University of Washington

iSchool Info 340

# Module 03 - SQL Queries

In this module, we will **start programming with the Structured Query Language (SQL).**

"More than five years ago, the global business consultancy McKinsey predicted **a shortage**

**of as many as 1.5 million managers and analysts with analytics know-how by 2018**,

writing that “data have swept into every industry and business function and are now an

important factor of production, alongside labor and capital.”

There are **currently more than 60,000 jobs in “data analytics” listed on Indeed.com**. And

a report from LinkedIn recently listed data visualization, data presentation, statistical

analysis, and data mining as top in-demand skills by employers in **2016**." (The State of Data

Education in 2016, https://www.tableau.com/data-education-2016)

## Outline

Here is a general outline of what we will be doing this module:

|  |
| --- |
| **Module03: SQL Queries** |
| Session01 Lectures and Labs < 110 mins |
| Select Basics - 10 |
| Lab 1: Using Basic Selects - 10 |
| Select Options - 20 |
| Lab 2: Using Select Options - 20 |
| Break - 10 |
| Additional Select Options - 10 |
| Lab 3: Using Additional Select Options - 10 |
| Aggregate Functions - 20 |
| Session02 - Lab |
| Lab 4: Selecting Aggregate Results - 50 |
| Session03 Lectures and Labs < 110 mins |
| Selecting with Common Functions - 30 |
| Lab 5: Using Common Functions - 20 |
| Break - 10 |
| Basic Transactions - 20 |
| Lab 6: Processing Basic Transactions - 30 |

**Note**: Times are only estimates and may change without notice!

# Session01 < 110 mins

In this session, we explore how to program in the **Structured Query Language (SQL).** We will first look at the **most common commands** used and move toward less common ones. By the end of this module, you will know **enough SQL commands to extract basic data from a database**.

## Select - 10

The SQL Select statement is **the one you will use the most in this course and in real life**. While it can be the most complex of SQL statements, its commonly used basic form is not difficult at all!

The SELECT statement’s main clauses are **SELECT, FROM, and WHERE**

**SELECT** <col1,col2,col3,...>

**FROM** <DBName>.<Schema>.<TableName>

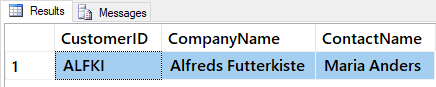
**WHERE** (<Filter Expression>);

Here is an example:

**SELECT** CustomerID, CompanyName, ContactName

**FROM** Northwind.dbo.Customers

**WHERE** (CustomerID = 'alfki');



In most cases, **SQL ignores** **SPACES, TAB,** and **CARRIAGE RETURNS**. It is also **not case-sensitive** (on most systems!) So, you will often see the Select Clause written like this:

SELECT -- Stacking the column listing; easier to read, but takes more space.

CustomerID,

CompanyName,

ContactName

FROM Northwind.dbo.Customers

WHERE

(CustomerID = 'alfki'

OR

CustomerID = 'anatr'

);

It even works like this, but it just does not look professional!

SELECT

CustomerID,

CompanyName,

ContactName

FROM Northwind.dbo.Customers

WHERE (

CustomerID = 'alfki'

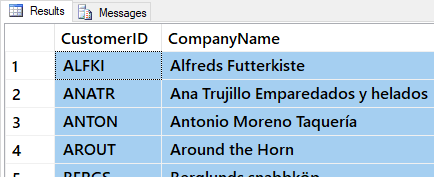
OR

CustomerID = 'anatr' )

### The SELECT Clause

Most of the time you use the **Select clause** to **list the columns** you want in your result set from a table's data.

SELECT CustomerID, CompanyName FROM Customers;



*Figure: Result of previous SQL query*

You can use select to **evaluate an expression, without a From clause.**

SELECT 5 + 4, 5 \* 4, GetDate();

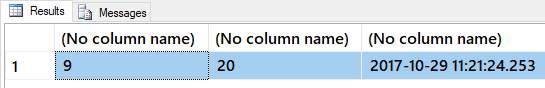


Figure: Result of previous SQL query

You can even include **expressions as columns that do not exist** in the database table.

SELECT CustomerID, CompanyName, GetDate() FROM Customers;

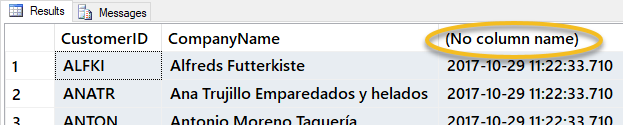


Figure: Result of previous SQL query

#### Column Aliases

Alias are used to **rename a column** in the result set.

SELECT [Sum] = 5 + 4, [Product] = 5 \* 4;

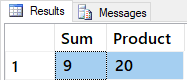


Figure: Result of previous SQL query

Column Aliases have many **STYLE variations**. This can be confusing at first, but you will soon get used to them (as long as you practice using the variations).

-- Alias in Front style

USE Northwind;

SELECT 'AVG Sales Price' **=** AVG(UnitPrice) FROM [Order Details]

SELECT "AVG Sales Price" **=** AVG(UnitPrice) FROM [Order Details]

SELECT [AVG Sales Price] **=** AVG(UnitPrice) FROM [Order Details]

-- Alias in Back style

SELECT AVG(UnitPrice) **AS** 'AVG Sales Price' FROM [Order Details]

SELECT AVG(UnitPrice) **AS** "AVG Sales Price" FROM "Order Details"

SELECT AVG(UnitPrice) **AS** [AVG Sales Price] FROM [Order Details]

-- Just being Lazy style

SELECT AVG(UnitPrice) 'AVG Sales Price' FROM [Order Details]

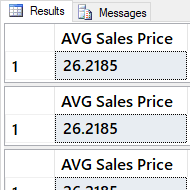


Figure: Result of previous SQL query

SET QUOTED\_IDENTIFIER off -- This setting changes the way Quotes are used in your scripts

SELECT AVG(UnitPrice) AS **"**AVG Sales Price"

FROM "Order Details" -- You will get an error!

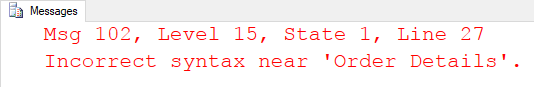


Figure: Result of SQL error message

-- So, using Square Brackets [] is recommended.

SELECT AVG(UnitPrice) AS [AVG Sales Price]

FROM [Order Details]

SET QUOTED\_IDENTIFIER on

#### Question: Why would people change the SET QUOTED\_IDENTIFIER setting?

### The From Clause

The From clause lets you **list one or more tables** you want to get data from.

-- Using one table

SELECT ProductName **FROM** Products;

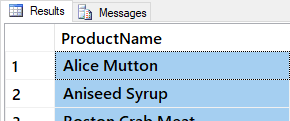


Figure: Result of previous SQL query

-- The From clause recognizes schemas(namespaces) and Database names

SELECT CompanyName FROM dbo.Customers; -- using the "Database Owner" schema

SELECT CompanyName

FROM Northwind.dbo.Customers; -- using the Database name as well as the schema

Use Northwind

SELECT CompanyName

FROM Northwind..Customers; -- using the Database name and the "default" schema

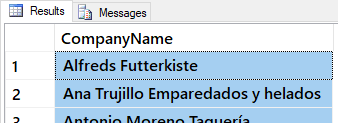


Figure: Result of previous SQL query

#### Joining Two Tables

-- Using 2 tables (The Improved ANSI way)

SELECT ProductName, CategoryName

FROM Products **JOIN** Categories

**ON** Products.CategoryId **=** Categories.CategoryId;

--(Table Aliases)--

-- Alias can be used to rename a table names in the query.

