Data Admin Concepts & Database Management

Table of Contents

Lab 05 – Physical Design and DDL

Overview

This lab is the fifth of ten labs in which we will build a database using the systematic approach covered in the asynchronous material. Each successive lab will build upon the one before and can be a useful guide for building your own database projects.

In this lab, we will use a diagram of properly normalized tables to create a set of tables in SQL Server.

Read this lab document once through before beginning.

Learning Objectives

In this lab you will

- Demonstrate data definition language (DDL) proficiency
- Demonstrate ability to convert from diagrams to SQL code

Lab Goals

This lab consists of two sections. The first section is a walkthrough of creating tables, columns, and constraints. In the second part, you will use similar code to build the rest of the database.

In both parts, you will code a few data manipulation (DML) statements to see your tables in action.

What You Will Need to Begin

- This document
- An active Internet connection (if using iSchool Remote lab)
- A blank Word (or similar) document into which you can place your answers. Please include your name, the current date, and the lab number on this document. Please also number your responses, indicating which part and question of the lab to which the answer pertains. Word docx format is preferred. If using another word processing application, please convert the document to pdf before submitting your work to ensure your instructor can open the file.
- To have completed Lab 04 – Normalization
- Understanding of database tables
- One of the following means of accessing a SQL Server installation
  - A connection to the iSchool Remote Lab (https://rds.syr.edu/)
  - A local installation of SQL Server (see Developer edition here https://www.microsoft.com/en-us/sql-server/sql-server-downloads-free-trial)
  - Regardless of how you access SQL Server, you will need to use SQL Server Management Studio to do so.

Part 1 – Creating Tables

Setup

After a thrilling round of design work, we have settled on a logical model design for the VidCast service. The current diagram is shown at the end of this document.

We will use this document along with some other narrative elements to build our database.

Formatting Note

Look for the "To Do" icon to point out sections of the lab you will need to do to complete the tasks.

SQL Server Management Studio

Most Relational Database Management Systems (RDBMS) are not software that we can directly interact with. Instead, they are services that run on a computer, server or otherwise, that manage access to the database and are responsible for handling the internal management of database objects. They are "headless" in that there isn't a user interface for them.

Instead, we use other applications to communicate with these services even if they are running on the same computer we're using to connect to them. In the case of SQL Server, we can use SQL Server Management Studio (SSMS) to do most of our work.

If you're running your own copy of SQL Server, you will have to ensure SSMS is installed (see links in "What You Will Need to Begin" above). If you are using the remote lab, this software is already installed on the remote computer you are accessing.

Screenshots used in this lab assume you are using the remote lab and are for the version of SSMS installed as of the last modified date of this document, but any differences in versions are mostly cosmetic.

Open SSMS by clicking the Window icon (formerly Start Button) in the lower left corner of the desktop and begin typing **SQL Server Management Studio**.

Click the SSMS icon that appears when typing. Note: this may appear before you've completed typing the name of the software and may have another version number than what is shown below.
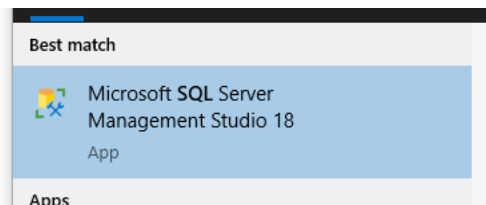
Once SSMS loads, you will have to tell SSMS how to connect to the server. You will be prompted with this dialog:
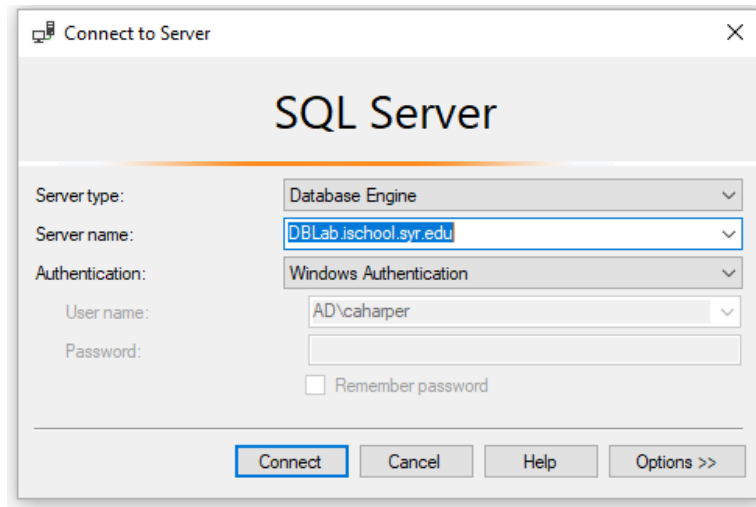
Let's unpack what SSMS is asking for here.

Server Type
Make sure **Database Engine** is selected for Server Type. The SQL Server ecosystem has many services that do different things related to data. All but Database Engine are outside the scope for this class.

Server Name
The value to enter here is, aptly, the name of the server. This name is a domain name services (DNS) name of the server on your network. It can also be an IP address if no name has been set for the computer on which the server is running.

If you are using the remote lab, the correct value for this is **DBLab.ischool.syr.edu**

If you are running the server on your own computer, you can simply enter **.** here. (That is a "dot" "period" or "full stop") "." (without the quotes) is a shorthand for saying "this computer". You might also have seen this as **localhost** or the IP address **127.0.0.1**. All of these are considered "loopback" names in that they tell the connection to use the current computer as the endpoint.

Authentication
Because SQL Server is managing access to our data and ensuring that only the right connections get through, we must authenticate with the server when we connect. In a sense, we are telling the server who we are.

