University of Washington

IT Foundations 130

# Module07-Functions

In this module you will learn about how to uses SQL Functions to retrieve information from a database.

[Module07-Functions 1](#_Toc65149848)

[Aggregate Functions (Review) 1](#_Toc65149849)

[Selecting with Common Functions 4](#_Toc65149850)

[Partitioned or Windowed Functions 11](#_Toc65149851)

[Using Functions for Reporting 15](#_Toc65149852)

[User Defined Functions 18](#_Toc65149853)

[Using UDFs for Check constraints 19](#_Toc65149854)

[Creating Advanced GitHub Pages 21](#_Toc65149855)

[Creating a Markdown File 21](#_Toc65149856)

[Formatting the Page 22](#_Toc65149857)

## Aggregate Functions (Review)

"Aggregate functions **perform a calculation on a set of values and return a single value**. Except for COUNT, aggregate functions ignore null values. Aggregate functions are frequently used with the GROUP BY clause of the SELECT statement." (<https://docs.microsoft.com/en-us/sql/t-sql/functions/aggregate-functions-transact-sql>, 2017)

These functions are some of the **most useful** ones. They include Max, Min, Avg, Sum, and Count.

This example shows who placed orders on the **most recent recorded** day.

USE Northwind

SELECT OrderID, CustomerID

FROM Orders

WHERE OrderDate = (SELECT **MAX**(OrderDate) FROM Orders);

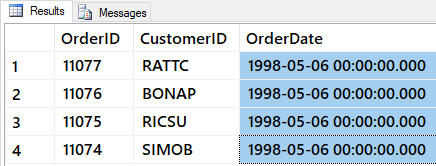


Figure: Result of previous SQL query

Most aggregate functions **exclude Null** values.

SELECT ShippedDate FROM dbo.Orders;

SELECT MAX(ShippedDate) FROM dbo.Orders;

SELECT MIN(ShippedDate) FROM dbo.Orders;

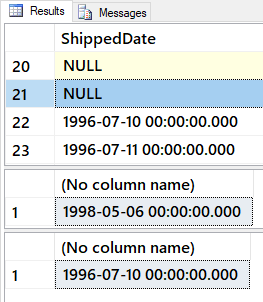


Figure: Result of previous SQL query

Count All (\*) is the exception.

SELECT Count(\*) as [All Orders] FROM dbo.Orders; -- INCLUDES nulls

SELECT Count(ShippedDate) as [ShippedOrders] FROM dbo.Orders; -- does NOT include nulls

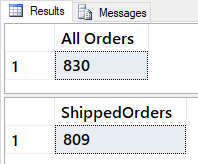


Figure: Result of previous SQL query

Determining the **average of the UnitPrice for all products** in the Products table can be done with the AVG function.

SELECT AVG(Price) as [avg price] FROM Pubs.dbo.Titles;

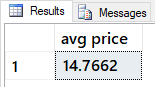


Figure: Result of previous SQL query

You can also use **multiple functions** in one Select statement.

SELECT

[grand total] = SUM(ytd\_sales),

[average sales] = AVG(ytd\_sales),

[number of sales] = COUNT(ytd\_sales),

[number of entries] = COUNT(\*)

FROM Pubs.dbo.Titles;

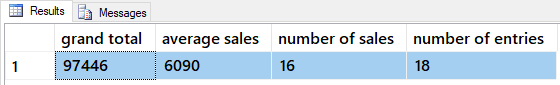


Figure: Result of previous SQL query

You can create your own calculations as well by **combining them,** like this:

SELECT

[Custom Average Sales] = SUM(ytd\_sales) / COUNT(\*),

[Standard Average Sales] = AVG(ytd\_sales)

FROM pubs.dbo.titles;

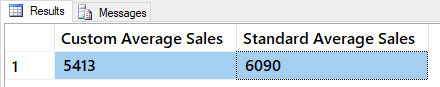


Figure: Result of previous SQL query

### Grouping for Sub-Totals

"The GROUP BY statement is **often used with aggregate functions** (COUNT, MAX, MIN, SUM, AVG) to **group the result-set by one or more columns**." (<https://www.w3schools.com/sql/sql_groupby.asp>, 2017)

To understand group by, let's first look at some results without it:

SELECT \* FROM Pubs.dbo.Titles WHERE title\_id = 'BU1032';

SELECT \* FROM Pubs.dbo.Sales WHERE title\_id = 'BU1032';

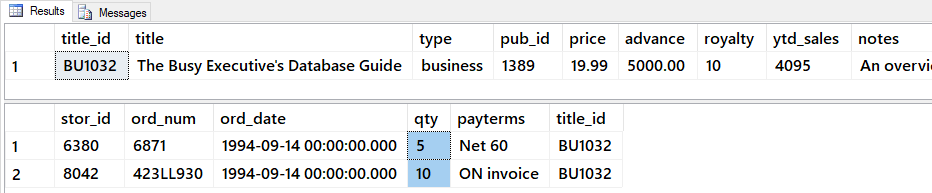


Figure: Result of previous SQL queries

Now, let's add the Group By clause and note how it returns totals.

SELECT Title\_id, SUM(qty) AS 'Quantity'

FROM Pubs.dbo.Sales

WHERE title\_id = 'BU1032'

GROUP BY Title\_id;

### Having

After data has been grouped, you can add a filter on the results using the Having option. This placement in the Select statement is different from Where clause it is applied after the grouped totals are created.

SELECT Stor\_id, Title\_id, SUM(qty) AS 'Quantity'

FROM Pubs.dbo.Sales

GROUP BY Stor\_id,Title\_id

WITH Cube

HAVING sum(Qty) > 100

ORDER BY Stor\_id, Title\_id -- Order by is always last

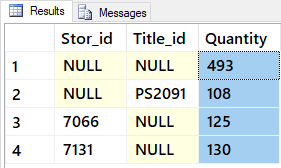


Figure: Result of previous SQL query

## Selecting with Common Functions

Functions are a named collection of SQL programming code. All RDMS include built-in functions, and some even let you create your own. We will see how to make your own custom MS SQL functions in a later module, but for now, let's look at some built-in MS SQL functions as examples.

Most SQL functions **return a single value**.

Select GetDate(), IsNull(null,0);

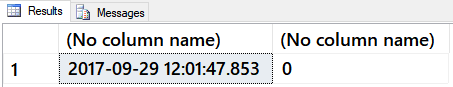


Figure: Result of previous SQL query

But, you can use them in a Select-From statement to **apply the function to many rows**.

Select GetDate(), IsNull(Price, 0), Title

**From** Pubs.dbo.Titles;

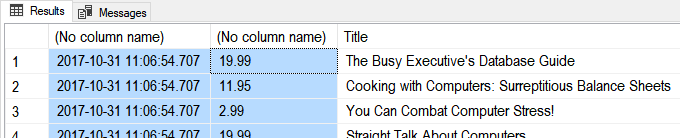


Figure: Result of previous SQL query

You can **combine functions** to create better looking results:

Select

**Cast**(GetDate() as Date) as [TodaysDate]

, IsNull(Price, 0) as [CurrentPrice]

, IsNull(Cast(Price as varchar(50)), 'Not For Sale!') as [CurrentPriceAsTEXT]

, Title

From Pubs.dbo.Titles

Go

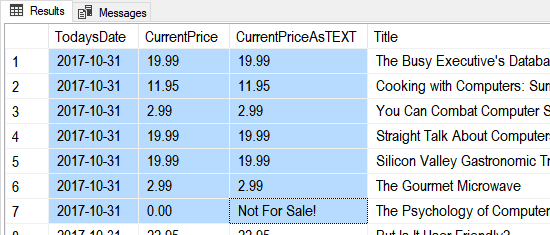


Figure: Result of previous SQL query

### Cast and Convert

**Cast and Convert** are both conversion functions.

Select Cast('1' as int), Cast('1' as decimal(3,2)), Cast(1 as nVarchar(50));

Select Convert(int,'1'), Convert(decimal(3,2),'1'), Convert(nVarchar(50), 1);

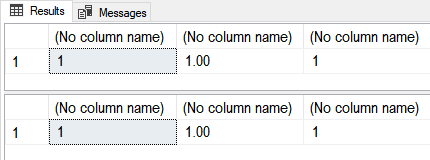


Figure: Result of previous SQL query

**Convert** has more features than Cast.

Select

[Simple Cast] = Cast(GetDate() as Date)

,[Simple Convert] = Convert(Date, GetDate())

,[US with Slash] = Convert(varchar(50), GetDate(), 101)

,[US with Dash] = Convert(varchar(50), GetDate(), 110)

,[ANSI YearMonthDay] = Convert(varchar(50), GetDate(), 112)

;

Go

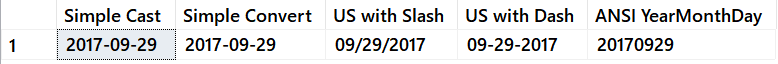


Figure: Result of previous SQL query

Logical Functions allow you to look for a condition the **evaluates to true or false and then return an appropriate value**.

