SO.SOID = SOL.SOLID Join Product P On SOL.SOLID = P.ProductID Join Tire T On P.ProductID = T.ProductID Join Vendor Ven On T.VendorID = Ven.VendorID Where Month(OrderDate) = '__', ModelName = '__' , MakeName = '__', Year = '__', VendorName = '__', State = '__'

			Group by OrderDate, ModelName, MakeName, Year, VendorName, State
2	Total sales (in dollars) by a customer in a given year.	PaymentIn, SalesOrder, Customer	Select CustomerID, LastName, FirstName, OrderDate, SUM(Amount) as TotalSales From Customer C Join SalesOrder SO On C.CustomerID = SO.SOID Join PaymentIn PI On SO.SOID = PI.PayInID Where Year(OrderDate) = '__' Group by CustomerID, LastName, FirstName, OrderDate
3	The five highest selling tires.	<SalesOrderLine>, Product, Tire We added Tire and Service as sub-types of Product.	Select TOP 5 TireName, MAX(Quantity) From Tire T Join Product P On P.ProductID = T.ProductID Join SaleOrderLine SOL On P.ProductID = SOL.SOLID Group by TireName
4	Itemized invoices for jobs for each customer that need to include tires purchased/tire rotation/tire repair/tire protection.	SalesOrder, <SalesOrderLine>, Customer, Product	Select CustomerID, LastName, FirstName, ProductType, P.Price, Quantity From Customer C Join SalesOrder SO On C.CustomerID = SO.SOID Join SalesOrderLine SOL On SO.SOID = SOL.SOLID Join Product P On P.ProductID = SO.ProductID Join Tire T On T.ProductID = P.ProductID Group By CustomerID
5	The number and type of job	Employee	Select EmpID, FirstName, LastName, ServiceTypeID

	performed by each of our employees.		
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).	SalesOrder, <SalesOrderLine>, Product Still not sure how to use sub type super type. Also do not know how to group by tire repair and replacement	Select SalesOrder From SaleOrderLine On SOL on P.Product ID= SOL.SOLIDID From Product P Join Service Serv On P.ProductP = Serv.ProductID
7	The following items for Purchase Orders: manufacturer name, number of POs, total cost.	PurchaseOrder, <PurchaseOrderLine>, Vendor, Tire	Select COUNT(POID) as NumPOS, SUM(Cost) as TotalCost From (Select Distinct VendorID, POID, VendorName From PurchaseOrder PO Join PurchaseOrderline POL On PO.POID = POL.POID Join Tire T On T.ProductID = POL.ProductID Join Vendor V On T.VendorID = V.VendorID) SQ
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	Customer, SalesOrder, <SalesOrderLine>, Product	Select CustomerID, LastName, FirstName, SUM(Quantity*SalePrice) as TotalSales, COUNT(SOID) as NumOrders, OrderDate From Customer C Join SalesOrder SO On C.CustomerID = SO.SalesOrderID Join SalesOrderLine SOL On SO.SalesOrderID = SOL.SalesOrderID Join Product P

	returning customers.		On SOL.ProductID = P.ProductID Where Year(OrderDate) = GETDATE() -2 Group by CustomerID, LastName, FirstName, OrderDate
9	List of tires that have not been purchased within the last 6 months (in order to better manage inventory).	Product, SalesOrderLine, SalesOrder	Select TireName, OrderDate From Product P Join Tire T On P.ProductID = T.ProductID Left Join SalesOrderLine SOL On P.ProductID = SOL.SOLID Join SalesOrder SO On SOL.SOLID = SO.SOID Where ProductType = “Tire” and Month(OrderDate) Between Month(GetDate())-6 And Month(GetDate())
10	Names of customers who took advantage of the financing option, date purchased, total amount purchased, credit limit, number of payments made, the total amount paid, outstanding amount, is time to pay-off less than 6 months, all displayed from	Finance, PaymentTerms, Customer, SalesOrder, PaymentIn	Select FirstName, LastName, OrderDate, SUM(OrderTotal), CreditLimit, COUNT(PayInID), SUM(Amount), OrderTotal – SUM(Amount) As Outstanding_Amount From Finance F Join Customer C On F.FinanceID = C.FinanceID Join PaymentTerms PT On C.TermsID = PT.TermsID Join SalesOrder SO On C.CustomerID = SO.CustomerID Join PaymentIn P On SO.PayInID = P.PayInID Where C.FinanceID Not Null Group By FirstName, LastName, OrderDate, CreditLimit Having Length < 6 Order By OrderDate Desc, Outstanding_Amount Desc