SELECT ProductName, CategoryName

FROM Products **as P** JOIN Categories **as C**

ON P.CategoryId = C.CategoryId;

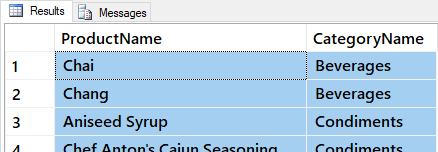


Figure: Result of previous SQL query

#### Question: Why would you want to join data from different tables?

### The Where Clause

The Where clause is used as a **boolean filter**. If a statement is true, then a row of data is returned. If not, then the row is not returned. By default, a Select statement **evaluates** the Where expression for **every row** in a table.

USE Northwind

SELECT ProductName, UnitsInStock

FROM Products

WHERE ProductName = 'Chai';

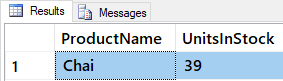


Figure: Result of previous SQL query

## Lab 1: Using Basic Selects - 10

In this lab, you create a select statement using Northwind database.

You will work on your own for the first 5 minutes, then we will review the answers together in the last 5 minutes.

**Note**: This lab should be done individually.

### 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;

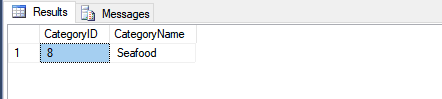
### Step 2: Create a Query

Answer the following questions by writing and executing SQL code.

**Notes**:

* The following image is what your results should look like.

**Question 1-1:** Select the Category Id and Category Name of the Category 'Seafood'.



##### Figure 1: The result of question 1-1

### Step 3: Review Your Work

Now, you will review your work with your instructor.

**NOTE: Unlike assignments, labs do not need to be turned in to Canvas!**

## Select Options - 20

Although a simple Select-From-Where is the most common way to start writing SQL code, **you can use other options to extract more than simple information**. Let's look at some you will find useful.

### Using Wild Cards

You can use "Wild Card" place holders in your where clause by using **the "LIKE" operator** instead of the "=" operator.

“A wildcard character is used to substitute any other character(s) in a string. Wildcard characters are used with the SQL LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.” (<https://www.w3schools.com/sql/sql_wildcards.asp>, 2017)(external site)

Here are a couple of examples using the Northwind database:

SELECT ProductName, UnitsInStock FROM Products

WHERE ProductName LIKE 'Ch%'; -- % means zero or more characters

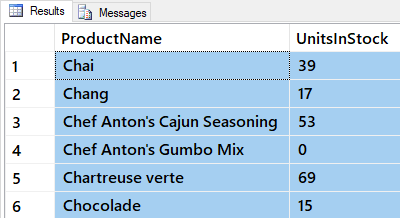


Figure: Result of previous SQL query

SELECT ProductName, UnitsInStock FROM Products

WHERE ProductName LIKE 'C\_a%'; -- \_ means one character

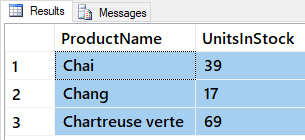


Figure: Result of previous SQL query

SELECT ProductName, UnitsInStock FROM Products

WHERE ProductName LIKE 'Cha[i,n]%'; -- \_ means either i or n character

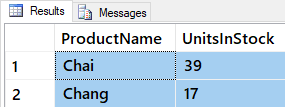


Figure: Result of previous SQL query

SELECT ProductName, UnitsInStock FROM Products

WHERE ProductName LIKE 'Cha[n-r]%'; -- \_ means n through r characters

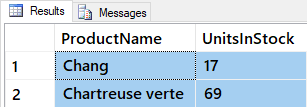


Figure: Result of previous SQL query

### Using Common Conditional Operators

Operators are **keywords or symbols that perform a function**. They are used in all programming languages, including SQL. The following is a sampling of ones you will use most often.

**BETWEEN** evaluates as a TRUE expression when a value is between two other values (inclusive).

SELECT ProductName, UnitsInStock

FROM Products

WHERE UnitsInStock BETWEEN '0' AND '15';

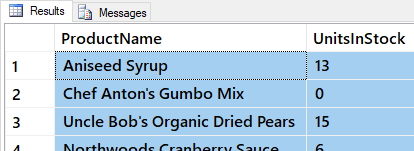


Figure: Result of previous SQL query

The **IN** keyword is used to test if a value is in a list of other values.

SELECT ProductName, UnitsInStock

FROM Products

WHERE UnitsInStock IN ('0','15');

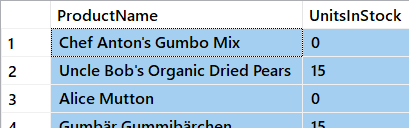


Figure: Result of previous SQL query

You can use **Logical Operators** for more complex queries.

SELECT ProductName, UnitsInStock

FROM Products

WHERE ( productname LIKE 'T%' )

OR ( productid = 17 AND UnitsInStock = 0.00 );

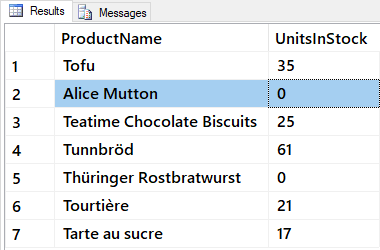


Figure: Result of previous SQL query

You can use **Negation Operators** to reverse the filter.

SELECT ProductName, UnitsInStock FROM Products

WHERE ProductName Not in ('Chai');

SELECT ProductName, UnitsInStock FROM Products

WHERE Not ProductName = 'Chai';

SELECT ProductName, UnitsInStock FROM Products

WHERE ProductName != 'Chai';

SELECT ProductName, UnitsInStock FROM Products

WHERE ProductName <> 'Chai';

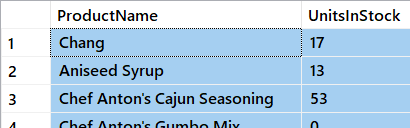


Figure: Result of previous SQL query

Avoid using the "not”, “!=”, or “<>” operators with Null, since this logic can be changed by a connection setting.