### The String Function:

SELECT STR(3.147);

SELECT STR(3.147, 5, 2);

SELECT STR(3.147, 3, 3);

SELECT STR(123.456, 3, 0);

Go

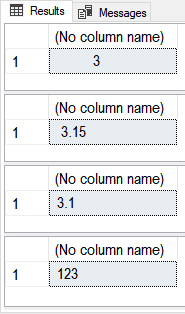


Figure: Result of previous SQL query

**Str()** is **similar** to **CONVERT**(char(15), 123.456), but if you do not allow enough room Convert() will throw an error, while str() will not.

SELECT CONVERT(char(3), 123.456)

Msg 8115, Level 16, State 5, Line 18 Arithmetic overflow error converting numeric to data type varchar.

### CONCAT

-- https://docs.microsoft.com/en-us/sql/t-sql/functions/concat-transact-sql

Select CONCAT(1, 'a', Cast('1/1/2020' as date), 5.6); -- Different data types

go

Select CONCAT(rtrim(ltrim(str((3 \* 100 / 9),5,2) )), '%') union

Select CONCAT(rtrim(ltrim(str((3 \* 100 / 3.21),5,2) )), '%');

go

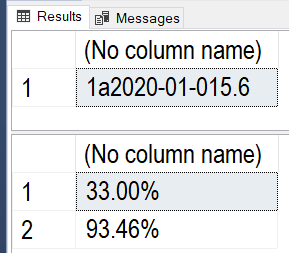


Figure: Result of previous SQL query

### The Format Function

Select Format(GetDate(), 'd', 'en-US' ) AS 'US Result'

,Format(GetDate(), 'd', 'en-gb' ) AS 'Great Britain Result'

,Format(GetDate(), 'd', 'de-de' ) AS 'Germany Result'

,Format(123.456, 'C', 'en-US') AS 'US Format'

,Format(123.456, 'C', 'en-gb') AS 'Great Britain Format'

,Format(123.456, 'C', 'de-de') AS 'Germany Format'

;

go



Figure: Result of previous SQL query

### The Immediate IF function

Select IIF(5 = 5, 'T', 'F');

Select

[ProductName] = IIF(ProductID = 3, ProductName + ' (Not For Sale!)', ProductName)

From Northwind.dbo.Products;

go

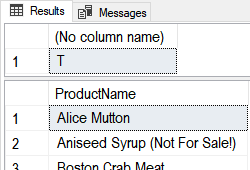


Figure: Result of previous SQL query

### The Case Function (directive)

Select Case (5 + 5)

When 10 Then 'Ten'

When 9 Then 'Nine'

End;

Select

ProductName

,[Category] = Case CategoryID

When 1 Then 'A'

When 2 Then 'B'

When 3 Then 'C'

End

From Northwind.dbo.Products;

go

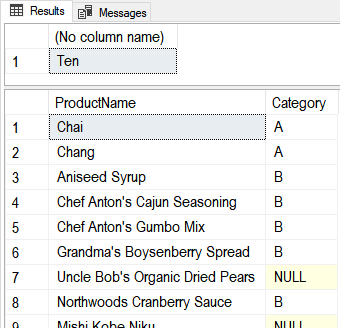


Figure: Result of previous SQL query

**A better use of Case** would be when you need to do a comparison between two columns and generate a third one. For example, say we have the following shipping data and want to see if orders shipped before their due date.

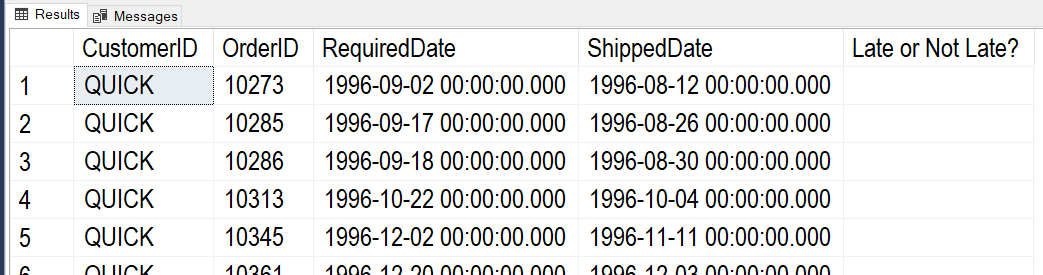


Figure: Result of previous SQL query

**Using a "Select – Case" statement will make this easy!**

Select

CustomerID

,OrderID

,RequiredDate

,ShippedDate

,[OnTime] = Case

When RequiredDate > ShippedDate Then 'Early'

When RequiredDate = ShippedDate Then 'On Time'

When RequiredDate < ShippedDate Then 'Late'

Else 'No Info Yet'

End

From Northwind.dbo.Orders

Where CustomerID = 'QUICK'

Order by [OnTime];

Go

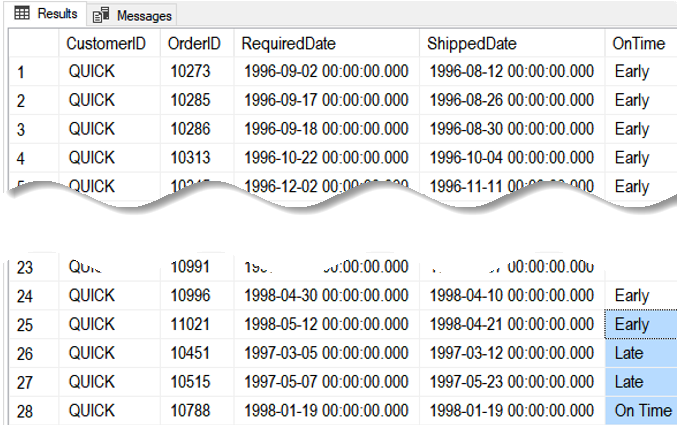


Figure: Result of previous SQL query

### The IsNumeric Funtion:

Select IsNumeric('1'), IsNumeric('a1'), IsNumeric('1.23');

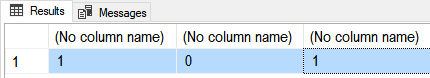


Figure: Result of previous SQL query

### The IsDate Function:

Select

IsDate('1/1/2001')

,IsDate('01-01-2001')

,IsDate('20010101')

,IsDate('Jan,01,2001')

,IsDate('1st of Jan,2001');

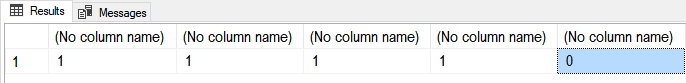


Figure: Result of previous SQL query

### The Left and Right Functions

DECLARE @string varchar(100) = 'This is some data'

SELECT [Left] = Left(@string,4),[Right] = Right(@string,4)

;

Go

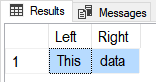


Figure: Result of previous SQL query

### The LTrim and RTrim Functions

DECLARE @string\_to\_trim varchar(100) = ' This is some data '

SELECT

[Without spaces] = '|' + LTrim(RTrim(@string\_to\_trim)) + '|'

,[With spaces:] = '|' + @string\_to\_trim + '|'

;

go

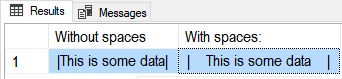


Figure: Result of previous SQL query

### The REPLACE Function

( string\_expression , string\_pattern , string\_replacement )

Select Replace('Bob Smith','Bob','Robert');

Select Replace('Bob Jim-Bob Smith','Bob','Robert');

go

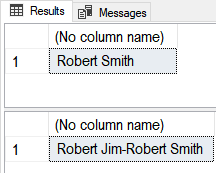


Figure: Result of previous SQL query

### The PATINDEX Function

( '%pattern%' , expression )

