# Lab 08

## Part 1 – Introducing Functions, Views, and Stored Procedures

### Describing the use of the *dbo.vc\_VidCastCount* function

These statements are designed to select the top 10 values (from largest to smallest) for the counted number of videos all users in the vc\_VidCast table. These results are then listed in descending order from the user with the most broadcasted videos to the least (up to a count of ten users).

The code knows that user 20 (username “ecstatic”) has 20 videos because it looked at all the rows for user 20 in the vc\_VidCast table and then added all those rows to come up with the “VidCastCount” of 22 total videos broadcasted.

### Describing the use of the *dbo.vc\_TagIDLookup* function

The select statements from the dbo.vc\_TagIDLookup provides the TagID (the unique ID) for the tagtext passed to the function, so ‘Music’ has a TagID of 4 and there is not Tag ‘Tunes’ so that is why it returns NULL.

### Describing the use of VIEW *dbo.vc\_MostProlificUsers*

The select statement from the view provides a provide the same results as what we did in our first dbo.vc\_VidCastCount function by giving us all the information in the vc\_User table and then the total number of videos broadcast by each of those users. However, it subsets the return to only the top 10 users with the most video broadcast (the Most Prolific users).

### Describing the use of STORED PROCEDURE *vc\_ChangeUserEmail*

The stored procedure is basically a function we can call to do an update (in this case). We provide the function the username and a new email address for that username. The procedure does an UPDATE on the vc\_User table updating the user Email Address to the one passed to the procedure.

### Describing the use of STORED PROCEDURE *vc\_AddUserLogin*

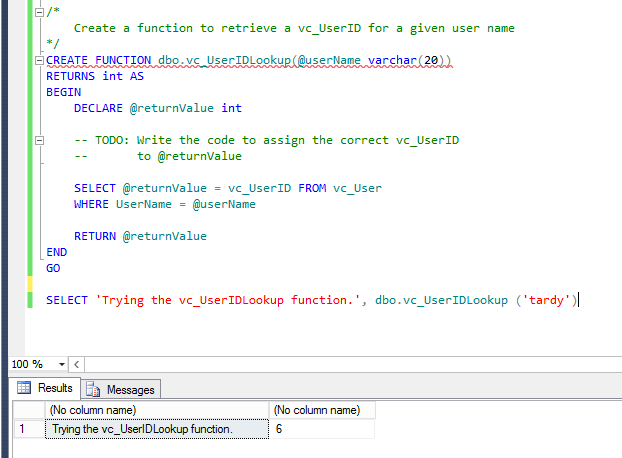
The timestamp when I execute this stored procedure is my current time/date because it reflects the “GetDate()” that we put in as a default for the UserLoginTimeStamp when we created the table and I just created the line for username ‘tardy’.