SET ANSI\_NULLS ON;

SELECT companyname, fax FROM suppliers WHERE Not fax = NULL;

SELECT companyname, fax FROM suppliers WHERE fax != NULL;

SELECT companyname, fax FROM suppliers WHERE fax <> NULL;

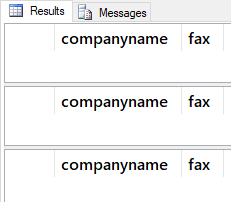


Figure: Result of previous SQL query

SET ANSI\_NULLS OFF;

SELECT companyname, fax FROM suppliers WHERE Not fax = NULL;

SELECT companyname, fax FROM suppliers WHERE fax != NULL;

SELECT companyname, fax FROM suppliers WHERE fax <> NULL;

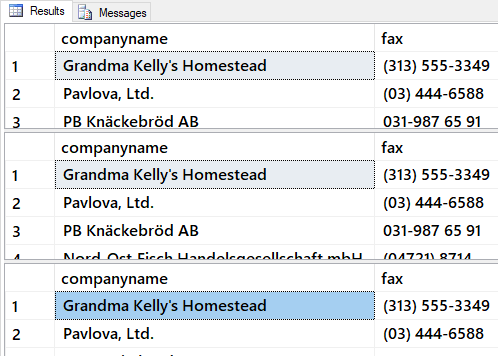


Figure: Result of previous SQL query

To find rows that have null values use the "IS NULL" operator.

SELECT companyname, fax FROM suppliers WHERE fax IS NULL;

#### The Order By Clause

The ORDER BY clause **sorts a query result** by one or more columns. A sort can be ascending (ASC) or descending (DESC).

SELECT Pub\_id, Type, Title\_id, Price FROM Pubs..Titles

ORDER BY Pub\_id DESC, Type, Price;

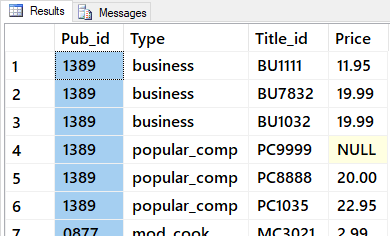


Figure: Result of previous SQL query

SELECT Pub\_id, Type, Title\_id, Price FROM Pubs..Titles

ORDER BY Pub\_id, Type DESC, Price;

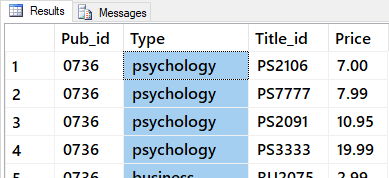


Figure: Result of previous SQL query

SELECT Pub\_id, Type, Title\_id, Price FROM Pubs..Titles

ORDER BY Pub\_id, Type DESC, Price DESC;

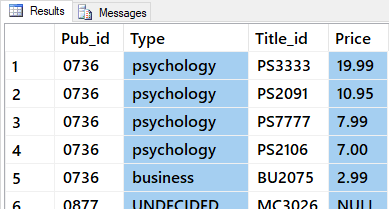


Figure: Result of previous SQL query

-- You can also use the column number in the Order By clause

SELECT Pub\_id, Type, Title\_id, Price FROM Pubs..Titles

ORDER BY 2 DESC;

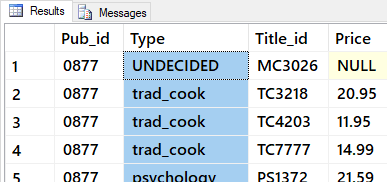


Figure: Result of previous SQL query

#### The TOP Clause

The Top n keyword specifies that the **first n rows of the result set** are to be returned. It **relies on the Order By clause** to determine which results are on "top."

USE northwind

SELECT TOP 5 productid, quantity

FROM [order details]

ORDER BY quantity DESC;

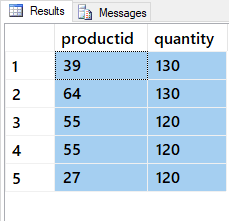


Figure: Result of previous SQL query

This result set lists a total of 10 products, because **additional rows with the same values** as the last row are also included.

SELECT TOP 5 WITH TIES productid, quantity

FROM [order details]

ORDER BY quantity DESC;

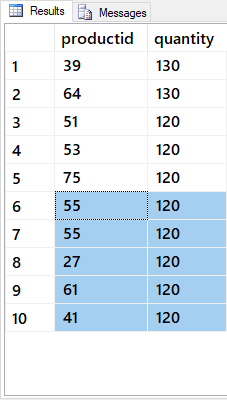


Figure: Result of previous SQL query

#### The DISTINCT Clause

The DISTINCT keyword **eliminates duplicate** rows. Note that this is dependent on the entire row being a duplicate.

SELECT DISTINCT Orders.CustomerID, Orders.Orderdate FROM Orders;

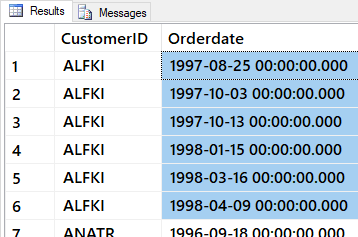


Figure: Result of previous SQL query

## Lab 2: Using Select Options - 20

In this lab, you create some select statements using Northwind database. These statements will use some of the more common Select options.

You will work on your own for the first 10 minutes, then we will review the answers together in the last 10 minutes.

**Note**: This lab should be done individually.

### 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.

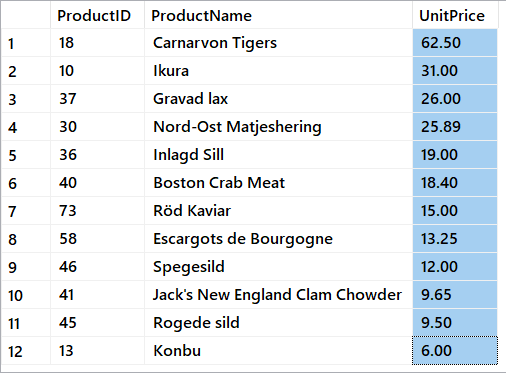
Select \* From Northwind.Sys.Tables Where type = 'u' Order By Name;

### Step 2: Create a Query

Answer the following questions by writing and executing SQL code.

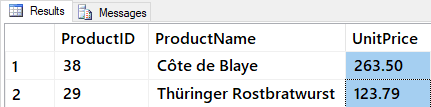
**Note**: The following image is what your results should look like.

**Question 2-1:** Select the Product Id, Product Name, and Product Price of all Products with the Seafood's Category Id. Ordered by the highest to the lowest products price.



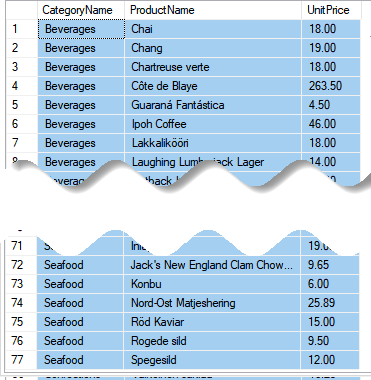
##### Figure 2: The results of questions 2-1

**Question 2-2:** Select the product Id, product name, and product price ordered by the products price highest to the lowest. Show only the products that have a price greater than $100.



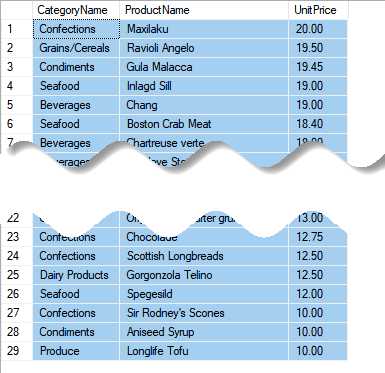
##### Figure 3: The results of questions 2-2

**Question 2-3:** Select the category name, product name, and product price from both categories and products. Order the results by category name and then product name, in alphabetical order. (Hint: Join Products to Category)



##### Figure 4: The results of questions 2-3

**Question 2-4:** Select the Category Name, Product Name, and Product Price from both Categories and Products. Order the results by price highest to lowest. Show only the products that have a price from $10 to $20.



