Data Admin Concepts & Database Management

Table of Contents

Lab 06 – Querying, Inserting, Updating, and Deleting

Overview

This lab is the sixth 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 structured query language (SQL) data manipulation commands to populate the database tables created in Lab 05.

Read this lab document once through before beginning.

Learning Objectives

In this lab you will

- Demonstrate data manipulation language (DML) proficiency

Lab Goals

This lab consists of two sections. The first section is a walkthrough of inserting, updating, querying, and deleting data. In the second part of the lab, you will code your own DML queries to solve the problems presented.

TIP: *If you are new to SQL or programming in general, you may benefit from run through of the SQL Tutorial at https://www.w3schools.com/sql/ . While not required reading, it can be a helpful resource for new programmers to get some coding in.*

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 05 – Physical Design and DDL
- Understanding of database tables and have reviewed the asynchronous material for Week 6
- 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 – Filling our Tables with Data

Setup

The action is heating up as we now have a fully-made database and now need to start adding data. Our application developers haven't yet written the front-end software. We will need to write some code to add the initial data and to work out what SQL our app devs will have to use to populate and retrieve the data.

We will have to use each of the CRUD commands to manipulate the data in our tables.

Formatting Note

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

Inserting Data – The INSERT Statement

To "create" data, we use the SQL `INSERT` statement. We saw a brief example of that in Lab 05 when we added three rows to our user table. Let's add some more.

Insert One Row at a Time

The basic format of an INSERT statement is:

```
INSERT INTO table_name (columnn1, column2, …, columnN)
VALUES (value1, value2, …, valueN)
```

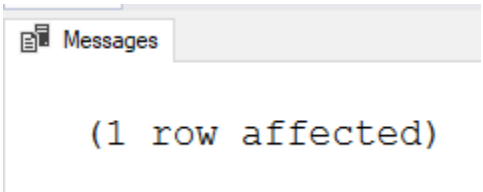
This will add a row to some table (represented by table_name) providing values for each column in the column list. These values will populate the columns in the same order shown in the column list.

To add a row to our vc_Status table, we would use the following code:

```
1  -- Adding a row to the vc_Status table
2  INSERT INTO vc_Status (StatusText)
3  VALUES ('Scheduled')
```

Code and execute the SQL in the preceding image to add a row to the vc_Status. Remember you can highlight lines of code in SSMS and when you click "Execute" it will only execute the highlighted code.

You should see this result:

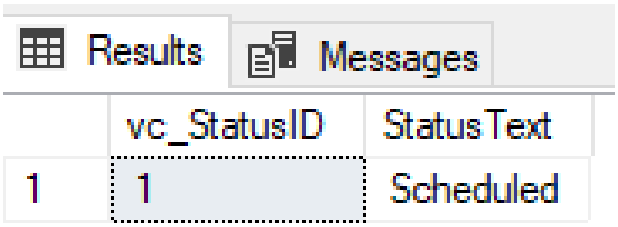


Only execute this code once! After you've INSERTed the data, it is there. Running the code again will try to add another row to the vc_Status table with the same data. We don't want this! Luckily, we coded a unique constraint to disallow duplicate status texts. If this constraint was properly coded, you will get an error message if you try to INSERT 'Scheduled' into vc_Status a second time.

To see your inserted data, code and execute the following statement:

```
5  -- The following line shows all of the rows in vc_Status
6  SELECT * FROM vc_Status
```

Your results should look like:



TIP: *As you code these INSERTs, you **won't** provide a value for the primary key columns that have the identity property specified. You also won't include it in your column list. This means the system is going to provide that number for us, guaranteeing it will always be unique. Because of this, the server will not reclaim used numbers, so it's possible that your ID columns will differ from the example images.* ***This is OK!***

Inserting Multiple Rows with One Statement

If you have multiple rows to add, you can use a single INSERT statement to do so. The following example adds three more rows to the vc_Status table:

```
8  -- Adding three more rows to the vc_Status table
9  INSERT INTO vc_Status (StatusText)
10 VALUES ('Started'), ('Finished'), ('On time')
```

Code and execute the SQL in the preceding image to add three rows to the vc_Status. Remember you can highlight lines of code in SSMS and when you click "Execute" it will only execute the highlighted code.