For now, set this to **Windows Authentication** (regardless of whether you're using remote lab or your own hardware). Note that the user name and password are greyed out (cannot be edited) and that SSMS has set User Name to the domain\userid you used when logging into the local computer.
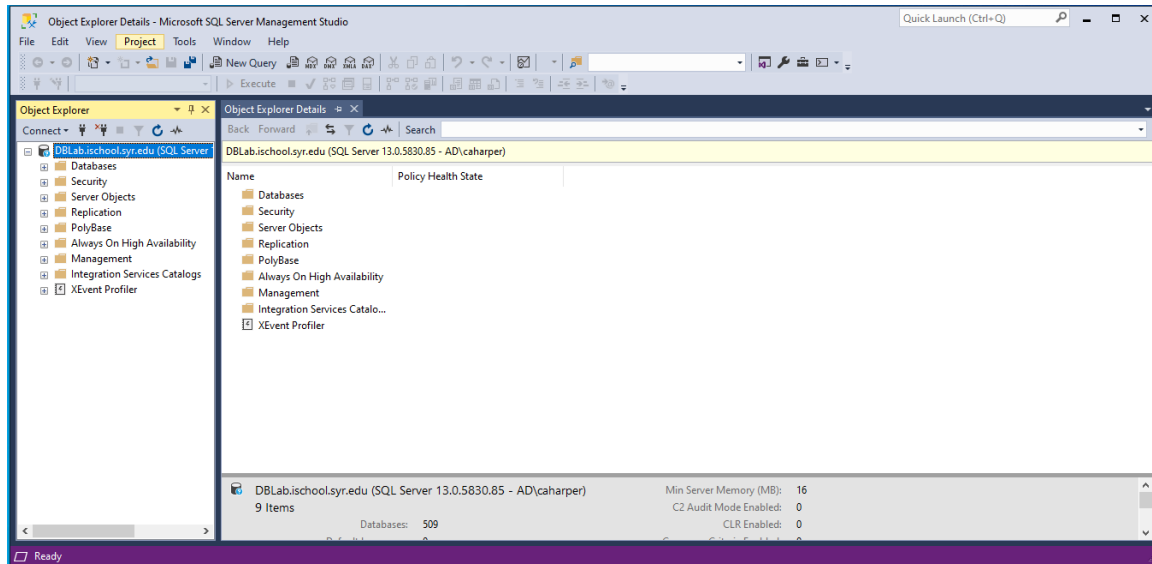
Subsequent Connections
SSMS should remember your settings for the next time you login, so you shouldn't have to reset all these values every time you connect, but it may forget what you used before, so keep these settings handy for future connections.
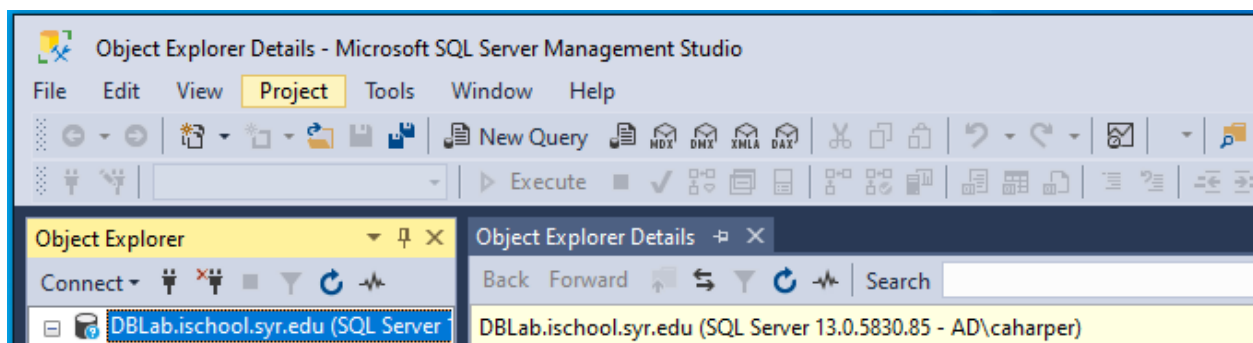
Click Connect
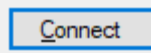
SSMS connects to the SQL Server and presents us with the main interface, ready to go!

There are a few elements we'll need to master to get our work done.

Menu Bar and Toolbar

We'll discuss each of the menus and toolbar items as we need them, but if you're unfamiliar with SSMS, take a moment to familiarize yourself with where the menu commands are and which toolbar buttons are available to you. Of interest to us are the New Query button, the save button, and the open File button.

Hover your mouse pointer over the buttons for tooltips that tell us what each button is.

Object Explorer
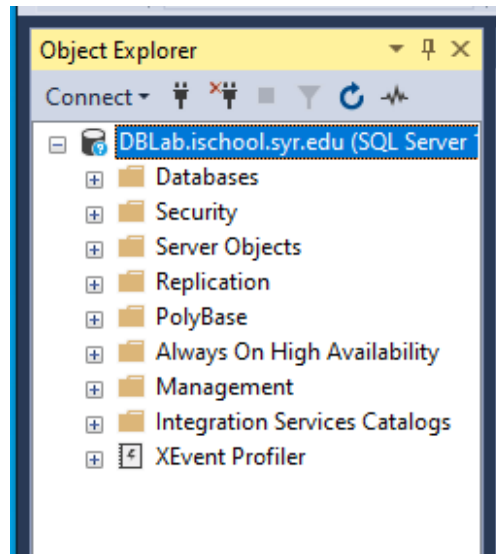
The Object Explorer is where we can see all the objects on our server and in our database. This is a hierarchical tree view of objects. At its root is the server to which you're connected and as what user you've authenticated.

There are several items below the server node, but the only one we'll be working with this term is Databases. Click the + sign to expand Databases.

If you're using the Remote Lab, you will see a long list of databases. They are listed alphabetically and, fortunately, they are named based on a pattern. Your database will follow the pattern, **IST659_M999_netid** where M999 is the IST659 section you're registered for. For example, caharper registered for IST659, section M400. This is their database:

If you're running your own server, you will not see any databases here except system databases. It's best to leave these alone and instead create a new database. To do so, right click on your **Databases** folder in Object Explorer and click New Database. Give your database a fitting name, perhaps **IST659_Labs**, and click okay. As a rule, try to avoid spaces where possible.

Expand your database and take note of the folders contained within. SSMS groups the objects of your database by their type.

