Part 1 – Introducing Functions, Views, and Stored Procedures

1. FUNCTIONS
2. Declare a variable (we’ll talk about variables in a minute)
3. DECLARE @isThisNull VARCHAR(30) -- Starts out as NULL
4. SELECT @isThisNull, ISNULL(@isThisNull, 'Yep, it is null') -- See?
5. Set the variable to something other than NULL
6. SET @isThisNull = 'Nope. It is not NULL'
7. SELECT @isThisNull, ISNULL(@isThisNull, 'Yep, it is null') -- How about now?
8. CREATE FUNCTION dbo.AddTwoInts(@firstNumber int, @secondNumber int)
9. RETURNS int AS
10. BEGIN
11. Declare the variable to hold temporarily
12. DECLARE @returnValue int -- data type matches RETURNS clause
13. Perform the function task
14. SET @returnValue = @firstNumber + @secondNumber
15. --Return the value to the calling statement
16. RETURN @returnValue
17. END;
18. Check that the function runs correctly
19. SELECT dbo.AddTwoInts(7, 11);

1. Functions that are more…well…functional

1. --Abstract routine calculation
2. -- Function to count the VidCasts made by a given user
3. CREATE FUNCTION dbo.vc\_VidCastCount (@userID int)
4. RETURNS int AS -- COUNT() is an integer value, so return it as an int
5. BEGIN
6. DECLARE @returnValue int -- matches the function's return type
7. SELECT @returnValue = COUNT(vc\_UserID) FROM vc\_VidCast
8. WHERE vc\_VidCast.vc\_UserID = @userID
9. RETURN @returnValue
10. END;
11. SELECT TOP 10 \*, dbo.vc\_VidCastCount(vc\_UserID) as VidCastCount
12. FROM vc\_User
13. ORDER BY VidCastCount DESC;

What this code does is call all the columns, plus our new VidCastCount function with the VidCastCount alias, from the vc\_User table. The query orders the table in descending order using the VidCastCount column. It then selects the first ten rows of the table.

The function knows that vc\_UserID = 20 has 22 vc\_VidCast records because in the function, we are setting @userID as an integer, which the function will then pair to the values in the vc\_UserID column of the vc\_VidCast table.

2. Performing data lookups

1. --Perform Data Lookups
2. --Function to retrieve the vc\_TagID for a given tag's text
3. CREATE FUNCTION dbo.vc\_TagIDLookup (@tagText varchar(20))
4. RETURNS int AS
5. BEGIN
6. DECLARE @returnValue int
7. SELECT @returnValue = vc\_TagID FROM vc\_Tag
8. WHERE TagText = @tagText
9. RETURN @returnvalue
10. END;
11. SELECT dbo.vc\_TagIDLookup('Music');
12. SELECT dbo.vc\_TagIDLookup('Tunes');

What lines 48 and 49 are doing is checking that the input ‘Music’ and ‘Tunes’ is within the vc\_Tag table. The reason why ‘Tunes’ returns NULL is because that tag is not in the vc\_Tag table.

3. VIEWS

1. VIEWS
2. Create a view to retrieve the top 10 vc\_Users and VidCast counts
3. CREATE VIEW vc\_MostProlificUsers AS
4. SELECT TOP 10 \*, dbo.vc\_VidCastCount(vc\_UserID) as VidCastCount
5. FROM vc\_User
6. ORDER BY VidCastCount DESC;
7. SELECT \* FROM vc\_MostProlificUsers;

From the vc\_User table, we are selecting all the columns plus the vc\_VidCastCount function that we defined earlier. We are ordering the table by this last column in descending order. We select the first ten rows of the table and store these in the vc\_MostProlificUsers view that we create in line 56.

4. STORED PROCEDURES

1. --STORED PROCEDURES
2. --Create a procedure to update a vc\_User's email address
3. --The first parameter is the user name for the user to change
4. --The second is the new email address
5. CREATE PROCEDURE vc\_ChangeUserEmail(@userName varchar(20), @newEmail varchar(20)) AS
6. BEGIN
7. UPDATE vc\_User SET EmailAddress = @newEmail
8. WHERE UserName = @userName
9. END;
10. --Run the procedure
11. EXEC vc\_ChangeUserEmail 'tardy', 'kmstudent@syr.edu';
12. --Check procedure
13. SELECT \* FROM vc\_User WHERE UserName = 'tardy';

This section of the query creates a procedure that takes a username and an email address as inputs and updates the email address by matching the username input to the UserName column in the vc\_User table.

