```
-- MS SQL NOTES 2017 (PGH)

-- Execute Shortcuts
Cmd + E or Alt + x


-- Notes regarding single vs double quotes vs brackets
-- THE ANSWER IN SHORT (BEFORE THE DETAIL BELOW) is use "" for the AS statement
  instead of square brackets, and single quotes for strings.

-- Return all Tables in DB
SELECT * FROM sys.Tables;

-- Ways to Alias
select FirstName as [First Name], Title as 'Job Title';

-- Concatenation
concat(FirstName,' ',Title) as 'Full';

-- Distinct (Show only unique entries)
select distinct ContactTitle
from customers;

-- Inner Join (Joined 3 tables together) with an aggregate function (SUM) and grouped
  by and ordered by
select products.ProductName, Total = round(sum([Order Details].UnitPrice * [Order
  Details].Quantity * (1 - [Order Details].Discount)),2)
from Products
inner join [Order Details] on Products.ProductID = [Order Details].ProductID -- First
  Join
inner join Orders on [Order Details].OrderID = Orders.OrderID -- Second Join
where Products.ProductName like '%LAY%' -- Between
or Products.ProductName like '_arte%' -- between missing first letter and end
or Products.ProductName like 'Racl%' -- starts with
or Products.ProductName like '%Mutton' -- ends with
group by Products.ProductName
order by Total DESC
;

-- LIKE Keyword for finding things fuzzy
select *
from Products
where ProductName LIKE '%an%'

select *
from Customers
where CompanyName
LIKE 'c%' OR CompanyName LIKE 'a%' OR CompanyName LIKE 't%'
;

-- Using [] for Like Statement with Multiple items
select *
```

```
from Customers
where CcompanyName LIKE '[cat]%' -- This does NOT look for cat, it looks for company
  names starting with c, or a or t
;
-- You can also use this for ranges:
select *
from Customers
where CcompanyName LIKE '[c-t]%' -- Looks for companies that start with letters c
  through to t
;

-- Using the 'NOT' Keyword (also works with the ^ symbol (karat))
Select *
from Customers
Where CompanyName not Like '[c-t%]'
;
Select *
from Customers
Where CompanyName Like '[^c-t%]' -- Does the same thing as above....but carret symbol
  makes negative (i.e. Not Like)
;

-- using Between & And to find an item within an exclusive range
select *
from Orders
where ShippedDate between '1996-09-01' and '1996-10-31'
-- also works with where ShippedDate between 'September 1, 1996' and 'October 31,
  1996'
;
select *
from Products
where UnitsInStock between 50 and 100
;
select *
from Products
where UnitsInStock between 'G' and 'S'
;

-- Using the IN keyword.  Use to create a list of criteria for your WHERE clause
-- ie. use as alternative to: where City in ('Seattle', 'Tacoma')
where City IN ('Seattle', 'Tacoma')
;
select *
from products
where SupplierID in (1, 7, 16)
;
-- Using NULL or NOT NULL (test to see if data is in a field or no data in a field)
select *
from Customers
where Fax is NULL
;
select *
```

```
from Customers
where Fax is NOT NULL
;
-- Finding something with just the year if there is a date string.  Use the DATE FUNCTION
select CustomerID, OrderDate, ShippedDate
from orders

-- where ShippedDate between '1-1-1998' AND '12-31-1998' (Long way)
where YEAR(ShippedDate) = '1998' -- <-- This is the best way to do it using the DATE FUNCTION
;

-- Using the In Keyword
select *
from products
where SupplierID in (1, 7, 16)
;
select *
from Customers
where ContactTitle = 'Owner'
and Country in ('France','Spain')
;

-- Using And with Like in Where Clause, also using one positive, one negative (AND NOT) <> <--- means AND NOT
select *
from Employees
where Title like '%rep%' and city = 'London'
;
select *
from Employees
where Title like '%rep%' and not city = 'London'
-- Can also use where Title like '%rep%' and city <> 'London'
;

-- Using "Where Not" vs. '<>'...I prefer the latter since it is easier to read
select *
from Employees
where not Title = 'Sales Representative'
where Title <> 'Sales Representative' -- Means same thing as above, but easier to read
;

-- ORDER BY SECTION
SELECT ColumnName(s)
FROM TableName
ORDER BY ColumnName ASC/DESC -- ASC is default if you do not specify!