Expand the `Tables` node. If you haven't yet created any tables, your tree structure should look like this:

> ⊟ 📋 IST659_M400_caharper
>   ⊞ 📁 Database Diagrams
>   ⊟ 📁 Tables
>   ⊞ 📁 System Tables
>   ⊞ 📁 FileTables
>   ⊞ 📁 Views
>   ⊞ 📁 Synonyms
>   ⊞ 📁 Programmability
>   ⊞ 📁 Service Broker
>   ⊞ 📁 Storage
>   ⊞ 📁 Security
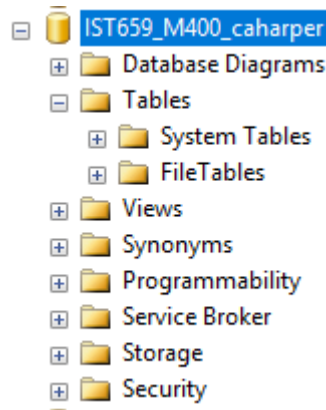
The rest of this lab is going to deal exclusively with the Tables in your database. We just have to create some.

In the toolbar, click the New Query **button.** This opens a new blank query window.

Take note of a couple elements that are now available. Particularly, there is a new toolbar. Also, this document has a tab at the top that lists the file name, the current server, the current database, and the currently authenticated user. This is a lot of data for one tab, so you may have to hover the mouse pointer over the tab to see all the information.

The Editing Toolbar

> IST659_M402_caharper ▾ | ▷ Execute ■ ✓ 🔲 🔲 🔲 | 🔲 🔲 🔲 | 🔲 🔲 🔲 | 🔲 🔲 | 🔲 🔲 | 🔲 ₊

One of the new elements to appear after we open any query file, including a new one, is the editing toolbar. There are some elements here we need to note for later use.

Available Databases

> | IST659_M400_caharper ▾ |
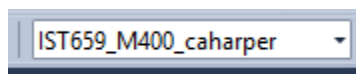
This is a drop-down list of all the available databases on this server. This is important to note because whatever you type in your query editor window and, subsequently, execute will affect the currently selected database specifically. If you are on the remote lab, this should default to your database. If you are on your own server, the default is the system database, master, a database we do not want to mess with.

In either case, at this point, **ensure the correct database is selected**. If necessary, change to the correct database by clicking the drop-down and scrolling to the correct database. You can also type the name of the correct database here and it will change accordingly.

Execute and Parse
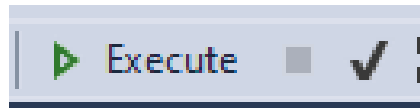
Just typing a query in the query editor window doesn't automatically execute the query against the database. The file you're viewing on the screen is merely a text file with a `.sql` extension, so SSMS knows how to set the font colors. Instead, we must type our SQL commands and click the Execute button to send those commands to the server.

If you have written particularly complex SQL code and would like to verify that it works before executing it against the database, you can click the ✔ parse button to validate your code. You do not need to do this every time, but if you're just starting out, it can be a good way to ensure your code is syntactically correct before executing it.
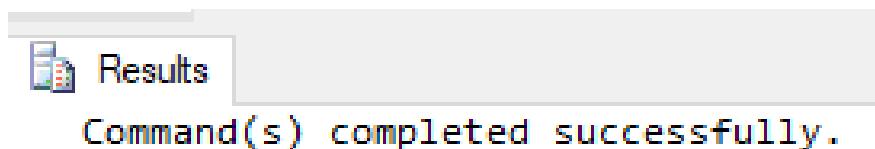
Try It

In the blank query editor window, **type the following line of SQL code**:

```
select 'Hello, SQL' as Greeting
```

Click the ✔ **parse button** to validate your code.

The Results pane appears and tells us this code is good to go.


```
Results
Command(s) completed successfully.
```

If you do not get this message, double check that you have entered the code exactly as above.

Click the **Execute button** in the toolbar to execute your code. A grid view results pane appears.

We will unpack this view later. For now, if you have never coded in SQL before, you just ran your first SQL query!

Line Numbers

When you first use SSMS, new query windows will not show any line numbers. While you can still see the line number in the lower right portion of the window:

it is still helpful to have the line numbers next to the code in the editor window for many reasons. It will be helpful when screen sharing with a colleague or your section instructor. Also, when SQL reports an error in the code, it will tell you which line has the problem. It may be helpful to be able to see at a quick glance what that code is.

To add line numbers to your query editor window, **click the Tools menu, then Options.**

Expand the Text Editor group and click on All Languages. On the right-hand side of the dialog, **check the Line Numbers box** below Display. **Click OK** to confirm.
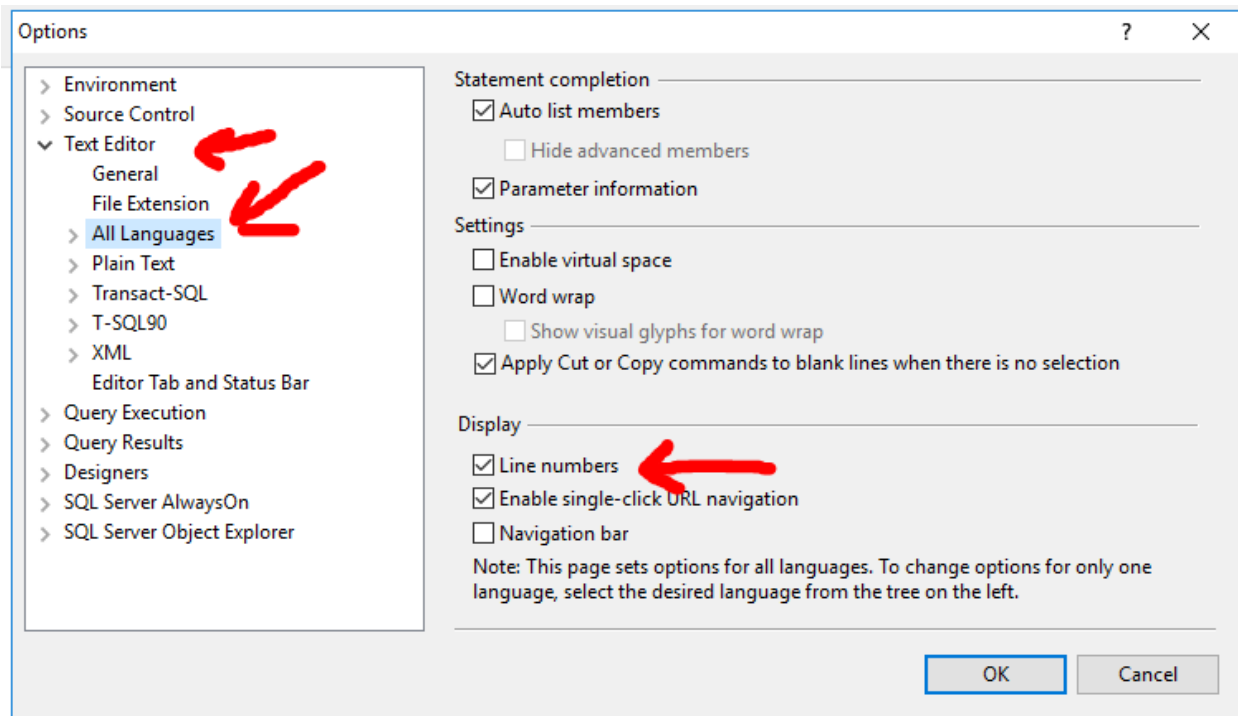
Note: You may have to reset this next time you log in.