5. @@identity

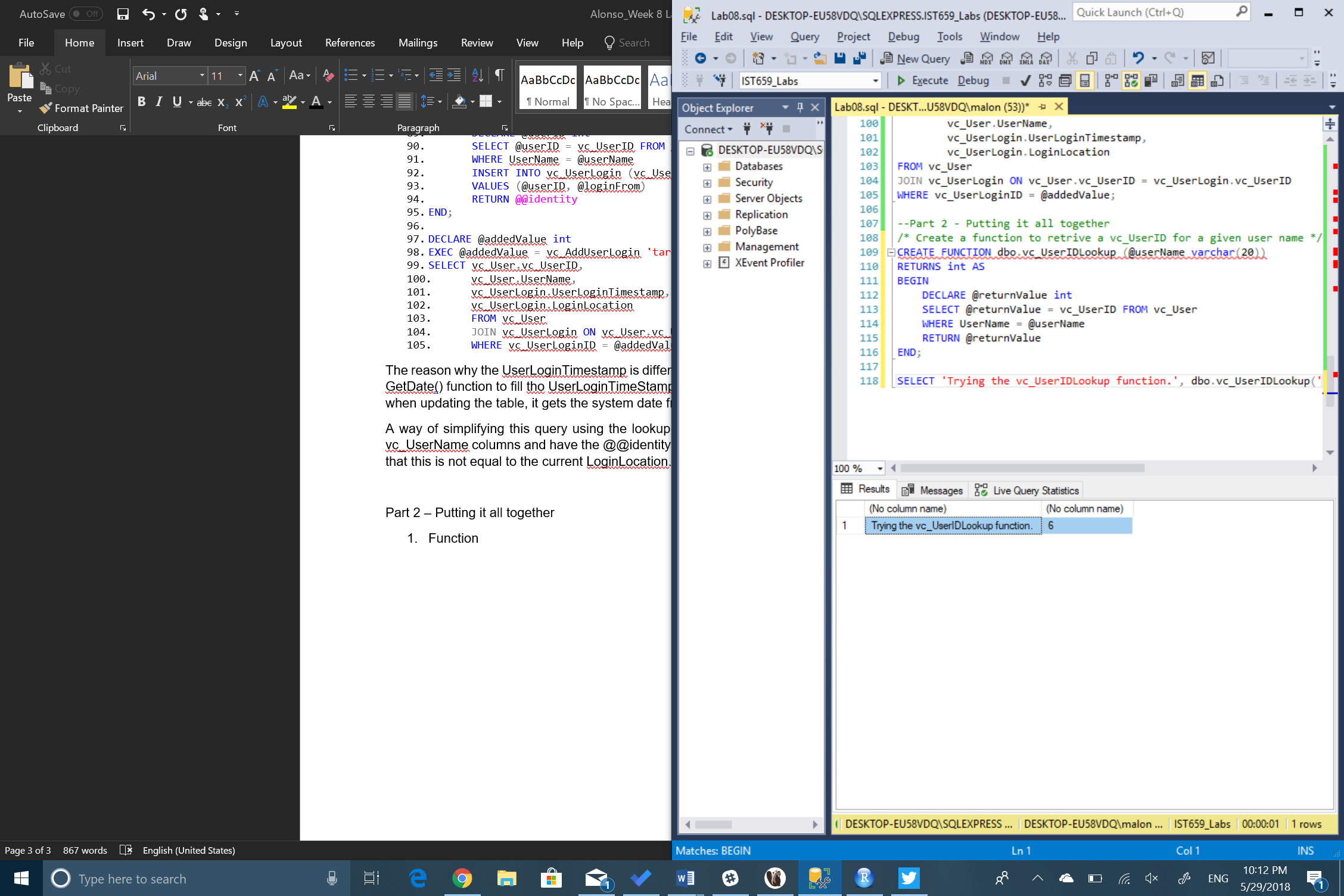
1. @@identity
2. INSERT INTO vc\_Tag (TagText) VALUES ('Cat Videos')
3. SELECT \* FROM vc\_Tag WHERE vc\_TagID = @@identity
4. /\*Create a procedure that adds a row to the UserLogin table.
5. This procedure is run when a user logs in and we need to
6. record who they are and from where they're logging in.\*/
7. CREATE PROCEDURE vc\_AddUserLogin (@userName varchar(20), @loginFrom varchar(50))
8. AS
9. BEGIN
10. DECLARE @userID int
11. SELECT @userID = vc\_UserID FROM vc\_User
12. WHERE UserName = @userName
13. INSERT INTO vc\_UserLogin (vc\_UserID, LoginLocation)
14. VALUES (@userID, @loginFrom)
15. RETURN @@identity
16. END;
17. DECLARE @addedValue int
18. EXEC @addedValue = vc\_AddUserLogin 'tardy', 'localhost'
19. SELECT vc\_User.vc\_UserID,
20. vc\_User.UserName,
21. vc\_UserLogin.UserLoginTimestamp,
22. vc\_UserLogin.LoginLocation
23. FROM vc\_User
24. JOIN vc\_UserLogin ON vc\_User.vc\_UserID = vc\_UserLogin.vc\_UserID
25. WHERE vc\_UserLoginID = @addedValue;

The reason why the UserLoginTimestamp is different is because the table vc\_UserLogin uses the GetDate() function to fill tho UserLoginTimeStamp column every time a query is run. This is why, when updating the table, it gets the system date from the computer and stores it into the column.