;
-- NOTE - You can also sort by column Names!!! ORDER by 3 ASC/DESC
SELECT ColumnName(s)
FROM TableName
ORDER BY 2 ASC/DESC
```

```
-- Can also order by multiple column names to keep thigns together, i.e. below, this
  would be order by region, then order by city.
Select *
from Employees
Order by Region, City
;
Select *
from Employees
Order by Region DESC, City -- Can also add DESC to first order by column
;
select *
from Products
where UnitsInStock < 50
and Discontinued = 0
order by UnitsInStock DESC -- or can say  order by 7 since the column I'm looking for
  is column 7
;

-- AGGREGATE FUNCTIONS!! eg. sum(), count(), avg()....you always have to tell sql what
   you want to group those by!
-- Note also, if you want to filter results that have been aggregated, you DONT USE
  'WHERE' clause, you the 'HAVING' Clause~!!!!!
select CustomerID, Count(OrderID) AS [Number of Orders]
from orders
Group by CustomerID
;
select CustomerID, Count(OrderID) AS [Number of Orders]
from orders
Group by CustomerID
having count(OrderID) >= 20 -- <--- This filters the aggregated results like a where
  clause...but you cant use a where clause here
;
select ProductID, sum(quantity) as "Quantity Sold"
from "Order Details"
group by ProductID
having sum(quantity) > 1000 -- Filters results of aggregate function sum, giving only
  those over 1000 (remember, use "having" not the "where" clause)
order by ProductID ASC
;
-- Count Function
select count(*) as "Customers" -- Easy way to run a count on records
from Customers
;
select CustomerID, count(OrderID) as "No. of Orders"
from Orders
group by CustomerID
having count(OrderID) > 0
order by count(OrderID) DESC
;
-- Min Max Avg
-- Finding Lowest Price per product, highest price per product, average price per
```

```
  product
select count(ProductName) as "No. of Products", min(UnitPrice) as "Min Price", max
  (UnitPrice) as "Max Price", avg(UnitPrice) as "Avg Price"
from Products
;
-- Find the most expensive product, and total number of products in the Products
  table.
select count(ProductName) as "Number of Products", max(UnitPrice) as "Max Price"
from Products
;
-- Using mathematical operation within an aggregate function (sum)...find the order
  total for each order when there are unit price and quantities
select OrderID, sum(UnitPrice * Quantity) as "Order Total"
from "Order Details"
group by OrderID
order by "Order Total" DESC
;

-- DATE TIME OPERATIONS
select SYSDATETIME() -- This gives you the current year, month, day, time formatted to
   a fraction of a second
select SYSDATETIMEoffset() -- This gives you the current offset from UTC
select sysutcdatetime() -- This gives you the current UTC date and time.
