

Contents

Iteration Introduction	2
Adding Attributes to the DBMS Physical ERD	4
Driving Questions	4
Example of Adding Attributes to the Car EERD	5
Car EERD With Attributes	7
Adding Attributes to Your DBMS Physical ERD	7
TrackMyBuys Attributes	7
Normalizing DBMS Physical ERDs	10
Normalizing Your DBMS Physical ERD	11
TrackMyBuys Normalization	11
SQL Scripts	13
Oracle Scripts	14
SQL Server Scripts	17
Postgres SQL Scripts	21
Tying it Together	24
Creating Tables from a DBMS Physical ERD	24
Car EERD With Attributes	25
Car CREATE TABLE Statement	25
Ferrari CREATE TABLE Statement	26
Car Sequence Creation	26
Creating your Tables, Columns, Constraints, and Sequences	26
TrackMyBuys Create Script	27
Summary and Reflection	27
TrackMyBuys Summary and Reflection	28
Items to Submit	28
Evaluation	29

Iteration Introduction

In Iteration 3, you added specialization-generalization to your design, and started on a detailed design by creating an initial DBMS physical ERD. In this iteration, you add attributes to your design, normalize your tables to BCNF, then create your tables, constraints, and sequences in SQL. Your database structure comes to life in this iteration!

To help you keep a bearing on where you have been and where you are headed, let's again look at an outline of what you created in prior iterations, and what you will be creating in this and future iterations.

Prior Iterations	Iteration 1	<p><i>Project Direction Overview</i> – You provide an overview that describes who the database will be for, what kind of data it will contain, how you envision it will be used, and most importantly, why you are interested in it.</p> <p><i>Use Cases and Fields</i> – You provide use cases that enumerate steps of how the database will be typically used, also identify significant database fields needed to support the use case.</p> <p><i>Summary and Reflection</i> – You concisely summarize your project and the work you have completed thus far, and additionally record your questions, concerns, and observations, so that you and your facilitator or instructor are aware of them and can communicate about them.</p>
	Iteration 2	<p><i>Structural Database Rules</i> – You define structural database rules which formally specify the entities, relationships, and constraints for your database design.</p> <p><i>Conceptual Entity Relationship Diagram (ERD)</i> – You create an initial ERD, the universally accepted method of modeling and visualizing database designs, to visualize the entities and relationships defined by the structural database rules.</p>
	Iteration 3	<p><i>Conceptual Extended Entity Relationship Diagram (EERD)</i> – You add specialization-generalization into your conceptual ERD and structural database rules.</p> <p><i>Initial DBMS Physical ERD</i> – You create an initial DBMS physical ERD, which is tied to a specific relational database vendor and version, with SQL-based constraints and datatypes.</p>
Current Iteration	Iteration 4	<p><i>Full DBMS Physical ERD</i> – You define the attributes for your database design and add them to your DBMS Physical ERD.</p> <p><i>Normalization</i> – You normalize your DBMS physical ERD to reduce or eliminate data redundancy.</p> <p><i>Database Structure</i> – You create your tables, sequences, and constraints in SQL.</p>
Future Iterations	Iteration 5	<p><i>Reusable, Transaction-Oriented Store Procedures</i> – You create and execute reusable stored procedures that complete the steps of transactions necessary to add data to your database.</p> <p><i>Questions and Queries</i> – You define questions useful to the organization or application that will use your database, then write queries to address the questions.</p> <p><i>Index Placement and Creation</i> – To speed up performance, you identify columns needing indexes for your database, then create them in SQL.</p>
	Iteration 6	<p><i>History Table</i> – You create a history table to track changes to values, and develop a trigger to maintain it.</p> <p><i>Data Visualizations</i> – You tell effective data stories with data visualizations.</p>

Before you proceed, it's critical to revise what you completed in prior iterations, if necessary. If there are issues with your database design or with your initial DBMS physical ERD, correct it now. You are implementing your design in SQL this week, and so changes will become more expensive; it's better to make them now rather than later.

Adding Attributes to the DBMS Physical ERD

After all of the relationships in the conceptual ERD are mapped to a DBMS physical ERD, the next step is to add attributes to the entities. After all, the primary purpose of a database is to provide long-term data storage, so a database that contains only primary and foreign keys, but no other fields, is not useful. The structural database rules, the initial conceptual ERDs, and initial DBMS physical ERDs you have seen thus far have not identified any attributes. The focus thus far has been on the entity and relationship structure, because you generally want to create that structure first before filling in the details. We need to learn more about adding attributes.

Just as with the entity and relationship structure, we rely on analysis of the organization and on how the application will be used to know what attributes we need. However, adding attributes is much less structured than creating the entity and relationship structure. We rely on one somewhat loose principle: *add the attributes that support the data the organization needs based on how the database will be used.* We do not need to create a list of detailed rules which identify the attributes first, and we do not need to add the attributes to the conceptual ERD first.

Because we are working with a DBMS physical ERD, we are now very close to implementing the SQL in our chosen database. Therefore, adding attributes requires involved knowledge of how databases use data, and exactly what the database will store. In contrast, the conceptual ERD only requires us to understand the organization and database usage at a higher level. Adding attributes is an iterative process and requires paying attention to specific details.

When you create a database for an organization, you may be assisted by analysts who help perform the organizational analysis. In that scenario, you and the analysts have many discussions, and the analysts answer many of your questions, to arrive at a suitable list of attributes. If you develop something by yourself or on a small team, such as a mobile application or other small project, you may not have the luxury of analysts; in that case you are both the designer and the analyst. In this course, you are your own team and play the part of answering your own design questions. You will need to do your own research, and make intelligent decisions.

Driving Questions

Here are some questions you can ask yourself to decide what attributes you need.

Question	Reasoning
What fields do other similar applications and databases store?	Taking a look at similar applications and databases can give you many ideas for attributes for your own database. For example, if you are creating a database that tracks people's workouts, then take a look at existing websites and mobile applications that help track people's workouts. There will be many fields you can identify from doing so.
What fields are obvious for my entities?	There are many obvious attributes for many entities. For example, if you have any kind of Person entity, you will likely

	need name attributes such as FirstName and LastName. If you have any kind of address entity, you will likely need the standard fields such as Street1, City, and so on. Many entities need obvious fields.
What fields are unique for my database?	You may have very specific ideas about your database, how it will store just the data you need. You know about these attributes because you know what you intend. For example, imagine someone is creating a database to track recent stock performance by hour the purpose of selling in just the right hour. Such a database will have a field to track the hour of the day, which is somewhat unique to that database, since many databases don't track hours.
What would be presented on a user interface that uses my database?	Since end users do not directly interact with the database but an application, what kinds of screens would you envision for such an application? Identifying what fields would be on those screens will help you identify needed attributes for your database.
What fields do the use cases require?	You have already defined use cases and the significant fields they require, so you can also look there to determine what attributes your database needs.

As you can see from the questions above, adding attributes centers around the driving principle of adding the attributes that support the data the organization needs based on how the database will be used, but the process is not an exact science.

Example of Adding Attributes to the Car EERD

Let's take a look at adding attributes to Car EERD we use earlier. Recall that our structural database rule for this is "A car is a Ferrari, Aston Martin, Lamborghini, or none of these." Because we haven't any additional details of what we are trying to store beyond the structural database rule and ERDs, let's define that now. Let's assume that we are creating a database for a car reseller that specializes in selling used, high-end cars. Let's further assume that this is only a subset of the database, a subset that stores just the car information but not the other important entities such Customer, Purchase, and the like. Armed with this information, I can now run through the questions.

Question	Reasoning
What fields do other similar applications and databases store?	Based upon my experience with websites such as http://kbb.com , http://cars.com , and many others, there are many obvious fields we need to store. I visited these sites to refresh my memory. We need to store a VIN number and a price for each car, of course. We need to store a make and a model. We need to store car mileage because customers will want to know. We need to store color as well. These are the basic attributes any reseller would need. If I could meet with the reseller the database is for, I would likely come up with more fields, but for now I'll stick with these basic ones.
What fields are obvious for my entities?	
What fields are unique for my	I looked up information on the Ferrari, Aston Martin, and

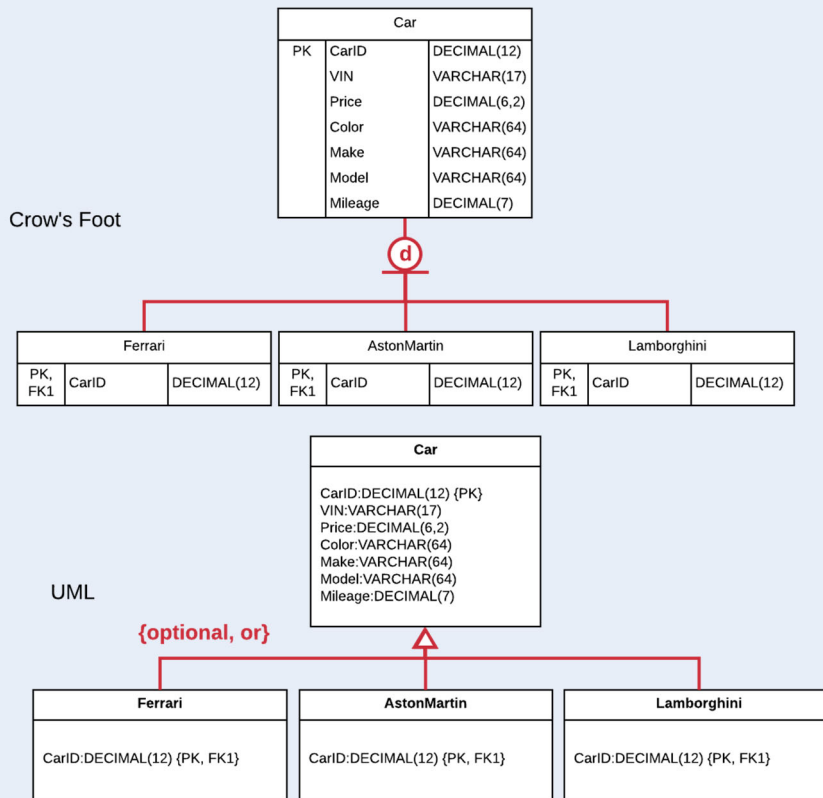
database?	Lamborghini lines, and while they each have unique items and properties in the real-world that customers care about, I could not identify useful fields to store about each of them in the database. Again, if I could meet with the reseller, I may come up with attributes to add here. I will stick with the more general fields already defined for the Car entity.
What would be presented on a user interface that uses my database?	I did not identify any additional fields from this question beyond what I have already included.