	the latest date and then the largest amount owed.		
11	Total profit per tire type and manufacturer type in the past 6 months.	Product, Tire, <SalesOrderLine>, SalesOrderLine, Make Not sure if TotalProfit is correct.	Select SUM((P.Price-Cost) * Quantity) as TotalProfit, TireName, ModelName, OrderDate From Make Mk Join Model Md On Mk.MakeID = Md.ModelID Join ModelTire MT On Md.ModelID = MT.ModelTireID Join Product P On MT.ModelTireID = P.ProductID Join SalesOrderLine SOL On P.ProductID = SOL.SOLID Join SalesOrder SO ON SOL.SOLID = SO.SOID Where Month(OrderDate) = GETDATE() – 6 Group by TireName, ModelName, OrderDate
12	List of all customers that have not made a purchase within the last 12 months from the current date.	Customer, SalesOrder	Select CustomerID, LastName, FirstName,OrderDate From Customer C Left Join SalesOrder SO On C.CustomerID = SO.SOID Where Month(OrderDate) between Month(GetDate())-12 And Month(GetDate())
13	List of customers whose average sales is less than the average of all sales. This will help us to find	Customer, SalesOrder	Select Customer, AVERAGE(COST) as Average Cost, Average(OrderTotal) as Total Average Sales Join Customer C On SalesOrder SOI C.Customer = C.SalesOrder Where Average Cost < Total Average Sales Order By (MAX) Average Cost ascen

	customers whom we should target to get a higher volume of sales.		

Logical Design

The logical design phase occurs after the conceptual design phase. In this phase, the ERD has already been created, so now it is important to ensure that the database is designed correctly so that it can run without issues. In the logical design phase, entities are converted into relations. Before we can begin writing out the relations, we undergo the process of normalization. Normalization is a crucial part of the logical design process. Then, there are several rules and constraints that need to be followed when converting the entities into relations, which we will discuss further in detail.

Normalization

The process of normalization is intended to make the database reliable and efficient. To normalize the data structure, we must ensure that each column is "atomic" meaning it cannot be broken down any further. For example, it is best practice not to have any multi-valued attributes like "name" in the database. We must break it down into its base components of first name and last name. Second, the columns must not contain redundant data, which increases the time it takes to run queries. By doing these two things we can ensure our database will have data integrity and run efficiently. Lastly, we must remove dependencies. To do this, we make sure there are not any dependency constraints. Constraints will be described further in the *data integrity* section.

Normalization of the Data Provided by the Client

To normalize the data that was provided by the client, we first had to ensure that each column was broken down into its most basic form and did not have multiple values. Then, we ensured that there were no partial or functional dependencies in the tables. To accomplish this, every column needed to be predicted by the key element within the table and that key element only.

TMake(MakeID, MMakeName)

TCarModel(ModelID, CMMakeID, CMModelName, CMYear)
Foreign Key CMMakeID references TMake
Not Null
On delete Restrict

TModelTire(MTID, ModModelID, ModProductID-Tire)
Foreign Key ModModelID references TCarModel
Not Null
On delete restrict

TCar(CarID, CCarManufacturer, CModelID, CModel, CYear, CTire)
Foreign Key CModelID references TCarModel
Not Null
On delete Restrict

TTires(TireID, TireName, TireManufacturer, TireGoodFor, TireMileage, TireDescription, TireCost, TireSalesPrice, TireQTY_OH, TireQTY_Committed, TireReorderPoint)

Normalized Relations

Revenue Cycle

TCustomer(CustomerID, CustFirstName, CustLastName, CustStreetAddress, CustZipcode, CustCity, CustState, CustEmail)

TEmployee(EmpID, EmpFirstName, EmpLastName)

TDiscount(DiscountID, DDiscountType)

TSalesOrder(SOID, SOPayInID, SOCustomerID, SOEmpID-Sales, SOEmpID-Service, SODiscountID, SOVehicleID, SOTechFirstName, SOTechLastName, SOOrderDate, SOMileage, SODropOffTime, SOPickUpTime, SOTimeStartService, SOTimeEndService, SOIsFinancing)