One way to simply the code is: ANSWER THIS

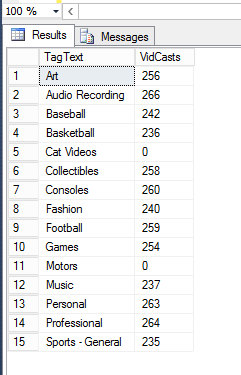
## Part 2 – Putting it All Together

### Coding your Own Functions

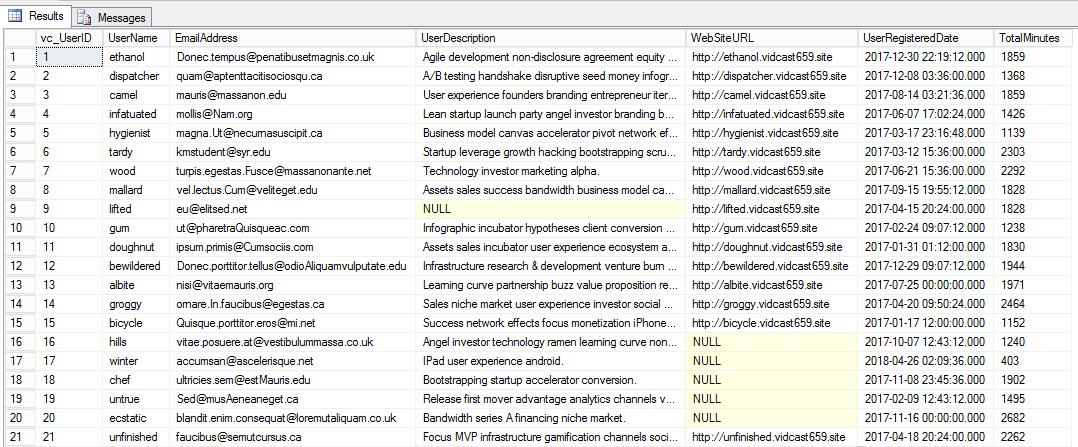
#### dbo.vc\_UserIDLookup



#### dbo.vc\_TagVidCastCount

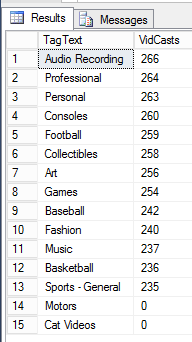


#### vc\_VidCastDuration

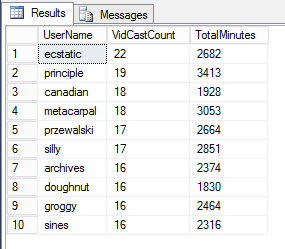


### Coding your own Views

#### Vc\_TagReport View

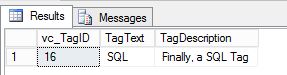


#### Altered vc\_MostProlificUsers

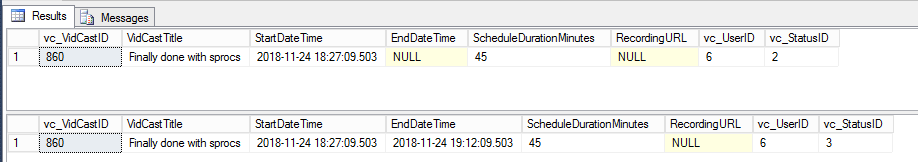


### Coding your own Stored Procedures

#### Add new Tag Stored Procedure



#### Vc\_FinishVidCast



# SQL Used

-- Declare a variable

declare @isThisNull varchar(30) -- Starts out as NULL

SELECT @isThisNull, ISNULL(@isThisNull, 'Yep, it is null') -- see?

-- Set the variable to somethin gother than NULL

SET @isThisNull = 'Nope. It is not NULL'

SELECT @isThisNull, ISNULL(@isThisNull, 'Yep, it is no null') -- how about now??

CREATE FUNCTION dbo.AddTwoInts(@firstNumber int, @secondNumber int)

RETURNS int AS

BEGIN

-- First, declare the variable to temporarily hold the result

DECLARE @returnValue int -- the data time matches the RETURNS clause

-- Do whatever needs to be done to set that variable to the correct value

SET @returnValue = @firstNumber + @secondNumber

-- Return the value to the calling statement

RETURN @returnValue

END

GO

SELECT dbo.AddTwoInts(5,10)

-- Function to count the VidCasts made by given User

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

RETURNS int AS -- COUNT() is an integer value, so return it as an int

BEGIN

DECLARE @returnValue int -- matches the function's return type

/\*

Get the count of the VidCasts for the provided UserID and

assign that value to @returnValue. Note that we use the

@userID parameter in the WHERE clause to limit our count

to that user's VidCast recrods.