I've identified attributes I'd like to add, all of them to the Car entity, but there's still one more important step. I need to carefully select the datatype for each attribute, so that it will support storage of the data that will be in each field. Selecting the correct datatypes can take some research and time. I outline in a table below my datatype selections and reasoning for each attribute, along with example data for each to help the reader understand what kind of data fits within the table.

Attribute	Datatype	Reasoning	Example Data
VIN	VARCHAR(17)	A quick web lookup reveals that VIN numbers are 17 characters in length. Also, VIN numbers are a combination of numbers and uppercase characters. So I use a VARCHAR(17) to support this attribute.	1C4RJEAG8CC115980
Price	DECIMAL(6,2)	Even these expensive cars will not cost more than \$999,999, so I allow for 6 digits. Due to the prices of these cars, it is unlikely the reseller would like to use decimal points in their prices such as \$99,999.99; however, I cannot say for sure, so I allow for the standard two points after the decimal.	175,000.00
Color	VARCHAR(64)	I use VARCHAR(64) for color just in case there is a really long color name I am not aware of. Since VARCHAR does not take up storage for unused characters, this will not harm the database. I can see that the colors will be duplicated in this attribute, so this would be normalized out of the table during normalization, but in this part of the design it is in the table as is.	Giallio Modena Solid
Make	VARCHAR(64)	The same as with Color, I use VARCHAR(64) in case there is a long make name.	Ferrari
Model	VARCHAR(64)	I use VARCHAR(64) here in case there is a long model name.	LaFerrari
Mileage	DECIMAL(7)	While it's unlikely any car being sold will have over 999,999 miles, I allow for 7 digits just in case.	125,000

Now that I've identified the attributes and their datatypes, I can now enhance the Car EERD by adding the attributes into the DBMS physical ERD. The updated ERD is diagrammed below.

Car EERD With Attributes



The primary and foreign keys I mapped previously are still present in the diagram. The additional attributes for Car are added in addition to the existing keys. You can see this diagram taking shape now. Further, you can envision how we could create SQL statements to create the four tables in the diagram, along with all of their columns and keys. Exciting!

Adding Attributes to Your DBMS Physical ERD

At this point, you know enough to add the attributes important to your database to your physical ERD. Add significant attributes to each entity now, making sure to provide reasoning why you selected the attributes and datatypes. Below is a process I go through to add attributes to the TrackMyBuys DBMS physical ERD.

TrackMyBuys Attributes

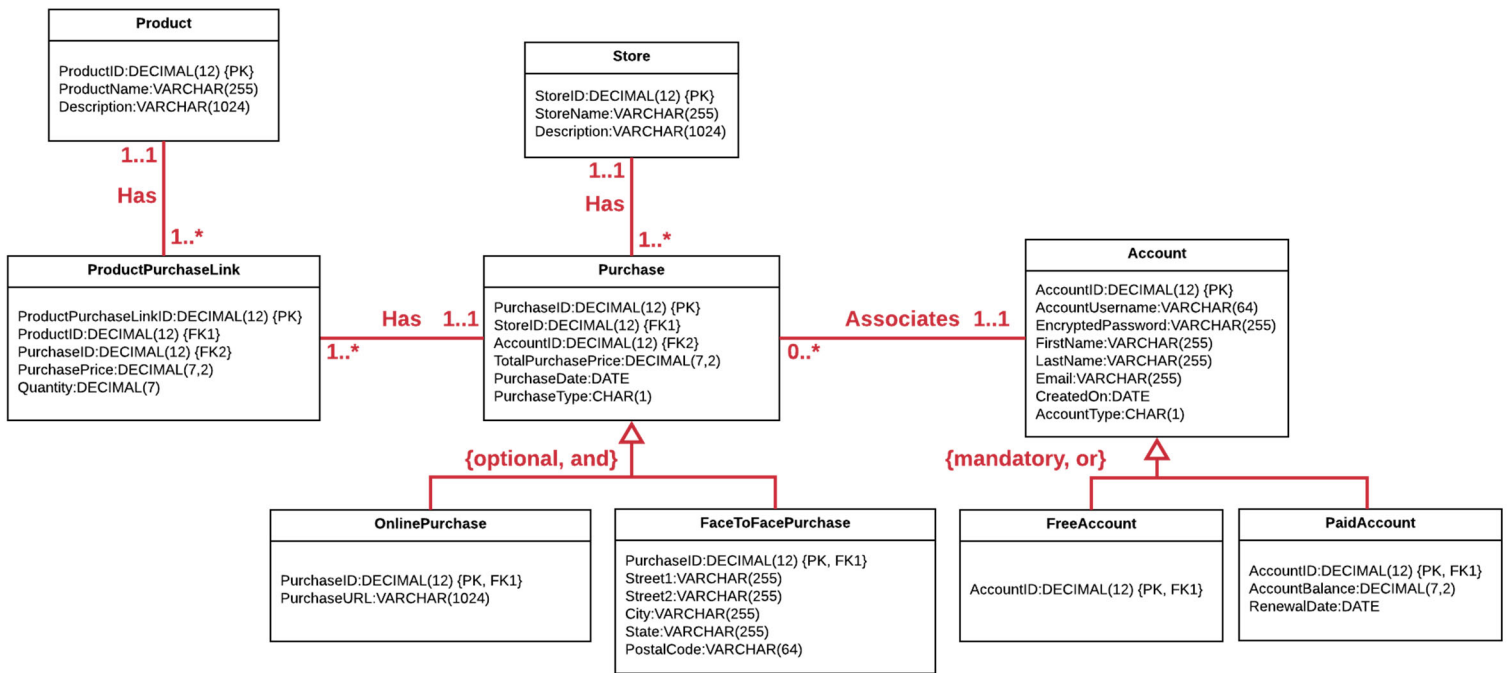
I am now going through the process of adding attributes and their datatypes table-by-table. My choices and reasoning are in the table below.

Table	Attribute	Datatype	Reasoning	Example Data
Product	ProductName	VARCHAR(255)	Every product has a name which acts like the identifier for the product when people are looking it up in TrackMyBuys. I allow for up to 255 characters just in case of something extraordinary.	Fuzzy Sweater
Product	ProductDescription	VARCHAR(1024)	Every product may have a description. People may want to describe the product more than just with the name. I allow for 1,024 characters so that people can type in something long if they need to.	The coziest sweater in the universe.
ProductPurchaseLink	PurchasePrice	DECIMAL(7,2)	When people buy a product, they buy it at a price. Though the price can change over time, there is a specific price they buy it at, which may include discounts. Although it's unlikely a price will ever be recorded in the millions, I allow for up to 7 digits. I also allow for the standard two decimal points.	79.99
ProductPurchaseLink	Quantity	DECIMAL(7)	When people buy a product, they can buy one or more of them. I allow for up to 7 digits as a safe upperbound.	1
Purchase	TotalPurchasePrice	DECIMAL(7,2)	When people make a purchase, the purchase can include one or more products. This attribute captures the total price including all products purchased. I allow for up to 7 digits as a safe upperbound, and the standard two decimal points.	79.99
Purchase	PurchaseDate	DATE	A purchase happens on a specific date.	10/23/2022
Purchase	PurchaseType	CHAR(1)	In a prior iteration, I indicated that there can be at least two types of purchases, online and face-to-face. This attribute is the subtype discriminator to indicate which it is.	O (for "Online")
Account	AccountUsername	VARCHAR(64)	Every account has a username	basechomp

			associated with it, which will be used to login to TrackMyBuys. I allow usernames to be up to 64 characters.	
Account	EncryptedPassword	VARCHAR(255)	Every account has a password. It will be stored in encrypted text format in the database. 255 characters should be a safe limit to store encrypted text.	xyz
Account	FirstName	VARCHAR(255)	This is the first name of the account holder, up to 255 characters of the name.	Apollo
Account	LastName	VARCHAR(255)	This is the last name of the account holder, up to 255 characters of the last name.	Ikaros
Account	Email	VARCHAR(255)	This is the email address of the account holder. 255 characters should be a safe upperbound.	Apollo.Ikaros@gmail.com
Account	AccountType	CHAR(1)	As identified in a prior iteration, there are two types of accounts – free and paid. This attribute is the subtype discriminator indicating which it is.	P (for “Paid”)
PaidAccount	AccountBalance	DECIMAL(7,2)	This is the unpaid balance, if any, for the paid account. I allow for up to 7 digits, though it will likely never get near this high.	0.00
PaidAccount	RenewalDate	DATE	This is the date on which the account needs to be renewed with a new payment.	12/19/2022

I feel I have captured all of the necessary fundamental attributes for TrackMyBuys in the table above. I see that I could be more detailed with storing account information balances and payment information. But given the use cases and structural database rules I’ve developed thus far, these attributes suffice.

Here is my ERD with the attributes included.



