University of Washington

IT Foundations

# Views, Functions, and Stored Procedures

As your **SQL** Select **statements** become complex, you may decide to **save** them **in** a **text** **file** **for repeated use**. (A text file with SQL code in it is **called a SQL Script**!) **Alternatively**, you can **save** your Select statements **within a database's file as a SQL View, Function, or Stored Procedure**.

**SQL Views and Functions are "Named" Select statements** whose code is stored in a database. **Stored Procedures** are similar, but they are **not restricted to just Select** statements. **This module** **looks** at how you **create and use these tools for selecting report data**.

Contents

[Views, Functions, and Stored Procedures 1](#_Toc65060089)

[Views 2](#_Toc65060090)

[Views and the Order By Clause 2](#_Toc65060091)

[Order By Issues in Views and Functions 3](#_Toc65060092)

[Partitioning Data with Views 5](#_Toc65060093)

[Lab 1: Creating Reporting Views 7](#_Toc65060094)

[Creating Base Views 8](#_Toc65060095)

[Schema Binding 10](#_Toc65060096)

[Functions 11](#_Toc65060097)

[Functions with Parameters 12](#_Toc65060098)

[Scalars Functions 13](#_Toc65060099)

[Lab 2: Creating Reporting Functions 14](#_Toc65060100)

[Stored Procedures 15](#_Toc65060101)

[Lab 3: Creating Reporting Stored Procedures 16](#_Toc65060102)

[GitHub 17](#_Toc65060103)

## Views

**Any view that is used to extract data for reporting purposes is called a "Reporting View."** They can save simple or complex Select statements, but more complex ones are typical.

Here is an **example** of **a view saving a complex select statement**. The select statement includes a join, column aliases, concatenation, and the choose function.

Create -- Drop

View vAuthorsByTitles

AS

Select

[Title] = T.title

,[Author]= A.au\_fname + ' ' + A.au\_lname

,[Order On Title] = Choose(TA.au\_ord, '1st', '2nd', '3rd')

From pubs.dbo.titles as T

Join pubs.dbo.titleauthor as TA

On T.title\_id = TA.title\_id

Join pubs.dbo.authors as A

On TA.au\_id = A.au\_id

Note that creating the view does not run the select code, so the **Create statement results are its success**, not data.



Figure: The results of the previous SQL statement

However, we can **run** **the saved code** **using** only a **simple Select statement, which makes** **complex** **SQL code** much **easier to use**.

Select **\* from** vAuthorsByTitles;

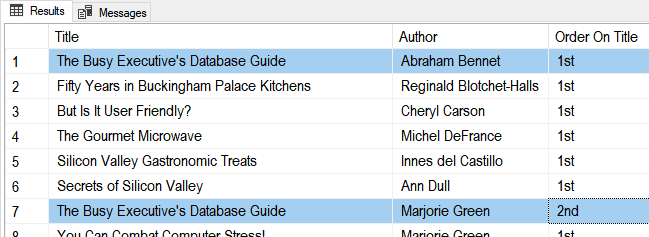


Figure: The results of the previous SQL statement

### Views and the Order By Clause

**By** **design**, **views cannot** **dictate how that data is sorted**. In some RDMS you can **use the TOP clause** to **trick** the system into allowing the Order By clause like this example:

Alter View vAuthorsByTitles

AS

Select **TOP 1000000000** --< You Need this,

[Title] = T.title

,[Author]= A.au\_fname + ' ' + A.au\_lname

,[Order On Title] = Choose(TA.au\_ord, '1st', '2nd', '3rd')

From pubs.dbo.titles as T

Join pubs.dbo.titleauthor as TA

On T.title\_id = TA.title\_id

Join pubs.dbo.authors as A

On TA.au\_id = A.au\_id

**Order By T.title, TA.au\_ord;** --< if you want to use this!

go

go

Select \* from vAuthorsByTitles

However, t**his MAY NOT be considered a best practice**. Instead, you can always use the Order By clause when you select from the View like this:

Select \* from vAuthorsByTitles Order By Title, Author;

***Note:*** *I have seen this trick used so often in the industry that you should know what it does and how to use it. Just remember to ask if your organization does or does not want to use it.*

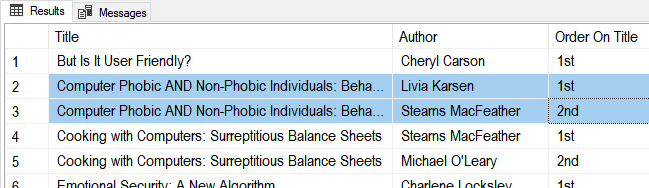


Figure: The results of the previous SQL statement

### Order By Issues in Views and Functions

Order By is not supposed to be used in a view or function. However, it is something you will see coded anyway. Once issue to be aware of is the odd way the Order By clause works when the Top command uses the Percent option. Here are some examples that show the issue.