Now highlight and execute the select statement from before:

```
5  -- The following line shows all of the rows in vc_Status
6  SELECT * FROM vc_Status
```

Now your results show

Results Messages

	vc_StatusID	StatusText
1	3	Finished
2	4	On time
3	1	Scheduled
4	2	Started

TIP: *Your rows may show in a different sort order. This is okay for now if you have four rows with the shown values in StatusText. We'll see how to change how they're sorted later!*

Inserting Data into Tables with Foreign Keys

Adding rows to tables with foreign keys is precisely the same as inserting in any other table, but the values to provide for the foreign key columns can sometimes be confusing to new coders.

If we add a row to the vc_VidCast table, we will need to provide a value for the two columns that have foreign key constraints on them, vc_UserID and vc_StatusID. Both columns have data type int and reference the primary key columns of the vc_User and vc_Status tables respectively.

For our first vc_VidCast, the user with UserName SaulHudson recorded a 30-minute video titled "December Snow". Its details are below:

Table Column Name	**Value?**
VidCastTitle	'December Snow'
StartDateTime	'3/1/2018 14:00'
EndDateTime	'3/1/2018 14:30'
ScheduledDurationMinutes	30

RecordingURL	'/XVF1234'
vc_UserID	'SaulHudson'
vc_Status	'Finished'

The eagle-eyed viewer will notice that 'SaulHudson' and 'Finished' are not integers, not even a little bit. This code will absolutely not work (DON'T TRY IT. Just take our word for it.)

```
12  INSERT INTO vc_VidCast
13      (VidCastTitle, StartDateTime, EndDateTime, ScheduleDurationMinutes
14      , RecordingURL, vc_UserID, vc_StatusID)
15  VALUES
16      ('December Snow', '3/1/2018 14:00', '3/1/2018 14:30', 30
17      , '/XVF1234', 'SaulHudson', 'Finished')
```

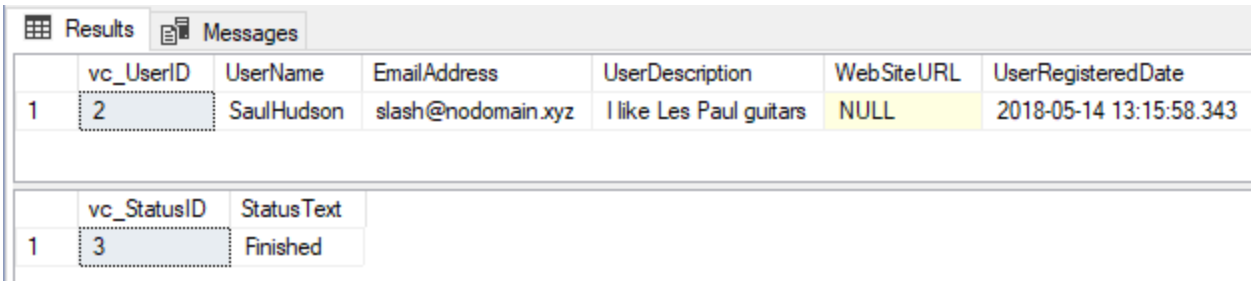
TIP: *Don't be thrown off by the number of lines used. Remember that SQL Server doesn't care about new lines in a single statement. Lines 12 through 17 above are one statement made into several lines for readability on this page*

Instead, we must find the value of vc_UserID for the vc_User with UserName SaulHudson and the vc_StatusID for the vc_Status with a StatusText of Finished.

Code and execute the following SQL in SSMS. Remember you can highlight lines of code in SSMS and when you click "Execute" it will only execute the highlighted code. We will unpack how to build each of these statements later.

```
12  SELECT * FROM vc_User WHERE UserName = 'SaulHudson'
13  SELECT * FROM vc_Status WHERE StatusText = 'Finished'
```

When you execute both of those SELECT statements, SQL Server will return two separate result sets.



Results | Messages

	vc_UserID	UserName	EmailAddress	UserDescription	WebSiteURL	UserRegisteredDate
1	2	SaulHudson	slash@nodomain.xyz	I like Les Paul guitars	NULL	2018-05-14 13:15:58.343

	vc_StatusID	StatusText
1	3	Finished

Take note of the values in vc_UserID and vc_StatusID. In our sample database they are 2 and 3, respectively. Yours may be different. **That's okay. Just make sure to use the integers for your database even if they differ from the sample.**

Revisiting the data we need to enter into the table, we can replace 'SaulHudson' with 2 and 'Finished' with 3.