Declare @Email varchar(50) = 'BSmith@MyCo.com';

SELECT

[Name Ends] = PatIndex('%@%', @Email)

,[Domain Starts] = PatIndex('%.%', @Email)

;

go

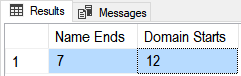


Figure: Result of previous SQL query

### The SUBSTRING Function

( expression ,start , length )

Declare @Email varchar(50) = 'BSmith@MyCo.com';

SELECT

[Name] = SubString(@Email,0,PatIndex('%@%',@Email))

,[Company] = SubString(@Email,PatIndex('%@%',@Email) + 1, patindex('%.%',@Email) - patindex('%@%',@Email) - 1)

,[Domain] = SubString(@Email,PatIndex('%.%',@Email) + 1,20)

go

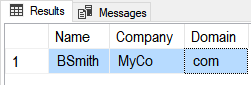


Figure: Result of previous SQL query

### Date/Time Functions

Declare @Date as DateTime = GetDate();

Select

[Isdate()] = Isdate(@Date)

,[Datename()] = DateName(mm,@Date) + ', ' + DateName(Weekday,@Date)

,[Datepart()] = str(DatePart(mm, @Date)) + ', ' + str(DatePart(Weekday,@Date))

,[Dateadd()] = DateAdd(mm, 1, @Date)

,[Datediff()] = DateDiff(yy, '20000101', @Date)

,[Day()Month()Year()] = str(Day(@Date)) + ', ' + str(Month(@Date)) + ', ' + str(Year(@Date));

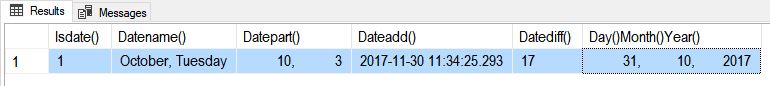


Figure: Result of previous SQL query

### Soundex()

Often you need to clean-up incorrect data. Soundex() is an one function that can help you do a clean-up task that would normally be difficult for humans to do quickly.

<https://docs.microsoft.com/en-us/sql/t-sql/functions/soundex-transact-sql>

Select SOUNDEX ('Patient'), SOUNDEX ('Pateint'), SOUNDEX ('Peteint'), SOUNDEX ('Patint'), SOUNDEX ('Patit');

go

If Exists(Select Name from Sys.tables where Name = 'LookupBySound')

Drop Table LookupBySound

go

Create Table LookupBySound

(ID int Primary Key Identity,[SoundExValue] nvarchar(100), Word nvarchar(100))

go

Insert Into LookupBySound Values ('P353', 'Patient');

go

If Exists(Select Name from Sys.tables where Name = 'DirtyData') Drop Table DirtyData

Create Table DirtyData

(ID int Primary Key Identity, DirtyDataValue nvarchar(100))

go

Insert Into DirtyData

Values

('Patient'), ('Pateint'), ('Peteint'), ('Patint'), ('Patit'), ('test');

go

Select \* from LookupBySound;;

Select \* from DirtyData;

go

Select DirtyData.DirtyDataValue, LookupBySound.Word

From DirtyData

Left Join LookupBySound

ON Soundex(DirtyData.DirtyDataValue) = LookupBySound.SoundExValue

## Partitioned or Windowed Functions

<https://docs.microsoft.com/en-us/sql/t-sql/queries/select-over-clause-transact-sql>

Partitioned or Windowed Functions allow you to group data differently than the standard Group By clause. Here is an example of a simple select statement with a Group By clause.

Select

stor\_id

,[store min was] = min(qty)

,[store max was] = max(qty)

From Pubs.dbo.sales

Group By stor\_id

Order By stor\_id;

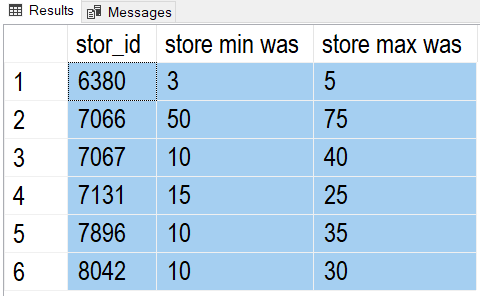


Figure: Result of previous SQL query

Note how just adding Qty doesn't work correctly, because each line becomes part of its own group

Select

stor\_id

,qty -- does not work correctly

,[store min was] = min(qty)

,[store max was] = max(qty)

From Pubs.dbo.sales

Group By stor\_id, qty -- qty is used for grouping

Order By stor\_id;

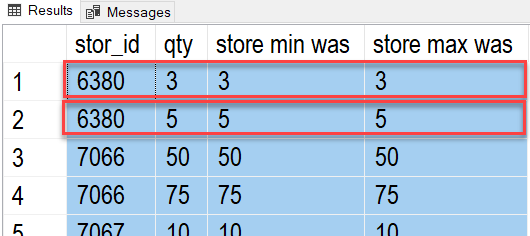


Figure: Result of previous SQL query

Using the Over-Partition function lets you group by only some of the columns.

Select

stor\_id

,qty

,[store min was] = min(qty) over(partition by stor\_id)

,[store max was] = max(qty) over(partition by stor\_id)

From Pubs.dbo.sales

Order By stor\_id;

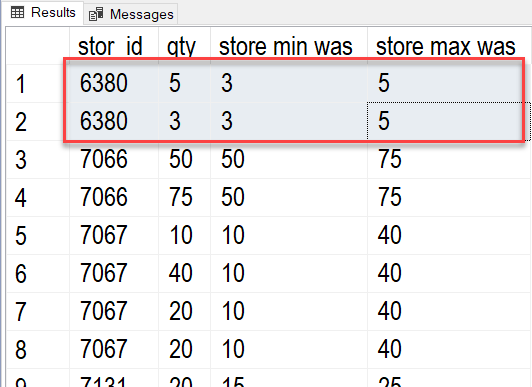


Figure: Result of previous SQL query

If I **add** both stor\_id and **qty** to the partition the results are the **same as Group By.**

Select

stor\_id

,qty

,[store min was] = min(qty) over(partition by stor\_id, qty)

,[store max was] = max(qty) over(partition by stor\_id, qty)

From Pubs.dbo.sales

Order By stor\_id;

### Ranking Functions

"Ranking functions return a ranking value for each row in a partition. Depending on the function that is used, some rows might receive the same value as other rows." -- <https://docs.microsoft.com/en-us/sql/t-sql/functions/ranking-functions-transact-sql>

SELECT o.OrderID, o.CustomerID, od.Quantity,

-- Ignore duplicate Values in the (Quantity) column

ROW\_NUMBER() OVER(ORDER BY od.Quantity) AS rownum,

-- Group duplicate Values in the (Quantity) column and tells how many rows have come before it

RANK() OVER(ORDER BY od.Quantity) AS rank,

-- Groups duplicate Values in the (Quantity) column and tells what the last row NUMBER came before it

DENSE\_RANK() OVER(ORDER BY od.Quantity) AS dense\_rank,

-- Divide rows into groups based in the number of groups you ask for. Ntile(2) would be 2 groups.

NTILE(2) OVER(ORDER BY od.Quantity) AS ntile

FROM Northwind.dbo.Orders as o Join Northwind.dbo.[Order Details] as od

On o.OrderID = od.OrderID

Where CustomerID = 'ALFKI'

ORDER BY od.Quantity;

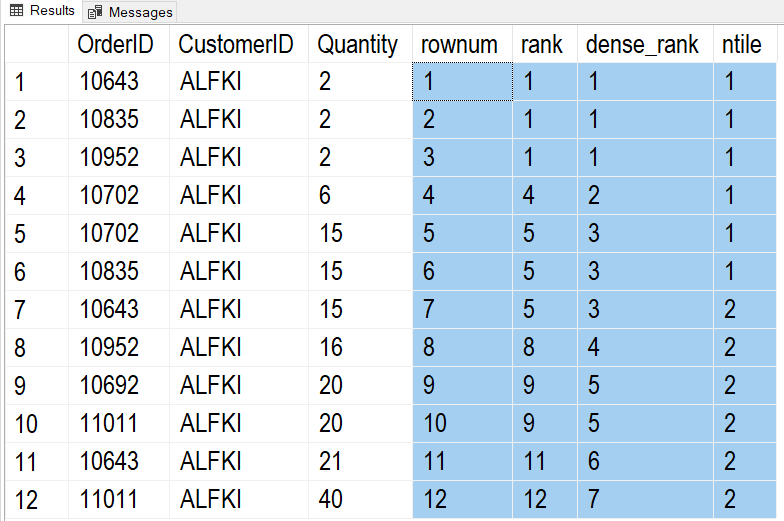


Figure: Result of previous SQL query

#### Extracting a Percentage

A common task is the calculate the percentage of one value compared to another. Here is how it can be done with SQL code.