\*/

SELECT @returnValue = COUNT(vc\_UserID) FROM vc\_VidCast

WHERE vc\_VidCast.vc\_UserID = @userID

-- Return @returnvalue to the calling code

RETURN @returnValue

END

GO

SELECT TOP 10

\*,

dbo.vc\_VidCastCount(vc\_userID) as VidCastCount

FROM vc\_User

ORDER BY VidCastCount DESC

GO

-- Function to retrieve the vc\_TagID for a given tag's text

CREATE FUNCTION dbo.vc\_TagIDLookup(@tagText varchar(20))

RETURNS int AS -- vc\_TagID is an int, so we will match that

BEGIN

DECLARE @returnValue int -- Matches the function's return type

/\*

Get the vc\_TagID of the vc\_Tag record whose TagText

matches the parameter and assign that value to @returnValue

\*/

SELECT @returnValue = vc\_TagID FROM vc\_Tag

WHERE TagText = @tagText

-- Send the vc\_TagID back to the caller

RETURN @returnValue

END

GO

-- Test our function

SELECT dbo.vc\_TagIDLookup('Music')

SELECT dbo.vc\_TagIDLookup('Tunes')

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

-- VidCast counts

CREATE VIEW vc\_MostProlificUsers AS

SELECT TOP 10

\*,

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

FROM vc\_User

ORDER BY VidCastCount DESC

GO

-- Select all from this new view

SELECT \* from vc\_MostProlificUsers

-- Create a procedure to update a vc\_User's email address

-- the first parameter is the user name for the user to change

-- the second is the new email address

CREATE PROCEDURE vc\_ChangeUserEmail(@userName varchar(20), @newEmail varchar(50))

AS

BEGIN

UPDATE vc\_User SET EmailAddress = @newEmail

WHERE UserName = @userName

END

GO

-- Execute the stored procedure

EXEC vc\_ChangeUserEmail 'tardy', 'kmstudent@syr.edu'

SELECT \* from vc\_User WHERE UserName = 'tardy'

INSERT INTO vc\_Tag (TagText) VALUES ('Cat Videos')

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

/\*

Create a procedure that adds a row to the UserLogin tabl

This procedure is run when a user logs in and we need

to record who they are and from where they're logging in

\*/

CREATE PROCEDURE vc\_AddUserLogin(@userName varchar(20), @loginFrom varchar(50))

AS

BEGIN

-- We have the user name, but we need tthe ID for the login table

-- First, declare a variable to hold the ID

DECLARE @userID int

-- Get the vc\_userID for the UserName provided and store it in the UserID

SELECT @userID = vc\_UserID FROM vc\_User

WHERE UserName = @userName

-- Now we can add the row using the INSERT statement

INSERT INTO vc\_UserLogin (vc\_userID, LoginLocation)

VALUES (@userID, @loginFrom)

-- Now return the @@identity (unique identity key) so the calling code

-- knows where the data ended up

RETURN @@identity

END

GO

DECLARE @addedValue int

EXEC @addedValue = vc\_AddUserLogin 'tardy', 'localhost'

SELECT

vc\_User.vc\_UserID,

vc\_User.UserName,

vc\_UserLogin.userLoginTimestamp,

vc\_UserLogin.LoginLocation

FROM vc\_user

JOIN vc\_UserLogin on vc\_User.vc\_UserID = vc\_UserLogin.vc\_UserID

WHERE vc\_UserLoginID = @addedValue

/\*

Create a function to retrieve a vc\_UserID for a given user name

\*/

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

RETURNS int AS

BEGIN

DECLARE @returnValue int

-- TODO: Write the code to assign the correct vc\_UserID

-- to @returnValue

SELECT @returnValue = vc\_UserID FROM vc\_User

WHERE UserName = @userName

RETURN @returnValue

END

GO

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

/\*

Create a fucntion called dbo.vc\_TagVidCastCount that calculates the count

of vc\_VidCastIDs for a given vc\_TagID.

\*/

CREATE FUNCTION dbo.vc\_TagVidCastCount(@Tag\_ID int)

-- COUNT() is an integer value, so return it as an int

RETURNS int AS

BEGIN

DECLARE @returnValue int -- matches the function's return type

/\*

Get the count of the VidCasts for the provided TagID and

assign that value to @returnValue. Note that we use the

@TagID parameter in the WHERE clause to limit our count

to those VidCast with the TagID passed to the function.