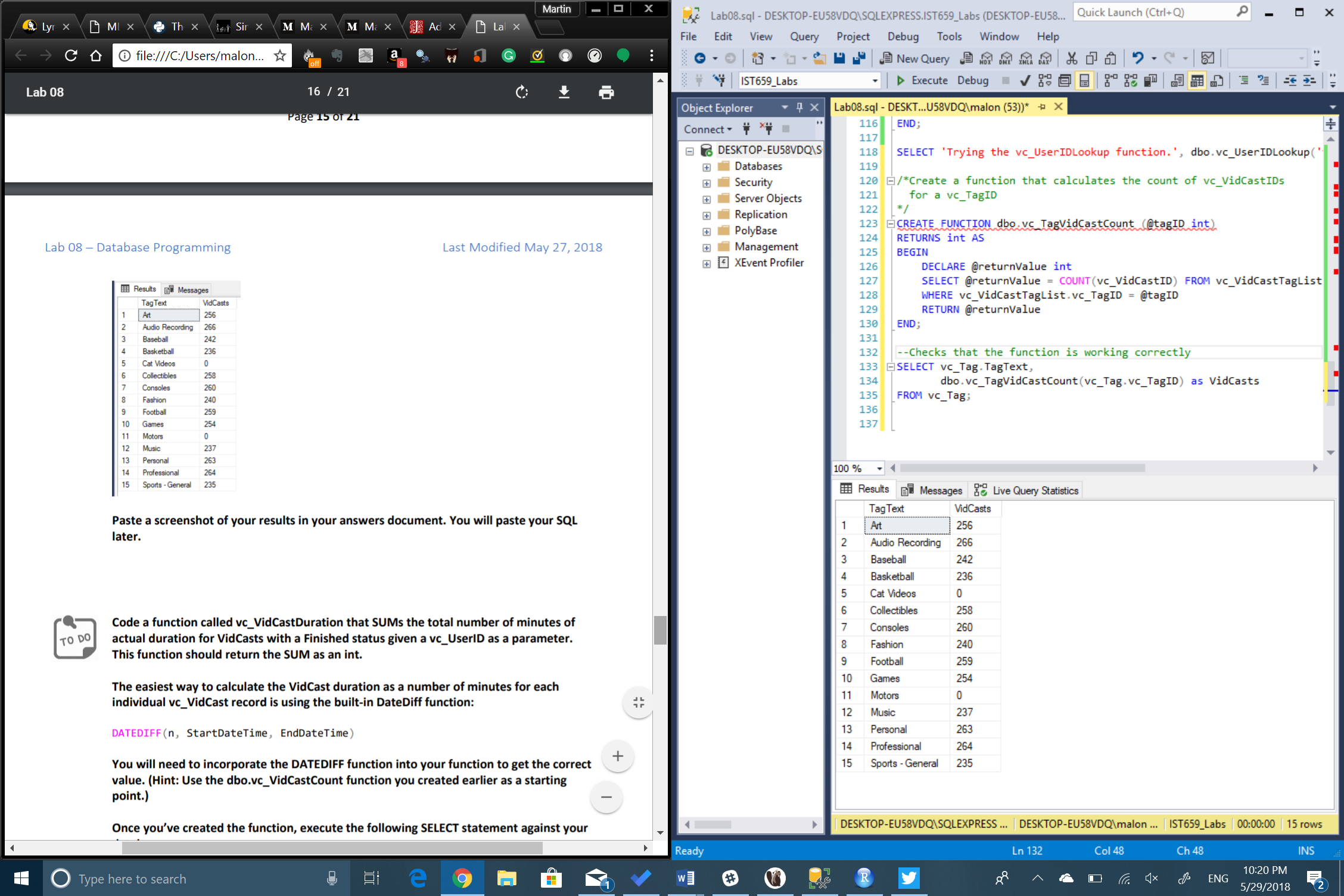
A way of simplifying this query using the lookup function would be to drop the vc\_UserID and vc\_UserName columns and have the @@identity script look for the LoginLocation tag and check that this is not equal to the current LoginLocation.