Your First Table

Delete any text you have in your query editor window. Also make sure the correct database is selected in Available databases.
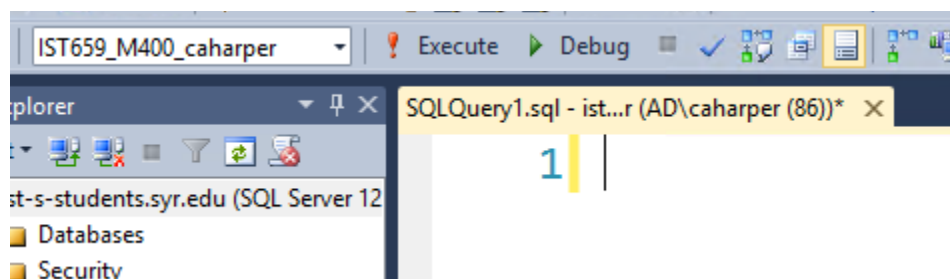
Comments

A Comment is a line or lines of code that are written to let the reader of the code know what the code is intended to do. They are an important part of any programming exercise to help demystify the code for readers, identify the authors and their intent, and can help explain decisions made by the author.

Because comments are meant to be read by humans and not by a computer, they are not processed as part of the code when executing. Comments are also used on actual lines of code to prevent those lines from executing. This is helpful in debugging sections of code and for temporarily removing code from the execution plan.

It's important to get into the habit of commenting code whenever possible, so let's start on the right foot. The correct foot, as it were.

In SQL Server, anything after a double-hyphen, regardless of where it is on the line, is treated as a comment. SSMS will format commented text in a light green color.

```
1  -- This is a comment. It is for you and me and will not be executed
2  |
```

You can also create a multi-line comment, usually several lines of code, by starting your comment block with /* and ending with */ and everything between those characters will be treated as comment text.

```
1  /*
2  This is a multiline comment. Line 1 starts the comment
3  We do not need to indicate that we are writing comments
4  on each subsequent line of code.
5  Line 6 ends the commented text.
6  */
7
```

Add a comment block containing your name as author, Lab 05 as the title of the script, IST659 and your section number as the course number, and the current quarter term. Your code should like the following:

```
1 /*
2       Author  : Saul Hudson
3       Course  : IST659 M400
4       Term    : April, 2018
5 */
6
```

Save Your Work

Save your file by clicking the 💾 Save icon in the toolbar, selecting Save from the File menu, or pressing Ctrl+S on your keyboard. Keep in mind that a SQL file is just a text file with a .sql extension, so we can save it wherever we like. If you're unsure where is best to save your file, the My Documents folder is a good starting point.

Save early, save often, especially if using the remote lab. There may be times when the remote lab ends your session, either because of a network error or from being logged out after an extended period of inactivity. If this happens, it is very likely you will lose any unsaved work. It's a good idea to get into the habit of saving as often as possible.

Formatting and Code Conventions

When we write code, we're not only writing it to be executed by a computer program like SQL Server, we're also writing it to be readable. Whether it is for you to diagnose an issue with your own code, or for someone else to try to make sense of what we've written after the fact, your code should be written for understanding by humans as well as computers. For this reason, it is important to devise and stick to a convention for formatting your code.

For example, SQL Server does not care overmuch for the format you've chosen for your code but consider the following two code blocks.

```
3 CREATE TABLE MyNewTable (MyNewTableID int identity primary key, MyDescription varchar(30) NOT NULL)
```

Versus

```
3 CREATE TABLE MyNewTable (
4       MyNewTableID int identity primary key,
5       MyDescription varchar(30) NOT NULL
6 )
```

The first and second examples are both syntactically correct and both will achieve the same goal of creating a table called MyNewTable. The second example, however, makes use of white space in the form of new lines and tabs to indent internal code to help the code make visual sense.

By taking four lines of text for a single SQL statement, we have made the task of working with the code after the fact that much easier.

Lines vs Statements

Part of understanding how to best format SQL code in the file is understanding the difference between a line in the file and a full SQL statement.

A **line**, simply put, starts at the beginning of a line of text and ends when a new line is reached (you create a new line by pressing the Enter key).

A **statement**, on the other hand, is the full text of a single SQL command. As we've seen in the preceding two examples, a statement is made up of several elements. When a statement is sent to SQL Server for execution (by way of clicking the Execute button in the toolbar), it will interpret the statement from its start, (mostly) ignoring hidden whitespace such as tabs and new lines, and proceeds until it has a complete statement with which to work. It then does the thing it was asked to do.

In basic terms, a SQL statement is made up of a few parts, all common to whatever the statement is trying to achieve.

We start with some verb that tells the RDBMS what we're about to do (CREATE, DROP, DELETE, SELECT, and UPDATE, to name a few). In many cases, particularly in Data Definition Language (DDL) statements, this is followed by some object type (TABLE, PROCEDURE, VIEW, FUNCTION, etc.).

For DDL commands, the object type is followed by the name of the object. The balance of the statement is made up of the specific code SQL Server needs to complete the command.

The Code for the User Table

Let's have another look at our previous example.

```
3  CREATE TABLE MyNewTable (
4      MyNewTableID int identity primary key,
5      MyDescription varchar(30) NOT NULL
6  )
```

Line 3 starts with the verb, CREATE. This tells SQL Server we're going to create something new. This is followed by TABLE. This tells SQL Server we are creating a table. The name of that table is MyNewTable.

The CREATE TABLE statement requires a comma-separated list of columns and constraints that comprise the table structure. These columns and constraints are contained within parentheses. Let's unpack this statement and create our first table.