##### Figure 5: The results of questions 2-4

### Step 3: Review Your Work

Now, you will review your work with your instructor.

**NOTE: Unlike assignments, labs do not need to be turned in to Canvas!**

## Break

We will take a short break.

## Additional Select Options - 10

There are many additional options that you can use with Select. These are considered **more advanced, but also more powerful**!

### Union Clause

The Union clause **combines rows of data**.

For example, while these select statements gives you back **two result sets**...

SELECT CompanyName, City

FROM Northwind.dbo.Customers

WHERE Region = 'WA';

SELECT e.FirstName, e.LastName, e.City

FROM Northwind.dbo.Employees as e

WHERE Region = 'WA';

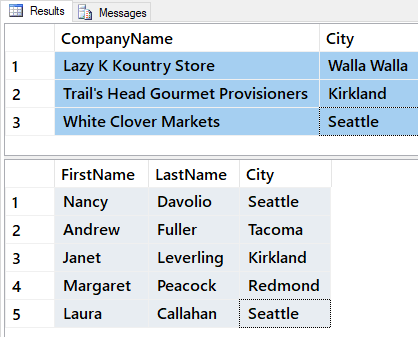


Figure: Result of previous queries

... this example gives you **one result set**.

SELECT CompanyName, City

FROM Northwind.dbo.Customers

WHERE Region = 'WA'

Union

SELECT e.FirstName + ' ' + e.LastName as EmployeeName, e.City

FROM Northwind.dbo.Employees as e

WHERE Region = 'WA'

Order By City;

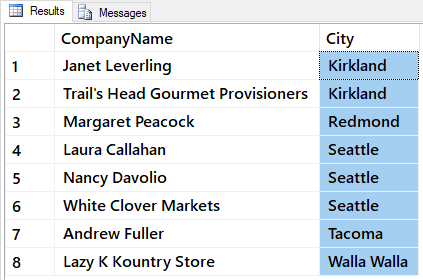


Figure: Result of previous query

### SubQueries

A subquery is a select statement used **inside another select statement**.

SELECT CompanyName, City

FROM Northwind.dbo.Customers

WHERE Region = 'WA'

And CustomerId in (Select CustomerId

From Orders

Where Year(OrderDate) > 1997)

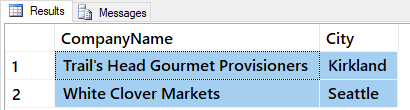


Figure: Result of previous SQL query

### Joins

Joins **combine columns of data** into one result set. To understand how this works, let's look at some simple data from two result sets.

Select CustomerID, CompanyName

From Northwind.dbo.Customers

Order By CustomerID;

Select CustomerId, OrderID, OrderDate

From Northwind.dbo.Orders

Order By CustomerID;

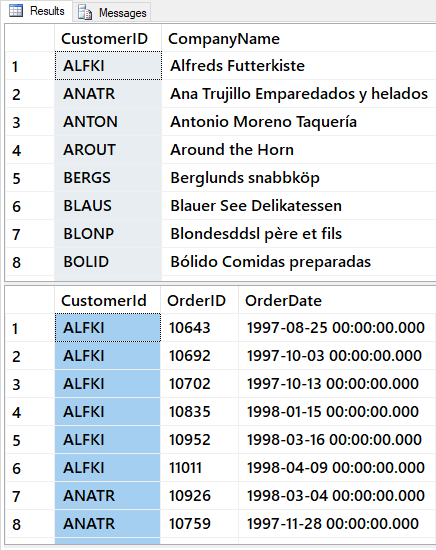


Figure: Result of previous SQL queries

We can combine (or join) all of these columns of data into one result set using the Join - On clause.

Select Customers.CustomerID, CompanyName, OrderID, OrderDate

From Northwind.dbo.Customers

**Join** Northwind.dbo.Orders

On Customers.CustomerID = Orders.CustomerID

Order By Customers.CustomerID;



Figure: Result of previous SQL query

We can make our query easier to read using **Table Aliases**.

Select c.CustomerID, CompanyName, OrderID, OrderDate

From Northwind.dbo.Customers as c

Join Northwind.dbo.Orders as o

ON c.CustomerID = o.CustomerID

Order By c.CustomerID;

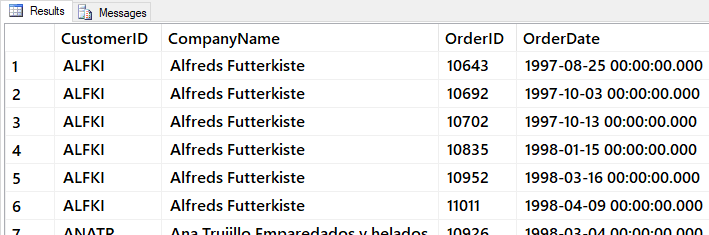


Figure: Result of previous SQL query

You can combine data from more than two tables by adding another Join-On clause.

Select c.CustomerID, CompanyName, o.OrderID, OrderDate, od.ProductID

From Northwind.dbo.Customers as c

Join Northwind.dbo.Orders as o

On c.CustomerID = o.CustomerID

Join Northwind.dbo.[Order Details] as od

On o.OrderID = od.OrderID

Order By c.CustomerID;

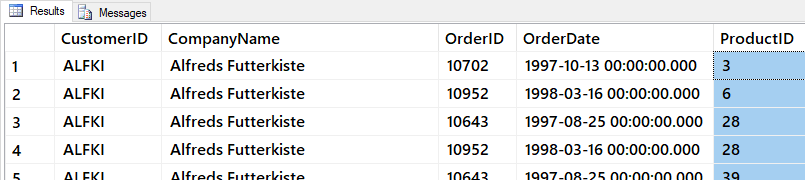


Figure: Result of previous SQL query

#### Note: We will spend more time learning Joins in a later module.

**Note**: We will spend more time learning Joins in a later module.

## Lab 3: Using Additional Select Options - 10

In this lab, you create some select statements using Northwind database. These statements will use some of the more advanced Select options.

You will work on your own for the first 5 minutes, then we will review the answers together in the last 5 minutes.

**Note**: This lab should be done individually.

### 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.

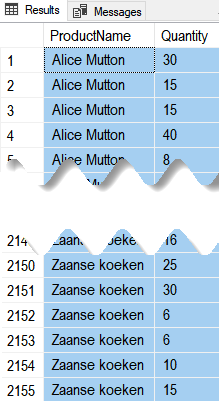
Select \* From Northwind.Sys.Tables Where type = 'u' Order By Name;

### Step 2: Create a Query

Answer the following questions by writing and executing SQL code.

**Note**: The following image is what your results should look like.

**Question 3-1:** Select the product name and the order quantity all products in the Northwind database, ordered by the product name.