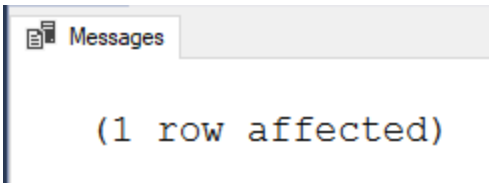
Table Column Name	Value
VidCastTitle	'December Snow'
StartDateTime	'3/1/2018 14:00'
EndDateTime	'3/1/2018 14:30'
ScheduledDurationMinutes	30
RecordingURL	'/XVF1234'
vc_UserID	2
vc_Status	3

Now we can code our INSERT statement using the proper values.

Code and execute the following SQL in SSMS. Remember you can highlight lines of code in SSMS and when you click "Execute" it will only execute the highlighted code. Remember to use the vc_UserID and vc_StatusID from your database which may differ from the sample.

```
17  -- Adding a vidcast record to the VidCast Table
18  INSERT INTO vc_VidCast
19      (VidCastTitle, StartDateTime, EndDateTime, ScheduleDurationMinutes
20      , RecordingURL, vc_UserID, vc_StatusID)
21  VALUES
22      ('December Snow', '3/1/2018 14:00', '3/1/2018 14:30', 30
23      , '/XVF1234', 2, 3)
```

You should see the following result in SSMS:



Code and execute the following SQL in SSMS. Remember you can highlight lines of code in SSMS and when you click "Execute" it will only execute the highlighted code. This will show you your record in the table

```
25  --Read all rows from vc_VidCast
26  SELECT * FROM vc_VidCast
```

Take a screenshot of your results grid and paste it into your answers document labeled 'First VidCast'

TIP: *Now that we have the INSERT statement for a vc_VidCast, you can copy paste this and replace values for adding new rows. In fact, if you want to add more than one vc_VidCast at a time, simply add a comma to the end of line shown on 23 and repeat the lines 22 and 23 for each subsequent record! (We'll do this later)*

Querying Data – The SELECT Statement

In this section, we will read the data back from the database. All the data we've inserted into our tables are currently sitting on disk somewhere. For it to be useful, we need a way to read it. We will use the SELECT statement to do so.

Because it's the statement we use so much, we'll spend more time unpacking the SQL SELECT statement here. Each SELECT statement can be broken into the following clauses:

Clause	Required?	Description
SELECT list	Yes	The list of columns to be returned and shown in the results
FROM clause	Yes	The table or tables in which the data are stored. Required only when the SELECT list contains column names.

WHERE clause	No	Optionally limits what data are retrieved by the statement. Is also used in SQL UPDATE and DELETE statements. If omitted, every row from the tables designated in the FROM clause are affected.
GROUP BY clause	No	Sets grouping levels when dealing with aggregates. If SQL Aggregate functions are used in the SELECT list and this clause is omitted or improperly coded, you will receive an error message. *
HAVING clause	No	Optionally limits the data retrieved by the statement. Differs from the WHERE clause in that it specifies conditionals for the results of aggregate functions. If omitted, all rows will be shown in the results *
ORDER BY clause	No	A comma-separated list of columns, taken from either the SELECT list or in any of the tables in the FROM clause that instructs the server to sort the results according to the needs of the user. If omitted, results are shown in the order they are stored on disk.
* - The GROUP BY and HAVING clauses are used when writing queries with aggregate functions. This lab does not cover these. We will revisit them in a future lab.		

The following narrative will help us build a SQL SELECT statement that shows us the following results:

Results Messages

	UserName	EmailAddress	VidCastTitle	StartDateTime	EndDateTime	ScheduledHours	StatusText
1	SaulHudson	slash@nodomain.xyz	December Snow	2018-03-01 14:00:00.000	2018-03-01 14:30:00.000	0.500000	Finished

SELECT List

The SELECT list is a comma-separated list of all the columns you want to see in the results. As a shorthand, you can use the * character to return all columns that are in the tables in the from clause. You've already written SELECT statements with * in the SELECT list:

```
25  --Read all rows from vc_VidCast
26  SELECT * FROM vc_VidCast
```

When reading line 26 aloud, we can read it as "Select all columns from the table called vc_VidCast".