Part 2 – Putting it all together

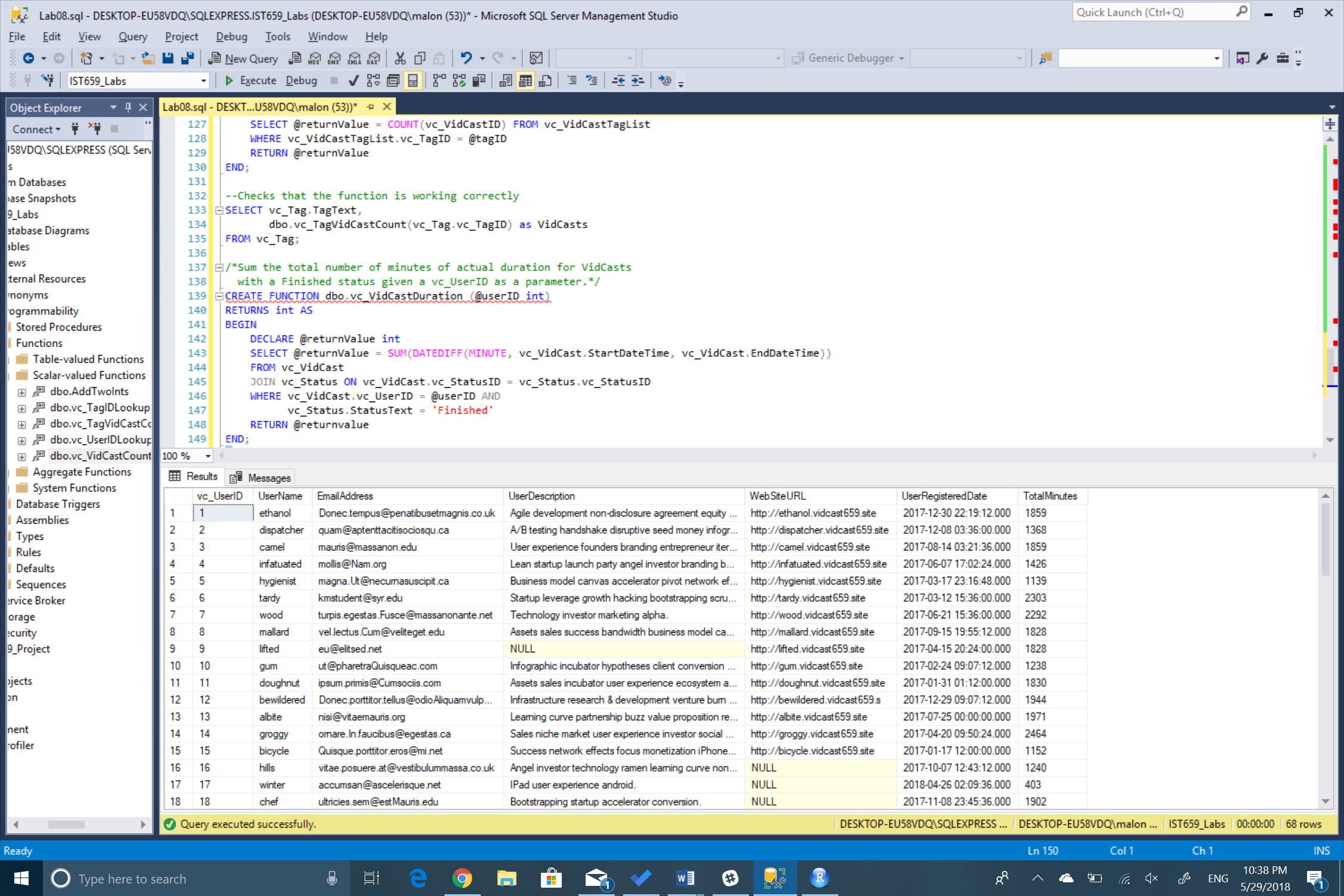
1. Lookup Function



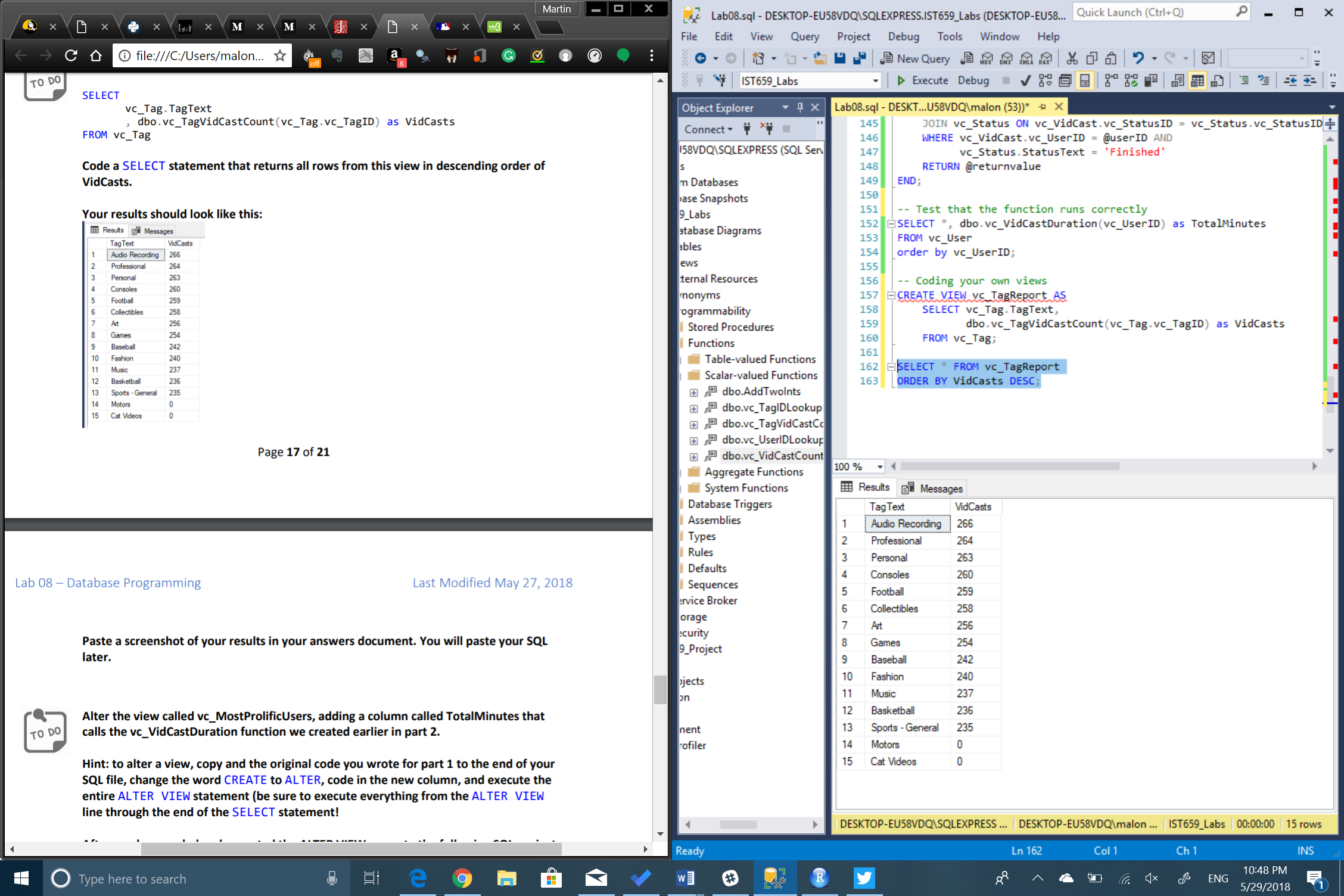
1. CountID Function