##### Figure 6: The results of questions 3-1

### Step 3: Review Your Work

Now, you will review your work with your instructor.

**NOTE: Unlike assignments, labs do not need to be turned in to Canvas!**

## Aggregate Functions - 20

"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 you will work with. 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);

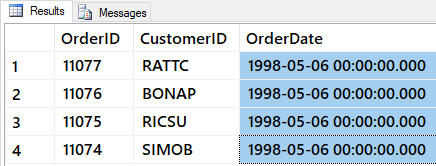


Figure: Result of previous SQL query

Most aggregate functions **exclude Null** values.

SELECT ShippedDate FROM dbo.Orders;

SELECT MAX(ShippedDate) FROM dbo.Orders;

SELECT MIN(ShippedDate) FROM dbo.Orders;

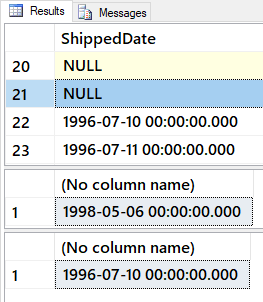


Figure: Result of previous SQL query

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

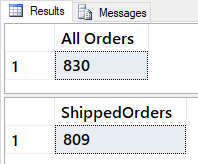


Figure: Result of previous SQL query

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;

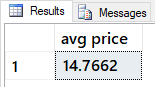


Figure: Result of previous SQL query

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;

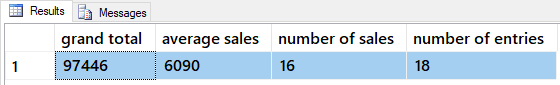


Figure: Result of previous SQL query

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;

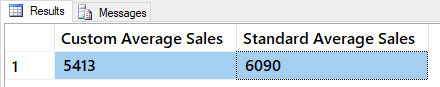


Figure: Result of previous SQL query

#### 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';

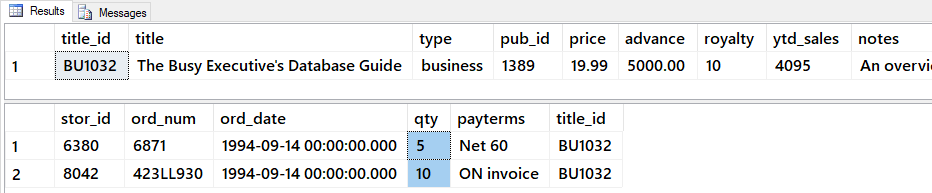


Figure: Result of previous SQL queries

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;

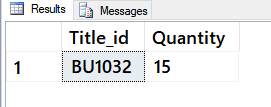


Figure: Result of previous SQL query

I can use Group By with the **Rollup option** If I want a grand total too.

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

From Pubs.dbo.Sales

GROUP BY Title\_id

WITH ROLLUP

ORDER BY Title\_id;

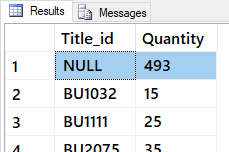


Figure: Result of previous SQL query

The Rollup command return subtotals with the grand total based on which title is first in the Group By clause.

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

FROM Pubs.dbo.Sales

GROUP BY Stor\_id, Title\_id

WITH Rollup; -- subtotals based on Stor\_id

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

FROM Pubs.dbo.Sales

GROUP BY Title\_id, Stor\_id

WITH Rollup; -- subtotals based on Title

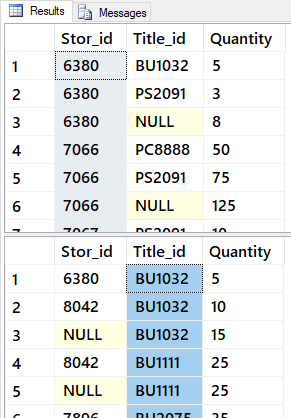


Figure: Result of previous SQL query

-- The Cube command returns all combinations of subtotals

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

FROM Pubs.dbo.Sales

GROUP BY Stor\_id,Title\_id

WITH Cube -- subtotals based on both Stor\_id and Title

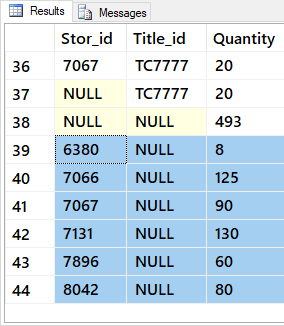


Figure: Result of previous SQL query

#### Having

After data has been grouped, you can add a filter on the results using the Having option. This is different than Where since it is applied after the 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

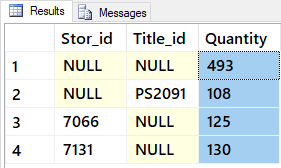


Figure: Result of previous SQL query

# Session02 - Lab

## Lab 4: Selecting Aggregate Results - 50

In this lab, you create advanced select statements, using a new Lab database.

You will work on your own for the first 30 minutes, then we will review the answers together in the last 20 minutes.

**Note**: This lab should be done individually or in groups of three or less.

### Step 1: Create a Lab Database

Run the copy and paste the following SQL code into a code window, then modify it to use your own name instead of "YourNameHere." Afterward, execute the code to make the Lab03Database.

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*--

-- Title: Module03\_Lab04

-- Author: YourNameHere

-- Desc: This file demonstrates how to select data from a database

-- Change Log: When,Who,What

-- 2017-01-01,YourNameHere,Created File

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*--

Use Master;

go

If Exists(Select Name from SysDatabases Where Name = 'MyLabsDB\_YourNameHere')

Begin

Alter Database [MyLabsDB\_YourNameHere] set Single\_user With Rollback Immediate;

Drop Database MyLabsDB\_YourNameHere;

End

go

Create Database MyLabsDB\_YourNameHere;

go

Use MyLabsDB\_YourNameHere;

go

-- Create Tables (Module 01)--

Create Table Categories

([CategoryID] [int] IDENTITY(1,1) NOT NULL

,[CategoryName] [nvarchar](100) NOT NULL

);

go

Create Table Products

([ProductID] [int] IDENTITY(1,1) NOT NULL

,[ProductName] [nvarchar](100) NOT NULL

,[CategoryID] [int] NULL

,[UnitPrice] [money] NOT NULL

);

go

Create Table Inventories

([InventoryID] [int] IDENTITY(1,1) NOT NULL

,[InventoryDate] [Date] NOT NULL

,[ProductID] [int] NOT NULL

,[Count] [int] NOT NULL

);

go

-- Add Constraints (Module 02) --

Alter Table Categories

Add Constraint pkCategories

Primary Key (CategoryId);

go

Alter Table Categories

Add Constraint ukCategories

Unique (CategoryName);

go

Alter Table Products

Add Constraint pkProducts

Primary Key (ProductId);

go

Alter Table Products

Add Constraint ukProducts

Unique (ProductName);

go

Alter Table Products

Add Constraint fkProductsToCategories

Foreign Key (CategoryId) References Categories(CategoryId);

go

Alter Table Products

Add Constraint ckProductUnitPriceZeroOrHigher

Check (UnitPrice >= 0);

go

Alter Table Inventories

Add Constraint pkInventories

Primary Key (InventoryId);

go

Alter Table Inventories

Add Constraint dfInventoryDate

Default GetDate() For InventoryDate;

go

Alter Table Inventories

Add Constraint fkInventoriesToProducts

Foreign Key (ProductId) References Products(ProductId);

go

Alter Table Inventories

Add Constraint ckInventoryCountZeroOrHigher

Check ([Count] >= 0);

go

Insert Into Categories

(CategoryName)