Although it can take a little more time to code, it is often better form to be specific about which columns we want to see in the results. Perhaps we don't care to see the surrogate primary keys, or we want the columns to show in a specific horizontal order.

For our query, we will need columns from many tables. In addition, we'll need to do some math to come up with the ScheduledHours column. That isn't even a column anywhere in our database. Instead we'll derive it from another column.

Our SELECT list looks like this:

```
28  SELECT
29      vc_User.UserName,
30      vc_User.EmailAddress,
31      vc_VidCast.VidCastTitle,
32      vc_VidCast.StartDateTime,
33      vc_VidCast.EndDateTime,
34      vc_VidCast.ScheduleDurationMinutes / 60.0 as ScheduledHours,
35      vc_Status.StatusText
```

Take special note of line 34. We have calculated the number of hours by dividing the ScheduledDurationMinutes by 60.0. The decimal is important here as it implicitly instructs SQL Server to allow for decimal places in an otherwise integer data point.

We've also *aliased* the column, giving it a new name so the column name in the results reflects what the data represent.

FROM Clause

The columns we need to show or derive other data points come from three different tables, vc_User, vc_VidCast, and vc_Status, so we will need them all in our FROM clause. We can't just list them out, though. SQL Server needs advice on how to match up the data points. For this, we will use the JOIN keyword and rely on our foreign key references to know what to link.

The FROM clause for our query is as follows:

```
36  FROM vc_VidCast
37  JOIN vc_User ON vc_VidCast.vc_UserID = vc_User.vc_UserID
38  JOIN vc_Status ON vc_VidCast.vc_StatusID = vc_Status.vc_StatusID
```

WHERE Clause

If we had fifty-million rows and we ran lines 28 through 38 as is, we would get all fifty-million records back in the result set. We can filter those results, if we would like, by using a WHERE clause. Although not necessary in this instance, we can use the following WHERE clause in our query:

```
39  WHERE vc_User.UserName = 'SaulHudson'
```

This will ensure that this query only ever retrieves vc_VidCasts for the vc_User with UserName 'SaulHudson'.

ORDER BY Clause

If we don't instruct SQL Server how to sort our results, it will use its own indexes to sort the results. Let's add an ORDER BY clause to sort by that start date and time of the VidCast record like this:

```
40  ORDER BY vc_VidCast.StartDateTime
```

The Complete Query

The full query to show the results is as follows:

```
28  SELECT
29      vc_User.UserName,
30      vc_User.EmailAddress,
31      vc_VidCast.VidCastTitle,
32      vc_VidCast.StartDateTime,
33      vc_VidCast.EndDateTime,
34      vc_VidCast.ScheduleDurationMinutes / 60.0 as ScheduledHours,
35      vc_Status.StatusText
36  FROM vc_VidCast
37  JOIN vc_User ON vc_VidCast.vc_UserID = vc_User.vc_UserID
38  JOIN vc_Status ON vc_VidCast.vc_StatusID = vc_Status.vc_StatusID
39  WHERE vc_User.UserName = 'SaulHudson'
40  ORDER BY vc_VidCast.StartDateTime
```

Code and execute the preceding SQL in SSMS. Remember you can highlight lines of code in SSMS and when you click "Execute" it will only execute the highlighted code. Paste a screenshot of your results into your answer document labeled as 'Saul's First VidCast'

Updating Data – the UPDATE statement

The only constant is change, and our data change constantly. We will use SQL UPDATE statements to change the data in our tables.

We need to change vc_User SaulHudson's UserRegisteredDate to March 1, 2018. We can use the following UPDATE statement to do so:

```
42  -- Correcting a User's UserRegisteredDate
43  UPDATE vc_User SET UserRegisteredDate = '3/1/2018' WHERE UserName = 'SaulHudson'
44
45  SELECT * FROM vc_User WHERE UserName = 'SaulHudson'
```

Code and execute the preceding SQL in SSMS. Remember you can highlight lines of code in SSMS and when you click "Execute" it will only execute the highlighted code. Paste a screenshot of your results into your answer document labeled as 'Update a User'

TIP: *When writing update and delete statements, never forget the WHERE clause. It is advisable to write the WHERE clause first, even if you wish to affect all rows in the table. Woe unto those who run a DELETE statement against a production table without a WHERE clause…*

Deleting Data – The DELETE Statement

The design team has decided that there will no longer be a vc_Status with StatusText 'On time' so we must delete it from the database.