Create or Alter View vTest

As

Select Top 100 Percent -- When this it...

CompanyName, Country From Customers

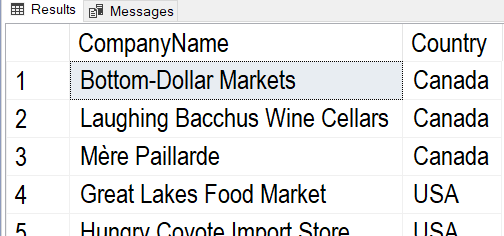
Where Country in ('USA', 'Canada')

Order By Country, CompanyName; -- Does Not Work!

Go

Select \* From vTest;

Go



Create or Alter View vTest

As

Select Top 100 Percent -- When this it...

CompanyName, Country From Customers

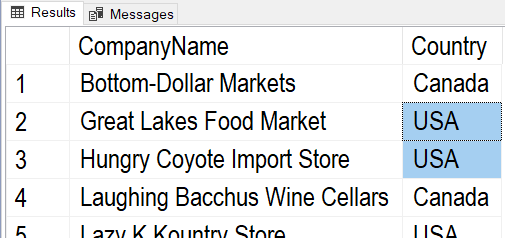
Where Country in ('USA', 'Canada')

Order By Country, CompanyName; -- Does Not Work!

Go

Select \* From vTest;

Go



Create or Alter View vTest

As

Select Top 100000 -- When this it...

CompanyName, Country From Customers

Where Country in ('USA', 'Canada')

Order By Country, CompanyName; -- **Works**!

Go

Select \* From vTest;

Go

Create or Alter View vTest

As

Select Top 99 Percent -- When this it...

CompanyName, Country From Customers

Where Country in ('USA', 'Canada')

Order By Country, CompanyName; -- Works, but you could lose rows!

Go

Select \* From vTest;

Go

Create or Alter Function fTest() -- Functions have the same issue

Returns Table

As

Return (

Select Top 100 Percent -- When this it...

CompanyName, Country From Customers

Where Country in ('USA', 'Canada')

Order By Country, CompanyName -- Does Not Work!

);

Go

### Partitioning Data with Views

Views allow you to "**split" data by rows (horizontal partitioning)** stored in one table (At least from a visual aspect). You can divide data by rows by **using a Where clause**. This is using when creating specialized reports.

Select \* from Pubs.dbo.Sales;

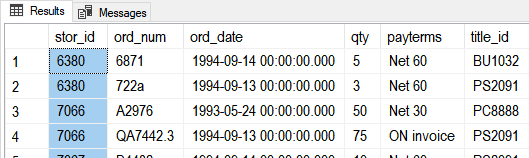


Figure: The results of the previous SQL statement

Create View **Store6380Sales** AS

Select St.stor\_name as Store, S.\*

From Pubs.dbo.Sales as S Join Pubs.dbo.stores as St

On S.stor\_id = St.stor\_id **Where S.stor\_id = 6380**;

go

Create View **Store7066Sales** AS

Select St.stor\_name as Store, S.\*

From Pubs.dbo.Sales as S Join Pubs.dbo.stores as St

On S.stor\_id = St.stor\_id **Where S.stor\_id = 7066**;

go

Select \* From Store6380Sales;

Select \* From Store7066Sales;

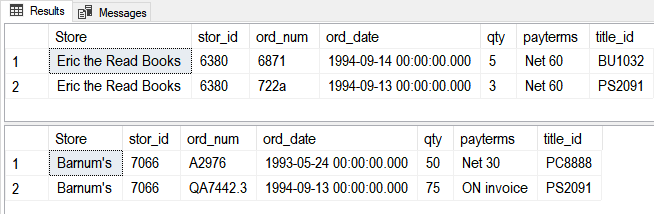


Figure: The results of the previous SQL statements

You can also **divide data by Columns** **(vertical partitioning)** too like this:

Select \* From Northwind.dbo.Employees;

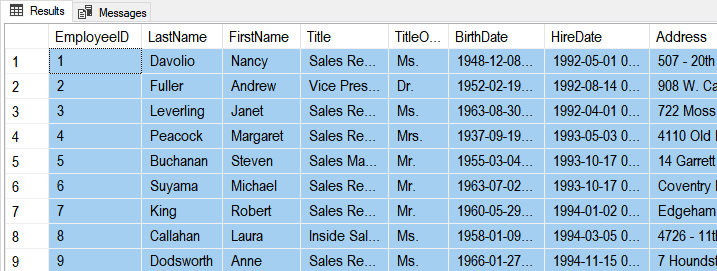


Figure: The results of the previous SQL statement

Create View **vPublicEmployeeInfo**

As

Select

TitleOfCourtesy

,FirstName

,LastName

,Title

From Northwind.dbo.Employees;

go

Create View **vPrivateEmployeeInfo**

As

Select

EmployeeID

,LastName

,FirstName

,Title

,TitleOfCourtesy

,BirthDate

,HireDate

,Address

,City

,Region

,PostalCode

,Country

From Northwind.dbo.Employees;

