Cory DioGuardi
Parker Johnson
Reese Jones
Louden Yandow
Principles of Data Management
Phase 1 Report

We designed our database with a focus on the relations between customers, dealers, and manufacturers, with the sales of vehicles. Our final design has a parent entity called Owner relating to a Sale entity. The Owner (which can be a customer, dealership, or manufacturer) buys or sells Vehicles. This is done through a Sale, which keeps information regarding the vehicle(s) sold, the cost of the sale, who is buying or selling the vehicle(s), and the date. Our vehicle entity keeps descriptive information such as the VIN (Vehicle Identification Number), make, model, year, color, engine, and transmission.

It is important to note that with this design in mind, our model suggests that Customers (who are not dealers or manufacturers) can sell cars as well. However, in our implementation, this will not be possible, as it is not our responsibility to keep information on cars after they are sold to a citizen.

We made a number of decisions that impacted the structure of our relations. We noticed that three of our entities, Manufacturer, Dealer, and Owner, have similar repeated information. This lead to us deciding to have all three entities inherit from a base class called Owner. We believe this will help reduce repetition between these entities, while still allowing us to differentiate them enough to effectively use them in our project.

Another similar decision we made was to have CustomerSale and BulkSale entities both inherit from a parent class called Sale. Like the previous example, they both have very similar information, but it is necessary to keep them separate as they behave differently. This way we can differentiate between the two types of sales easily, while both maintain the information they

need. We ended up completely merging these two entities anyway, as they were only different in the sense that BulkSale kept track of multiple VIN's, while CustomerSale only kept track of one. Merging them is more efficient in terms of modelling, because we determined that it didn't make too much of a difference if a sale was capable of keeping track of multiple VIN's, but was only used to keep track of one.

We faced some issues in our early design that forced us to make changes. The biggest problem we had was the number of ternary relations in our model. One of these relationships was between our Dealer, Customer, and Vehicle entities. We originally related these three entities with a "Sold" relationship. To eliminate this ternary relation, we fabricated the Sale entity. This new entity allowed us to remove the ternary relation and create three binary relations with Sale and the aforementioned entities.

Our other ternary relation was between our Manufacturer, Dealer, and Vehicle entities. These were all related by a "bulk sale" relation. We solved this problem in a similar way as before, by fabricating a class (we called it BulkSale) to relate the three entities to in order to remove the ternary relation and create binary relations. After this, as previously mentioned, we noticed Sale and BulkSale are similar, and had them inherit from a parent class. We changed Sale to CustomerSale to remove ambiguity, but ended up reverting back to a model with one sale entity by merging CustomerSale and BulkSale.

The final result of these decisions and solutions is a somewhat simple ER Model. We have an entity, Owner (which can be either a Manufacturer, Dealer, or Customer, but not a mix of these three), that relates to a Sale entity in two ways: buying or selling. The Sale entity then relates to the Vehicle entity such that at least one Vehicle is sold in each Sale.

Our database will hold many different kinds of data. The bulk of our data will be organized in our Sale, Vehicle, Owner, and Customer tables. We have other tables too for the purpose of relating our main entities to other entities, or for accessing specific kinds of information like a list of a Customer's phone numbers.

Our Sale table contains a SaleID that is unique to every Sale. Along with the SaleID, we store the ID's of the seller (usually a Dealer) and the buyer (usually a Customer), the Cost of the transaction, and the date (separated into Day, Month, and Year) of the Sale.

Our Vehicle table contains a list of VIN's (Vehicle Identification Numbers) that are unique to each vehicle. Along with each VIN, the Brand, Model, Year, Color, Engine, and Transmission are stored.

Our Owner table contains an OwnerID that is unique to each Owner. Along with this ID, we store the Street Number, Street Name, City, State, and Zip of the Owner. These attributes are shared among Dealers, Manufacturers, and Customers, so our tables for these entities only share an OwnerID attribute in common, and inherit all the attributes of the Owner table. Customer has extra attributes Gender and Annual Income.

For the sake of modeling our design, our sample data's ID attributes (SaleID, OwnerID, and VIN) are incrementing. When we actually implement our design, we will use more diverse identification numbers. For Vehicles, the Brand, Model, Engine, Transmission, and Color will all be String values. For example, one of our sample Vehicles is a Chevrolet (brand) Suburban (model), with a V8 engine and Automatic transmission with a red colored body.

Our Owner entity's Name attribute will be stored as a String, and will contain the name of the Owner (which is the name of the Manufacturer, Dealer, or Customer). Their Street

Number and Zip are stored as integers, with Street Name, City, and State as Strings. Customers' Gender attribute is stored as a String, while its Annual Income attribute is stored as an integer. We have a table associating OwnerID's (numbers) with Phone Numbers (also numbers) so that we can keep track of Phone Numbers if an Owner has multiple.

We made a lot of progress with our project during this phase. We have completed designing what he hope to be a good implementation for our database, and have generated our own sample data to see if our design works and is efficient (or not). Overall, we discovered how much time it takes to design a large database like this, because the nature of data and its interactions with other data can very quickly make what sounds simple very complex. Many issues were found and resolved, and we have a concrete plan going forward on how we will implement our design and how we want it to function for the users.