select CURRENT_TIMESTAMP -- This shows the standard format for dates and times (note,
  you don't need parenthesis)
select GETDATE() --
select GETUTCDATE() --
;
-- The Convert Function lets you extract certain things from the full date time
-- Formula is convert([type],[date/time function],[style number]
select convert(date, sysdatetime()) -- Answer: 2017-10-16
select convert(time, sysdatetime()) -- Answer: 12:03:51.7985245
select convert(varchar, sysdatetime()) -- Answer: 2017-10-16 12:03:51.7985245
select convert(varchar, sysdatetime(), 106) -- <-- The 106 is a number corresponding
  to a 'style' (you can look these up online) ANS: 16 Oct 2017
select convert(varchar, sysdatetime(), 113) -- <-- The 106 is a number corresponding
  to a 'style' (you can look these up online) ANS: 16 Oct 2017 12:08:02.4165897

-- DAY MONTH YEAR FUNCTIONS
select orderID, Year(OrderDate) as "Year", Month(OrderDate) as "Month", Day(OrderDate)
   as "Day"
from Orders
order by orderID ASC
;
select Year(getdate()) -- get the current year
;
-- DATE PART (extracts interval from date string)
select OrderDate, datepart(month, OrderDate) as "Order Month"
from Orders
;
-- DATE ADD (Adds/subtracts interval to an existing date for use in another column,
  example is add 2 days to order date to see when needs to ship by
```

```
select OrderDate, dateadd(day,2,orderDate) as "Taret ship date" -- formula is dateadd
  (interval type, incriment e.g 2 days, the column name incrementing)
from Orders
;
-- to subtract instead of add:
select ShippedDate, dateadd(Day,-2,ShippedDate) as "Email customer about shipment
  date"  -- only change is -2 (two days prior)
from Orders
;
-- Datediff Function (Calculate the difference between two dates) Formula asks for
  (type, then original date, second date)
select OrderDate, ShippedDate, datediff(day,OrderDate,ShippedDate) as "Days Between
  Ordered and Shipped Dates"
from Orders
;
select datediff(day,getdate(),'January 1, 2018') -- Quick way to get the days between
  now and New Year's day
;

-- E ERCISES ON DATE TIME (MY ANSWERS)
-- Find out how many months it has been since the orders have shipped.  Use the Orders
   table.
select OrderID, datediff(month,OrderDate,ShippedDate) as "Months it took to ship since
   order date"
from orders
order by 2 DESC
;
-- Using the Orders table, calculate the due date of the payments by adding 30 days to
   the Order Date field.
select OrderID, OrderDate, dateadd(day, 30, OrderDate) as "Payment Due Date"
from Orders
;
-- Using the Orders table, convert the Order Date field to be displayed with a Mon dd,
   yyyy format.
select OrderID, CONVERT(varchar(20), OrderDate, 107) as "Order Date"
from Orders
;
-- Display the Order ID field and month of the Order Date field for the records in the
   Orders table.
select OrderID, datepart(month,OrderDate) as "Order Month"
from Orders
;
-- Find the order id, and number of days that passed between the Order Date and the
  Shipped Date of orders. Use the Orders table.
select OrderID, datediff(day,OrderDate, ShippedDate) as "order to shipped delta"
from Orders
;

-- CONCATENATION
select concat(firstName,' ', Lastname) as "Full Name" -- <-- Best Way
from Employees
;
```

```
select FirstName + ' ' + LastName as "Full Name" -- <-- Not good
;

-- LOWER AND UPPER (Converts just like excel)
select upper(firstname), lower(Lastname)
from Employees
;

-- LEFT RIGHT FUNCTIONS (When you want to extract a portion of a string)
select concat(lastname,left(firstName,1),right(Extension,2)) as "Username"
-- or Lastname + left(firstname,1) + right(lastname,2) as "Username"
from Employees
;

-- SUBSTRING FUNCTION (Extracts portion of string starting anywhere in the string
  (that's the diff between this and left/right))
Select substring('isabel harrison on udemy', 1, 6)  -- Pulls part of string starting
  at 1st character and 6th ending character

-- LEN FUNCTION (Returns the length of a string)
-- LEN will take into account any opening spaces at the beginning of a string,
  however, it DOESN'T take into account trailing spaces at the end of the field, or
  string.
Len('String')

-- LTRIM and RTRIM (Left Trim and Right Trim)
-- The LTRIM function removes leading spaces in your strings, and RTRIM removes
  trailing spaces.
-- This can be accomplished in your fields, or handwritten strings.