Go

Select \* From vPublicEmployeeInfo;

Select \* From vPrivateEmployeeInfo;

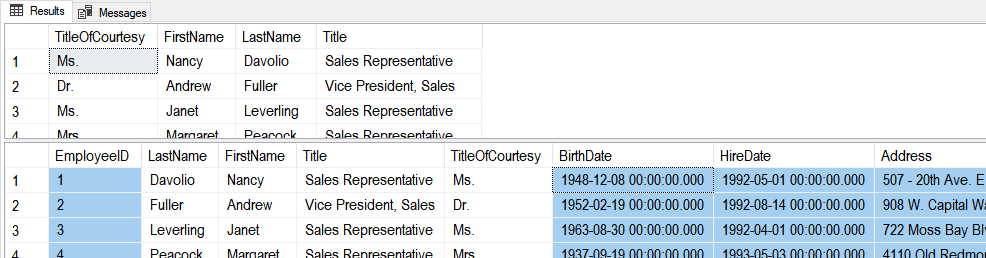


Figure: The results of the previous SQL statement

Now you would **protect the private data with permissions** and allow people to only access that employees table data through the appropriate views.

Use Northwind;

**Deny** Select On Employees to Public;

Use Module06Demos;

**Deny** Select vPrivateEmployeeInfo to Public;

**Grant** Select On vPublicEmployeeInfo to Public;

**Grant** Select vPrivateEmployeeInfo to HR;

## Lab 1: Creating Reporting Views

In this lab, you create views using the Northwind database from your own lab database.

**Step 1: Review Database Tables**

Run the following code in a SQL query editor and review the names of the tables you have to work with.

Select \* From Northwind.Sys.Tables Where type = 'u' Order By Name;

**Step 2: Create a Lab Database**

Create a new database for this lab called Mod06LabsYourNameHere (using your own name, of course!) Modify and use the following code to accomplish this:

Create Database Mod06LabsYourNameHere;

go

Use Mod06LabsYourNameHere;

go

**Step 3: Create a Query**

Answer the following questions by writing and executing SQL code.

**Question 1**: How can you create a view to show a list of **customer** names and their **locations? Use the IsNull()** function to display null region names as the name of the customer's country? Call the view **vCustomersByLocation.**

Select \* From vCustomersByLocation Order By CustomerName;

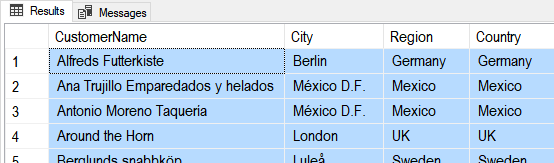


Figure: The query results of question 1

**Question 2**: How can you create a view to show a list of **customer** names, their **locations**, and the **number of orders** they have placed (hint: use the **count()** function)? Call the view **vNumberOfCustomerOrdersByLocation**.

Select \* From vNumberOfCustomerOrdersByLocation Order By CustomerName;



Figure: The query results of question 2

**Question 3**: How can you create a view to shows a list of **customer** names, their **locations**, and the **number of orders** they have placed (hint: use the **count()** function) on a given **year** (hint: use the **year()** function)? Call the view **vNumberOfCustomerOrdersByLocationAndYears**.

Select \* From vNumberOfCustomerOrdersByLocationAndYears Order By CustomerName, OrderYear;



Figure: The query results of question 3

**Step 4: Review Your Work**

Now, you will review your work with your instructor.

## Creating Base Views

In addition to any reporting views you create, **each table** in a database **should** **have a "Base" or "Basic" view** to show data from that table. When you **make a table**, you **create a base view** and then **restrict access to the table** while **allowing access to the View**. Base views allow people to use your data in the "Abstract."

Create -- Drop

Table tblCustomers -- Note: "tbl" is a common prefix in some databases

(CustomerID int Identity Primary Key, CustomerName nVarchar(100));

go

Insert Into tblCustomers (CustomerName) Values ('Bob Smith'),('Sue Jones');

go

-- Make a matching view!

Create View Customers

AS

Select CustomerID, CustomerName From dbo.tblCustomers;

go

-- Without a prefix in the name, like vCustomers, people may think your View is a table!