Foreign Key SOPayInID references TPaymentIn
Null Allowed
On delete set null
Foreign Key SOCustomerID references TCustomer
Not Null
On delete restrict
Foreign Key SOEmpID-Sales references TEmployee
Not Null
On delete restrict
Foreign Key SOEmpID-Service references TEmployee
Null allowed
On delete set null
Foreign Key SODiscountID references TDiscount
Null Allowed
On delete set null
Foreign Key SOVehicleID references TVehicle
Not Null
On delete restrict

TSalesOrderLine(SOLID, SOLSOID, SOProductID, SOLStatus, SOLSaleOrReturn, SOLQuantity, SOLPrice)

Foreign Key SOLSOID references TSalesOrder
Not Null
On delete restrict
Foreign Key SOProductID references TProduct
Not Null
On delete restrict

TPaymentType(PaymentTypeID, PTDescription)

TPaymentIn(PayInID, PayEmpID, PayPaymentType, PayDate, PayAmount, PayCardNumber, PayExpirationDate, PaySecurityCode)

Foreign Key PayPaymentType references TPaymentType
Null allowed
On delete set null
Foreign Key PayEmpID references TEmployee
Not Null
On delete Restrict

Expenditure Cycle

TVendor(VendorID, VVendorName, VSalesRepFirstName, VSalesRepLastName)

TPurchaseOrder(POID, POEmpID, PODatePaid, POAmountPaid)

Foreign Key POEmpID references TEmployee
Not Null
On Delete Restrict

TPurchaseOrderLine(POLID, POLPOID, POLProductID, POLQuantity, POLPrice)

Foreign Key POLPOID references TPurchaseOrder
Not Null
On Delete Restrict
Foreign Key POLProductID references TTire
Not Null
On Delete Restrict

TMake(MakeID, MMakeName)

TModel(ModelID, MOMakeID, MOModelName, MOYear)

Foreign Key MOMakeID references TMake
Not null
On delete Restrict

TVehicle(VehicleID, VEHCustomerID, VEHMakeID, VINNumber)

Foreign Key VEHCustomerID references TCustomer
Not null
On Delete Restrict
Foreign Key VEHMakeID references TMake
Not null
On delete restrict

TProduct(ProductID, PProductType, PSalePrice)

TService(ServProductID, ServLifeTimeProtection)

TTire(TireProductID, TireVendorID, TireName, TireDescription, TireMileage, TireCost, TireQty_OH, TireQTY_Committed, TireReorderPoint)

Foreign Key TIVendorID references TVendor
Not Null
On Delete Restrict

TTireModel(TireModelID, TMModelID, TMProductID)

Foreign Key TMModelID references TModel
Not Null
On Delete Restrict
Foreign Key TMProductID references Ttire
Not Null
On Delete Restrict

Differences between ERD and Normalized Relations

One difference between ERDs and normalized relations are that ERDs can have multi-valued attributes, while normalized relations should be broken down into smaller attributes in order to make the entity atomic. Atomicity is important so that reports made within the database are efficient and accurate. Similarly, normalized relations do not include derived attributes. This is important because derived attributes are calculated from other attributes, so they do not need to be included in normalized relations. Furthermore, the names of the entities in normalized relations are different from the names of the entities in the ERD. In normalized relations, we add a T to the beginning of the name of the entity to indicate that it is a table. Additionally, the attributes in normalized relations have unique names. This is beneficial because it will prevent us from getting unambiguous column names in our queries.

Database Integrity

Data integrity means that the reports generated from the database are trustworthy. Normalization is one way to ensure that data integrity is accomplished. There are three integrity constraints: entity, referential, and domain. The entity integrity constraint is that every entity must have a primary key that isn’t null and doesn’t change over time. The referential integrity applies to the relationships between entities. It states that for each relationship, the foreign key in one entity must match the primary key in the other entity, or null if applicable. The last is the domain integrity constraint. This constraint says that every value in a column must be of the same data type, like integer or string. We ensured that these constraints were enforced by having a related primary key, none of which that are null, and foreign key for each of the relationships in our diagram. We enforced the referential constraint by not allowing any multi-value attributes.

Physical Design and Implementation

The Physical Design phase is the part of the database building process where we choose which relational database management system (RDBMS) we will we be using. This is important because different RDBMS have different data types that they use to store information. We will be using Microsoft SQL Server as our RDBMS. The next step is the actual implementation of data into the database which we did using dummy data. The purpose of this was to make sure everything in the database was working without any errors. Without this phase of the database design process, we wouldn't be able to create a database. Rather, we would just have the conceptual design all planned out on paper.