Select CategoryName

From Northwind.dbo.Categories

Order By CategoryID;

go

Insert Into Products

(ProductName, CategoryID, UnitPrice)

Select ProductName,CategoryID, UnitPrice

From Northwind.dbo.Products

Order By ProductID;

go

Insert Into Inventories

(InventoryDate, ProductID, [Count])

Select '20170101' as InventoryDate, ProductID, ABS(CHECKSUM(NewId())) % 100 as RandomValue

From Northwind.dbo.Products

UNION

Select '20170201' as InventoryDate, ProductID, ABS(CHECKSUM(NewId())) % 100 as RandomValue

From Northwind.dbo.Products

UNION

Select '20170302' as InventoryDate, ProductID, ABS(CHECKSUM(NewId())) % 100 as RandomValue

From Northwind.dbo.Products

Order By 1, 2

go

-- Show all of the data in the Categories, Products, and Inventories Tables

Select \* from Categories;

go

Select \* from Products;

go

Select \* from Inventories;

go

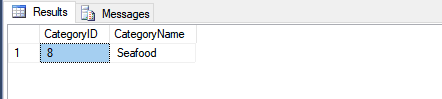
### Step 2: Create Some Queries

Answer the following questions by writing and executing SQL code. We start with some review questions to get you warmed up, and then move on to new ones!

**Notes**:

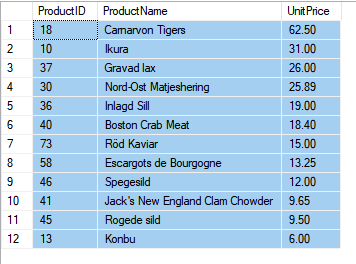
* The following image is what your results should look like.
* Inventory Quantities may vary, since I use a random function to create the data!
* Make sure your code is well formatted, consistent, and produces the same result!

**Question 1**: Select the Category Id and Category Name of the Category 'Seafood'.



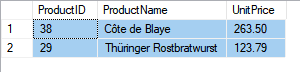
##### Figure 9: The results of question 1 in Lab 4

**Question 2**: Select the Product Id, Product Name, and Product Price of all Products with the Seafood's Category Id. Ordered By the Products Price highest to the lowest



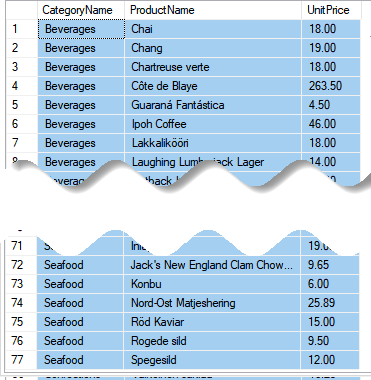
##### Figure 10: The results of question 2 in Lab 4

**Question 3**: Select the Product Id, Product Name, and Product Price Ordered By the Products Price highest to the lowest. Show only the products that have a price greater than $100.



##### Figure 11: The results of question 3 in Lab 4

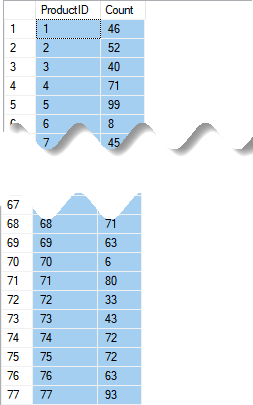
**Question 4**: Select the CATEGORY NAME, product name, and Product Price from both Categories and Products. Order the results by Category Name and then Product Name, in alphabetical order. (Hint: Join Products to Category)



##### Figure 12: The results of question 4 in Lab 4

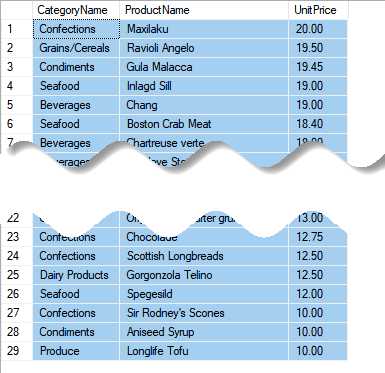
**Question 5**: Select the Product Id and Number of Products in Inventory for the Month of JANUARY. Order the results by the ProductIDs.

(**Note**: Quantities may vary, since I use a random function to create the data!)



##### Figure 13: The results of question 5 in Lab 4

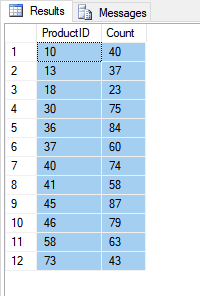
**Question 6**: Select the Category Name, Product Name, and Product Price from both Categories and Products. Order the results by price highest to lowest. Show only the products that have a PRICE FROM $10 TO $20.



##### Figure 14: The results of question 6 in Lab 4

**Question 7**: Select the Product Id and Number of Products in Inventory for the Month of JANUARY. Order the results by the ProductIDs and where the ProductID are only the ones in the seafood category (Hint: Use a subquery to get the list of productIds with a category ID of 8)

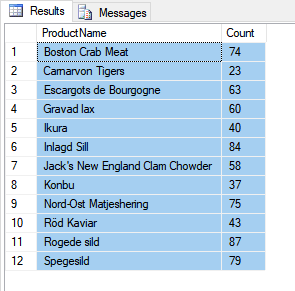
(**Note**: Quantities may vary, since I use a random function to create the data!)



##### Figure 15: The results of question 7 in Lab 4

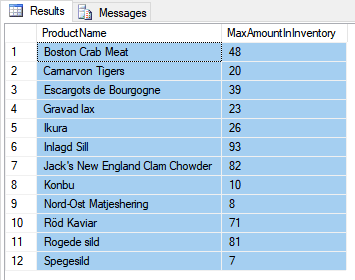
**Question 8**: Select the PRODUCT NAME and Number of Products in Inventory for the Month of January. Order the results by the Product Names and where the ProductID as only the ones in the seafood category (Hint: Use a Join between Inventories and Products to get the Name)

(**Note**: Quantities may vary, since I use a random function to create the data!)



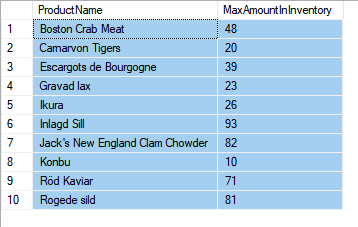
##### Figure 16: The results of question 8 in Lab 4

**Question 9:** Select the Product Name and Number of Products in Inventory for both JANUARY and FEBURARY. Show what the MAXIMUM AMOUNT IN INVENTORY was and where the productID as only the ones in the seafood category and Order the results by the Product Names. (Hint: If Jan count was 5, but Feb count was 15, show 15) (**Note**: Quantities may vary, since I use a random function to create the data!)