Select \* from Customers;

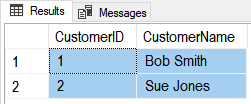


Figure: The results of the previous SQL statement

**NOTE: All tables should have a Base view that all other queries build on!**

**Abstraction layer objects,** like views, **make changes to a table design** **easier.** They help you maintain the way applications access the data. When a database table needs to be changed, **applications** that use the View can **continue to work if** **you** **modify the View's code to hide those changes**.

-- Step 1) Make a backup code of the data and drop the existing table

Select \* Into #TempCustomers from tblCustomers;

-- Step 2) Make the changes to the original table and replace its data

-- 2.1

Drop Table tblCustomers;

go

-- 2.2

Create Table tblCustomers

( CustomerID int Identity Primary Key

, CustomerFirstName nVarchar(100)

, CustomerLastName nVarchar(100)

);

go

-- 2.3

Insert Into tblCustomers (CustomerFirstName, CustomerLastName)

Select

CustomerFirstName = Substring(CustomerName, 1,3) -- use a function to split the data

,CustomerLastName = Substring(CustomerName, 4,100) -- use a function to split the data

From #TempCustomers;

go

-- Step 3) Make it look like the data was not split!

Alter

View Customers

AS

Select CustomerID, CustomerName = CustomerFirstName + ' ' + CustomerLastName

From dbo.tblCustomers;

go

-- Step 5) Also, add a new view to show the new design changes

Create

View CustomersNormalized

AS

Select CustomerID, CustomerFirstName, CustomerLastName

From dbo.tblCustomers;

go

-- Step 6) You set permissions to force developers into using the View and not the actual table

Deny Select On tblCustomers to Public;

Grant Select On Customers to Public;

-- Step 7) Verify the changes

Select \* From tblCustomers;

Select \* From Customers;

Select \* From CustomersNormalized;

go

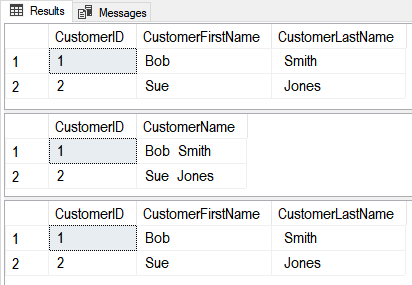


Figure: The results of the previous SQL statements

### Schema Binding

Foreign Keys will not protect a Parent table from being Dropped, so some developers will create Views using the Schema Binding option, which **keeps tables from changing so much the View does not work anymore**!

Use Northwind;

go

Create -- Drop

View vCategories

WITH SCHEMABINDING -- this Requires you to use the table's 2-part name!

AS

Select CategoryID, CategoryName as [CatName] From dbo.Categories; --<< 2-part name

go

Select \* From Categories;

Select \* From vCategories;

go

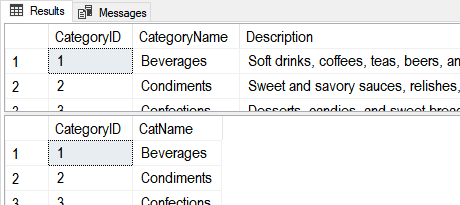


Figure: The results of the previous SQL statements

If a table's **changes will not break its View**, then the change **is allowed**.

Alter Table Categories Add IsDiscontinued int; -- SYNTAX: Don't use the word "Column"

go

Select \* From Categories;

Select \* From vCategories;

go

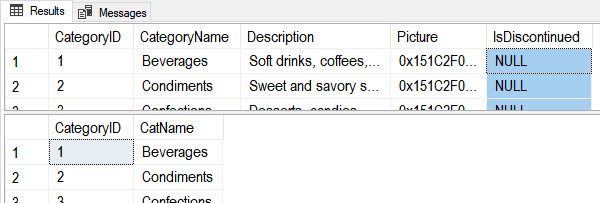


Figure: The results of the previous SQL statements

..., but **changes that would break the table's views** are **not** **allowed**.

Alter Table Categories Drop Column CategoryName; -- SYNTAX: Use the word "Column"

go

Msg 5074, Level 16, State 1, Line 15

The object 'vCategories' is dependent on column 'CategoryName'.

Msg 5074, Level 16, State 1, Line 15

The index 'CategoryName' is dependent on column 'CategoryName'.

Msg 4922, Level 16, State 9, Line 15

ALTER TABLE DROP COLUMN CategoryName failed because one or more objects access this column.

Also, you **cannot drop the whole table**!

Alter Table Products Drop Constraint [FK\_Products\_Categories];

Drop Table Categories;

Msg 3729, Level 16, State 1, Line 26

Cannot DROP TABLE 'Categories' because it is being referenced by object 'vCategories'.