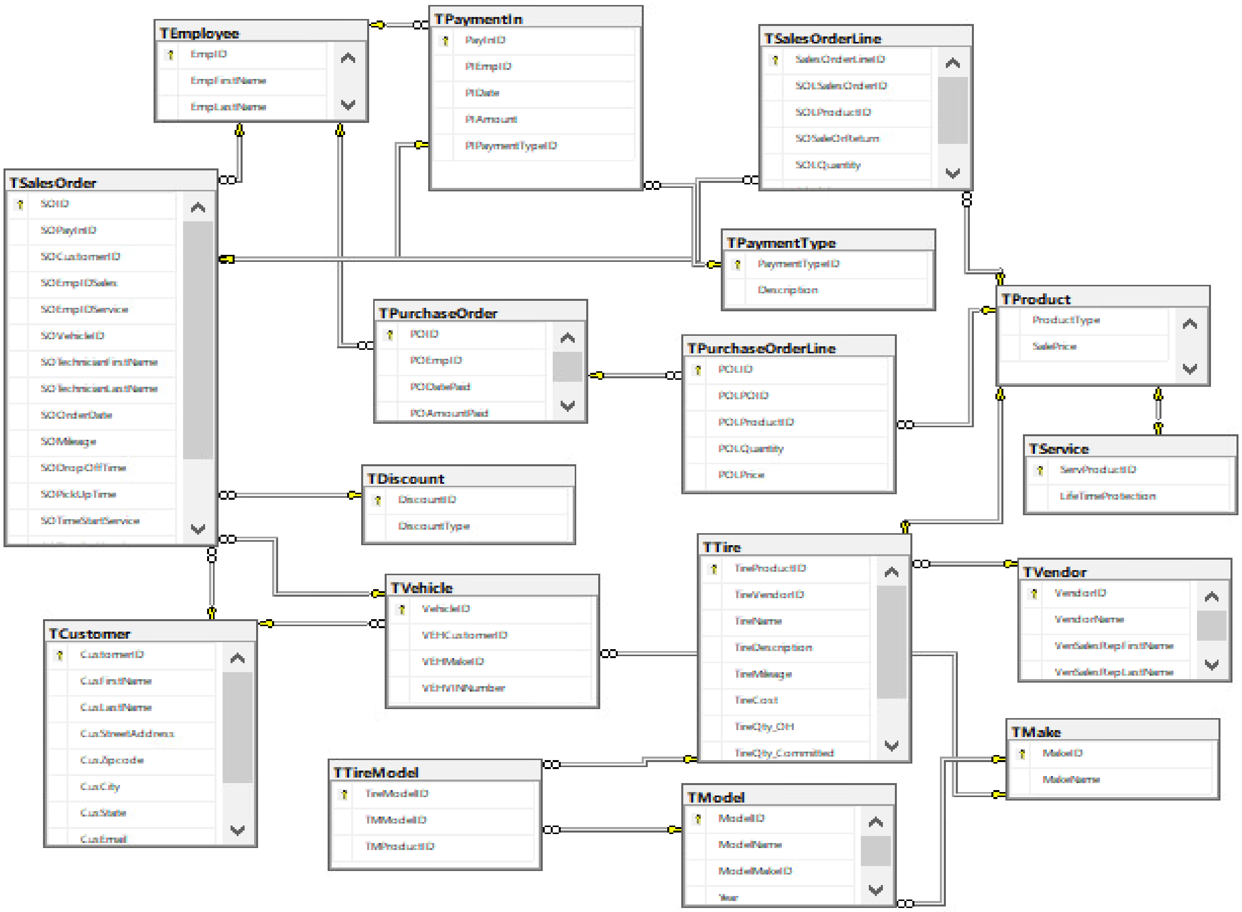
Data Dictionary

A data dictionary is a collection of information describing the data included in a database and the relationships between the information. It is used as a way to better understand the structure and information within a database. It includes things like entity names, attributes, and their data types. As an example, the data dictionary for our project includes things like the Customer table and its attributes being things like their first name, last name, and the customer type. The data types, whether or not it is allowed to be null, what table it references if it is a foreign key, and a sample of the key will be included on the same row as the attribute.

Denormalization

Denormalization is the process of removing of some of the normalized relations of the data in order to improve performance. When denormalizing the data, it will make it more efficient to run SQL queries that include a long list of join statements. While we recognize that denormalization results in data duplication and redundancy, we made the choice to denormalize some of our data. First, we decided to remove the year table and list the year in the model table instead. We also decided to list the customer's state and zip code as attributes in the customer table, rather than having separate tables for them. We decided to do this with the vendor's address as well. These changes make it easier for us and the client to write queries and will allow the queries to run quicker.