\*/

SELECT @returnValue = COUNT(vc\_VidCastID)

FROM vc\_VidCastTagList

JOIN vc\_Tag on vc\_VidCastTagList.vc\_TagID = vc\_Tag.vc\_TagID

WHERE vc\_VidCastTagList.vc\_TagID = @Tag\_ID

-- Return @returnvalue to the calling code

RETURN @returnValue

END

GO

-- Test function

SELECT

vc\_Tag.TagText,

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

FROM vc\_Tag

/\*

Function vc\_VidCastDuration that sums the total number of minutes of

actual duration of VidCasts with a Finished Status given a specific

vc\_UserID as a parameter. Should return the SUM as an integer

\*/

CREATE FUNCTION dbo.vc\_VidCastDuration(@User\_ID int)

-- SUM() is an integer value, so return it as an int

RETURNS int AS

BEGIN

DECLARE @returnValue int -- matches the function's return type

/\*

Get the sum of the vidcast actual durations for the

provided @User\_ID tag and assign that value to @returnValue.

Note that we use the @User\_ID parameter in the WHERE clause

limit our sum to those VidCast associated with @User\_ID

that was passed to the function.

\*/

SELECT @returnValue = SUM(DATEDIFF(n, StartDateTIme, EndDateTime))

FROM vc\_VidCast

INNER JOIN vc\_User ON vc\_VidCast.vc\_UserID = vc\_User.vc\_UserID

INNER JOIN vc\_Status on vc\_VidCast.vc\_StatusID = (SELECT vc\_Status.vc\_StatusID FROM vc\_Status where vc\_Status.StatusText = 'Finished')

WHERE vc\_VidCast.vc\_UserID = @User\_ID AND vc\_Status.StatusText = 'Finished'

GROUP BY vc\_VidCast.vc\_UserID

-- Return @returnvalue to the calling code

RETURN @returnValue

END

GO

-- Test function

SELECT

\*,

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

FROM vc\_User

/\*

Create a View to hold the TagText and the total number of VidCasts

that are tagged with that text

\*/

CREATE VIEW vc\_TagReport AS

SELECT TOP 15

vc\_Tag.TagText,

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

FROM vc\_Tag

ORDER BY VidCasts DESC

GO

-- test new view

SELECT \* from vc\_TagReport

-- Alter a view to retrieve the top 10 vc\_Users and their

-- VidCast counts adding in the total VidCastDuration

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

GO

-- Select all from this altered view

SELECT UserName, VidCastCount, TotalMinutes

FROM vc\_MostProlificUsers

/\*

Create a stored procedure to add a new Tag to the database

Inputs:

@Tag\_Text : the text of the new tag

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

Returns:

@@identity with the value inserted

\*/

CREATE PROCEDURE vc\_AddTag(@Tag\_Text varchar(20), @description varchar(100)=NULL) AS

BEGIN

--- Code Procedure Here

INSERT INTO vc\_Tag

(TagText, TagDescription)

VALUES

(@Tag\_Text, @description)

RETURN @@identity

END

GO

DECLARE @newTagID int

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

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

/\*

vc\_FinishVidCast Stored Procedure

Inputs : @Vid\_CastID (integer) - that will need to be marked as Finished

To accomplish this, we need to change the EndDateTime in the vc\_VidCast table to the current date/time - GetDate()

and then change the StatusID on the Vid\_CastID to match the vc\_StatusID in the Status table for Finished

- the change vc\_StatusID will be updated to this value in the vc\_VidCast table

\*/

CREATE PROCEDURE vc\_FinishVidCast(@Vid\_CastID int) AS

BEGIN

--- Code Procedure Here

-- Update the VidCast EndDate to now

UPDATE vc\_VidCast

SET

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

EndDateTime = GETDATE()

WHERE vc\_VidCastID = @Vid\_CastID

RETURN @@identity

END

GO

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