## Functions

In addition to SQL Server's built-in functions, you can create **custom functions**. These are **often called** **User Defined Functions** or just **UDFs**. There are **two basic types of functions**; functions that **return a table of values** and functions that **return a single value**.

Functions and Views similar. Here are **two examples that show how similar**:

-- View

Create **View** vProducts

AS

Select ProductID, ProductName,CurrentPrice = UnitPrice, CategoryID, Discontinued

From Northwind.dbo.Products;

go

**Select \* from vProducts;** -- 77rows

Go

-- Function

Create **Function dbo.**fProducts() # Using the dbo prefix is common in Microsoft SQL Server

Returns Table

AS

Return(

Select ProductID, ProductName, CategoryId, Discontinued

From Northwind.dbo.Products

);

go

**Select \* from dbo.fProducts();** -- 77rows

go

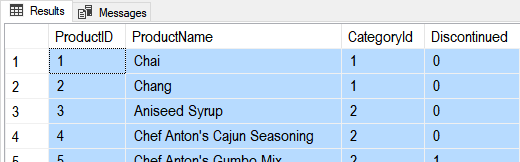


Figure: The results of the previous SQL statements

### Functions with Parameters

**Unlike views, functions can use parameters** to change the results of the query as it is executed like this:

Alter Function dbo.fProducts(@CategoryId int)

Returns Table

AS

Return(

Select ProductID, ProductName

From Northwind.dbo.Products

Where CategoryID = @CategoryId

);

go

Select \* From dbo.fProducts(**1**); -- 12 rows

go

Parameters in a function **may seem a significant advantage**, **but** remember that **you can always apply a Where clause while using a View** like this:

Select \* From vProducts Where CategoryID = 1; -- 12 rows



Figure: The results of the previous SQL statements

**TIP**: Since table functions are more complex and provide similar functionality, **use Views to 'Keep It Simple' when you can!**

### Scalars Functions

Unlike views, you can create **UDFs to return a single (scalar)** **value as an expression**. (Note: In MS SQL, you **must use include the schema name in scaler UDFs**, in this case, dbo).

**Unlike parameters in table functions, parameters in scalar functions are very useful!**

Create Function **dbo**.**MultiplyValues**(@Value1 Float, @Value2 Float)

Returns Float

As

Begin

Return(Select @Value1 \* @Value2);

End

go

-- Calling the function

Select Tempdb.**dbo**.**MultiplyValues**(4, 5);

go

If you want **to apply the function to each row** of a result set, you use the new function like this:

Create table dbo.SalesDetails

(SalesId int, SalesLineItemId int, ProductId int, SalesPrice money, SalesQty int,

Primary key(SalesId, SalesLineItemID)

);

go

Insert Into dbo.SalesDetails (SalesId,SalesLineItemId,ProductId,SalesPrice,SalesQty)

Values (1,1,100,$9.99,10),(1,2,200,$1.00,5)

Go

**Select**

SalesId

,SalesLineItemId

,ProductId

,SalesPrice

,SalesQty

,**dbo**.**MultiplyValues**(SalesPrice,SalesQty) as ExtendedPrice

**From dbo.SalesDetails**

Here are the results:

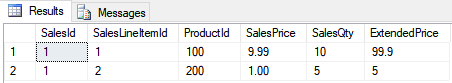


Figure: Results of using the Multiply-Values function in a query

**Important:** Custom Scalar functions are sometimes used for Check constraints because you cannot otherwise reference a column in another table; here is a video of an example (<https://youtu.be/NxNJJvG7FzU>).

## Lab 2: Creating Reporting Functions

In this lab, you create reporting functions using the code you created in lab 1.

**Step 1: Complete Lab 1**

You must complete lab 1 before you continue with this lab.

**Step 2: Copy and Convert**

Using the same Select code you created lab1, create three functions in the Mod06LabsYourNameHere database (using your own name, of course!) Call the functions dbo.f**CustomersByLocation, dbo.fNumberOfCustomerOrdersByLocation, and dbo.fNumberOfCustomerOrdersByLocationAndYears**

Select \* From dbo.fCustomersByLocation() Order By CustomerName;

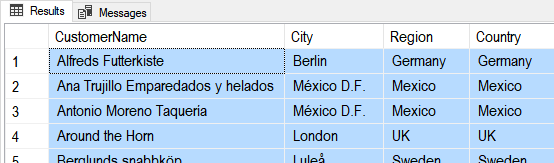


Figure: The query results of question 1

Select \* From dbo.fNumberOfCustomerOrdersByLocation() Order By CustomerName;



Figure: The query results of question 2

Select \* From dbo.fNumberOfCustomerOrdersByLocationAndYears() Order By CustomerName, OrderYear;



Figure: The query results of question 3

**Step 3: Review Your Work**

Now, you will review your work with your instructor.