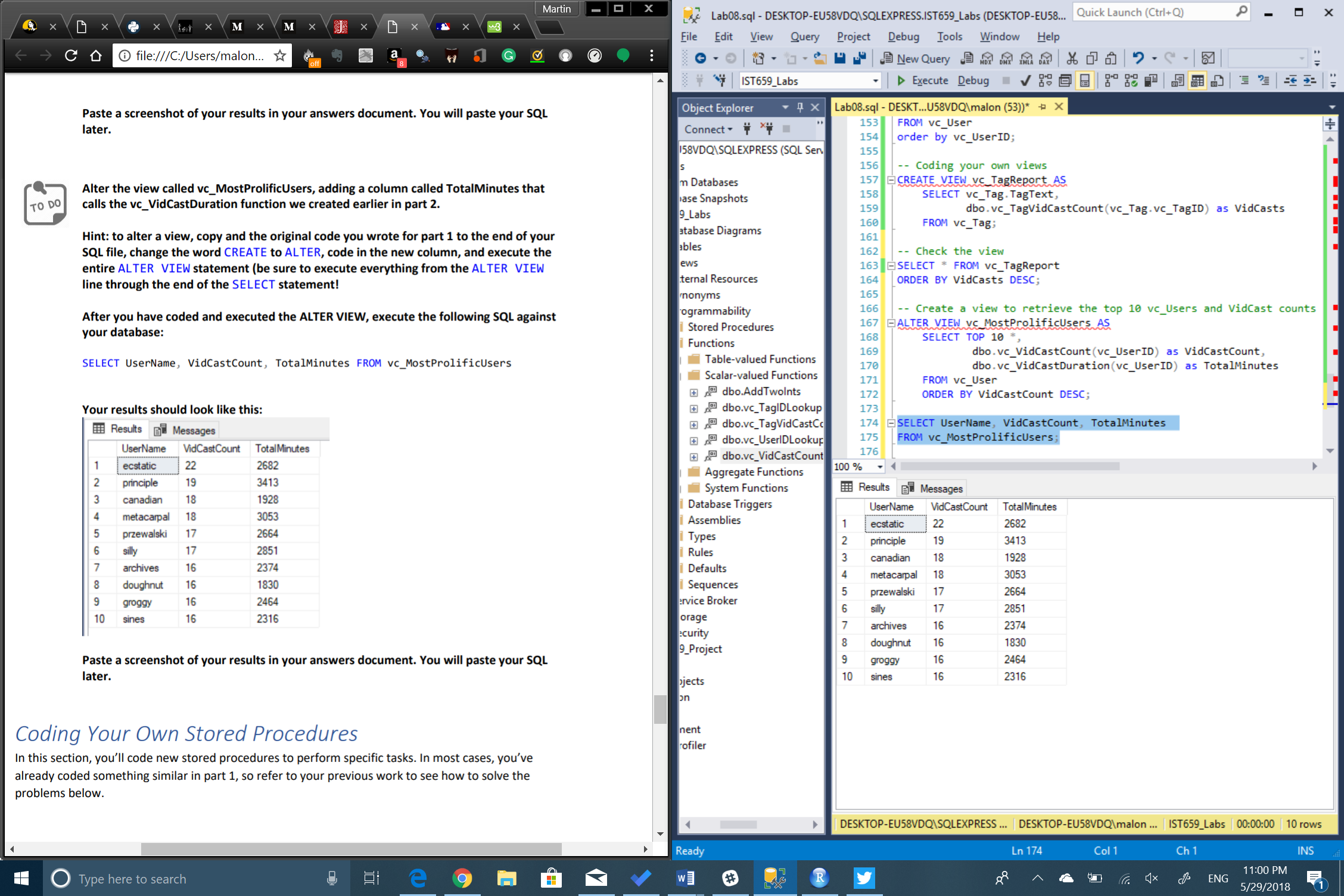
1. SUM DateDiff Function



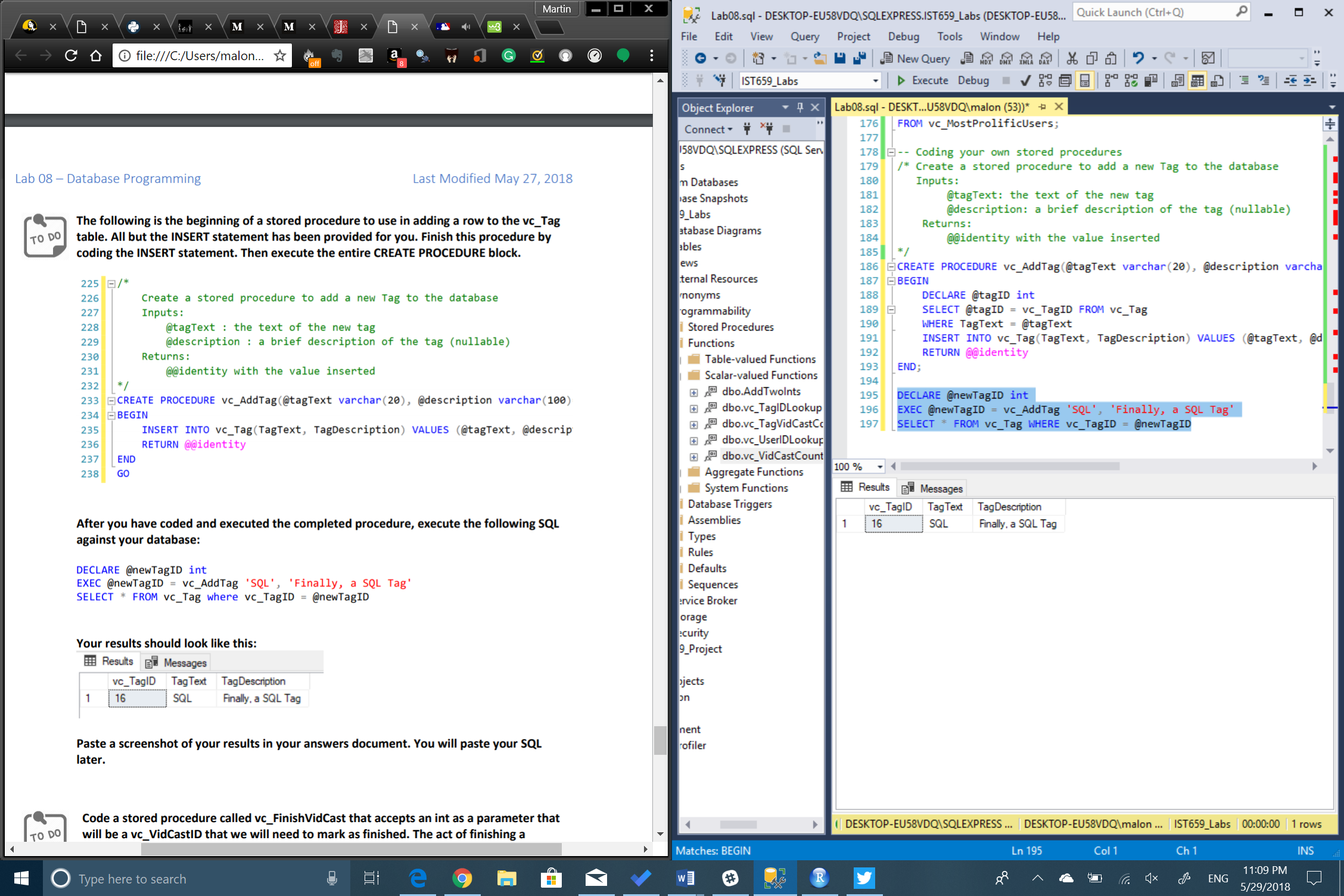
1. TagReport View