Implemented Physical Design



Strengths and Weaknesses Encountered During Implementation

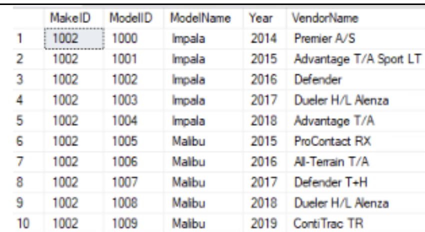
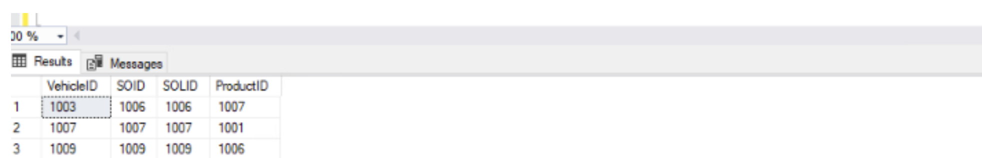
One of our strengths was that our ERD was close to finalized before going into the implementation phase, which made it easier for us to create tables in the database. Creating the tables in the database was also one of our strengths because we simply had to create the tables in the order that we wrote our normalized relations. However, one of our weaknesses when creating the tables in the database was that the attributes of the tables were named inconsistently since we were working on them separately. The inconsistency of the attribute names made it more difficult for us to write the SQL queries and implement them into our database. To fix this issue, we would need to delete the tables and create them again with the appropriately named attributes. Given the time constraints, we decided to leave the tables the way they are.

Specific SQL Statements Requested

Here we will list the specific programs we were asked to execute by the client in the database, as well as additional queries we believe would be useful for the operation. We have included the request asked of us, the SQL code needed to implement the program, and an image of the result of the program. Some of the ouputs are empty because none of the sample data applied to the requested query. When more data is added to the database, it will show more results.

<table>
<tr><th>Query #</th><th>Question</th><th>SQL</th><th>Partial Output</th></tr>
<tr>
<td>1</td>
<td>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).</td>
<td>SELECT MakeID, ModelID, MOModelName, MOYear, VendorName, Sum(Quantity*SalePrice) TotalSales
FROM View1 JOIN View2
ON View1.ProductID=View2.ProductID
WHERE month(date)=[Parameter]

View1
SELECT MakeID, ModelID, ModelName, Year, VendorName
FROM TMake
JOIN TModel
ON MakeID = ModelMakeID
JOIN TTireModel
ON ModelID = TMModelID
JOIN TTire
ON TMProductID = TireProductID
Join TVendor
ON VendorID = TireVendorID

View2</td>
<td>
<table>
<tr><th></th><th>MakeID</th><th>ModelID</th><th>ModelName</th><th>Year</th><th>VendorName</th></tr>
<tr><td>1</td><td>1002</td><td>1000</td><td>Impala</td><td>2014</td><td>Premier A/S</td></tr>
<tr><td>2</td><td>1002</td><td>1001</td><td>Impala</td><td>2015</td><td>Advantage T/A Sport LT</td></tr>
<tr><td>3</td><td>1002</td><td>1002</td><td>Impala</td><td>2016</td><td>Defender</td></tr>
<tr><td>4</td><td>1002</td><td>1003</td><td>Impala</td><td>2017</td><td>Dueler H/L Alenza</td></tr>
<tr><td>5</td><td>1002</td><td>1004</td><td>Impala</td><td>2018</td><td>Advantage T/A</td></tr>
<tr><td>6</td><td>1002</td><td>1005</td><td>Malibu</td><td>2015</td><td>ProContact RX</td></tr>
<tr><td>7</td><td>1002</td><td>1006</td><td>Malibu</td><td>2016</td><td>All-Terrain T/A</td></tr>
<tr><td>8</td><td>1002</td><td>1007</td><td>Malibu</td><td>2017</td><td>Defender T+H</td></tr>
<tr><td>9</td><td>1002</td><td>1008</td><td>Malibu</td><td>2018</td><td>Dueler H/L Alenza</td></tr>
<tr><td>10</td><td>1002</td><td>1009</td><td>Malibu</td><td>2019</td><td>ContiTrac TR</td></tr>
</table>