Each of the attributes have been added to their respective entities in the ERD. The previously added primary and foreign keys have also been retained. One item worth noting is that I did not identify any attributes necessary for the FreeAccount entity at this time. I may identify some as the application is further developed.

While there is room for more detail, I feel that this is a solid DBMS physical ERD for the use cases and structural database rules I have added thus far in the design.

Normalizing DBMS Physical ERDs

Once a conceptual ERD is mapped to a DBMS physical ERD, and attributes are added, there is only one more necessary step before implementation in SQL can occur – normalization. We want to normalize our entities before they are implemented to minimize data redundancy in our database. Recall that normalization is a formal process of eliminating data redundancy. Further recall that normalization works by identifying dependencies between columns, and moving keys and the values they determine into their own tables.

The scope and technical complexity on how to perform normalization from scratch is too broad and deep to be a part of this document. Please use the textbook, online lectures, and live classroom sessions to learn about normalization, and the steps to follow to normalize a table. Keep in mind that it is best practice to normalize tables to BCNF when possible. If a table is not normalized to BCNF for specific reasons, we should be aware of those reasons and make a conscious choice to do so.

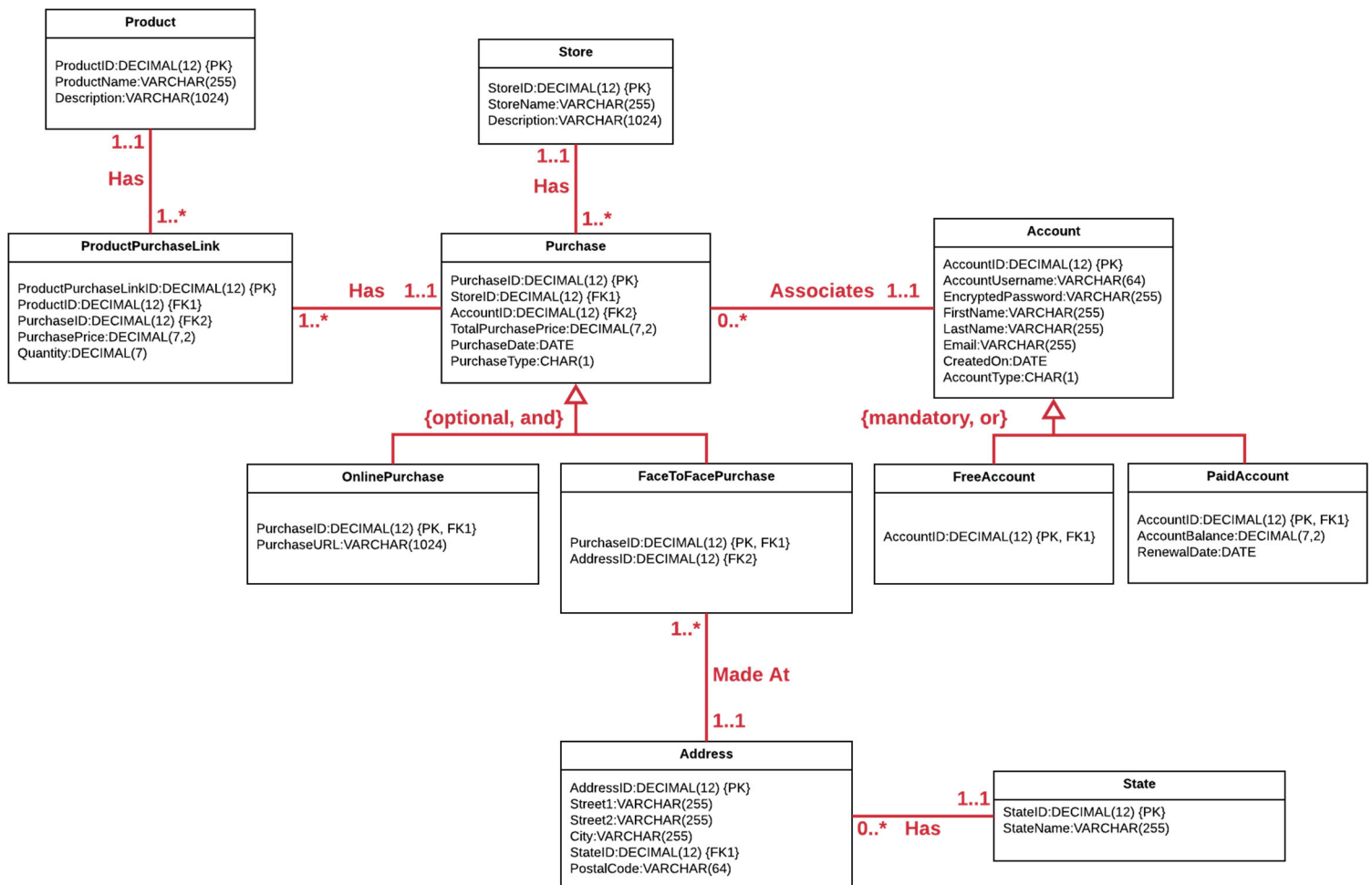
While normalization is performed on the DBMS physical ERD, it affects the conceptual ERD as well as the structural database rules. Why? Because normalization results in additional entities and relationships between those entities. Since it's important to keep the structural database rules, conceptual ERD, and DBMS physical ERD in sync, normalizing the DBMS physical ERD also results in changes to the structural database rules and the conceptual ERD.

Normalizing Your DBMS Physical ERD

You are very close to implementing your database in SQL. All that's left is to normalize your entities in your DBMS physical ERD, and update your conceptual ERD and structural database rules to match. Do so now by normalizing your DBMS physical ERD. Here is an example with TrackMyBuys.

TrackMyBuys Normalization

I notice only one place where there is redundancy in my physical ERD, and that is with the address information in the FaceToFacePurchase entity. If many purchases are made at that same face-to-face store, the address information will repeat. Here is my ERD with the address information normalized.



There are two additional entities after normalization – Address and State. By moving the primary address information into its own entity, I do not need to repeat the address every time a purchase is made. I can reference the address instead. Likewise, rather than repeating the state name every time a purchase is made, the address entity references the state entity instead.

The Address entity is not normalized to BCNF. In fact, I could create more tables that further breakdown street into its number and street name and normalize out street name. I could associate cities and states with postal codes. I could break out city so it does not repeat. I chose

not to normalize address to full BCNF because it is not necessary for my database. That would add unnecessary complexity and add no real benefit.

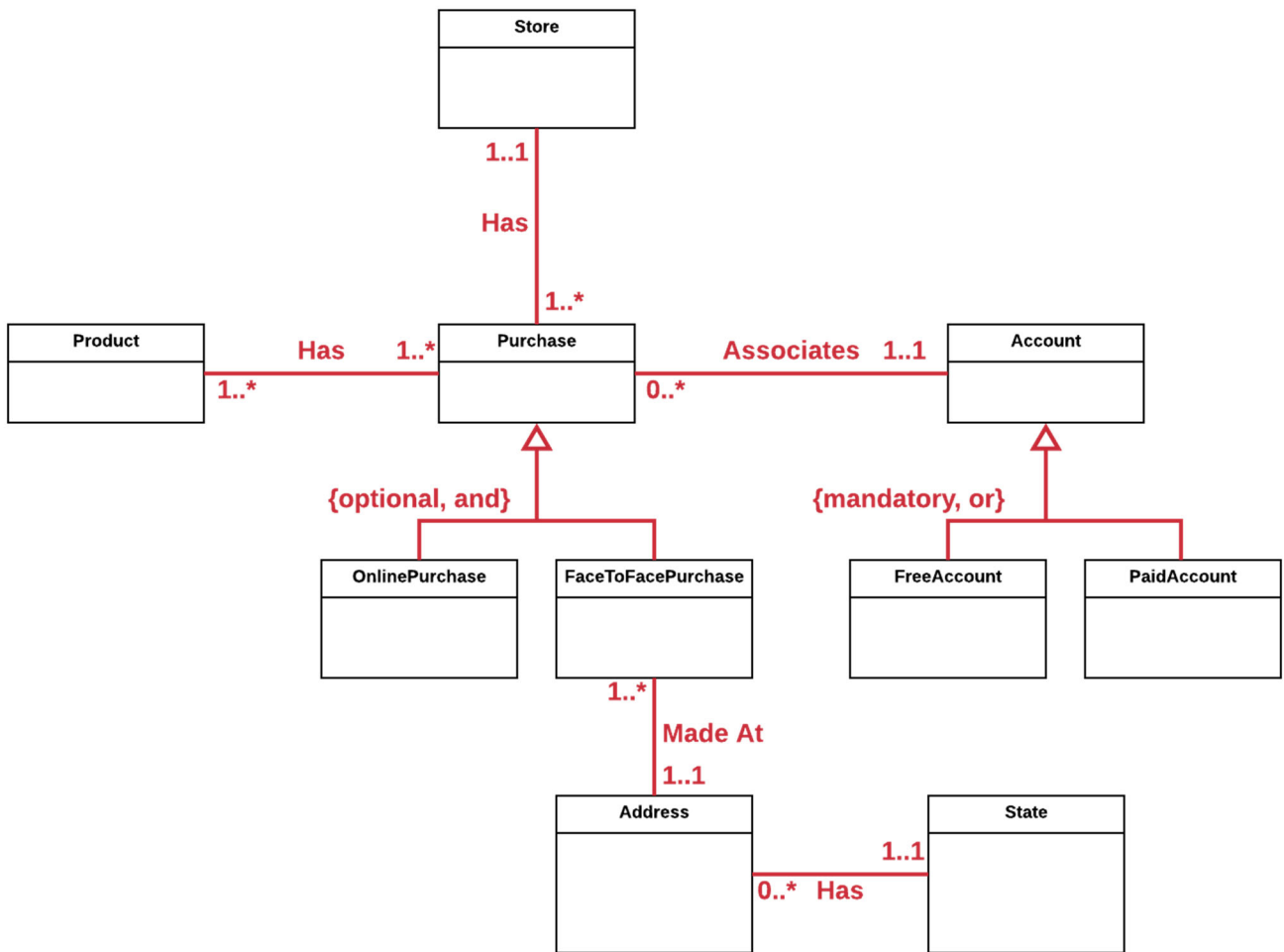
The State entity is normalized to BCNF.