To start, we're going to create the User table from our VidCast database. Let's have another look at the User table from the diagram:
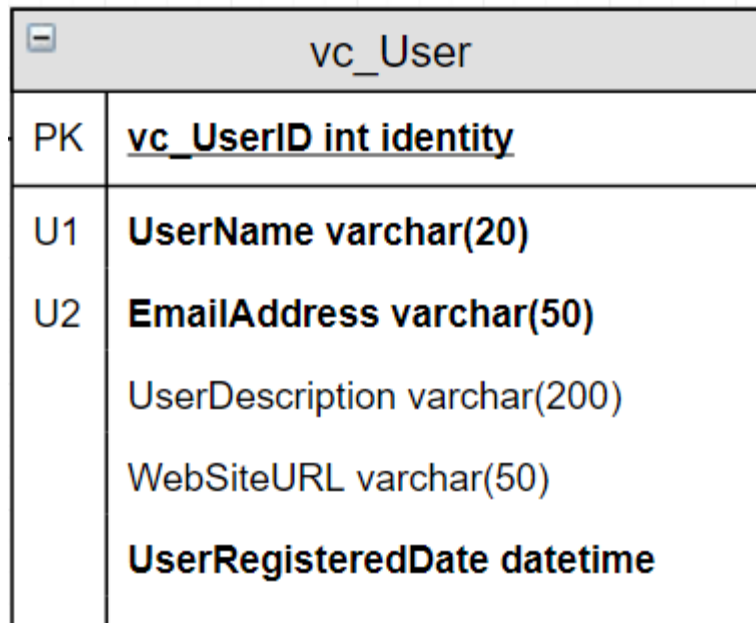
We have most of the information we need to create this table in this shape. The only thing not shown here is that we want to default the UserRegisteredDate column to whatever the current date is at the time a row is entered. We could force the person adding the record to provide it, but to be user-friendly, let's use a SQL default that tells the database what to use if no UserRegisteredDate is provided.

If we decompose this diagram into the columns and constraints we'll need to provide in our SQL CREATE TABLE statement, we can see the different components in the shape and how they might be coded.

vc_User Columns		
Column Name	Data Type	Properties
vc_UserID	int	identity
UserName	varchar(20)	not null
EmailAddress	varchar(50)	not null
UserDescription	varchar(200)	
WebSiteURL	varchar(50)	
UserRegisteredDate	datetime	not null default GetDate()

vc_User Constraints

Constraint Name	Constraint Type	Properties
PK_vc_User	PRIMARY KEY	vc_UserID
U1_vc_User	UNIQUE	UserName
U2_vc_User	UNIQUE	EmailAddress

You don't need to build these grids for every table you create. These tables are presented here to illustrate how the diagram shapes translate to the SQL CREATE TABLE code.

TO DO   **In your query editor window, type the following code:**

```
 7  -- Creating the User table
 8  CREATE TABLE vc_User (
 9        -- Columnns for the User table
10      vc_UserID int identity,
11      UserName varchar(20) not null,
12      EmailAddress varchar(50) not null,
13      UserDescription varchar(200),
14      WebSiteURL varchar(50),
15      UserRegisteredDate datetime not null default GetDate(),
16      -- Constraints on the User Table
17      CONSTRAINT PK_vc_User PRIMARY KEY (vc_UserID),
18      CONSTRAINT U1_vc_User UNIQUE(UserName),
19      CONSTRAINT U2_vc_User UNIQUE(EmailAddress)
20  )
21  -- End Creating the User table
22
```

TIP: *Be mindful of the commas at the ends of lines 1-15 and 17-18 above. Also note that there is not a comma at the end of line 19 as it is the last item in this comma-separated list.*

At this point, we can probably execute all our code without a problem, but let's get into the habit of being selective about what we execute. **Select the lines of code from your first comment through the last comment:**

```
 7   -- Creating the User table
 8   CREATE TABLE vc_User (
 9       -- Columnns for the User table
10       vc_UserID int identity,
11       UserName varchar(20) not null,
12       EmailAddress varchar(50) not null,
13       UserDescription varchar(200),
14       WebSiteURL varchar(50),
15       UserRegisteredDate datetime not null default GetDate(),
16       -- Constraints on the User Table
17       CONSTRAINT PK_vc_User PRIMARY KEY (vc_UserID),
18       CONSTRAINT U1_vc_User UNIQUE(UserName),
19       CONSTRAINT U2_vc_User UNIQUE(EmailAddress)
20   )
21   -- End Creating the User table
```
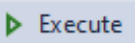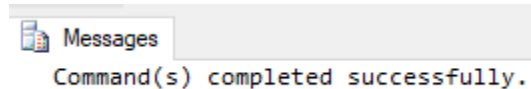
Now, click ▶ Execute **in the toolbar**. SSMS will send your code to the SQL Server for execution.

It should take a fraction of a second for your code to execute and return the success message.

Messages
Command(s) completed successfully.

If you get any message other than this, double check that you have typed the code exactly as written in the sample image above. If you need to make any edits, simply do so and re-execute the code.

We've created a table! There's nothing in it yet, but now it is ready for us to add data. Have a look at your Object Explorer to see your ta… wait…

```
☐ 🛢 IST659_M400_caharper
    ⊞ 📁 Database Diagrams
    ☐ 📁 Tables
        ⊞ 📁 System Tables
        ⊞ 📁 FileTables
    ⊞ 📁 Views
    ⊞ 📁 Synonyms
    ⊞ 📁 Programmability
    ⊞ 📁 Service Broker
    ⊞ 📁 Storage
    ⊞ 📁 Security
```
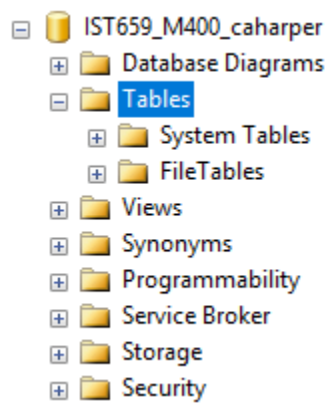
Because the Object Explorer is built based on a snapshot of the database objects, it may not yet show your table. If it doesn't, right-click the Tables folder and click Refresh