00 %
Results Messages
<table>
<tr><th></th><th>VehicleID</th><th>SOID</th><th>SOLID</th><th>ProductID</th></tr>
<tr><td>1</td><td>1003</td><td>1006</td><td>1006</td><td>1007</td></tr>
<tr><td>2</td><td>1007</td><td>1007</td><td>1007</td><td>1001</td></tr>
<tr><td>3</td><td>1009</td><td>1009</td><td>1009</td><td>1006</td></tr>
</table>
</td>
</tr>
</table>

		SELECT VehicleID, SOID, SOLID, ProductID FROM TVehicle JOIN TSalesOrder ON VehicleID = SOVehicleID JOIN TSalesOrderLine SOL ON SOID = SOLID JOIN TProduct ON SOLProductID = ProductID WHERE ProductType='Tire'	
2	Total sales (in dollars) by a customer in a given year.	SELECT CusFirstName, CusLastName, Sum(PIAmount) TotalSales FROM TCustomer JOIN TSalesOrder ON CustomerID = SOCustomerID JOIN TPaymentIn ON SOID = PISOID GROUP BY CusFirstName, CusLastName	| CusFirstName | CusLastName | TotalSales 1 | Jakeem | Bryan | 863.94 2 | Medge | Kirk | 831.94 3 | Reese | Levy | 60 4 | Quincy | Williams | 353.98
3	The five highest selling tires.	SELECT TOP 5 TireProductID, ProductName, Count(SOLID) TimesSold FROM TSalesOrderLine JOIN TProduct ON SOLProductID = ProductID JOIN TTire ON TireProductID = ProductID GROUP BY TireProductID, ProductName ORDER BY TimesSold DESC	| TireProductID | TireName | TimesSold 1 | 1001 | 235/50R19 | 2 2 | 1002 | 235/50R19 | 1 3 | 1004 | 235/50R19 | 1 4 | 1006 | 265/40R21 | 1 5 | 1007 | 265/40R21 | 2

<table>
<tr>
<td>4</td>
<td>itemized invoices for jobs for each customer that need to include tires purchased/tire rotation/tire repair/tire protection</td>
<td>SELECT CustomerID, SOID, VehicleID, SOOrderDate, SOLQuantity, ProductID, CusFirstName, CusLastName, SalePrice, (SOLQuantity*SalePrice) LineTotal
FROM TCustomer JOIN TVehicle on CustomerID=VEHCustomerID join TSalesOrder on VehicleID=SOVehicleID join TSalesOrderLine on SOID = SOLSOID join TProduct on SOLProductID = ProductID
WHERE CustID=[] AND CarID=[] AND Date=[]</td>
<td>(with the WHERE clause commented out)
<table>
<tr><th>CustomerID</th><th>SOID</th><th>VehicleID</th><th>SOOrderDate</th><th>SOLQuantity</th><th>ProductID</th><th>CusFirstName</th><th>CusLastName</th><th>SalePrice</th><th>LineTotal</th></tr>
<tr><td>1007</td><td>1000</td><td>1007</td><td>2021-04-17</td><td>1</td><td>1007</td><td>Quincy</td><td>Williams</td><td>148.99</td><td>148.99</td></tr>
<tr><td>1003</td><td>1010</td><td>1003</td><td>2020-10-20</td><td>4</td><td>1001</td><td>Jakeem</td><td>Bryan</td><td>133.99</td><td>535.96</td></tr>
<tr><td>1001</td><td>1011</td><td>1001</td><td>2020-11-22</td><td>2</td><td>1010</td><td>Reese</td><td>Levy</td><td>15</td><td>30</td></tr>
<tr><td>1003</td><td>1003</td><td>1003</td><td>2020-11-18</td><td>2</td><td>1006</td><td>Jakeem</td><td>Bryan</td><td>163.99</td><td>327.98</td></tr>
<tr><td>1007</td><td>1004</td><td>1007</td><td>2020-07-25</td><td>1</td><td>1002</td><td>Quincy</td><td>Williams</td><td>170.99</td><td>170.99</td></tr>
<tr><td>1009</td><td>1005</td><td>1009</td><td>2021-03-22</td><td>1</td><td>1007</td><td>Medge</td><td>Kirk</td><td>148.99</td><td>148.99</td></tr>
<tr><td>1009</td><td>1006</td><td>1009</td><td>2020-11-07</td><td>4</td><td>1001</td><td>Medge</td><td>Kirk</td><td>133.99</td><td>535.96</td></tr>
<tr><td>1001</td><td>1007</td><td>1001</td><td>2020-06-19</td><td>2</td><td>1010</td><td>Reese</td><td>Levy</td><td>15</td><td>30</td></tr>
<tr><td>1007</td><td>1008</td><td>1007</td><td>2020-04-28</td><td>2</td><td>1011</td><td>Quincy</td><td>Williams</td><td>17</td><td>34</td></tr>
<tr><td>1009</td><td>1009</td><td>1009</td><td>2020-05-27</td><td>1</td><td>1004</td><td>Medge</td><td>Kirk</td><td>146.99</td><td>146.99</td></tr>
</table>
</td>
</tr>
<tr>
<td>5</td>
<td>The number and type of job performed by each of our employees.</td>
<td>SELECT EmpID, EmpFirstName, EmpLastName, Count(SOEmpIDSales) Sales, Count(SOEmpIDService) Service
FROM TEmployee Join TSalesOrder On EmpID = SOEmpIDService
GROUP BY EmpID, EmpFirstName, EmpLastName</td>
<td>
<table>
<tr><th></th><th>EmpID</th><th>EmpFirstName</th><th>EmpLastName</th><th>Sales</th><th>Service</th></tr>
<tr><td>1</td><td>1001</td><td>Troy</td><td>Pruitt</td><td>3</td><td>3</td></tr>
<tr><td>2</td><td>1002</td><td>Paul</td><td>Trujillo</td><td>1</td><td>1</td></tr>
<tr><td>3</td><td>1005</td><td>Barry</td><td>Wolf</td><td>2</td><td>2</td></tr>
<tr><td>4</td><td>1006</td><td>Jessamine</td><td>Haynes</td><td>2</td><td>2</td></tr>
<tr><td>5</td><td>1008</td><td>Mona</td><td>Horne</td><td>2</td><td>2</td></tr>
</table>
</td>
</tr>
<tr>
<td>6</td>
<td>Number of times tire protection has been purchased for a particular tire</td>
<td>SELECT TireProductID, TireName, Count(TireProductID) NumPurchases
FROM TSalesOrderLine JOIN TProduct On SOLProductID = ProductID Join TTire On ProductID = TireProductID Join TService On ProductID = ServProductID</td>
<td>
<table>
<tr><th>TireProductID</th><th>TireName</th><th>NumPurchases</th></tr>
</table>
</td>
</tr>
</table>