## Stored Procedures

Like **Views or Functions, Stored Procedures** (Sprocs or Procs) **are** a **Named Set of SQL Statements**. Creating Views, Functions, and Stored Procedures are all **similar, as shown in these three examples:**

-- View

Create **View** vProducts

AS

Select ProductID, ProductName,CurrentPrice = UnitPrice, CategoryID, Discontinued

From Northwind.dbo.Products;

go

**Select \* from vProducts;** -- 77rows

Go

-- Function

Create **Function dbo.**fProducts() # Using the dbo prefix is common in Microsoft SQL Server

Returns Table

AS

Return(

Select ProductID, ProductName, CategoryId, Discontinued

From Northwind.dbo.Products

);

go

**Select \* from dbo.fProducts();** -- 77rows

Go

-- Stored Procedure

Create **Procedure pProducts**()

AS

Select ProductID, ProductName, CategoryId, Discontinued

From Northwind.dbo.Products;

go

**Execute pProducts();** -- 77rows

Go

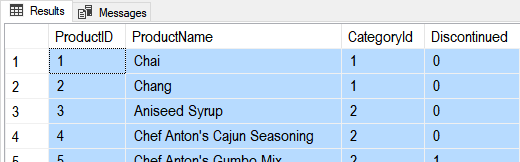


Figure: The results of the previous SQL statements

## Lab 3: Creating Reporting Stored Procedures

In this lab, you create reporting stored procedures using the code you created in lab 1.

**Step 1: Complete Lab 1**

You must complete lab 1 before you continue with this lab.

**Step 2: Copy and Convert**

Using the same Select code you created lab1, create three stored procedures in the Mod06LabsYourNameHere database (using your own name, of course!) Call the stored procedures **pCustomersByLocation, pNumberOfCustomerOrdersByLocation, and pNumberOfCustomerOrdersByLocationAndYears**

Exec pCustomersByLocation;

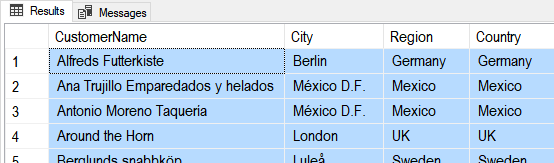


Figure: The query results of question 1

Exec pNumberOfCustomerOrdersByLocation;



Figure: The query results of question 2

Exec pNumberOfCustomerOrdersByLocationAndYears;



Figure: The query results of question 3

**Step 3: Review Your Work**

Now, you will review your work with your instructor.

# GitHub

Even when your code is safely stored in a database as a view, function, or stored procedure, **you should still save the script file you used for your code.**

In fact, it has always been a **good idea to make backups of your code files** and to make them available for others to access. **Traditional** this has been accomplished using a **network share on an organization's server**. While this is still a common practice, more and **more organizations have embraced storing this code on the Internet** **via source control software.**

**One of the most popular source control software is Git**, which can store your files on the Internet at GitHub.com. Here is an excerpt from an article on what GitHub is:

"Version control systems **keep** these **revisions straight**, storing the modifications in a **central repository**. This allows developers to easily **collaborate**, as they can download a new version of the software, make changes, and upload the newest revision. Every developer can see these new changes, download them, and contribute.

…

Git is the preferred version control system of most developers, since it has multiple advantages over the other systems available. It stores file changes more efficiently and ensures file integrity better. If you're interested in knowing the details, the [Git Basics page](http://git-scm.com/book/en/Getting-Started-Git-Basics) has a thorough explanation on how Git works."

(<http://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/>)

**To get started** using GitHub.com, **you will need an account**. This process is like creating most web software accounts and is **tied to an email account**.

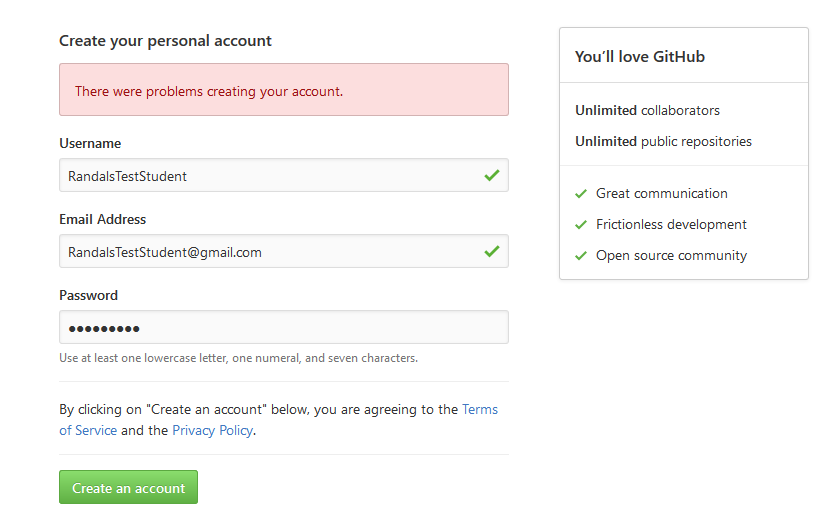


Figure: Creating an account on GitHub

**You can set up multiple user accounts and each account can have multiple repositories.** You can think of a **repository** asa set of **shared folders** where your files are stored and **managed through GitHub's web server.**