When we write a DELETE statement, we simply provide the table name and a conditional telling SQL Server which rows to remove. Line 51 of the following code is the DELETE statement in action. By executing this line of code, we will remove all rows from vc_Status that have a StatusText equal to 'On time'

```
47  -- See what rows we have in status
48  SELECT * FROM vc_Status
49
50  -- Delete the On time status
51  DELETE vc_Status WHERE StatusText = 'On time'
52
53  -- See the effect
54  SELECT * FROM vc_Status
```

You don't always need to SELECT before and after. We're just doing so here as an illustrative exercise.

Code and execute the preceding SQL in SSMS. Remember you can highlight lines of code in SSMS and when you click "Execute" it will only execute the highlighted code. Paste a screenshot of your results into your answer document labeled as 'No more on time'

Part 2 – Putting All Together

In this part, you'll add some more data to the VidCast tables and write some queries to read the new data.

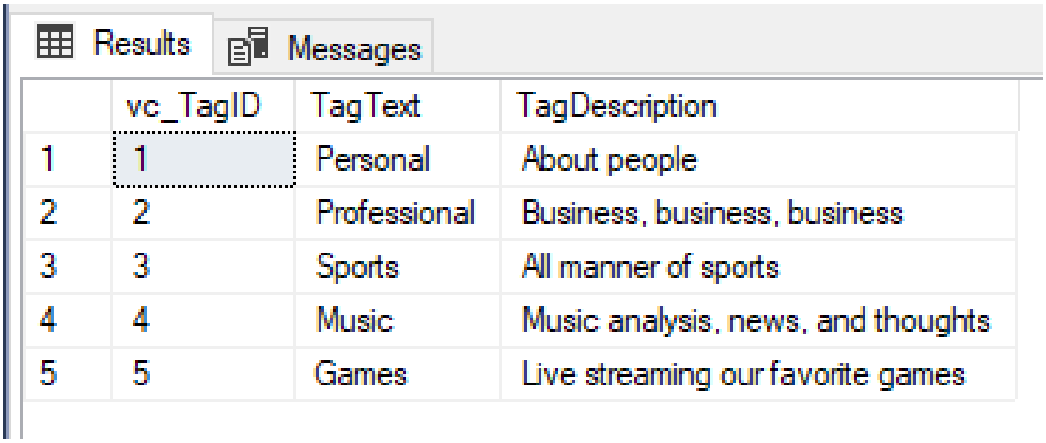
First, we'll add some Tags to the vc_Tag table.

The Tags we want to add are as follows:

TagText	TagDescription
Personal	About people
Professional	Business, business, business
Sports	All manner of sports
Music	Music analysis, news, and thoughts
Games	Live streaming our favorite games

Code and execute the SQL INSERT statement(s) to add the preceding values to the vc_Tag table. When finished, write a SELECT statement that retrieves all rows from vc_Tag. Paste a copy of your code and a screenshot of the results to your answer doc labeled 'Tags'

Your vc_Tag table results should look like this:



	vc_TagID	TagText	TagDescription
1	1	Personal	About people
2	2	Professional	Business, business, business
3	3	Sports	All manner of sports
4	4	Music	Music analysis, news, and thoughts
5	5	Games	Live streaming our favorite games

We also need a few more users for our database. They are as follows:

UserName	EmailAddress	UserDescription
TheDoctor	tomBaker@nodomain.xyz	The definite article
HairCut	S.todd@nodomain.xyz	Fleet Street barber shop
DnDGal	dnd@nodomain.xyz	NULL

Code and execute the SQL INSERT statement(s) to add the preceding values to the vc_User table. When finished, write a SELECT statement that retrieves all rows from vc_User. Paste a copy of your code and a screenshot of the results to your answer doc labeled 'Users'

Your vc_User table results should look like this (your dates and IDs may be different):

Results Messages

	vc_UserID	UserName	EmailAddress	UserDescription	WebSiteURL	UserRegisteredDate
1	1	RDwight	rdwight@nodomain.xyz	Piano Teacher	NULL	2018-05-14 13:15:58.343
2	2	SaulHudson	slash@nodomain.xyz	I like Les Paul guitars	NULL	2018-03-01 00:00:00.000
3	3	Gordon	sumner@nodomain.xyz	Former cop	NULL	2018-05-14 13:15:58.343
4	4	TheDoctor	tomBaker@nodomain.xyz	The definite article	NULL	2018-05-15 00:57:51.027
5	5	HairCut	S.todd@nodomain.xyz	Fleet Street barber shop	NULL	2018-05-15 00:57:51.027
6	6	DnDGal	dnd@nodomain.xyz	NULL	NULL	2018-05-15 00:57:51.027

Each user may elect to add tags to their user account. Some have already done so. The code to INSERT these tags into the vc_UserTagList is in Appendix B at the end of this document.

