Part 1 Securing Data objects

Creating a database user

```
-- creating a guestuser database user
create user guestuser for login guestuser
```

All rows from vc UserLogin table

III	Results	₽ Messages						
	vc_UserLoginID		vc_UserID	UserLoginTimestamp	LoginLocation			
1	1		6	2020-11-19 21:44:13.473	localhost			
2	2		66	2020-11-25 12:32:52.817	Gallifrey			

vc_Vidcast that was modified by guestuser

	■ Results										
	vc_VidCastID	VidCast Title	Start Date Time	EndDateTime	Schedule Duration Minutes	RecordingURL	vc_UserID	vc_StatusID			
1	838	Rock Your Way To Success	2018-03-01 13:12:00.000	2020-11-25 13:24:39.510	63	NULL	62	3			

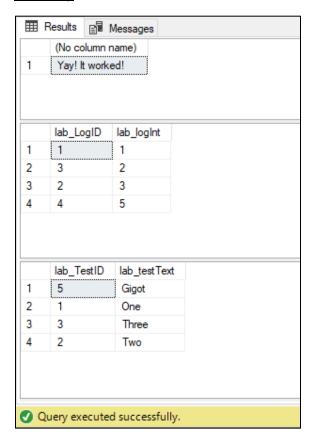
Code from my query tab

```
-- creating a guestuser database user
create user guestuser for login guestuser
-- grant read permission on the user table
grant select on vc_user to guestuser
-- revoke the select permission!
revoke select on vc User to guestuser
-- give them the view instead
grant select on vc_MostProlificUsers to guestuser
-- allow guestuser to run some stored procedures
grant execute on vc_AddUserLogin to guestuser
grant execute on vc_FinishVidCast to guestuser
-- determine parameters to finish VidCast titled 'Rock Your Way to Success'
select VidCastTitle ,UserName, StatusText
from [IST_659_Vidcast].[dbo].[vc_VidCast]
join vc_User on vc_User.vc_UserID = vc_VidCast.vc_UserID
join vc_Status on vc_Status.vc_StatusID = vc_VidCast.vc_StatusID
where VidCastTitle = 'Rock Your Way to Success'
-- grant permissions to guestuser to finish VidCast titled 'Rock Your Way To Success'
grant select, insert, update, delete on vc_Status to guestuser
grant select, insert, update, delete on vc_VidCast to guestuser
grant select, insert, update, delete on vc_User to guestuser
```

```
-- retrieve all rows from the vc_UserLogin table
select * from vc_UserLogin
-- retrieve only vc_VidCast record titled 'Rock Your Way To Success'
select * from [IST_659_Vidcast].[dbo].[vc_VidCast]
where VidCastTitle = 'Rock Your Way To Success'
Code from guestusers query tab
-- guestuser's tab
select * from vc_User
go -- attempt after security change
select * from vc_MostProlificUsers
go -- execute procedure to add userlogin for the TheDoctor
declare @addedValue int
exec @addedValue = vc_AddUserLogin 'TheDoctor', 'Gallifrey'
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
go -- execute procedure to finish VidCast titled 'Rock Your Way to Success'
DECLARE @newVC int
INSERT INTO vc VidCast
(VidCastTitle, StartDateTime, ScheduleDurationMinutes, vc_UserID,
vc StatusID)
VALUES (
'Rock Your Way to Success'
, DATEADD(n, -45, GETDATE())
, (SELECT vc_UserID FROM vc_User WHERE UserName = 'humdrum')
, (SELECT vc_StatusID FROM vc_Status WHERE StatusText='Finished'))
SET @newVC = @@identity
SELECT * FROM vc_VidCast WHERE vc_VidCastID = @newVC
EXEC vc FinishVidCast @newVC
SELECT * FROM vc_VidCast WHERE vc_VidCastID = @newVC
```

Part 2 Data Integrity Through Transactions

The Setup



Count the VidCastID's for a given TagID function

The function behaved as I would expect to, that is, it did not allow for a record to be entered with testText = 'One' because it would be a duplicate which we stated in the code is not allowed or else rollback. It did, however, allow for a record to be entered with testText = my last name because that record did not previously exist in the database. I did not expect the lab_TestID in this case to be entered as '5', but if we wanted it to be '4' we could adjust the code accordingly.

Code for Part 2

```
--add records to lab_Test and lab_Log
insert into lab_Test (lab_testText) values ('One'), ('Two'), ('Three')
insert into lab_Log (lab_logInt) select lab_TestID from lab_Test
-- use a transaction to make sure our data conform to our wierd rules
-- Step 1: begin the transaction
begin transaction
      --Step 2: assess the state of things
      declare @rc int
      set @rc = @@rowcount -- initially 0
      -- Step 3: make the change
      -- on success, @@rowcount is incremented by 1
      -- on failure, @@rowcount does not change
      insert into lab_Test (lab_testText) values ('Gigot')
      --Step 4: check the new state of things
      if(@rc = @@rowcount) -- if @@rowcount was not changed, fail
      begin
              -- Step 5, if failed
             select 'Bail out! It Failed!'
             rollback
      end
      else -- Success! Continue
      begin
             -- Step 5, if succeeded
             select 'Yay! It worked!'
             insert into lab_Log (lab_logInt) values (@@identity)
             commit
      end
select * from lab_Log
select * from lab_Test
```