Below are my structural database rules modified to reflect the new entities. The new ones are italicized.

1. Each purchase is associated with an account; each account may be associated with many purchases.
2. Each purchase is made from a store; each store has one to many purchases.
3. Each purchase has one or more products; each product is associated with one to many purchases.
4. An account is a free account or a paid account.
5. A purchase is an online purchase, a face-to-face purchase, both, or none of these.
6. *Each face-to-face purchase is made at an address; each address is associated with one or more face-to-face purchases.*
7. *Each address has a state; each state may be associated with many addresses.*

I added #6 and #7 to reflect the new Address and State entities.

Below is my new conceptual ERD to reflect the new entities.



The Address and State entities are now included in the conceptual ERD, and the conceptual ERD is in sync with the structural database rules and the DBMS physical ERD.

SQL Scripts

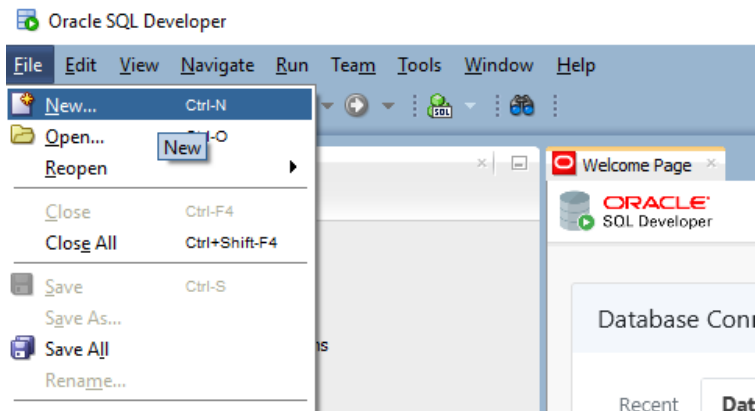
Before you dive into creating SQL code for your project, you need to know about saving your code in SQL scripts. A SQL script is simply a text file with a “.sql” extension at the end of the filename. It’s best practice to save your code in SQL scripts so that you can rerun the code anytime you need. For example, if you create several tables with SQL and save it in a script, then later decide you need to modify any of the table definitions, you could open up the script, make the modifications you need, and re-run the script. If you don’t have a script, you would need to recreate all of the SQL code again to make the modifications. The same goes for queries and stored procedures you create. You want to save those in scripts so you can reuse them as you need.

Saving and using SQL scripts is not difficult. In fact, the experience is very similar to using filed-based applications such as Word, Excel, Pages, or Numbers. To save a script, you type your code into the buffer window of your SQL client, then use the client’s save feature to save it as a SQL script. Later, if you want to use the script, use your client’s file open feature to open the script. Once a script is open, you can run the commands in it with the client’s execute options, the options you are already familiar with. The process is simple and familiar.

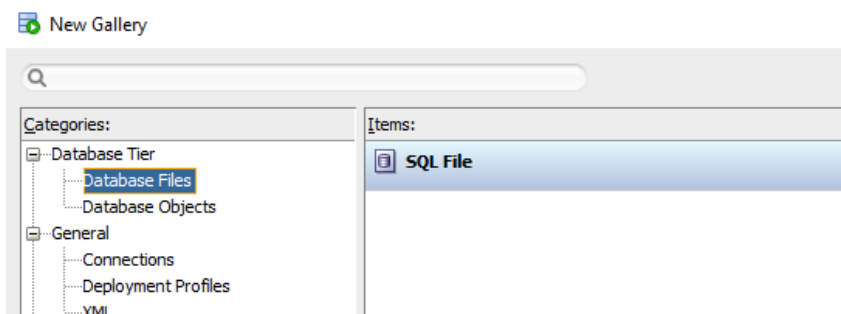
Let's take a look at saving and opening scripts for the course's supported databases. As we go through the examples, please keep in mind that as versions of the clients change, the precise names and locations of the menu options may change.

Oracle Scripts

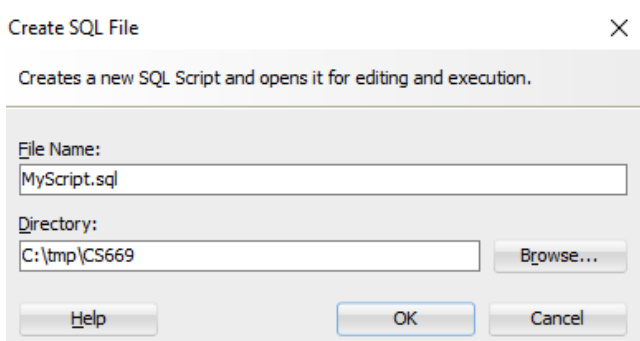
First, we'll start with Oracle in Oracle SQL Developer. First, open a buffer window by selecting File/New....



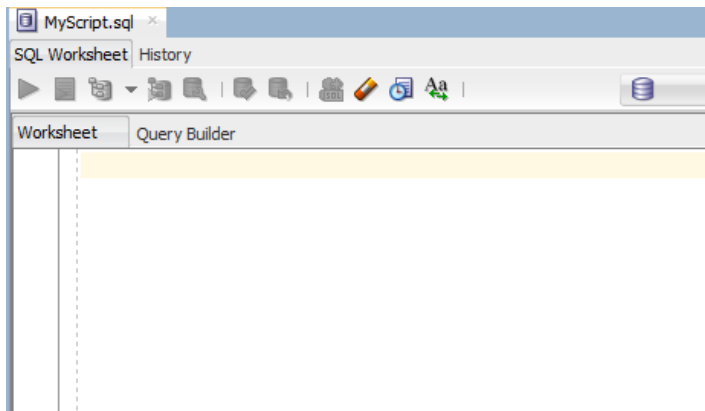
From there, select Database Files/SQL File and click the OK button.



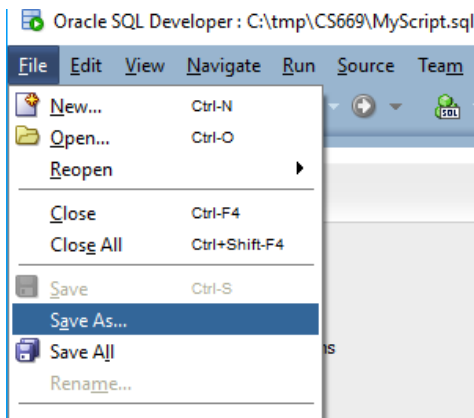
From there, give the file a name and choose a directory where you want to store the file. It's a good idea to store it in a directory you will remember. In the example below, I named my script "MyScript.sql", and put it into a C:\tmp\CS669 directory.



Once you click the OK button, a buffer (named a “SQL worksheet” in Oracle SQL Developer) appears where you can type SQL commands.

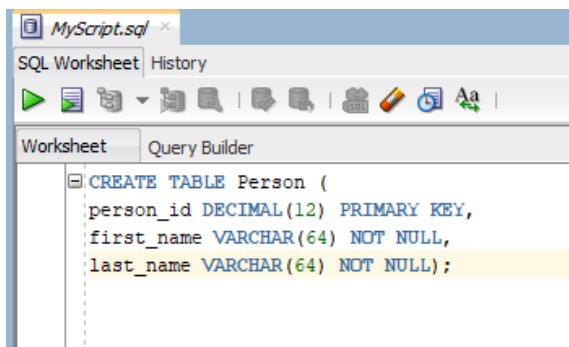


This is just one way to open a buffer window associated with a script. If you already have a buffer open and just want to save it to a script, you can select File/Save As... to save it to a script.

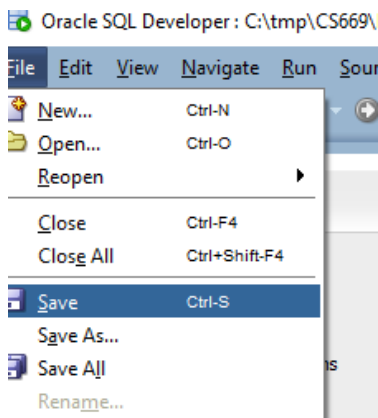


Either way, you’ve achieved the goal of opening a buffer window and tying that window to a specific SQL script saved on your drive.

Next, type in whatever SQL commands you need. In the example below, I add a simple create table statement for Person table, for illustrative purposes.



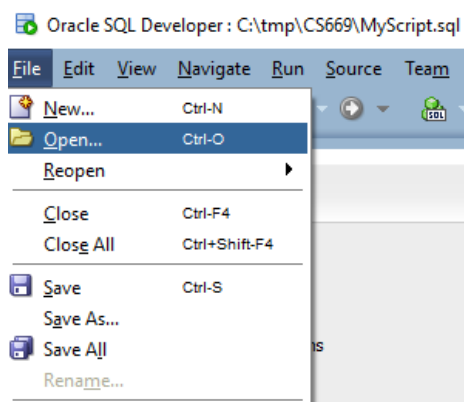
It’s a good idea to save regularly so you don’t lose work. You can either type Ctrl-S to save, or select File/Save....



