This is a transcript of my narration for Homework 3, a walkthrough/initial analysis of modernizing the sample eMovies database.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# This is Homework 3, modernizing the eMovies database model in Erwin

I am first going to look at the logical model because the logical precedes and is forward engineered into the physical. In the logical, we are dealing with entities and attributes.

My process will be the following: I am going to look at a particular entity and if I think a change should be made, I will make that change and apply it to all other entities. Then I will revisit the entity and see what else should be modernized.

Looking at the entity Payment, the first thing I notice is its naming convention: all caps. As a singular noun, it does follow object-oriented design, which is good. However, we the first letter of each descriptor is capitalized. That is the first change I will be making. Instead of PAYMENT in all caps, I am changing the entity’s label to Payment with just a capital P. Next, I would make the same change to the rest of the entities in the models: Customer, Customer Credit, Movie Rental Record, Movie Copy, Movie, MovieStore here I would remove the underscore, Store and Employee.

Now, I go back to Payment and look its attributes. I notice that just like the initial entity names, they are all lower case. So, following proper or initial case, I will capitalize the first letter of each word in the attribute. For example, lowercase payment transaction number to this (Payment Transaction Number). I would do this to all the attributes in each entity in the data model.

Also, I would change any words that are shortened to their full description. In Payment, the No in Customer No should be Number, the e in epay should be Electronic, and credit card “exp” (also in Customer Customer) should be expiration. The same goes for email in the Customer entity, it should be Customer Email, to match the rest of its attributes; soc sec number in Movie Rental Record to social security number; in Movie: description to Movie Description, genre to Movie Genre, Rental rate to Movie Rental Rate; Star1 and 2 to StarOne and StarTwo. In Store, address 2 to two spelled out.

With the entity name and attributes following Pascal Case, we have a standard look and feel across the database. Also, the logical model is more representative of the real world now because before even reaching the database or back-end, either on paper documents or say a web form on the front end, these attributes will follow this spelling convention when they are asked to be filled in so there is less work down the line. Furthermore, they are self-documenting.

Now I will go back and reassess the attributes in entities, which will be easier because now everything is written with standard Pascal Case. Anywhere there is an attribute with Number in it, I am going to add “Id” because it is being used as a unique identifier. And we know it reflects a primary key in the physical because it is separated by its own box on the top.

I am also going to see if I can normalize the database. Within the body of Payment, I notice Credit Card number, expiration, and type. Credit Card itself is a subtype of payment so instead of being in the body of Payment, it should be its own entity and have expiration, type, and number as its attributes. CreditCard will still have a relationship to Payment as child and Payment as a parent, established by a foreign key to PaymentTransactionNumberId. The same thing should happen with check, and electronicpay, which are also types of payments. Like Credit Card, they should be separate entities with their own bodies, as children of Payment with a foreign key to type. The Check entity should have CheckId as a primary key with bank number and number as attributes and the electronicpayment entity should have electronicpaymentid as a primary key with Vendor Number and Account Number as attributes.

Now going to Customer, the attributes address, state, and zipcode are shared with employee and store. So, these should be taken out of their bodies and instead added to the body of a separate address entity. This makes sense because in the real world, addresses generally have the same format regardless of who or what they belong to. Then the attributes CustomerAddress in Customer, StoreAddress in Store, and EmployeeAddress in Address can just relate to the Address entity with a foreign key.

Customer also shares the attributes first name, last name, and phone with Employee. These attributes should be also extracted from their body and put into a separate Person. Both Customer and Employee are a Person who has this personally identifiable information. So, first name, last name, email, and phone, and optionally social security number should be the attributes in Person and Customer and Employee should be children of Person through an is-a relationship. A Customer is a Person, and an Employee is a Person. A supervisor can also be a person, but more specifically, a higher ranked Employee who is already considered a Person. So on the physical side SupervisorId can be a self-referential key. Similarly, this is a foreign key to Store Manager because the Manager is just the highest-ranking employee.

Let’s now look at the bridge table MovieStore. This is a weak entity with a composite key. To make it into a strong entity, I would push MovieNumberId and StoreNumberId into the body of MovieStore and give MovieStore a primary key of MovieStoreId.

Moving down to Movie, I notice that there is MovieStar1 Name and MovieStar2 Name. In other words, a movie can have many stars and a star can have many movies. So, we should create a Star entity with StarId as its primary key and StarName as its attribute and StarMovie as a foreign key to the Star attribute in the Movie Table. The Movie and Star entities are in a many to many relationships. When you join the two tables on this foreign key, you can determine whether a movie star starred in the movie or not.

Now let’s look at the physical. In the physical, we are dealing with tables and columns. Because in the logical, we have already used those conventions, table names are easily converted. And the columns, instead of having an underscore separating them, would just need to have the spaces between them collapsed. But the table names should be fully qualified with schema.tablename, so for example, Movie.Payment. Lastly, because we have already done the thinking in the logical model, we just need to make a few more adjustments in the physical. For example, the attributes that we decided could be optional can be null, like social security number in the Person table. And because we are now on computers, we might have to adjust some of the data types. For example, using the business rule from the logical--an EmployeeNumberId because it is assigned by the company, does not necessarily have to be variable; it can be char which can save some space and if it’s an Id, it can just be a non-negative integer.

Those are a few of the changes I would make to modernize the eMovies database. Thank you.