IST659_M400_caharper
  Database Diagrams
  Tables
    System Tables
    FileTables
    dbo.vc_User
  Views
  Synonyms
  Programmability
  Service Broker
  Storage
  Security
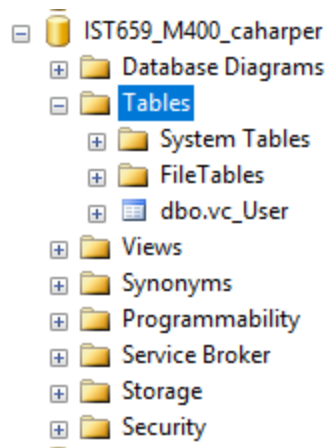
Much better. To see the columns and other properties we've defined for the vc_User table, expand the vc_User table, then expand Columns, then expand Keys. Your Object Explorer should look like this:
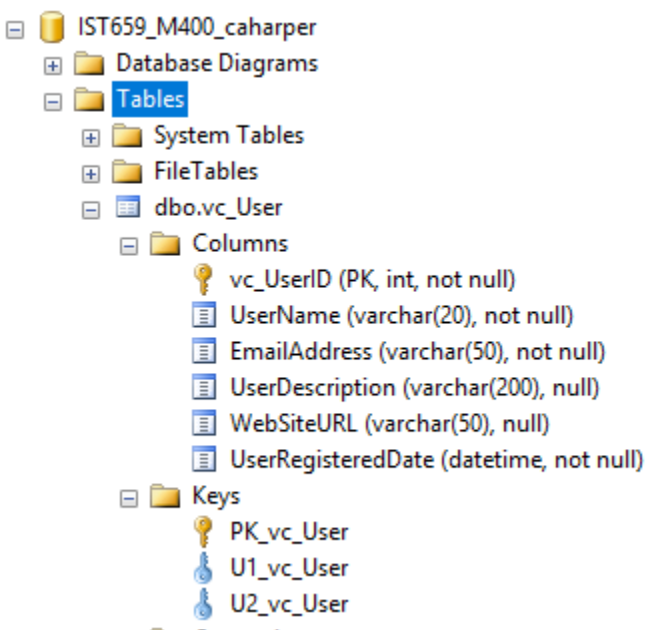
IST659_M400_caharper
  Database Diagrams
  Tables
    System Tables
    FileTables
    dbo.vc_User
      Columns
        vc_UserID (PK, int, not null)
        UserName (varchar(20), not null)
        EmailAddress (varchar(50), not null)
        UserDescription (varchar(200), null)
        WebSiteURL (varchar(50), null)
        UserRegisteredDate (datetime, not null)
      Keys
        PK_vc_User
        U1_vc_User
        U2_vc_User

Take a screenshot of your Object Explorer with your table and its columns and keys expanded. Paste this screenshot into your answer doc labeled as "User Table"

Another Table with a Foreign Key

Let's code the UserLogin table. This table will track when a user logs in and keep track of what IP they connected from. We've isolated the table from the diagram here:

To aid in coding this as a table, we can look at it in grid format with the column properties translated. Let's also default the timestamp to the current date and time.

vc_UserLogin Columns		
Column Name	Data Type	Properties
vc_UserLoginID	int	identity
vc_UserID	Int	not null
UserLoginTimestamp	datetime	not null default GetDate()
LoginLocation	varchar(50)	not null

vc_UserLogin Constraints		
Constraint Name	Constraint Type	Properties
PK_vc_UserLogin	PRIMARY KEY	vc_UserLoginID
FK1_vc_UserLogin	FOREIGN KEY	vc_UserID REFERENCES vc_User(vc_UserID)

Add the following code to your query editor window. Remember to pay attention to the parentheses and commas in this code. SQL is quite particular about those things. Highlight and execute only this code.

```
23  -- Creating the UserLogin table
24 ⊟CREATE TABLE vc_UserLogin (
25      -- Columns for the UserLogin table
26      vc_UserLoginID int identity,
27      vc_UserID int not null,
28      UserLoginTimestamp datetime not null default GetDate(),
29      LoginLocation varchar(50) not null,
30      -- Constraints on the User Login Table
31      CONSTRAINT PK_vc_UserLogin PRIMARY KEY (vc_UserLoginID),
32      CONSTRAINT FK1_vc_UserLogin FOREIGN KEY (vc_UserID) REFERENCES vc_User(vc_UserID)
33  )
34  -- End Creating the User Login Table
```

Take a screenshot of your Object Explorer with your vc_UserLogin table and its columns and keys expanded. Paste this screenshot into your answer doc labeled as "User Login Table"

Add Some Data

Before we proceed, let's see our tables in action.

Copy and paste the following code into your query editor window (or type it out – that makes learning code easier!). Then highlight this code (and only this code!) and execute it.

```
-- Adding Data to the User table
INSERT INTO vc_User(UserName, EmailAddress, UserDescription)
    VALUES
            ('RDwight', 'rdwight@nodomain.xyz', 'Piano Teacher'),
            ('SaulHudson', 'slash@nodomain.xyz', 'I like Les Paul guitars'),
            ('Gordon', 'sumner@nodomain.xyz', 'Former cop')

SELECT * FROM vc_User
```

TIP: *Again, be mindful of the quotation marks, commas, and parentheses in these statements. They are important!*

TIP: *After you have executed the INSERT statement, you will get an error message if you try to execute it again. This is by design. Once you run it, the data are there permanently, and we have guarded against duplication using our UNIQUE constraints.*

The first four lines are a single INSERT statement. When that statement is executed, it adds three rows to the database in the vc_User table. The last line is a SELECT statement that queries that table.

Upon completion of the execution, you should see a results grid like this:

	vc_UserID	UserName	EmailAddress	UserDescription	WebSiteURL	UserRegisteredDate
1	1	RDwight	rdwight@nodomain.xyz	Piano Teacher	NULL	2018-05-03 10:26:49.510
2	2	SaulHudson	slash@nodomain.xyz	I like Les Paul guitars	NULL	2018-05-03 10:26:49.510
3	3	Gordon	sumner@nodomain.xyz	Former cop	NULL	2018-05-03 10:26:49.510

"But, wait, we didn't specify a vc_UserID or UserRegisteredDate value. Where did they come from?" The identity property we added to vc_UserID told SQL Server to provide the next available value for us. When the first row went in, 1 was the next value. Then 2. Then… you get it. Because we specified a default of GetDate() on the UserRegisteredDate, but didn't include one in the INSERT statement, SQL Server ran the internal GetDate() function (more on that in a later lab) which returned the current date and time (down to the microsecond, actually).