<table>
<tr><td></td><td></td><td>WHERE ProductType='Serv' And ServDescription = 'tire protection'
GROUP BY TireProductID, TireName</td><td></td></tr>
<tr><td>7</td><td>The following items for Purchase Orders: manufacturer name, number of POs, total cost.</td><td>SELECT VendorID, VendorName, Count(POID) NumPurchases, Sum(POLQuantity*POLPrice) Total
FROM TPurchaseOrder Join TPurchaseOrderLine On POID = POLPOID Join TTire On POLProductID = TireProductID Join TVendor On TireVendorID = VendorID
GROUP BY VendorID, VendorName</td><td>
<table>
<tr><th></th><th>VendorID</th><th>VendorName</th><th>NumPurchases</th><th>Total</th></tr>
<tr><td>1</td><td>1001</td><td>Advantage T/A Sport LT</td><td>2</td><td>1267.92</td></tr>
<tr><td>2</td><td>1002</td><td>Defender</td><td>1</td><td>170.99</td></tr>
<tr><td>3</td><td>1004</td><td>Advantage T/A</td><td>1</td><td>182.99</td></tr>
<tr><td>4</td><td>1006</td><td>All-Terrain T/A</td><td>1</td><td>327.98</td></tr>
<tr><td>5</td><td>1007</td><td>Defender T+H</td><td>2</td><td>297.98</td></tr>
</table>
</td></tr>
<tr><td>8</td><td>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.</td><td>SELECT CustomerID, CusFirstName, CusLastName, Count(SOID)
FROM TCustomer Join TSalesOrder On CustomerID = SOCustomerID
WHERE SOOrderDate >= 2 years
GROUP BY CustomerID, CusFirstName, CusLastName</td><td>
<table>
<tr><th></th><th>CustomerID</th><th>CusFirstName</th><th>CusLastName</th><th>NumSOS</th></tr>
<tr><td>1</td><td>1000</td><td>Dalton</td><td>Paul</td><td>1</td></tr>
<tr><td>2</td><td>1004</td><td>Lars</td><td>Richmond</td><td>1</td></tr>
<tr><td>3</td><td>1005</td><td>Rylee</td><td>Merrill</td><td>2</td></tr>
<tr><td>4</td><td>1006</td><td>Hedwig</td><td>Dodson</td><td>2</td></tr>
<tr><td>5</td><td>1007</td><td>Quincy</td><td>Williams</td><td>2</td></tr>
<tr><td>6</td><td>1008</td><td>Salvador</td><td>Shepherd</td><td>2</td></tr>
</table>
</td></tr>
</table>