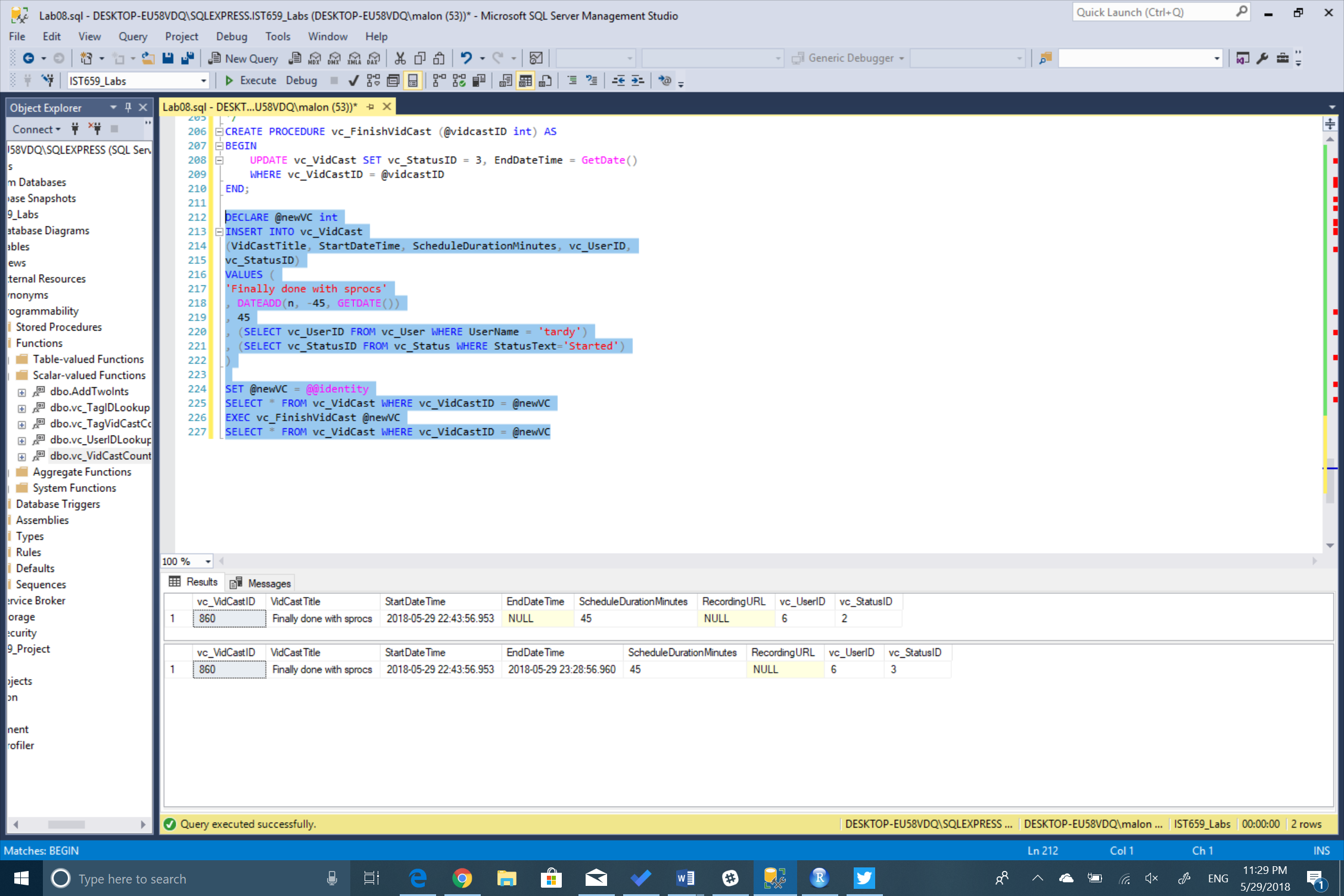
1. Altering the vc\_MostProlificUsers view



1. Create a new procedure



1. vc\_FinishVidCast



SQL Code

--Part 2 - Putting it all together

/\* Create a function to retrive a vc\_UserID for a given user name \*/

CREATE FUNCTION dbo.vc\_UserIDLookup (@userName varchar(20))