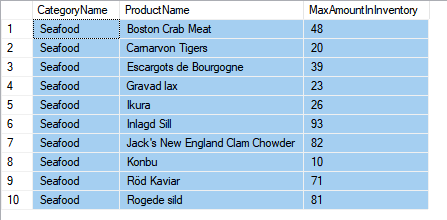
##### Figure 17: The results of question 9 in Lab 4

**Question 10:** Select the Product Name and Number of Products in Inventory for both JANUARY and FEBURARY. Show what the MAX AMOUNT IN INVENTORY was and where the ProductID as only the ones in the seafood category and Order the results by the Product Names. Restrict the results to rows with a MAXIMUM COUNT OF 10 OR HIGHER. (**Note**: Quantities may vary, since I use a random function to create the data!)



##### Figure 18: The results of question 10 in Lab 4

**Question 11**: Select the CATEGORY NAME, Product Name and Number of Products in Inventory for both JANUARY and FEBURARY. Show what the MAX AMOUNT IN INVENTORY was and where the ProductID as only the ones in the seafood category and Order the results by the Product Names. Restrict the results to rows with a maximum count of 10 or higher (**Note**: Quantities may vary, since I use a random function to create the data!)



##### Figure 19: The results of question 11 in Lab 4

### Step 3: Review Your Work

Now, you will review your work with your instructor.

# Session03 Lectures and Labs < 110 mins

## Selecting with Common Functions - 30

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);

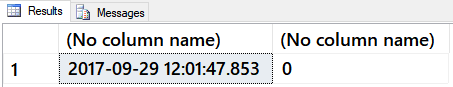


Figure: Result of previous SQL query

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;

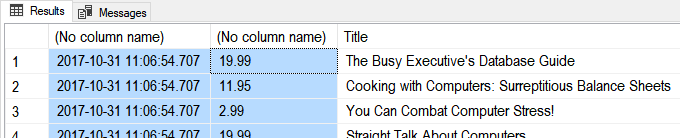


Figure: Result of previous SQL query

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

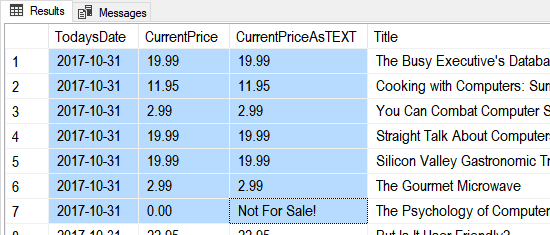


Figure: Result of previous SQL query

#### 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);

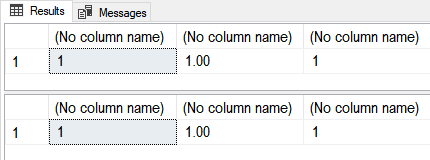


Figure: Result of previous SQL query

**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

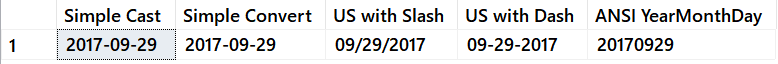


Figure: Result of previous SQL query

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

#### The Immediate IF function

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

Select

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

From Northwind.dbo.Products;

go

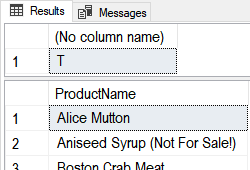


Figure: Result of previous SQL query

#### The Choose function

Select Choose(1, 'A', 'B', 'C'), Choose(2, 'A', 'B', 'C');

Select

ProductName

,[Category] = Choose(CategoryID, 'A','B','C')

From Northwind.dbo.Products;

go

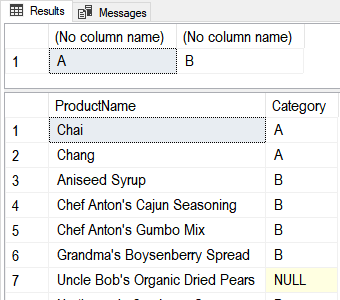


Figure: Result of previous SQL query

#### 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

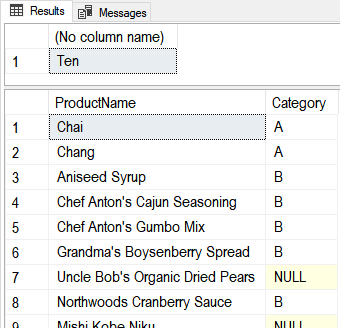


Figure: Result of previous SQL query

A Better example of Case:

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

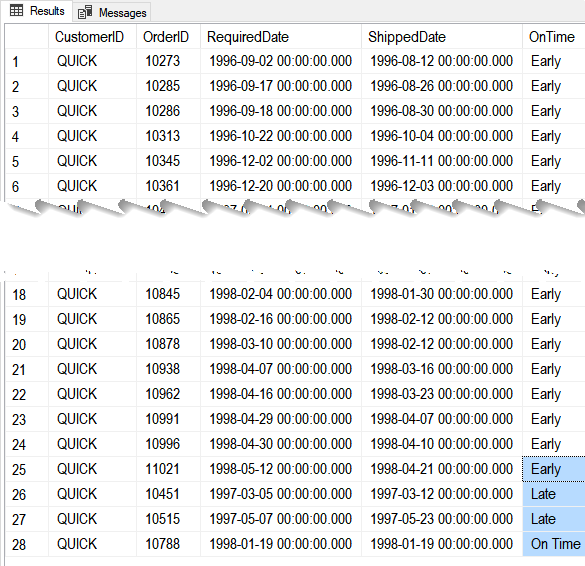


Figure: Result of previous SQL query

#### The IsNumeric Funtion:

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

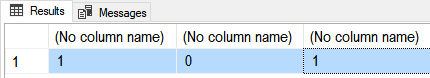


Figure: Result of previous SQL query

#### The IsDate Function:

Select

IsDate('1/1/2001')

,IsDate('01-01-2001')

,IsDate('20010101')

,IsDate('Jan,01,2001')

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

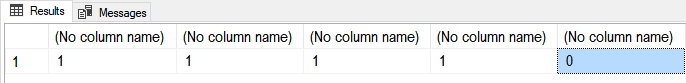


Figure: Result of previous SQL query

#### The 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

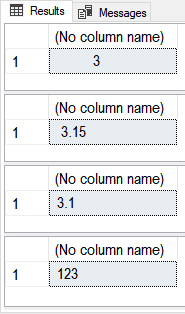


Figure: Result of previous SQL query

**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.

#### 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 Left and Right Functions

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

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

;

Go

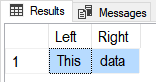


Figure: Result of previous SQL query

#### 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

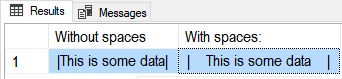


Figure: Result of previous SQL query

#### The Upper and Lower Functions

Select Upper('Bob Smith'), Lower('Bob Smith');

go

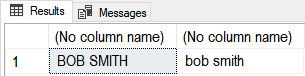


Figure: Result of previous SQL query

#### The STUFF Function

( character\_expression , start , length , replaceWith\_expression )

Select Stuff('Bob Smith',1,3,'Robert');