*Tip: Creating an artificial column just for sorting my data is useful in reporting!*

Select [sortcolumn] = 'a', [result] = (3 / 9) Union

Select 'b', (3 \* 100 / 9) Union

Select 'c', Cast((3 \* 100 / 9) as float) Union

Select 'd', (1. \* 3 \* 100 / 9) Union -- No need to cast to float now!

Select 'e', (1. \* 12 \* 100 / 706) – Ex. 12 products of 706 total products

Order By [sortcolumn]

Go

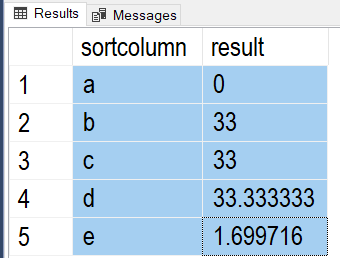


Figure: Result of previous SQL query

Here is an example of how that code could be used.

Select

[stor\_id]

,[title\_id]

,[QtyByTitle] = Sum(Qty) Over(Partition By [stor\_id])

,[QtyByStoreAndTitle] = Sum(Qty) Over(Partition By [title\_id], [stor\_id]) -- The Order Matters here!

,[Title Percent Per Store] = Cast( (1. \* (Sum(Qty) Over(Partition By [title\_id], [stor\_id])) \* 100) /

Sum(Qty) Over(Partition By [stor\_id]) as decimal(10,2))

From pubs.dbo.sales

Order By 1;

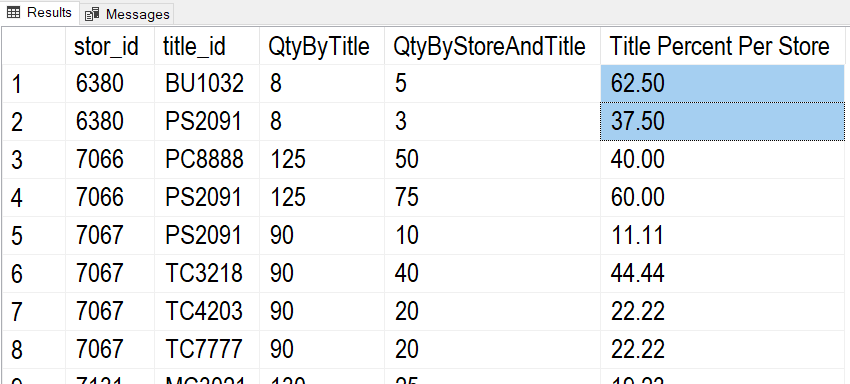


Figure: Result of previous SQL query

### Lag and Lead

These functions are very useful in reporting! It shows the previous or following values based on a given group of values.

Here is an example that shows the following year's values.

Select

[OrderYear] = Year(OrderDate)

,[YearlyTotalQty] = Sum(Quantity)

,[PreviousYearlyTotalQty] = lead(Sum(Quantity)) Over(Order By Year(OrderDate))

From Northwind.dbo.Orders as O

Join Northwind.dbo.[Order Details] as OD

On o.OrderID = o.OrderID

Group By Year(OrderDate);

Go

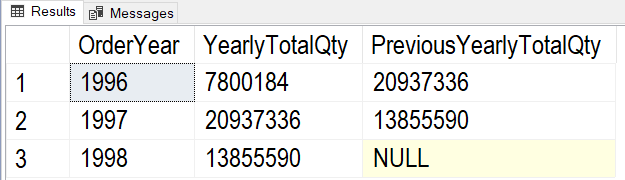


Figure: Result of previous SQL query

Here is another example that shows the shows the previous year's values.

Select

[OrderYear] = Year(OrderDate)

,[YearlyTotalQty] = Sum(Quantity)

,[PreviousYearlyTotalQty] = Lag(Sum(Quantity)) Over(Order By Year(OrderDate))

From Northwind.dbo.Orders as O

Join Northwind.dbo.[Order Details] as OD

On o.OrderID = o.OrderID

Group By Year(OrderDate);

go

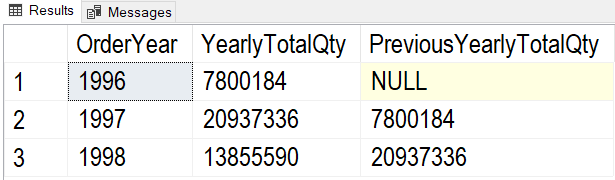


Figure: Result of previous SQL query

# Using Functions for Reporting

To create reporting queries, you start off with a simple Select statement and then build on it by adding more and more detailed code to finally get what you want as a result.

To create reporting queries, you start off with a simple Select statement like this one.

Select Distinct

OrderDate

From Northwind.dbo.Orders;

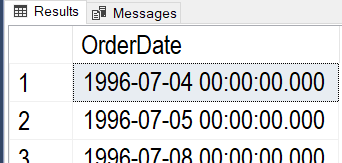


Figure: Result of previous SQL query

Next, add simple functions and test the results!

Select Distinct

[OrderYear] = Year(OrderDate)

From Northwind.dbo.Orders;

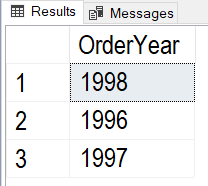


Figure: Result of previous SQL query

Then, add more columns, functions, or tables as needed.

Select Distinct

[OrderYear] = Year(OrderDate)

,[YearlyTotalQty] = Sum(Quantity)

From Northwind.dbo.Orders as O

Join Northwind.dbo.[Order Details] as OD

On o.OrderID = o.OrderID

Group By Year(OrderDate);

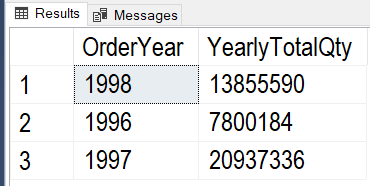


Figure: Result of previous SQL query

Keep adding more complex function as needed (like using the Lag Function).

Select

[OrderYear] = Year(OrderDate)

,[YearlyTotalQty] = Sum(Quantity)

,[PreviousYearlyTotalQty] = Lag(Sum(Quantity)) Over(Order By Year(OrderDate))

From Northwind.dbo.Orders as O

Join Northwind.dbo.[Order Details] as OD

On o.OrderID = o.OrderID

Group By Year(OrderDate);

go

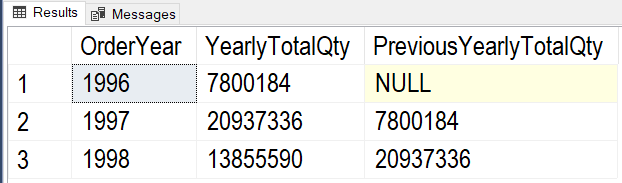


Figure: Result of previous SQL query

Continue adding more features until you get what you are looking for.

Select

ProductName

,[OrderYear] = Year(OrderDate)

,[YearlyTotalQty] = Sum(Quantity)

,[PreviousYearlyTotalQty] =

IIF(Year(OrderDate) = 1996, 0, Lag(Sum(Quantity)) Over (Order By ProductName,Year(OrderDate)))

,[Bad-PreviousYearlyTotalQty] = IsNull(Lag(Sum(Quantity)) Over (Order By ProductName,Year(OrderDate)), 0) -- Don't use Is Null!