Copy and paste the complete code from Appendix B into your SQL script file and execute it against your database. After you have inserted those 14 rows, Code and execute a SELECT statement to retrieve all vc_UserTagList records and paste a screenshot of your results to your answer doc labeled 'User Tag List'.

Results Messages

	vc_UserTagListID	vc_TagID	vc_UserID
1	5	1	3
2	14	1	4
3	6	1	6
4	3	2	1
5	12	2	3
6	8	2	5
7	2	2	6
8	4	3	2
9	1	3	6
10	9	4	4
11	13	4	5
12	11	5	2
13	7	5	3
14	10	5	6

Code and execute a SQL SELECT statement that retrieves the vc_User's UserName and EmailAddress and the vc_Tag TagText for all vc_User records, ordered by user name then tag. Your output should look like the following screenshot. Copy and paste your SQL and a results screenshot to your answers doc labeled 'User Tags Report'

Results Messages

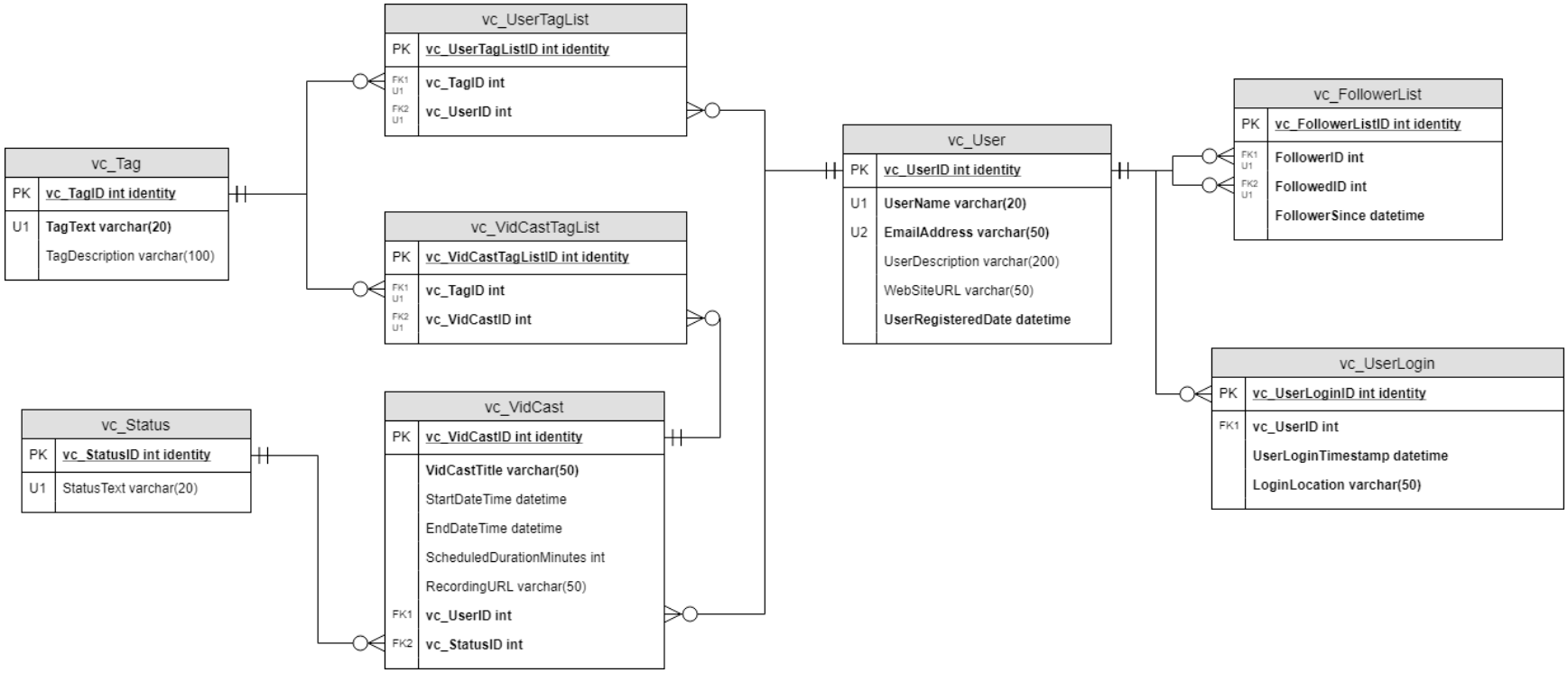
	UserName	EmailAddress	TagText
1	DnDGal	dnd@nodomain.xyz	Games
2	DnDGal	dnd@nodomain.xyz	Personal
3	DnDGal	dnd@nodomain.xyz	Professional
4	DnDGal	dnd@nodomain.xyz	Sports
5	Gordon	sumner@nodomain.xyz	Games
6	Gordon	sumner@nodomain.xyz	Personal
7	Gordon	sumner@nodomain.xyz	Professional
8	HairCut	S.todd@nodomain.xyz	Music
9	HairCut	S.todd@nodomain.xyz	Professional
10	RDwight	rdwight@nodomain.xyz	Professional
11	SaulHudson	slash@nodomain.xyz	Games
12	SaulHudson	slash@nodomain.xyz	Sports
13	TheDoctor	tomBaker@nodomain.xyz	Music
14	TheDoctor	tomBaker@nodomain.xyz	Personal