9	List of tires that have not been purchased within the last 6 months (in order to better manage inventory).	SELECT ProductID, ProductName FROM TSalesOrderLine LEFT JOIN TProduct WHERE SOLProductID IS Null AND ProductType = ‘Tire’	Results Messages ProductID TireName
10	Names of customers who took advantage of the financing option, date purchased, total amount purchased, credit limit, the number of payments made, the total amount paid, outstanding amount, is time to pay-off less than 6 months, all displayed from the latest date and then the	Payment type – cash, credit, check SELECT CusFirstName, CusLastName, Total, Paid, (Total - Paid) Remaining FROM TCustomer Join TSalesOrder On CustomerID = SOCustomerID Join SQ1 On SOCustomerID = SQ1.CustomerID JOIN SQ2 ON SQ1.SOID=SQ2.SOID WHERE IsFinancing = ‘Y’ AND Month(SOOrderDate) = Month(GETDATE()) – 6 SQ1 SELECT CustomerID, SOID, Sum(SOLQuantity*SOLPrice) Total FROM TVehicle Join TSalesOrder On VehicleID = SOVehicleID Join TSalesOrderLine On SOID = SOLSOID GROUP BY CustomerID, SOID SQ2	CustomerID SOID Total 1 1001 1004 170.99 2 1003 1000 148.99 3 1003 1006 731.96 4 1007 1001 30 5 1007 1005 148.99 6 1007 1007 30 7 1009 1003 327.98 8 1009 1008 34 9 1009 1009 718.95 CustomerID CusFirstName CusLastName Paid 1 1001 Reese Levy 16.89 2 1003 Jakeem Bryan 57.44 3 1007 Quincy Williams 42.39 4 1009 Medge Kirk 60.33

	largest amount owed.	SELECT CustomerID, CusFirstName, CusLastName, Sum(PIAmount) Paid FROM TCustomer Join TVehicle On CustomerID = VEHCustomerID Join TSalesOrder On SOVehicleID = VehicleID Join TPaymentIn On SOID = PISOID GROUP BY CustomerID, CusFirstName, CusLastName	
11	Total profit per tire type and manufacturer type in the past 6 months.	SELECT ProductID, ProductName, SUM((SOLPrice-POLCost)*Quantity) Profit FROM TSalesOrder join TSalesOrderLine on SOID=SOLSOID join TProduct P on SOLProductID=ProductID join TTire on ProductID=TireProductID WHERE Month(SOOrderDate) = Month(GETDATE()) – 6 GROUP BY ProductID, ProductName	 Results Messages ProductID / TireName / Profit 1 / 1001 / 235/50R19 / 296.92 2 / 1002 / 235/50R19 / 63.49 3 / 1004 / 235/50R19 / 105.49 4 / 1006 / 265/40R21 / 152.98 5 / 1007 / 265/40R21 / 155.48 ProductID / TireName / Profit
12	List of all customers that have not made a purchase within the last 12 months from the current date.	SELECT CustomerID, CusFirstName, CusLastName FROM TCustomer C JOIN TVehicle V on C.CustomerID=V.VEHCustomerID Left Join TSalesOrder on VehicleID= SOVehicleID WHERE SOID IS Null AND Month(SOOrderDate) = Month(GETDATE()) – 12	 CustomerID / CusFirstName / CusLastName

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 CusFirstName, CusLastName, AVG(SOLQuantity*SOLPrice) AVGPurchase FROM TCustomer Join TSalesOrder On CustomerID = SOCustomerID Join TSalesOrderLine On SOID = SOLSOID WHERE AVG(SOLQuantity*SOLPrice) < (SELECT AVG(SOLQuantity*SOLPrice) From TSalesOrderLine)	CusFirstName CusLastName AVGPurchase 1 Rylee Merrill 30 2 Dalton Paul 170.99 3 Lars Richmond 148.99 4 Quincy Williams 34