Take a screenshot of your results grid and paste it into your answer document. Label it "User records"

Creating the Follower List Table

When a user follows another user, we need to add a record to this table. Pay special attention to how the UNIQUE constraint differs from what we've done before. In previous statements, we have only put one column name in the constraint. Because we want to ensure that the combination of values in two different columns must be unique, we add them both, separated by a comma, to the constraint.

Also, this is the first time we have used a column name in a foreign key that differs from the referenced primary key in the other table. This is quite common. There is no hard and fast rule that says a foreign key column need be named the same as the referenced primary key column. In fact, it is more common to name them differently. When you do so, however, ensure that the meaning of the column name reflects the data you intend to keep. In our case, we have renamed our foreign key columns to indicate their semantic meaning.

FollowerID is the vc_UserID of the vc_User doing the following.

FollowedID is the vc_UserID of the vc_User being followed.

As before, we will default the datetime to GetDate().

Type and execute the following code in your query editor

```
45  -- Creating the Follower List table
46  CREATE TABLE vc_FollowerList (
47      -- Columns for the follower List table
48      vc_FollowerListID int identity,
49      FollowerID int not null,
50      FollowedID int not null,
51      FollowerSince datetime not null,
52      -- Constraints on the Follower List table
53      CONSTRAINT PK_vc_FollowerList PRIMARY KEY (vc_FollowerListID),
54      CONSTRAINT U1_vc_FollowerList UNIQUE (FollowerID, FollowedID),
55      CONSTRAINT FK1_vc_FollowerList FOREIGN KEY (FollowerID) REFERENCES vc_User(vc_UserID),
56      CONSTRAINT FK2_vc_FollowerList FOREIGN KEY (FollowedID) REFERENCES vc_User(vc_UserID)
57  )
58  -- End creating the Follower List table
```

TIP: *Note line 54, above. This is how we create a composite unique constraint. Because both FollowerID and FollowedID are marked as U1. This means that they are part of the same Unique constraint. When you see the same constraint specification for more than one column on a table, you define them like this.*

Your line numbers might be slightly different. That's okay.

In the Object Explorer, refresh your Tables folder and take a screenshot of this portion of the screen. Paste it into your answers document and label it "User Tables"
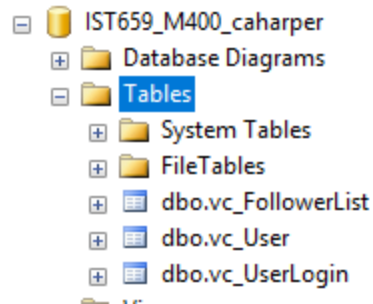
Part 2 - The Rest of the Tables

Code and execute the SQL CREATE TABLE statements to create the remaining tables. Remember to save often!!

Though not shown on the diagram below, make sure all vc_UserID columns in this section have a foreign key constraint that references vc_User(vc_UserID). **Also, do not add a default for the dates in VidCast**.

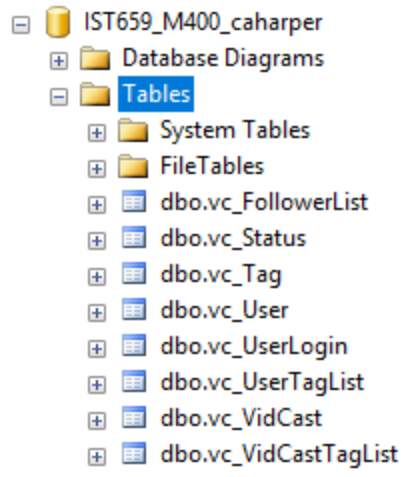The diagram for these tables is below, but be sure to create them in this order:

1. vc_Tag
2. vc_Status
3. vc_VidCast
4. vc_VidCastTagList *(be careful of the composite UNIQUE constraint)*
5. vc_UserTagList *(be careful of the composite UNIQUE constraint)*
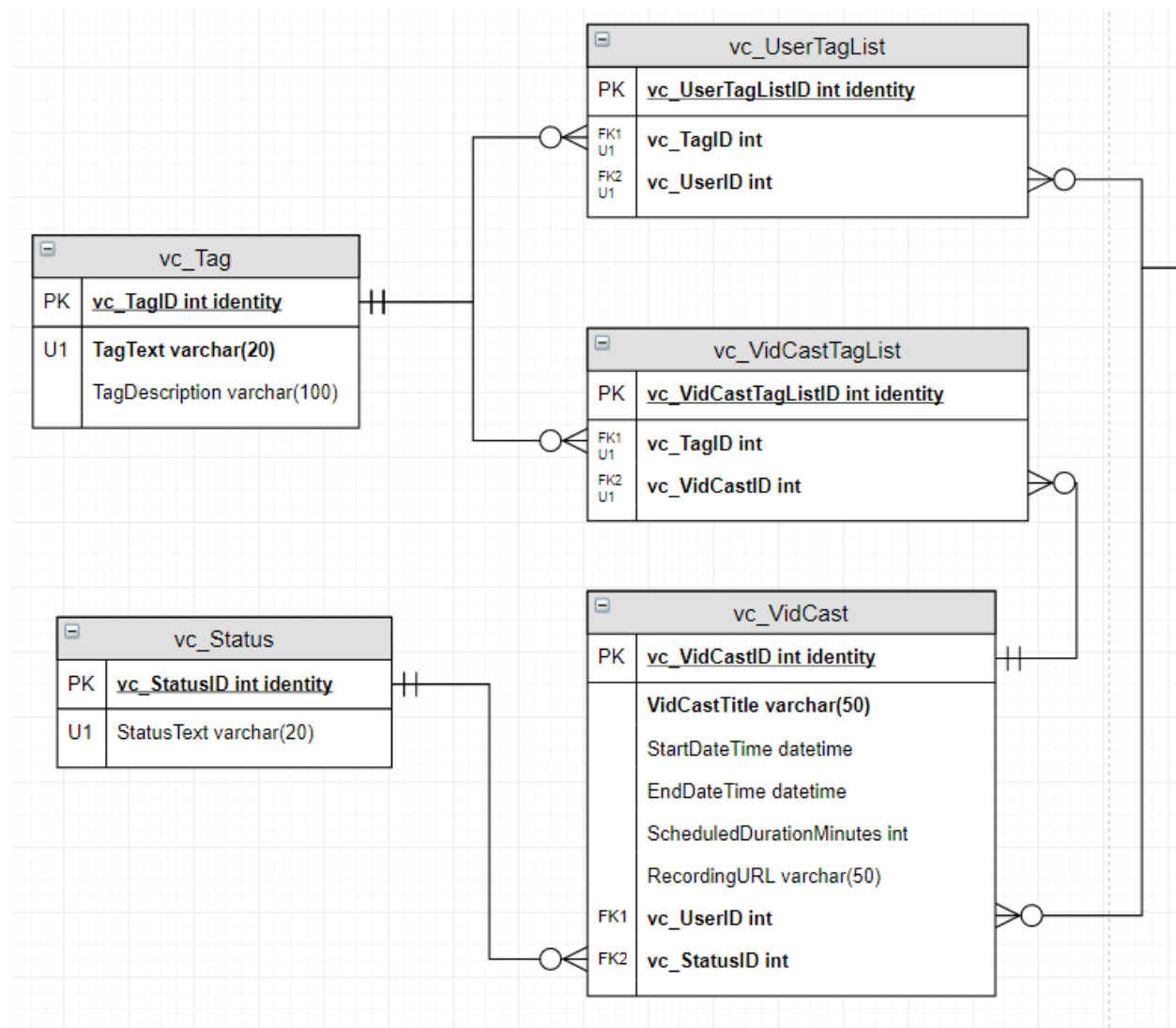
Why in this order? Well, if you try to put a foreign key constraint on a table and the referenced table doesn't yet exist, you will get an error message!