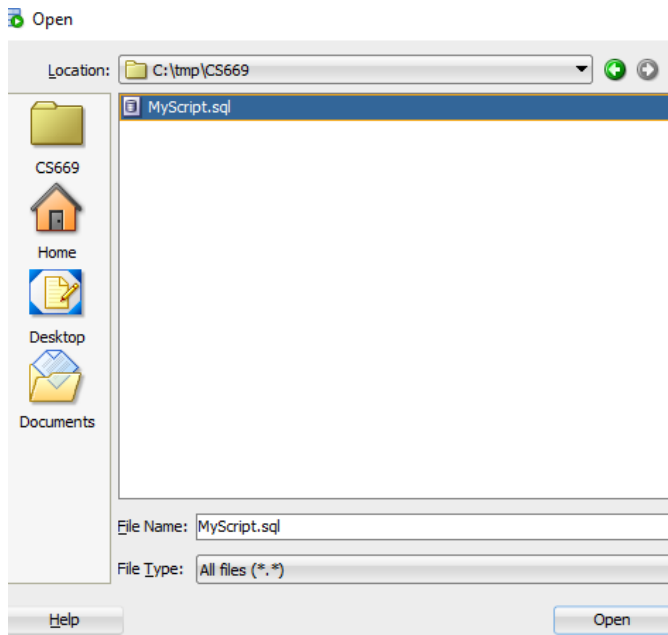
Either way, your latest commands will be saved to the file.

That's it! It's not too complex, and not much different than saving Word or Pages files. You are now saving your code to your drive and can work with it later, and will not lose your code the next time you close your client or restart your computer.

Later on, when you want to work with the script, you just use File/Open... to open up the script again.

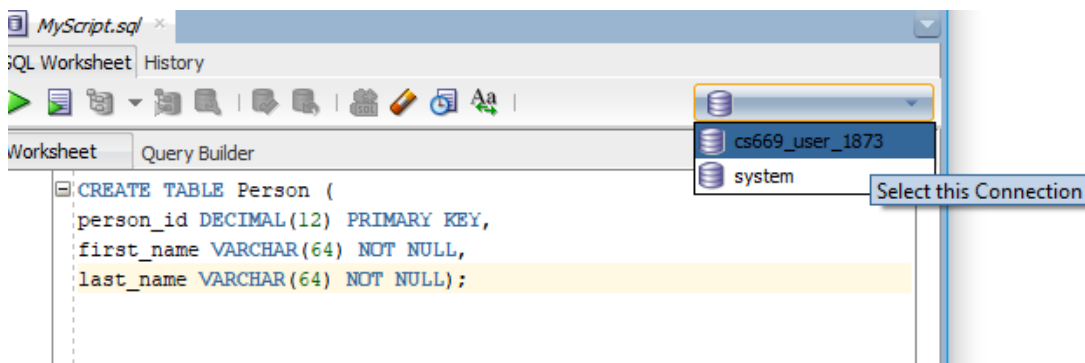


Select the file you want, and click the Open button.



And you'll see your file with your SQL commands in it again.

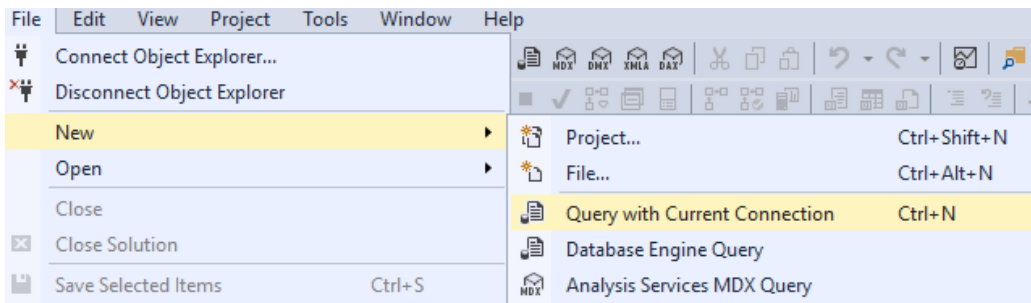
One other thing you need to know is that in order to execute your code, you need to tell Oracle SQL Developer which connection to use for it. On the right, there is a list of connections. Choose the one you want before executing your code.



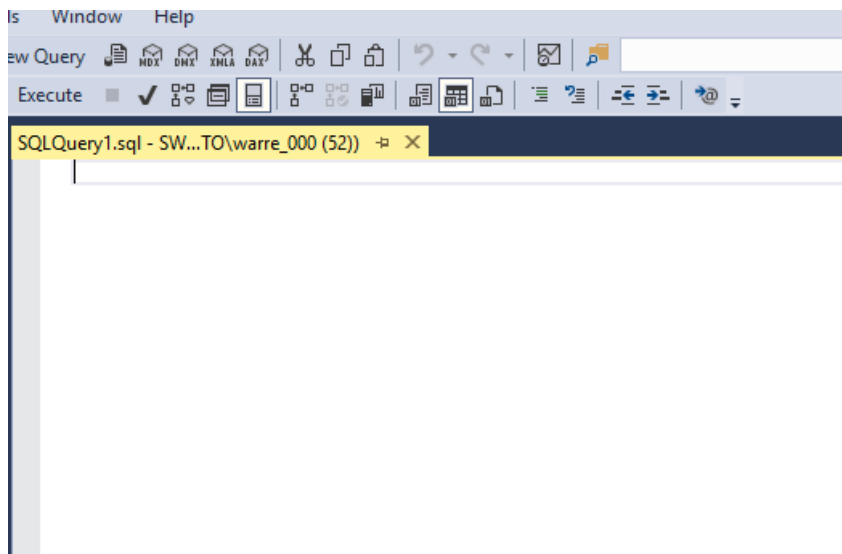
You can keep the same script open, and change connections if you need, before executing the commands therein.

SQL Server Scripts

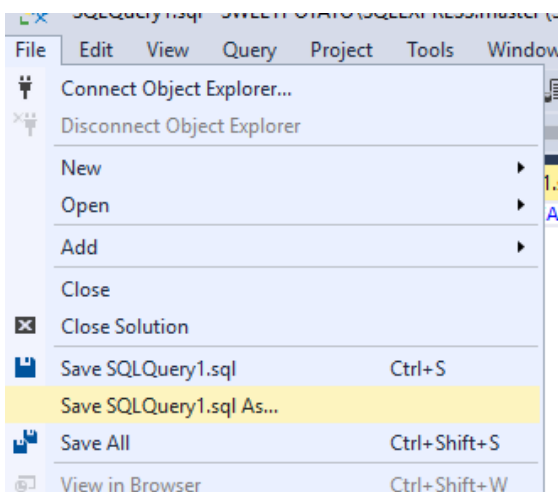
To open a buffer window with SQL Server Management Studio (SSMS), select File/New/Query with Current Connection.



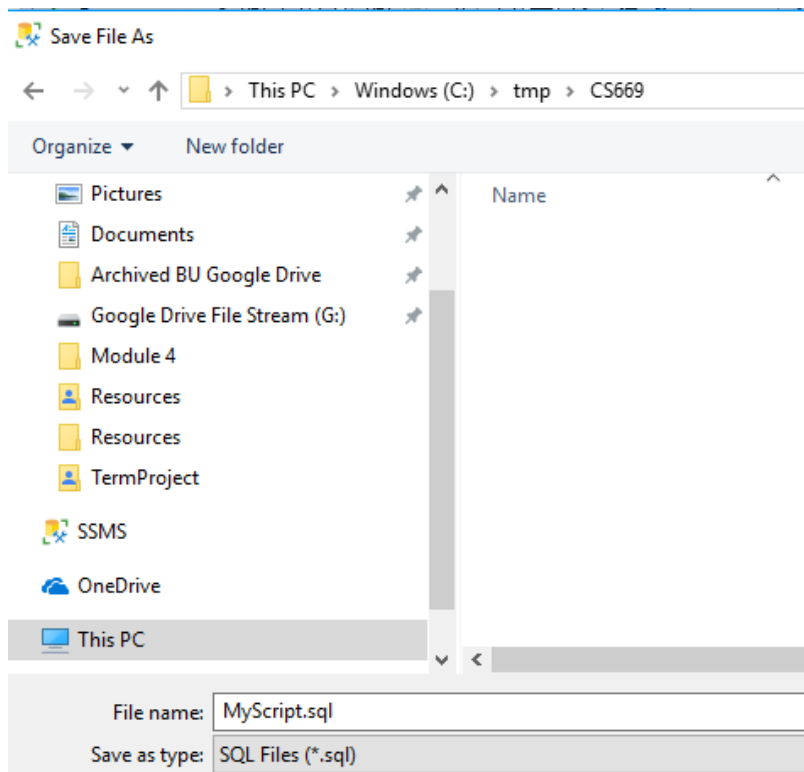
Note that when you start SSMS, it asks you to connect to a database, so you should already have a connection to your database. This command will create a new buffer window associated with that connection. You will see the buffer window such as the below.



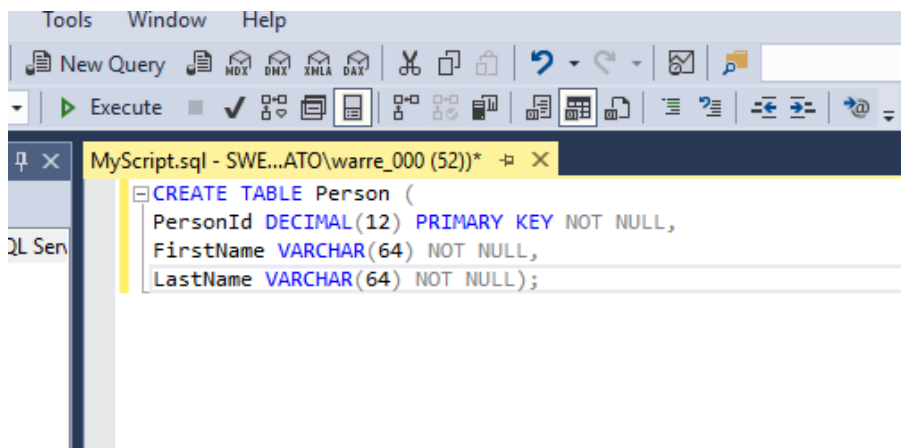
Next, you need to save the buffer to a file. To do so, click File/Save XYZ.sql As..., where XYZ is the default name of the script chosen by SSMS.