RETURNS int AS

BEGIN

DECLARE @returnValue int

SELECT @returnValue = vc\_UserID FROM vc\_User

WHERE UserName = @userName

RETURN @returnValue

END;

SELECT 'Trying the vc\_UserIDLookup function.', dbo.vc\_UserIDLookup('tardy');

/\*Create a function that calculates the count of vc\_VidCastIDs

for a vc\_TagID

\*/

CREATE FUNCTION dbo.vc\_TagVidCastCount (@tagID int)

RETURNS int AS

BEGIN

DECLARE @returnValue int

SELECT @returnValue = COUNT(vc\_VidCastID) FROM vc\_VidCastTagList

WHERE vc\_VidCastTagList.vc\_TagID = @tagID

RETURN @returnValue

END;

--Checks that the function is working correctly

SELECT vc\_Tag.TagText,

dbo.vc\_TagVidCastCount(vc\_Tag.vc\_TagID) as VidCasts

FROM vc\_Tag;

/\*Sum the total number of minutes of actual duration for VidCasts

with a Finished status given a vc\_UserID as a parameter.\*/

CREATE FUNCTION dbo.vc\_VidCastDuration (@userID int)

RETURNS int AS

BEGIN

DECLARE @returnValue int

SELECT @returnValue = SUM(DATEDIFF(MINUTE, vc\_VidCast.StartDateTime, vc\_VidCast.EndDateTime))

FROM vc\_VidCast

JOIN vc\_Status ON vc\_VidCast.vc\_StatusID = vc\_Status.vc\_StatusID

WHERE vc\_VidCast.vc\_UserID = @userID AND

vc\_Status.StatusText = 'Finished'

RETURN @returnvalue

END;

-- Test that the function runs correctly

SELECT \*, dbo.vc\_VidCastDuration(vc\_UserID) as TotalMinutes

FROM vc\_User

order by vc\_UserID;

-- Coding your own views

CREATE VIEW vc\_TagReport AS

SELECT vc\_Tag.TagText,

dbo.vc\_TagVidCastCount(vc\_Tag.vc\_TagID) as VidCasts

FROM vc\_Tag;

-- Check the view

SELECT \* FROM vc\_TagReport

ORDER BY VidCasts DESC;

-- Create a view to retrieve the top 10 vc\_Users and VidCast counts

ALTER VIEW vc\_MostProlificUsers AS

SELECT TOP 10 \*,

dbo.vc\_VidCastCount(vc\_UserID) as VidCastCount,

dbo.vc\_VidCastDuration(vc\_UserID) as TotalMinutes

FROM vc\_User

ORDER BY VidCastCount DESC;

-- Checking that everything worked fine

SELECT UserName, VidCastCount, TotalMinutes

FROM vc\_MostProlificUsers;

-- Coding your own stored procedures

/\* Create a stored procedure to add a new Tag to the database

Inputs:

@tagText: the text of the new tag

@description: a brief description of the tag (nullable)

Returns:

@@identity with the value inserted

\*/

CREATE PROCEDURE vc\_AddTag(@tagText varchar(20), @description varchar(100)) AS

BEGIN

DECLARE @tagID int

SELECT @tagID = vc\_TagID FROM vc\_Tag

WHERE TagText = @tagText

INSERT INTO vc\_Tag(TagText, TagDescription) VALUES (@tagText, @description)

RETURN @@identity

END;

DECLARE @newTagID int

EXEC @newTagID = vc\_AddTag 'SQL', 'Finally, a SQL Tag'

SELECT \* FROM vc\_Tag WHERE vc\_TagID = @newTagID

/\* Create a stored procedure that marks a VidCast as Finished

Inputs:

@vidcastID: the int of the vc\_VidCastID

@statusID: the new status

Returns:

@@identity with the value inserted

\*/

CREATE PROCEDURE vc\_FinishVidCast (@vidcastID int) AS

BEGIN

UPDATE vc\_VidCast SET vc\_StatusID = 3, EndDateTime = GetDate()

WHERE vc\_VidCastID = @vidcastID

END;

DECLARE @newVC int

INSERT INTO vc\_VidCast

(VidCastTitle, StartDateTime, ScheduleDurationMinutes, vc\_UserID,

vc\_StatusID)

VALUES (

'Finally done with sprocs'

, DATEADD(n, -45, GETDATE())

, 45

, (SELECT vc\_UserID FROM vc\_User WHERE UserName = 'tardy')

, (SELECT vc\_StatusID FROM vc\_Status WHERE StatusText='Started')

)

SET @newVC = @@identity

SELECT \* FROM vc\_VidCast WHERE vc\_VidCastID = @newVC

EXEC vc\_FinishVidCast @newVC

SELECT \* FROM vc\_VidCast WHERE vc\_VidCastID = @newVC