Go

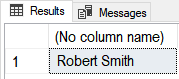


Figure: Result of previous SQL query

#### The REPLACE Function

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

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

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

go

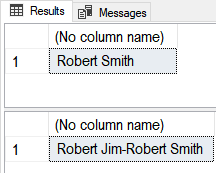


Figure: Result of previous SQL query

#### The PATINDEX Function

( '%pattern%' , expression )

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

SELECT

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

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

;

go

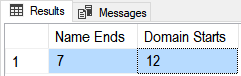


Figure: Result of previous SQL query

#### 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

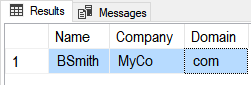


Figure: Result of previous SQL query

#### 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));

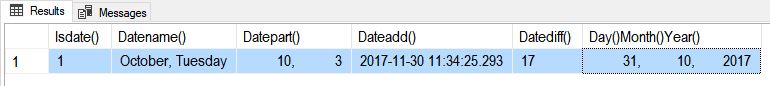


Figure: Result of previous SQL query

## Lab 5: Using Common Functions - 20

In this lab, you create advanced select statements with functions, using Northwind database.

You will work on your own for the first 10 minutes, then we will review the answers together in the last 10 minutes.

**Note**: This lab should be done individually or in groups of three or less.

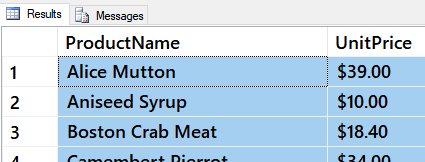
### Step 1: Review Database Tables

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;

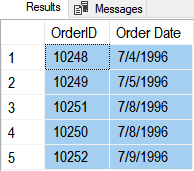
### Step 2: Create a Query

**Question 1**: Show a list of Product names, and the price of each product, with the price formatted as US dollars? Order the result by the Product!



##### Figure 22: The results of question 1 in Lab 4

**Question 2:** Show a list of the top five Order Ids and Order Dates based on the ordered date. Format the results as a US date with back-slashes. Use a column alias to rename orderdate to Order Date.



##### Figure 23: The results of question 2 in Lab 4

### Step 3: Review Your Work

Now, you will review your work with your instructor.

## Break

We will take a short break

## Basic Transactions - 20

"A transaction is a **single unit of work**. If a transaction is successful**, all of the data modifications made during the transaction are committed and become a permanent part of the database**. If a transaction encounters errors and must be canceled or rolled back, then all of the data modifications are erased." (<https://technet.microsoft.com/en-us/library/ms174377(v=sql.110).aspx>, 2017)

Consider a table like this one:

Create Table TempDB.dbo.Contacts

(ContactID int Identity, FirstName varchar(50),LastName varchar(50), EmailAddress varchar(50))

go

Use TempDB;

#### Insert

Insert allows you to add data to a table. The best way to type an insert statement is with an **explicit list** of columns. Here is a simple example:

Insert Into dbo.Contacts

(FirstName,LastName, EmailAddress)

Values

('Bob', 'Smith', 'BSmith@MyCo.Com');

go

Select \* from Contacts;

While this works, it is **not as good**.

Insert Into dbo.Contacts

~~-- (FirstName,LastName, EmailAddress)~~

Values

('Sue', 'Jones', 'SJones@MyCo.Com');

go

Select \* from Contacts;

When you add a new row to a table with an **Identity Option** you can **see what the new ID is** using this code:

Select @@IDENTITY

Trying to **add a ID value on an Identity column will not work**!

Insert Into dbo.Contacts

(ContactID, FirstName,LastName, EmailAddress)

Values

(3, 'Tim', 'Thomas', 'TThomas@MyCo.Com');

#### Update

Update statements allow you to **change existing data**. Here is a simple example:

Update Contacts

Set LastName = 'Smith'

Where ContactId = 2;

You can update **multiple columns** at a time like this:

Update Contacts

Set LastName = 'Smith'

,EmailAddress = 'SSmith@MyCo.com'

Where ContactId = 2;

And, **multiple rows** like this!

Update Contacts

Set LastName = 'Smith';

~~-- Where ContactId = 2;~~

**NOTE**: **Leaving off the Where clause will update ALL ROWS!** This is a really **BAD** thing to do by **accident**!

#### Delete

Delete statements allow you to **remove one or more rows**.

Delete

From dbo.Contacts

Where ContactId = 2;

**NOTE**: **Leaving off the Where clause will delete ALL ROWS!** This is a really **BAD** thing to do by **accident**!

Delete

From dbo.Contacts

~~-- Where ContactId = 2;~~

## Lab 6: Processing Transactions - 30

In this lab, you create transaction statements, using a new Lab database.

You will work on your own for the first 10 minutes, then we will review the answers together in the last 10 minutes.

**Note**: This lab should be done individually or in groups of three or less.

### Step 1: Create a Lab Database

Run the copy and paste the following SQL code into a code window, then modify it to use your own name instead of "YourNameHere." Afterward, execute the code to make the Module03\_Lab05 database.

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*--

-- Title: Module03\_Lab05DB

-- Author: YourNameHere

-- Desc: This file demonstrates how to process data in a database

-- Change Log: When,Who,What

-- 2017-01-01,YourNameHere,Created File

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*--

Use Master;

go

If Exists(Select Name from SysDatabases Where Name = 'MyLabsDB\_YourNameHere')

Begin

Alter Database [MyLabsDB\_YourNameHere] set Single\_user With Rollback Immediate;

Drop Database MyLabsDB\_YourNameHere;

End

go

Create Database MyLabsDB\_YourNameHere;

go

Use MyLabsDB\_YourNameHere;

go

-- Create Tables (Module 01)--

Create Table Categories

([CategoryID] [int] IDENTITY(1,1) NOT NULL

,[CategoryName] [nvarchar](100) NOT NULL

);

go

Create Table Products

([ProductID] [int] IDENTITY(1,1) NOT NULL

,[ProductName] [nvarchar](100) NOT NULL

,[CategoryID] [int] NULL

,[UnitPrice] [money] NOT NULL

);

go

-- Add Constraints (Module 02) --

Alter Table Categories

Add Constraint pkCategories

Primary Key (CategoryId);

go

Alter Table Categories

Add Constraint ukCategories

Unique (CategoryName);

go

Alter Table Products

Add Constraint pkProducts

Primary Key (ProductId);

go

Alter Table Products

Add Constraint ukProducts

Unique (ProductName);

go

Alter Table Products

Add Constraint fkProductsToCategories

Foreign Key (CategoryId) References Categories(CategoryId);

go

Alter Table Products

Add Constraint ckProductUnitPriceZeroOrHigher

Check (UnitPrice >= 0);

go

-- Show the Current data in the Categories, Products, and Inventories Tables

Select \* from Categories;

go

Select \* from Products;

go

### Step 2: Create SQL Transaction Statements

**Question 1**: How would you add data to the Categories table?

**Question 2**: How would you add data to the Products table?

**Question 3**: How would you update data in the Products table?

**Question 4**: How would you delete data from the Categories table?

### Step 3: Review Your Work

Now, you will review your work with your instructor.