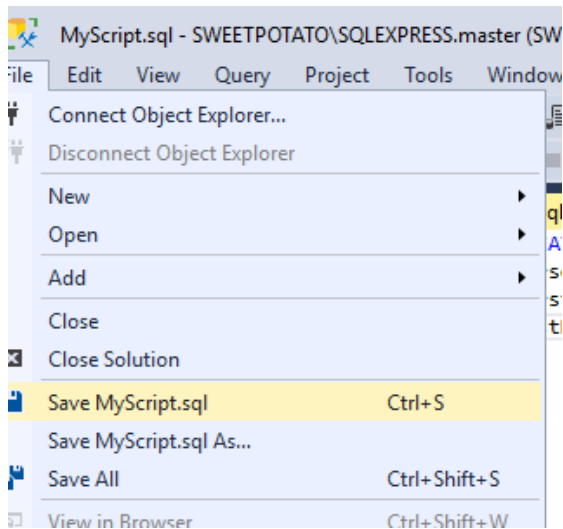
Then save it into a directory with the filename of your choosing. I chose to save it under C:\tmp\CS669\MyScript.sql, shown below.



At this point, the buffer is now associated with the file. Type in whatever SQL commands you need. In the example below, I add a simple create table statement for Person table, for illustrative purposes.

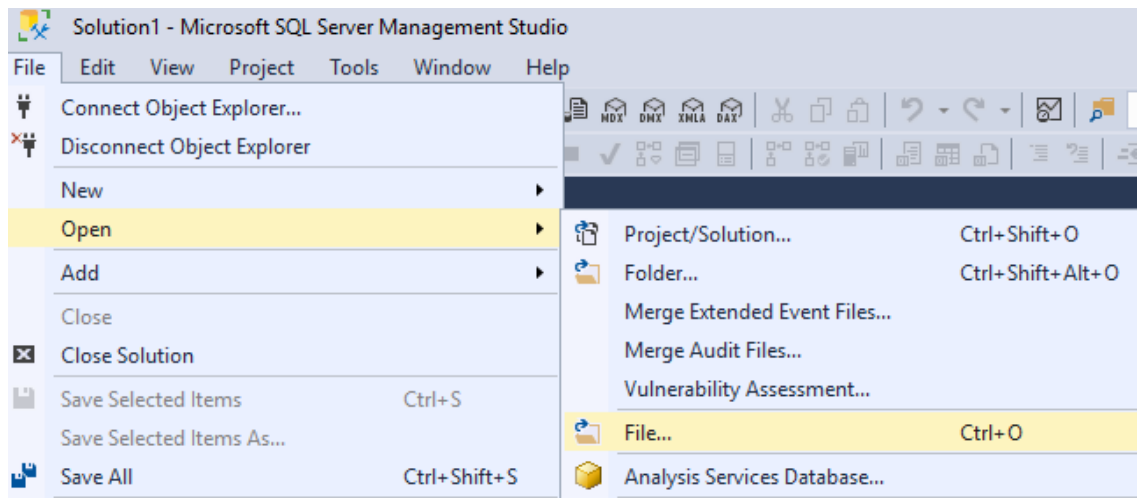


It's a good idea to save regularly so you don't lose work. You can either type Ctrl-S to save, or select File/Save MyScript.sql....

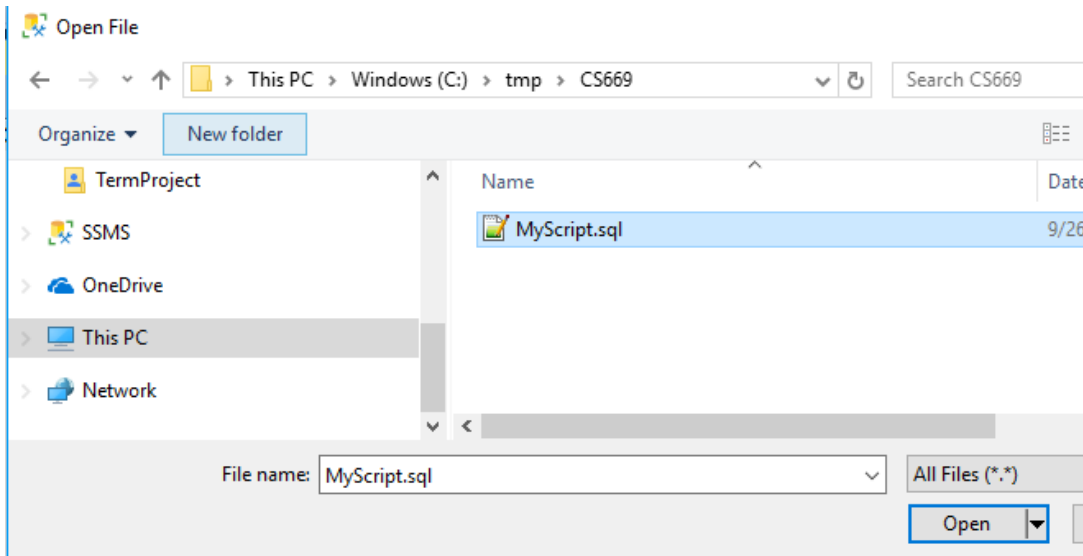



That's it! It's not too complex, and not much different than saving Word or Pages files. You are now saving your code to your drive and can work with it later, and will not lose your code the next time you close your client or restart your computer.

Later on, you may need to open up your script to continue working with it. To do so, click File/Open/File...



You will then need to navigate to your saved script and click the Open button.



After that, the script will be open again, and you can edit and execute the commands as needed. To execute the commands, just use the  **Execute** button like you have already been doing in the labs.

Postgres SQL Scripts

The first step in creating a SQL script in the pAdmin tool is to connect to your database, because queries and scripts cannot be created otherwise. Open up pgAdmin and expand the PostgreSQL tree item first. When you do so, it will prompt you to enter your password that you chose during installation. Enter the password and confirm to continue.

Connect to Server

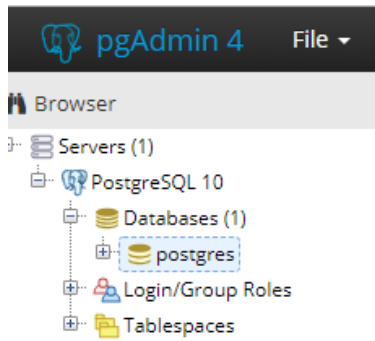
Please enter the password for the user 'postgres' to connect the server -
"PostgreSQL 10"

Password

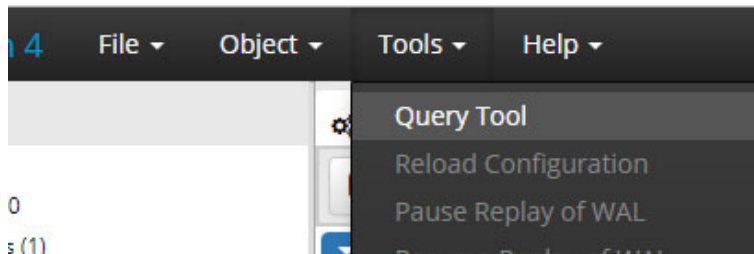
☐ Save Password

OK **Cancel**

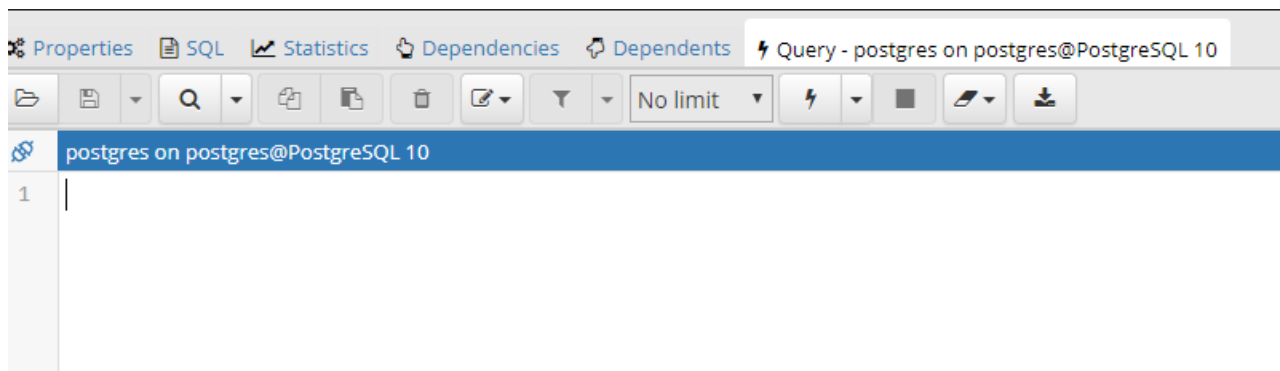
Once connected, you need to expand Databases and select the database you want to create the script for.