Refresh your Tables folder in the Object Explorer and take a screenshot of the listing as before. Paste this screenshot into your answers document labeled "VidCast tables"
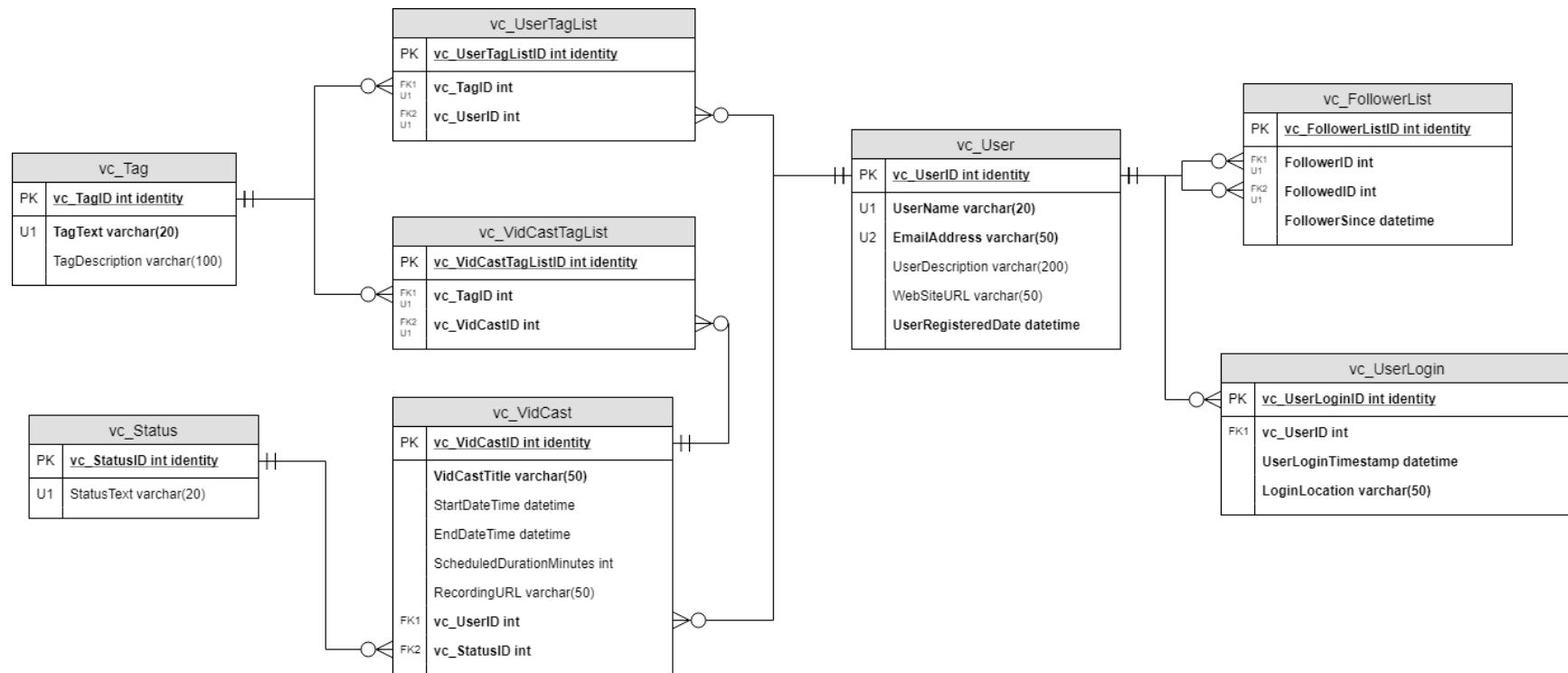
- IST659_M400_caharper
  - Database Diagrams
  - Tables
    - System Tables
    - FileTables
    - dbo.vc_FollowerList
    - dbo.vc_Status
    - dbo.vc_Tag
    - dbo.vc_User
    - dbo.vc_UserLogin
    - dbo.vc_UserTagList
    - dbo.vc_VidCast
    - dbo.vc_VidCastTagList

What to Submit

After completing Part 2, copy and paste the text of your SQL query file at the end of your answers document. Save this document and submit it to the appropriate section on the LMS.

Appendix A – VidCast Logical Model Diagram

For the full diagram, see https://drive.google.com/file/d/1KRqkSvQABuTMXqYAzojTCt9etTSR8Vea/view?usp=sharing