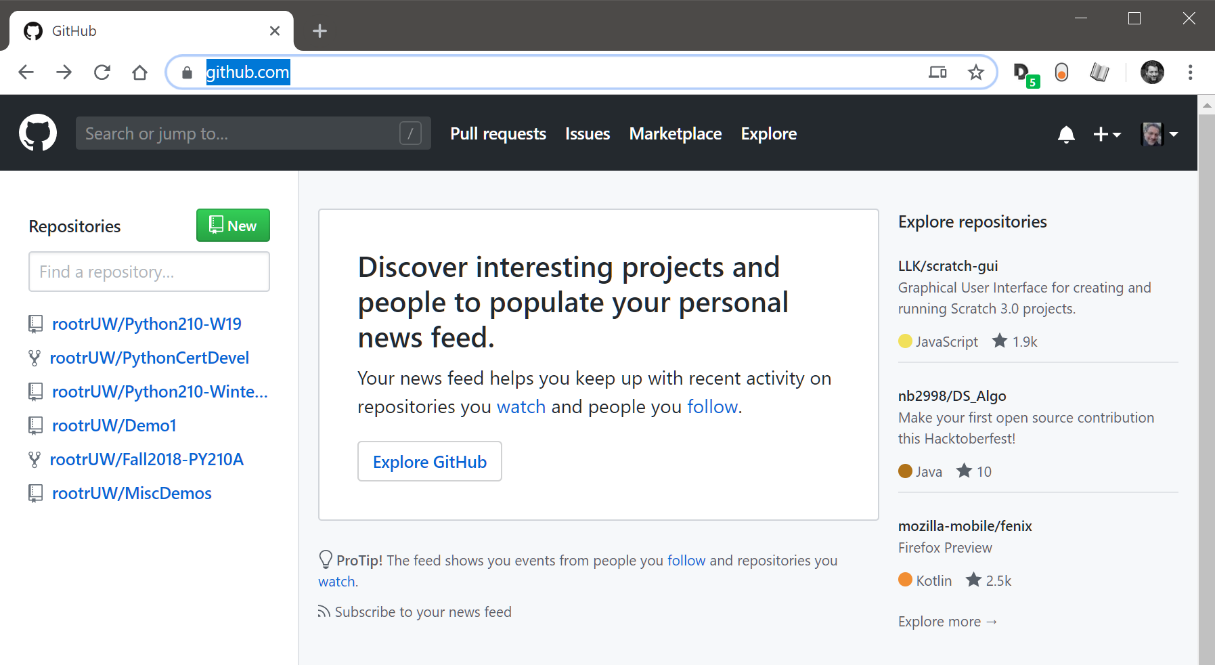


Figure: A list of repositories on the GitHub website

**To create a repository, you** need to **supply its name** and **configure it to be** either private or **public**. If the repository is public, it can easily be seen and shared with other **people.**

**You should include a starting readme file in the repository. If you don't,** you will not see the repository folders materialize until you start loading files into it via the Git command-line too**l.**

**Note: In this course, we use the Web UI to make using GitHub more accessible;** however, using the Git command tool is the preferred method used in the industry.

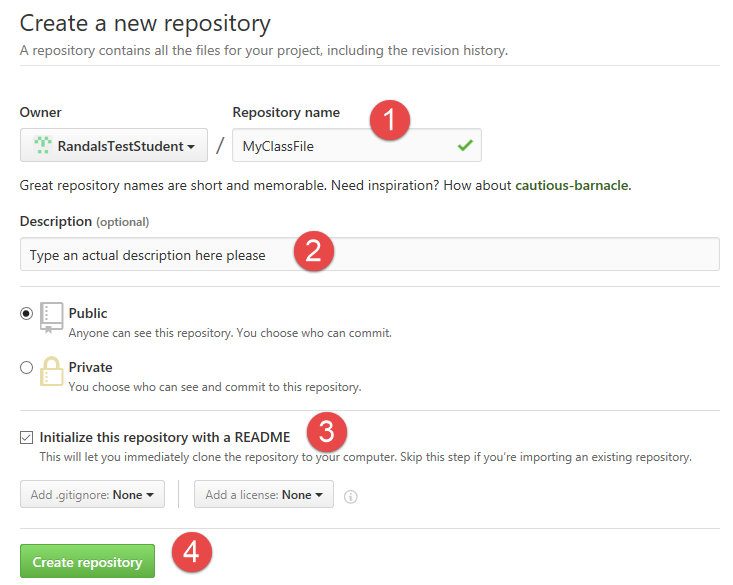


Figure: Creating a repository using the GitHub.com web interface

Once you create the repository, you upload files to it. These files can be viewed, downloaded, and modified by anyone with a link, so be careful what you upload!