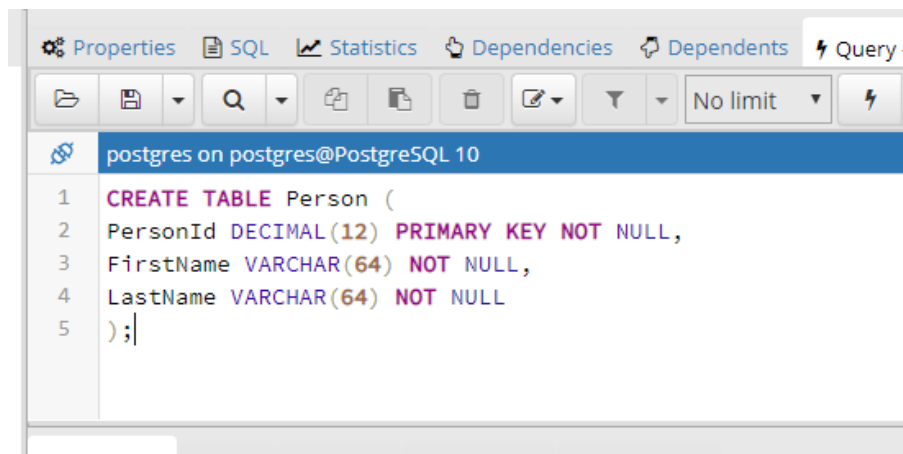
Once selected, the Tools/Query Tool option becomes enabled. Select it.




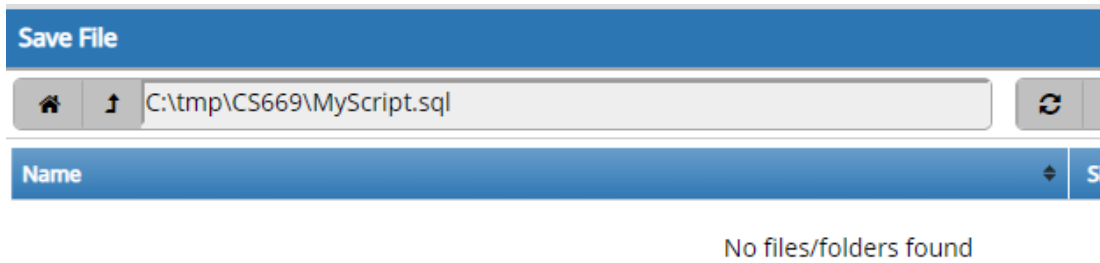
A buffer window appears where you may work with your SQL.




With pgAdmin, you first need to type something into the buffer before you can save it to a script. For illustrative purposes, I type in a command to create a simple Person table.




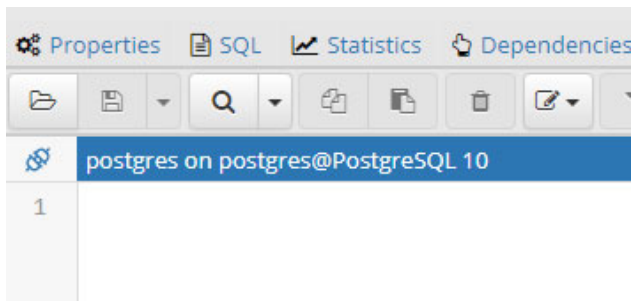
To save the buffer as a script, click the  icon. Then select the location and filename for the script. I opted for C:\tmp\CS669\MyScript.sql, as shown below.



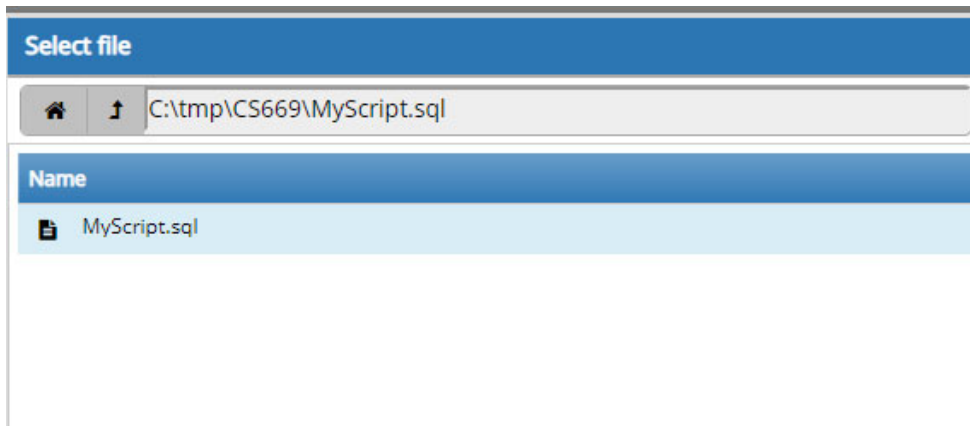
Make sure the click the Save button, and then your buffer will now be associated with this file. pgAdmin will tell you it successfully saved the file, and will also change the title of the buffer to your filename.

That's it! It's not too complex, and not much different than saving Word or Pages files. You are now saving your code to your drive and can work with it later, and will not lose your code the next time you close your client or restart your computer. Make sure to save your work often so you don't lose anything. To do so, just click the  icon whenever you need to save changes.

Later on, you may need to open up your script to continue working with it. To do so, open the query tool again with File/Query Tool, then click the  icon to open a saved file. It's the leftmost icon in the toolbar.



Once you do so, a file dialog will appear, and you choose the file you previously saved.



When you click the Select button, the file will open once again for you to continue editing and executing.

Tying it Together

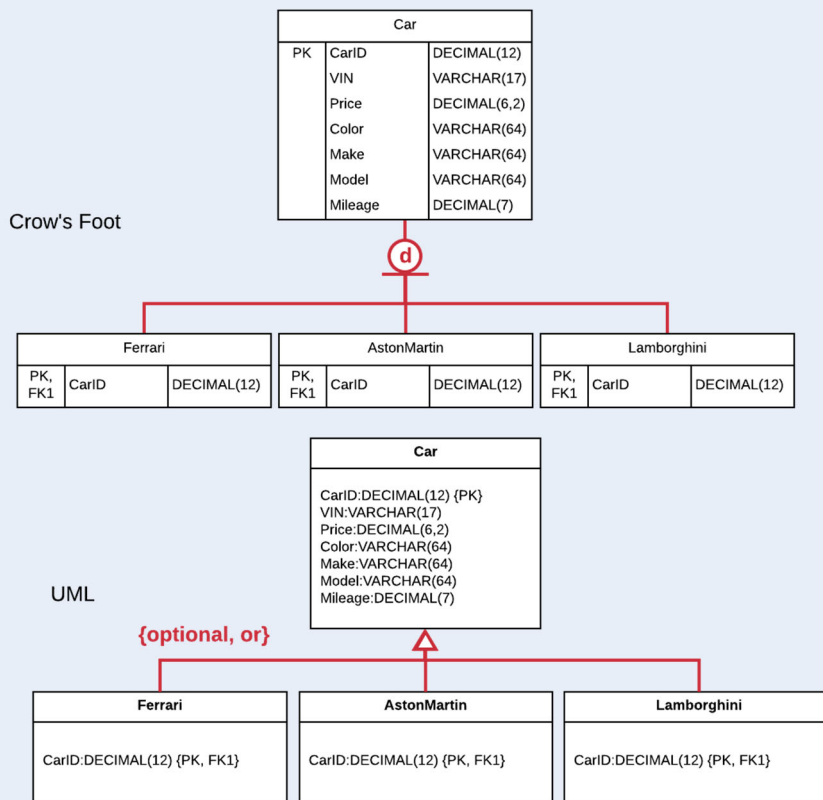
So how does all of this tie into your term project? Simply put, as you work on your SQL code, you want to save it off into a script so you can always edit and re-run them as needed. Imagine how frustrated you would be if you were to lose large amounts of SQL code, and you had to rewrite it all. You will have a SQL script containing your work, with a comment that briefly identifies and explains each section.

Creating Tables from a DBMS Physical ERD

As you may have predicted, it is straightforward to create the tables needed from a DBMS physical ERD. The entities in the ERD are created in SQL, one-for-one. Since the ERD already contains the attributes, datatypes, and constraints, creating the tables is a mostly mechanical process.

Let's again look at the car example we have been using in the iterations. Recall that the DBMS physical ERD looks as follows.

Car EERD With Attributes



To create the Car entity, we would use the following CREATE TABLE statement.

Car CREATE TABLE Statement

```

CREATE TABLE Car (
  CarID    DECIMAL(12) NOT NULL PRIMARY KEY,
  VIN      VARCHAR(17) NOT NULL,
  Price    DECIMAL(8,2) NOT NULL,
  Color    VARCHAR(64) NOT NULL,
  Make     VARCHAR(64) NOT NULL,
  Model    VARCHAR(64) NOT NULL,
  Mileage  DECIMAL(7) NOT NULL);
  
```

We use the entity name "Car" for the table name, and then create columns that are one-to-one matches of the entity's attributes. We add the primary key constraint to CarID as indicated, and NOT NULL to the attributes we know should not be null (which for this table is all attributes).

If we want to create the Ferrari table for example, it would look as follows.

Ferrari CREATE TABLE Statement

```
CREATE TABLE Ferrari (  
  CarID    DECIMAL(12) NOT NULL PRIMARY KEY,  
  FOREIGN KEY (CarID) REFERENCES Car(CarID));
```

Notice that the CREATE TABLE statement for Ferrari has a one-to-one mapping with its entity definition in the ERD, including the primary and foreign key constraints. There is only one attribute in the entity – CarID -- so only one attribute in the CREATE TABLE statement.

We also want to create one sequence for each table, the exception being tables whose primary keys consist solely of foreign keys. That is, any table that needs a new unique value for its primary key needs a sequence. Tables that are simply reusing primary keys from other tables do not need a sequence. In this example, *Car* needs a sequence, but the subtypes such as *Ferrari* do not, because their primary key is just a foreign key back to *Car*.