From Northwind.dbo.Orders as O

Join Northwind.dbo.[Order Details] as OD

On o.OrderID = o.OrderID

Join Northwind.dbo.Products as P

On OD.ProductID = P.ProductID

Group By ProductName, Year(OrderDate);

go

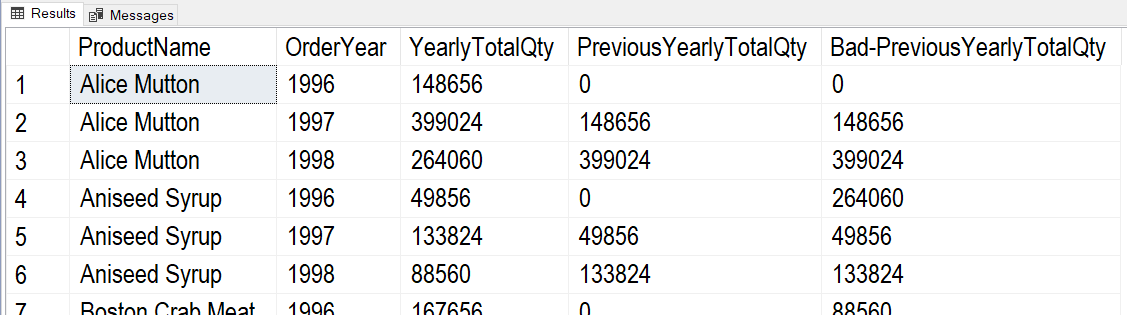


Figure: Result of previous SQL query

Once you feel getting too complex or if you think you might reuse the results, create a reporting View like this one.

Create -- Drop

View vProductOrderQtyByYear

AS

Select

ProductName

,[OrderYear] = Year(OrderDate)

,[YearlyTotalQty] = Sum(Quantity)

,[PreviousYearlyTotalQty] = IIF(Year(OrderDate) = 1996, 0, Lag(Sum(Quantity)) Over (Order By ProductName,Year(OrderDate)))

From Northwind.dbo.Orders as O

Join Northwind.dbo.[Order Details] as OD

On o.OrderID = o.OrderID

Join Northwind.dbo.Products as P

On OD.ProductID = P.ProductID

Group By ProductName, Year(OrderDate);

Go

When using the View, you can always add on more functions -- as needed. For example, our current view makes it easy to create a Key Performance Indicators (KPIs) report

Select

ProductName

,[OrderYear]

,YearlyTotalQty

,PreviousYearlyTotalQty

,[QtyChangeKPI] = Case

When YearlyTotalQty > PreviousYearlyTotalQty Then 1

When YearlyTotalQty = PreviousYearlyTotalQty Then 0

When YearlyTotalQty < PreviousYearlyTotalQty Then -1

End

From vProductOrderQtyByYear

Go

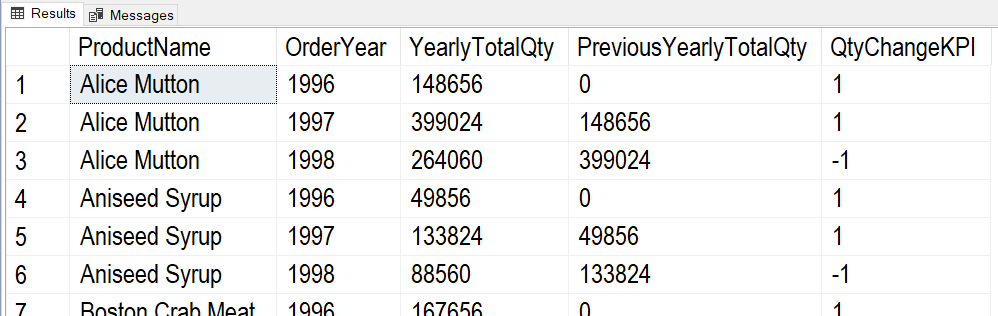


Figure: Result of previous SQL query

# User Defined 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**.

### Scalars Functions

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:

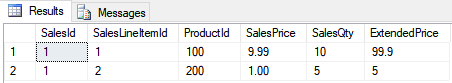


Figure: Results of using the Multiply-Values function in a query

## Using UDFs for Check constraints

Custom Scalar functions are sometimes used for Check constraints because you cannot otherwise reference a column in another table. Here is an example

Set NoCount ON; -- Turns of the (1 row affected) messages

Go

Use TempDB;

Go

If Exists (Select Name From Sys.Tables where Name = 'SignupForMeetings')

DROP TABLE SignupForMeetings;

Go

If Exists (Select Name From Sys.Tables where Name = 'Meetings')

DROP TABLE Meetings;

Go

-- Make dependent tables.

CREATE TABLE Meetings (MeetingID int Primary Key, MeetingDateAndTime datetime);

Go

INSERT INTO Meetings (MeetingID, MeetingDateAndTime)

VALUES (1,'1/1/2020 10:00:00');

Go

CREATE TABLE SignupForMeetings

( SignupID int PRIMARY KEY

, SignupDateTime datetime

, MeetingID int Foreign Key References Meetings(MeetingID)

);

Go

INSERT INTO SignupForMeetings (SignupID, SignupDateTime, MeetingID)

VALUES (1, '1/1/2020 11:00:00', 1) -- Opps! This is One hour AFTER the meeting

Go

SELECT MeetingID, MeetingDateAndTime From Meetings;

SELECT SignupID, SignupDateTime, MeetingID From SignupForMeetings;

Go

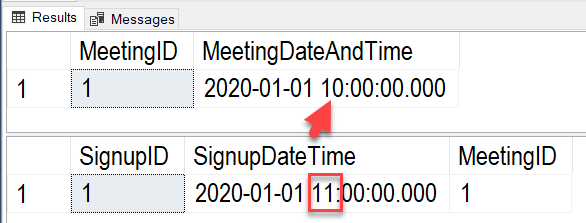


Figure: Incorrect meeting signup data is allowed

-- Remove that row and work on a way to Fix this issue!

DELETE FROM SignupForMeetings WHERE SignupID = 1;

Go

-- Make a function that will get the meeting date and time based on a meeting ID

CREATE or ALTER FUNCTION dbo.fGetMeetingDateTime (@MeetingId int)

RETURNS DATETIME

AS

BEGIN

RETURN (SELECT MeetingDateAndTime

FROM Meetings

WHERE Meetings.MeetingID = @MeetingID);

END

Go

-- Test the function

SELECT dbo.fGetMeetingDateTime(1);

SELECT IIF(CAST('1/1/2020 07:00:00' as datetime) < dbo.fGetMeetingDateTime(1), 'TRUE', 'FALSE'),'Before Start';

SELECT IIF(CAST('1/1/2020 11:00:00' as datetime) < dbo.fGetMeetingDateTime(1), 'TRUE', 'FALSE'), 'After Start';

Go

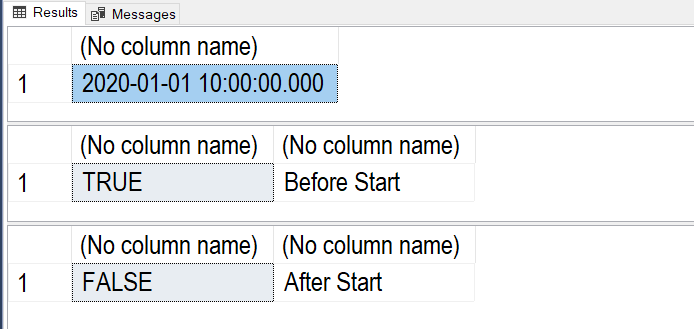


Figure: Results of the function test

-- Now, create a constraint that checks that a signup is before the meeting time

ALTER TABLE SignupForMeetings

ADD CONSTRAINT ckSignupVsMeetingDateTime

CHECK(SignupDateTime < dbo.fGetMeetingDateTime(MeetingID));

Go

-- Test the check constraint

INSERT INTO SignupForMeetings

(SignupID, SignupDateTime, MeetingID)

VALUES

(1, '1/1/2020 9:00:00', 1) -- One hour BEFORE the meeting

Go

INSERT INTO SignupForMeetings

(SignupID, SignupDateTime, MeetingID)

VALUES

(1, '1/1/2020 11:00:00', 1) -- One hour AFTER the meeting

Go

*Msg 547, Level 16, State 0, Line 69*

*The INSERT statement conflicted with the CHECK constraint "ckSignupVsMeetingDateTime". The conflict occurred in database "tempdb", table "dbo.SignupForMeetings".*

*The statement has been terminated.*

SELECT MeetingID, MeetingDateAndTime From Meetings;

SELECT SignupID, SignupDateTime, MeetingID From SignupForMeetings;

Go

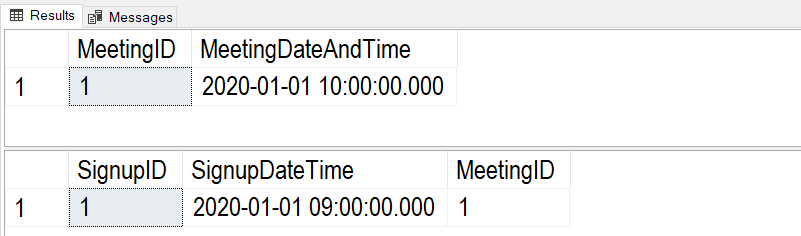


Figure: Results of the previous statements.