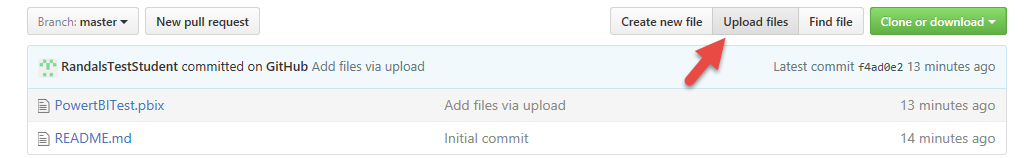


Figure: Using the GitHub.com web interface to upload files

You can store many different types of files on a GitHub repository and use subfolders to organize them. In each folder, including the main repository folder, you should make a ReadMe.md markdown file with notes about the folder's contents.

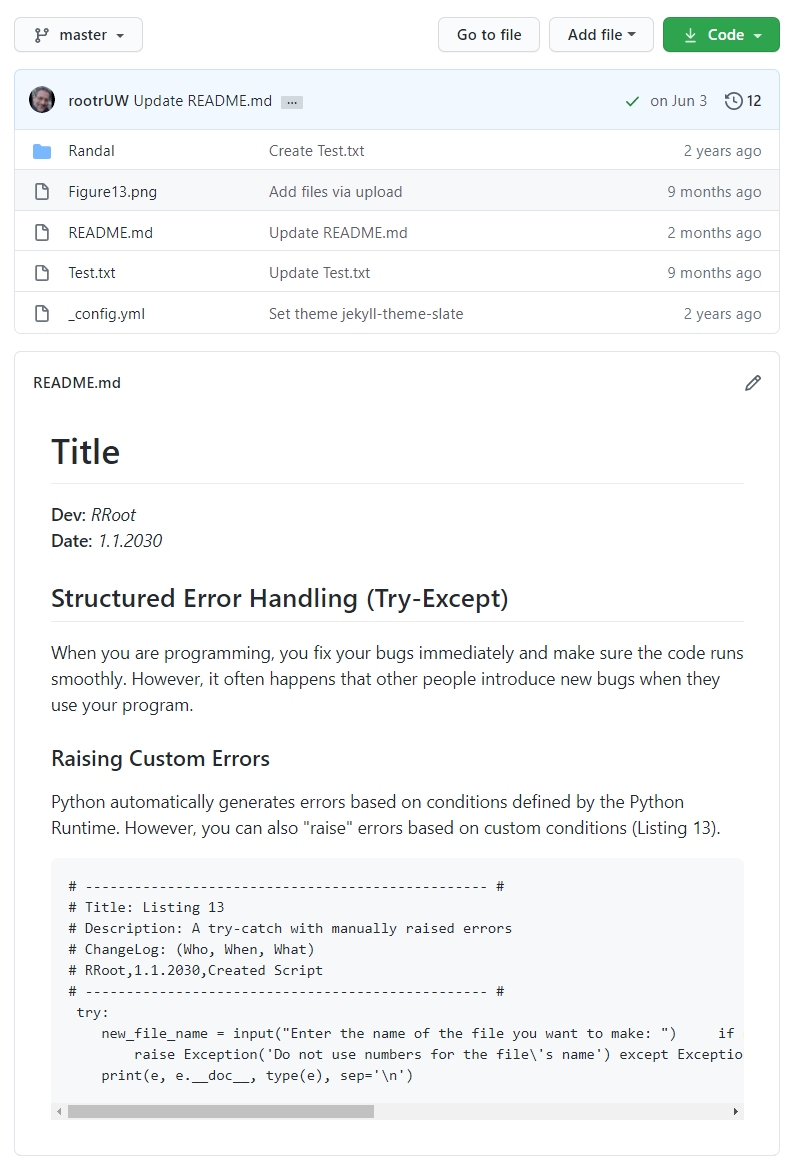


Figure: A ReadMe file in a repository folder

Files stored on GitHub can then be shared with co-workers, friends, or anyone in the World! They can even make their own copy of the files as a "Forked" Repository, which can be used to suggest, make, or track changes over time.