Below is the code to create the sequence for Car.

Car Sequence Creation

```
CREATE SEQUENCE car_seq START WITH 1;
```

As you can see, creating tables and sequences that correspond to entities is quite mechanical and straightforward, once the DBMS physical ERD is implemented well.

Creating your Tables, Columns, Constraints, and Sequences

Your task now is to create your term project SQL script, and add all table, column, constraint and sequence creations designed in your DBMS physical ERD. Use the one-to-one method previously illustrated. Note that to make the script re-runnable, it is best practice to add DROP TABLE and DROP SEQUENCE statements to the beginning of the script which drops all of the tables and sequences, then to follow that with the CREATE TABLE statements, ALTER TABLE statements if you are using them, and CREATE SEQUENCE statements. In this way, you can run the script over and over to drop all tables and sequences and re-create them, should you need to modify something. Make sure to comment this create section as well. Keep in mind that you will need to drop the tables in an order that honors the foreign key constraints. Sequences can be dropped in any order.

All of your tables should have associated sequences (except those whose primary key consists solely of foreign keys), so that all primary key and foreign key values will not be hardcoded.

When you submit this and future iterations, attach the SQL script. Please note that *it is required that you type the SQL yourself*. Do not use a tool to generate the SQL; such SQL will receive no credit. While many such tools exist, the SQL they produce is not perfect, and it is often necessary to add or edit SQL manually.

Capture a screenshot of the execution of the table and sequence creations. There's no need to capture all of the output in this instance. A single screenshot demonstrating the creations executed will suffice.

Below is an excerpt for TrackMyBuys. Note that in the excerpt, I do not exhaustively create all tables, but create enough to provide an example for you to follow. You should create all of your tables.

TrackMyBuys Create Script

Here is the topmost snippet of my TrackMyBuys create script. In this example, you can see the DROP TABLE and DROP SEQUENCE commands, followed by the first CREATE TABLE. The remainder of the creations are off the screen. Note that I am using SQL Server for TrackMyBuys.

```

DROP TABLE BalanceChange;
DROP TABLE FreeAccount;
DROP TABLE PaidAccount;
DROP TABLE Account;
DROP TABLE Address;
DROP TABLE State;
DROP SEQUENCE BalanceChangeSeq;
DROP SEQUENCE FreeAccountSeq;
DROP SEQUENCE PaidAccountSeq;
DROP SEQUENCE AccountSeq;
DROP SEQUENCE AddressSeq;
DROP SEQUENCE StateSeq;

CREATE TABLE Account (
    AccountID DECIMAL(12) NOT NULL PRIMARY KEY,
    AccountUsername VARCHAR(64) NOT NULL,
    EncryptedPassword VARCHAR(255) NOT NULL,
    FirstName VARCHAR(255) NOT NULL,
    LastName VARCHAR(255) NOT NULL,
    Email VARCHAR(255) NOT NULL,
    CreatedOn DATE NOT NULL,
    AccountType CHAR(1) NOT NULL);

```

I put my DROP TABLE and DROP SEQUENCE commands at the top so that the script is re-runnable, then follow it with the CREATE TABLE and CREATE SEQUENCE commands. All columns, constraints, and sequences are included as illustrated in the ERD.

Summary and Reflection

You have created all of your tables based off of a normalized DBMS physical ERD, and indexed your tables. Your database structure is now implemented in SQL, and your tables are primed to be filled with data. Great work! Just in the next iteration, you will be inserting data into these tables and writing useful queries against the data. Update your project summary to reflect your new work.

It's a common phenomenon that, once you start implementing your design in SQL, both the structures that work well as well as the structures that need improvement become acutely obvious. Write down

your questions, concerns, or observations you have about this, and other areas, so that you and your facilitator or instructor are aware of them.

Here is an updated summary as well as some observations I have about my progress on TrackMyBuys for this iteration.

TrackMyBuys Summary and Reflection

My database is for a mobile app named TrackMyBuys which records purchases made across all stores, making it the one stop for any purchase history. Typically, when a person purchases something, they can only see their purchase history with that same vendor, and TrackMyBuys seeks to provide a single interface for all purchases. The database must support a person entering, searching, and even analyzing their purchases across all stores.

The structural database rules and conceptual ERD for my database design contain the important entities of Store, Product, Purchase, and Account, as well as relationships between them. The design contains a hierarchy of Purchase/FaceToFacePurchase and Purchase/OnlinePurchase to reflect the two primary ways people purchase products. The design also contains a hierarchy of Account/PaidAccount and Account/FreeAccount to reflect the fact that people can signup for a free account or a paid account for TrackMyBuys. The DBMS physical ERD contains the same entities and relationships, uses the best practice of synthetic keys, and contains the important attributes needed by the database to support the application. A history of balances has been created as well as a query which tracks changes to balances for a specific month.

The SQL script that creates all tables and sequences follows the specification from the DBMS physical ERD exactly. Important indexes have been created to help speed up access to my database and are also available in an index script.

It's thrilling to create the tables and columns my database needs. My design is no longer abstract, but actually implemented in a real, modern database. I can't wait to add data to it and query it!

My database structure seems to map OK to SQL. I suppose I will know for sure next iteration, once I start putting data into the database.

Items to Submit

In summary, for this iteration, you start by updating your design document by revising sections from prior iterations. You then add attributes to your design, normalize your tables to BCNF, then create your tables, constraints, and sequences in SQL. Make sure to use the templates provided with this iteration to ensure you are submitting all necessary items.

MET CS 669 Database Design and Implementation for Business
Term Project Iteration 4

Evaluation

Your iteration will be reviewed by your facilitator or instructor with the criteria outlined in the table below. Note that the grading process:

- involves the grader assigning an appropriate letter grade to each criterion.
- uses the following letter-to-number grade mapping – A+=100,A=96,A-=92,B+=88,B=85,B-=82,C+=88,C=85,C-=82,D=67,F=0.
- provides an overall grade for the submission based upon the grade and weight assigned to each criterion.
- allows the grader to apply additional deductions or adjustments as appropriate for the submission.
- applies equally to every student in the course.

Criterion	What it Means	A+ Excellent	B Good	C Fair/Satisfactory	D Insufficient	F Failure
Sufficiency of Attributes (15%)	This measures how essential the attributes are to the needs of the database, the coverage (thoroughness) of the attributes, and how well justified the attributes are with explanations. Excellent solutions define important attributes for all entities, and are well justified.	Entirely essential Full coverage Well justified	Mostly essential Good coverage Justified	Secondary but useful Partial coverage Partially justified	Mostly unnecessary Insufficient coverage Insufficient justification	No attributes or Entirely unnecessary Unacceptable coverage Unacceptable justification
Attribute Datatypes (10%)	This measures how appropriate the datatypes are for the needs of the data. Excellent attributes are entirely appropriate for the needs of the data, and have entirely appropriate precision, scale, and other limits.	Entirely appropriate	Mostly appropriate	Somewhat appropriate	Mostly inappropriate	No attributes or Entirely inappropriate
Attribute Naming (10%)	This measures how appropriate the attribute names are given their purpose, and how usable the names are in modern relational databases. Excellent names identify their purpose very well, and have only characters that can legally be used in a modern relational database.	Entirely appropriate Entirely usable	Mostly appropriate Mostly usable	Somewhat appropriate Somewhat usable	Mostly inappropriate Mostly unusable	No attributes or Entirely inappropriate Entirely unusable

DBMS Physical ERD Quality (30%)	This measures how accurately the DBMS physical ERD represents the database design, and the correctness of the diagrammatic notation. Excellent DBMS physical ERDs accurately map all entities, relationships, and relationship constraints, enforce relationships properly with primary and foreign keys, contain all designed attributes and datatypes, and contain entirely correct diagrammatic notation throughout.	Entirely accurate Entirely correct	Mostly accurate Mostly correct	Somewhat accurate Somewhat correct	Mostly inaccurate Mostly incorrect	DBMS physical ERD missing or Entirely inaccurate Entirely incorrect
Normalization (10%)	This measures how well the tables are normalized. In an excellent solution, every table is either normalized to BCNF, or consciously normalized to a lesser normal form with excellent justification.	Excellent normalization	Good normalization	Partial normalization	Insufficient normalization	No normalization or Normalization implementation entirely incorrect
SQL Script Quality (15%)	This measures how accurately the SQL script implements the designed database structure, and the correctness of the SQL syntax. With excellent scripts: o the SQL script includes the table names, attribute names, attribute datatypes, and constraints exactly as defined in the DBMS physical ERD, as well as the identified sequences. o only designed elements are present in the script. o the syntax of the SQL script is entirely correct and could be executed in a modern relational database without modification	Entirely accurate Entirely correct	Mostly accurate Mostly correct	Somewhat accurate Somewhat correct	Mostly inaccurate Mostly incorrect	The SQL script is missing or The SQL script is generated by a tool without significant modification or Entirely inaccurate Entirely incorrect
Prior Work Soundness (10%)	This measures how well any issues from prior iterations have been improved in order to provide a frame of reference for this iteration.	Completely improved or No improvement necessary	Mostly improved	Somewhat improved	Mostly not improved	No improvements

Preliminary Grade:		Entities Deduction: At least 10 required at the conceptual level At least two subtypes required 3 point deduction for each missing		Lateness Deduction: 5 points per day 4 days maximum Contact your facilitator for any exceptions		Iteration Grade:
--------------------	--	--	--	---	--	------------------

Use the **Ask the Teaching Team Forum** if you have any questions regarding how to approach this iteration. Make sure to include your name in the filename and submit it in the *Assignments* section of the course.