# Creating Advanced GitHub Pages

**In this module's assignment, you create a GitHub webpage**. This web page should be **like the Word documents you created in previous assignments**. To create your page, you need to know some basic commands in a language called "markdown."

*"GitHub combines a syntax for formatting text called GitHub Flavored Markdown with a few unique writing features.*

*Markdown is an easy-to-read, easy-to-write syntax for formatting plain text.*

*We've added some custom functionality to create GitHub Flavored Markdown, used to format prose and code across our site." (*[*https://help.github.com/en/github/writing-on-github/about-writing-and-formatting-on-github*](https://help.github.com/en/github/writing-on-github/about-writing-and-formatting-on-github)*, 2019)*

## Creating a Markdown File

To demonstrate an example, I create a **new GitHub repository called "ITFnd100-Mod07"** as shown in this figure:

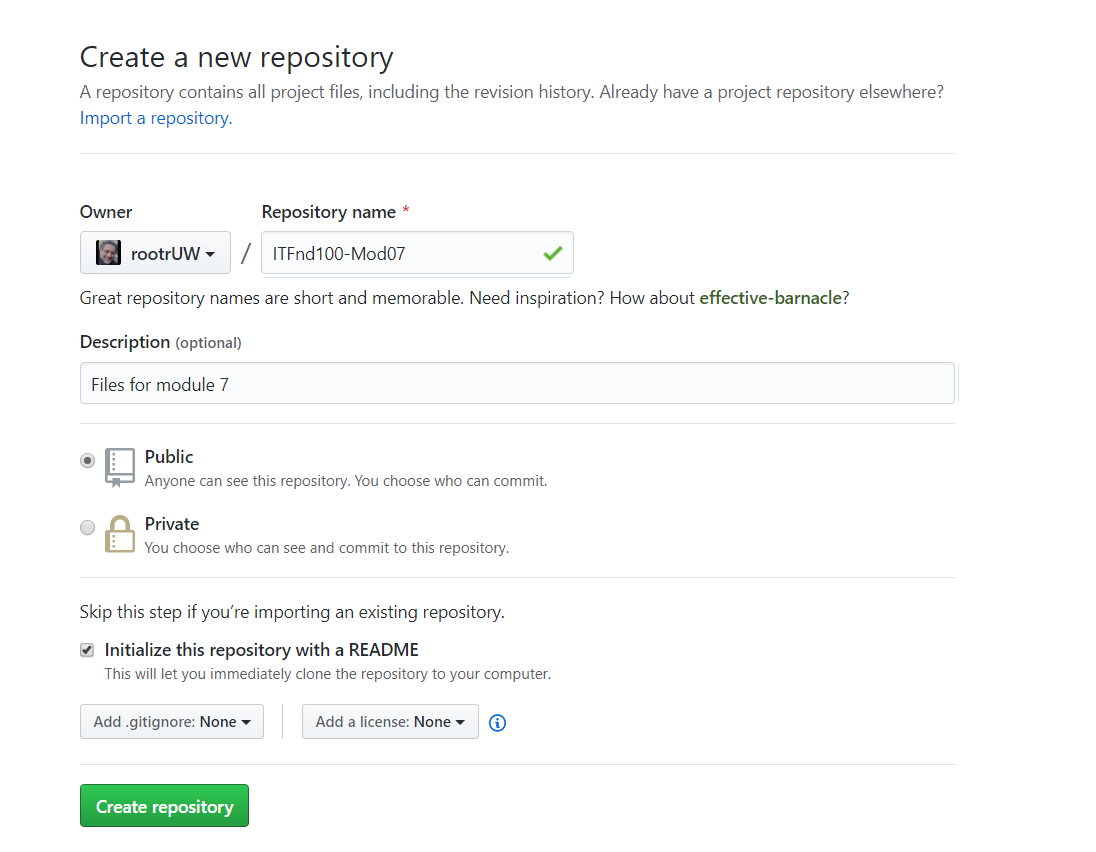


Figure. Creating a new GitHub repository

Next, I **create a new "docs" folder with a file called "index.md"** inside of it. When I do, I need to **type or paste in some text** for the new file before it is created in the folder. In Figure 17, I have typed in some **simple markdown commands to format my document**. Once I have at least some text in the file I can create the folder and file on my repository.

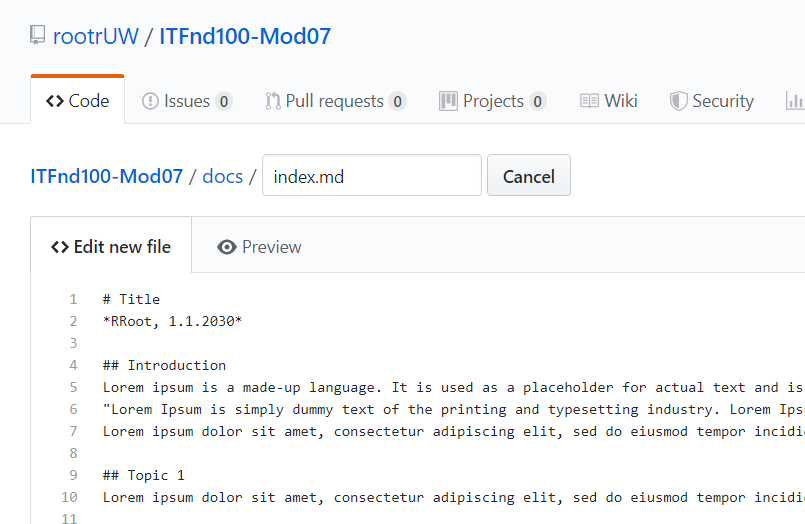


Figure. Creating a new Index.md file in the "docs" folder

## Formatting the Page

I **use the "Preview" tab to see what my new markdown file looks** like each time I modify the text (Figure 18). The look changes based on what markdown commands I use. While there are lots of commands available, let's **focus on the ones you need for the assignment.**

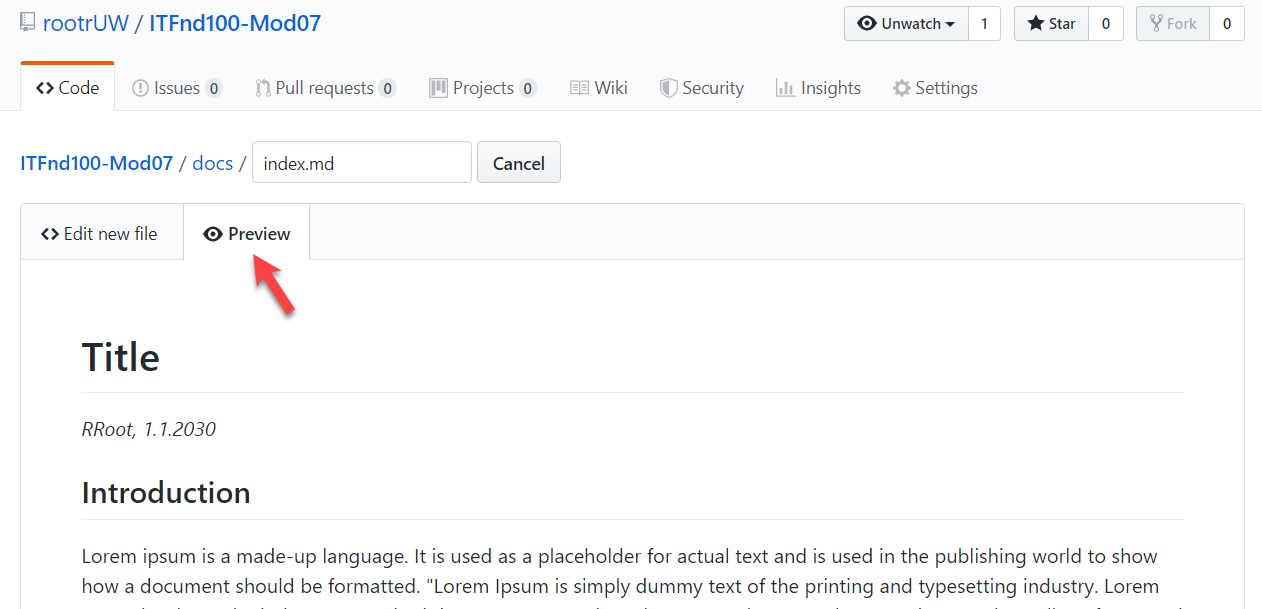


Figure. Previewing a GitHub page

### Creating Text Headers

The hash, **"#" symbol, indicates a header**. You use one or more hash symbols to define the level of the header. Oddly, the more hash marks you use, the smaller the header size.