TIP: *Look back at the SELECT statement we built in part one and use it as a model for building this query. You will need to JOIN three tables together here and will need to sort by two columns.*

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

Appendix B – User Tags INSERT Statement

```
INSERT INTO vc_UserTagList (vc_UserID, vc_TagID)
VALUES
	((SELECT vc_UserID FROM vc_User WHERE UserName='DnDGal'),
	(SELECT vc_TagID FROM vc_Tag WHERE TagText='Sports')),
	((SELECT vc_UserID FROM vc_User WHERE UserName='DnDGal'),
	(SELECT vc_TagID FROM vc_Tag WHERE TagText='Professional')),
	((SELECT vc_UserID FROM vc_User WHERE UserName='RDwight'),
	(SELECT vc_TagID FROM vc_Tag WHERE TagText='Professional')),
	((SELECT vc_UserID FROM vc_User WHERE UserName='SaulHudson'),
	(SELECT vc_TagID FROM vc_Tag WHERE TagText='Sports')),
	((SELECT vc_UserID FROM vc_User WHERE UserName='Gordon'),
	(SELECT vc_TagID FROM vc_Tag WHERE TagText='Personal')),
	((SELECT vc_UserID FROM vc_User WHERE UserName='DnDGal'),
	(SELECT vc_TagID FROM vc_Tag WHERE TagText='Personal')),
	((SELECT vc_UserID FROM vc_User WHERE UserName='Gordon'),
	(SELECT vc_TagID FROM vc_Tag WHERE TagText='Games')),
	((SELECT vc_UserID FROM vc_User WHERE UserName='HairCut'),
	(SELECT vc_TagID FROM vc_Tag WHERE TagText='Professional')),
	((SELECT vc_UserID FROM vc_User WHERE UserName='TheDoctor'),
	(SELECT vc_TagID FROM vc_Tag WHERE TagText='Music')),
	((SELECT vc_UserID FROM vc_User WHERE UserName='DnDGal'),
	(SELECT vc_TagID FROM vc_Tag WHERE TagText='Games')),
	((SELECT vc_UserID FROM vc_User WHERE UserName='SaulHudson'),
	(SELECT vc_TagID FROM vc_Tag WHERE TagText='Games')),
	((SELECT vc_UserID FROM vc_User WHERE UserName='Gordon'),
	(SELECT vc_TagID FROM vc_Tag WHERE TagText='Professional')),
	((SELECT vc_UserID FROM vc_User WHERE UserName='HairCut'),
	(SELECT vc_TagID FROM vc_Tag WHERE TagText='Music')),
	((SELECT vc_UserID FROM vc_User WHERE UserName='TheDoctor'),
	(SELECT vc_TagID FROM vc_Tag WHERE TagText='Personal'))
```