# Title

## Introduction

## Topic 1

### Subtopic

## Summary

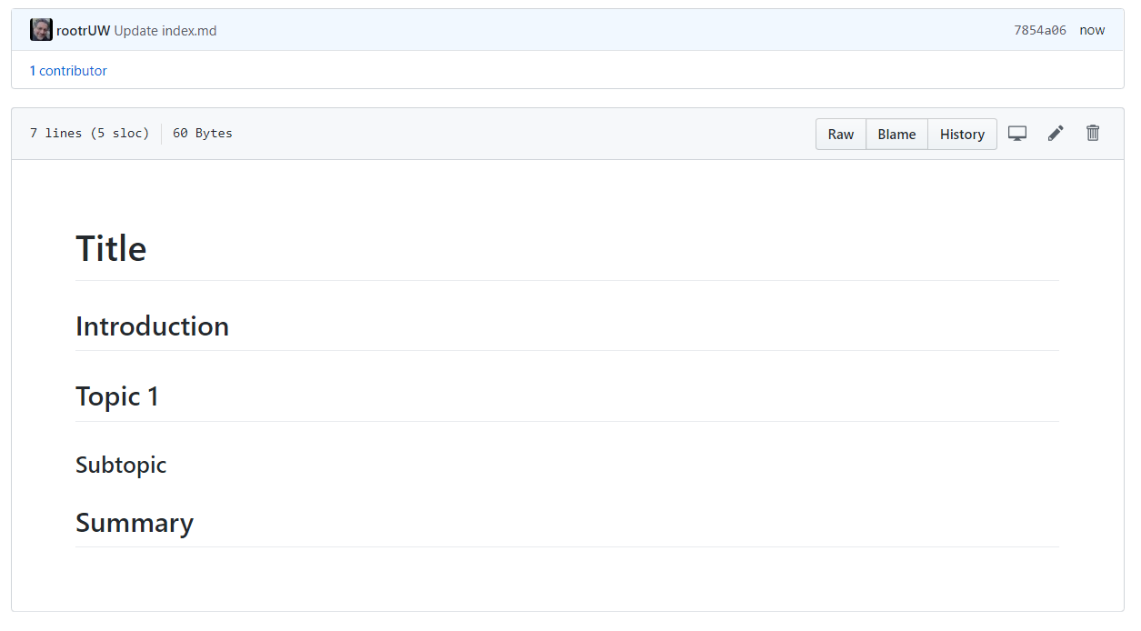


Figure. The results use header markdown commands

### Formatting Text

You can make text **bold using two Astrid "\*\*"** symbols or **italic using a single Astrid**.

\*\*Dev:\*\* \*RRoot\*

\*\*Date:\*\* \*1.1.2030\*

Listing 17

***Note:*** *There are* ***two invisible space characters after the text "\*RRoot\* that force a newline into the page****. Newlines can be tricky in this language, and you may need to experiment to style the page to your liking.*

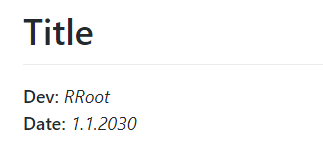


Figure. Creating told text

### Adding Code Samples

To add some **sample code** to the page, you **use the backtick (`) symbol,** as shown in Listing 18.

# Title

\*\*Dev:\*\* \*RRoot\*

\*\*Date:\*\* \*1.1.2030\*

## Structured Error Handling (Try-Except)

When you are programming, you fix your bugs immediately and make sure the code runs smoothly. However, it often happens that other people introduce new bugs when they use your program.

### Raising Custom Errors

Python automatically generates errors based on conditions defined by the Python Runtime. However, you can also "raise" errors based on custom conditions (Listing 13).

```

# ------------------------------------------------- #

# Title: Listing 13

# Description: A try-catch with manually raised errors

# ChangeLog: (Who, When, What)

# RRoot,1.1.2030,Created Script

# ------------------------------------------------- #

try:

new\_file\_name = input("Enter the name of the file you want to make: ")

if new\_file\_name.isnumeric():

raise Exception('Do not use numbers for the file\'s name')

except Exception as e:

print("There was a non-specific error!")

print("Built-In Python error info: ")

print(e, e.\_\_doc\_\_, type(e), sep='\n')

```

#### Listing 13

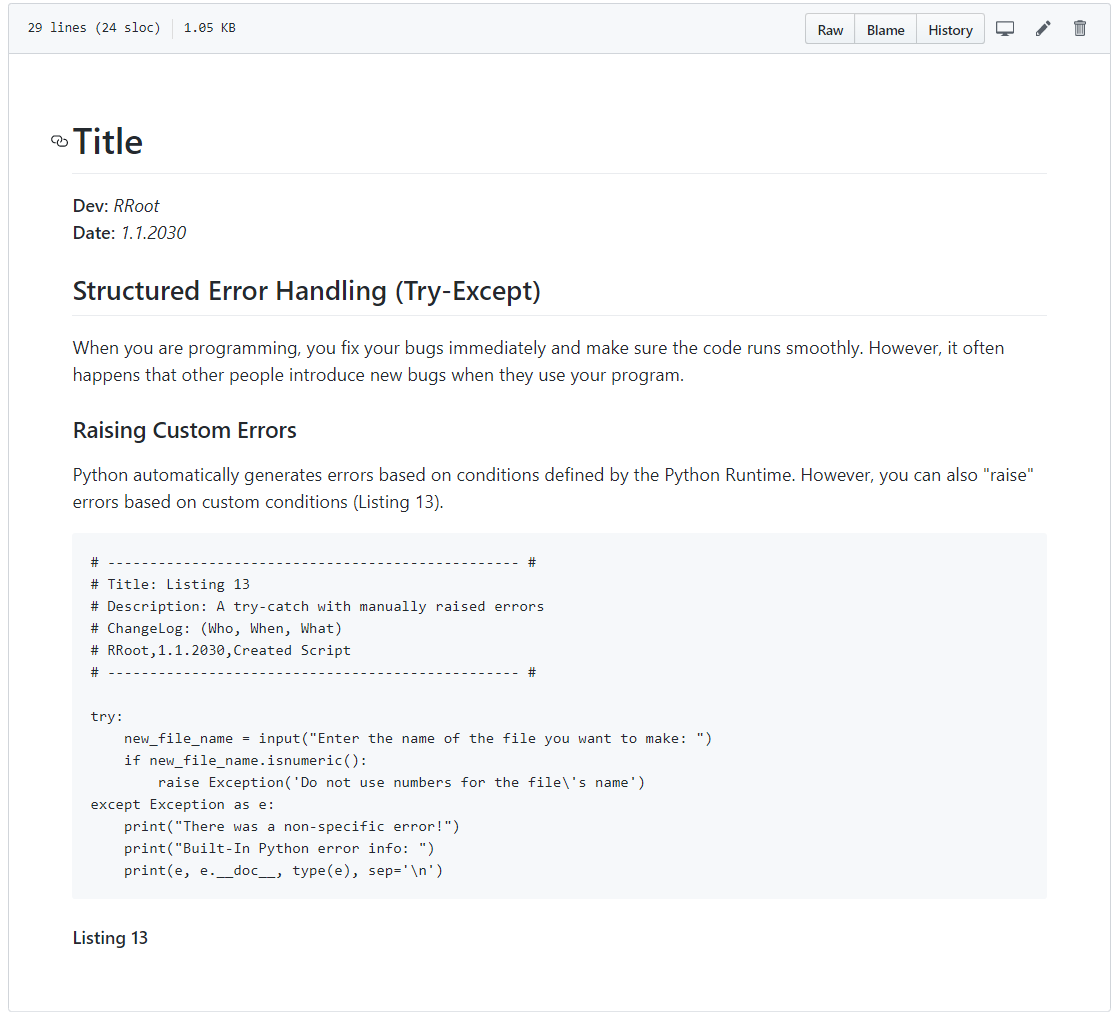


Figure. The results of adding code to the page

***Tip:*** *The keyboard key for the* ***backtick symbol (`)*** *is shared with the tilde symbol (~)* ***above the Tab key on both a Windows and Mac keyboard****.*

### Adding Images

To **add an image** to a page, perform the following steps.

1. **Save image to your computer's hard drive**. In MS Word, you do this by right-clicking the image and using the "Save as Picture…" option of the context menu.

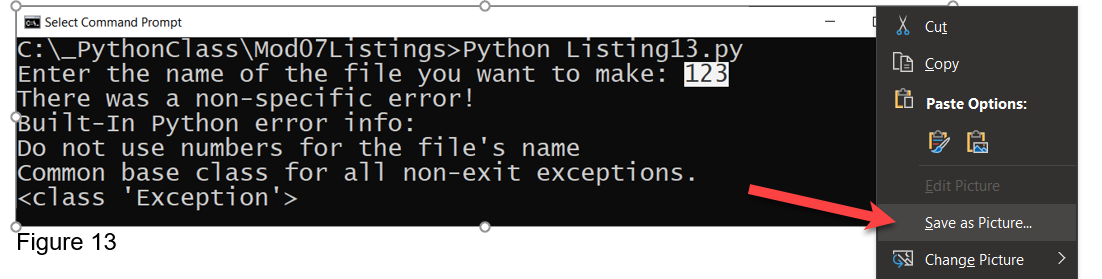


Figure. Saving an image from Word to a drive

2. **Upload the image to your GitHub** repository's "docs" folder (or a subfolder if you wish to be more organized.

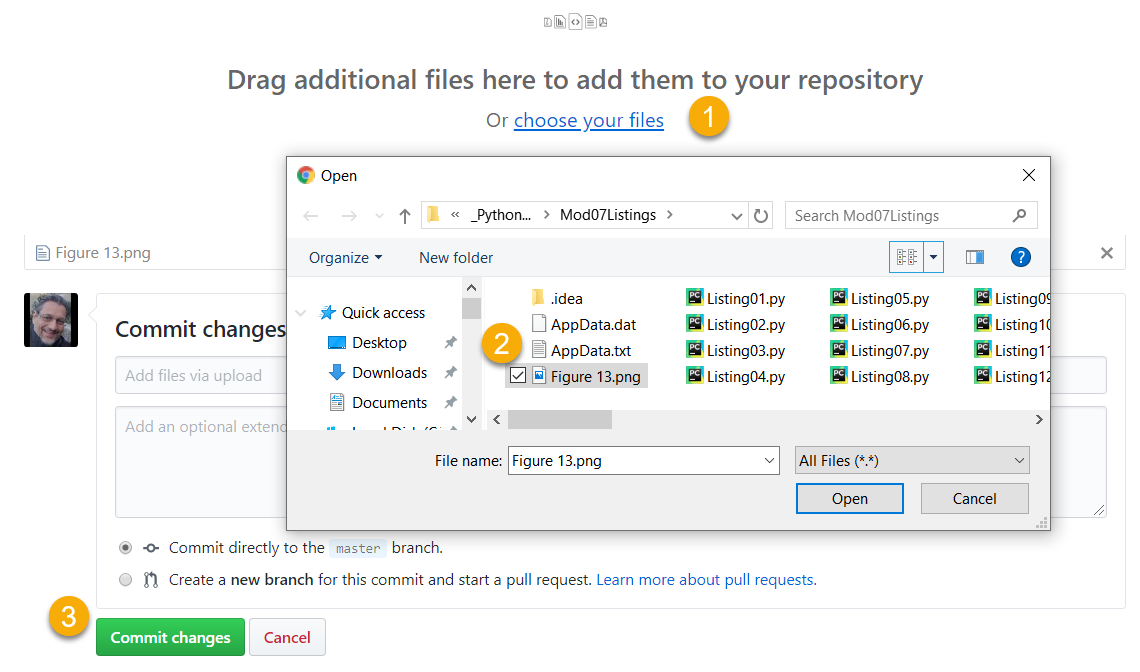


Figure. Uploading the saved image file to GitHub

3. **Copy the web address for the file**, by locating your file on the GitHub page, then right-clicking it to access the context menu. From there, use the "Copy link address" option, or an equivalent one if you are using a browser other than Chrome.

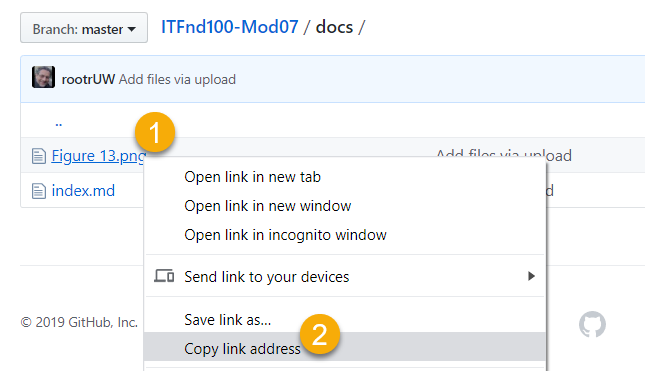


Figure. Copying the file's URL on GitHub with a Chrome browser

4. **Link your image to your page using an image link** **code.** The syntax for the image link is:

![alt text](web address "tooltip text")

**Here is an example:**

print("Built-In Python error info: ")

print(e, e.\_\_doc\_\_, type(e), sep='\n')

```

#### Listing 13

![Results of Listing 13](Figure13.png "Results of Listing 13")#### Figure 13. The results of Listing 13

**Note:** If you cannot get the images to work, try using HTML tags instead. <https://www.w3schools.com/tags/tag_img.asp>

The **first part of command** indicates the **alternate text** use by screen readers. The **second** part of the command indicates the **URL to the file**. The **third** part of the command indicates the text you want a **tool tip** to display. Figure 24 shows the results of the command.

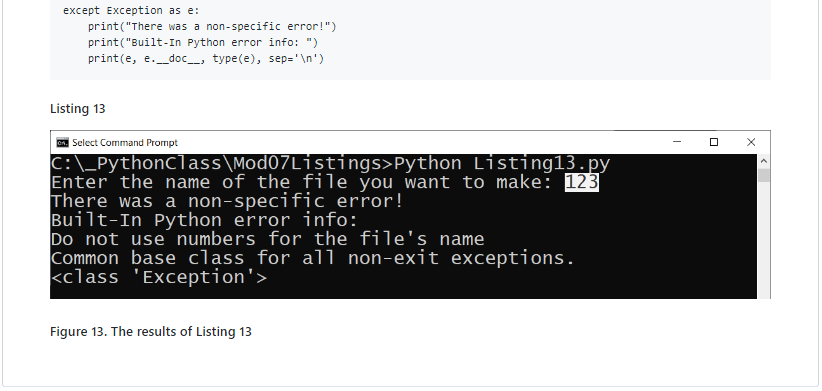


Figure. The results of using a Markdown image link on a GitHub page

***Important:*** *It is best to use a relative path for files that are in your folder, and only use the "hard coded" physical path for files outside of your website.*

### Learning More

There is much information on the Internet about the Markdown language, but you **should find all you need for this course on this one webpage**:

<https://help.github.com/en/github/writing-on-github/basic-writing-and-formatting-syntax>

***Important****: Learning to use Markdown and Jekyll could well be the topic of a complete course, but in this course,* ***you do NOT need to know much about Markdown*** *programming.* ***Please use only the basics shown in this module*** *instead of more advanced features* ***and do not worry about getting the format perfect!***