SELECT LTRIM('        Extra space be gone')
SELECT RTRIM('Extra space be gone      ')
;

-- REPLACE FUNCTION (This is temporary, not permanent unless using Update)
-- Forumla: REPLACE(Column Name, String want to replace, String replacing it
  with)...NOTE THIS IS A TEMPORARY CHANGE, NOT PERMANENT
select FirstName, LastName, Replace(City,'Red','Rich') as "City"
from Employees
;

-- SUB QUERIES SECTION (NOTE: SQL evaluates sub query FIRST! and then works on the
  outer query)
-- You can have a subquery anywhere there is a statement, within update statements
  etc.
-- Good example where we want quantity that is above the average quantity (we're just
  nesting another select statement)
select *
from "Order Details"
where Quantity > (Select AVG(Quantity) from "Order Details") -- The sub query is
  within parenthesis
;
-- Example where we're pulling from multiple tables (Products and Order Details
```

```
  tables)
select ProductID, UnitPrice
from Products -- <-- Looking at Products Table
where ProductID in (select Distinct productID from "Order Details" where Quantity >
  100) -- <-- Sub Query is looking at Order Details table
;

-- Using E ISTS keyword with Subqueries
-- The keyword E ISTS, works as an existence test for a subquery. Unlike the
  subqueries you learned in the last lecture, these subqueries do not produce any
  data. Instead they return a value of TRUE or FALSE.
SELECT Columns
FROM Table
WHERE [NOT] E ISTS (Subquery)
;
-- E ISTS statements usually contain an asterisk (*) because when you are testing
  whether rows (records) that meet the specified conditions in the subquery exist,
  there is no need to list columns.

-- The following query displays the Product Name and Unit Price of all products that
  were ordered.  In other words, where the Product ID is in the Order Details table.
select ProductName, UnitPrice
from Products
where Exists (select * from "Order Details" where Products.productID = "Order
  Details".ProductID)
;
-- Alternatively, if you want to know which products have NOT sold, you would type:
SELECT ProductName, UnitPrice
FROM Products
WHERE NOT E ISTS
(SELECT *
FROM [Order Details]
WHERE Products.ProductID = [Order Details].ProductID )
;
-- Example 2
select ProductName, UnitPrice
from Products
where ProductID in
(select ProductID from "Order Details" where Quantity >= 100)
;
-- Find the name of the company that placed order 10251.  Use the Customers and Orders
   table.
select companyName
from Customers
where CustomerID = (select customerID from Orders where OrderID = 10251) -- Substring
  yields an answer, and saying Where CustomerID = that result.
;
-- Find the name of the companies that placed orders between July 1, 1996 and July 31,
   1996.  Use the Customers and Orders table.
select CompanyName
from Customers
where CustomerID in
```

```
(select CustomerID from Orders where OrderDate between '1996-07-01' and '1996-07-31')
;
-- Select the Order ID and Customer ID of the customers who placed orders on the latest order date.  Use the Orders table.
Select OrderID, CustomerID
from orders
where OrderDate =
(select max(OrderDate) from Orders)

-- JOINS!!!! ..use Customers table (on left) and Orders table (on Right) as examples...
-- INNER JOIN = Matching value in both tables (Only the customers who have orders)
-- LEFT OUTER JOIN = All Customers regardless of  whether they have orders or not, but if they did place orders, would see those)
-- RIGHT OUTER JOIN = All Orders (regardless of customer, but if there is a match with Customers, will show those customers with orders)
-- FULL JOIN =
-- SELF JOIN =
-- UNION JOIN =

-- Join Syntax:
Select ColumnName(s) -- That you want to see in your query
from TableName1
(Type) Join TableName2 -- Type can be "Inner Join" or just "Join"....or 'Left Join', 'Right Join',
On ColumnName1 = ColumnName2 -- Tell SQL what the comment fields match between the two tables
;

-- Inner Join Example:
select Customers.CompanyName, Orders.OrderID, Orders.OrderDate, Orders.EmployeeID
from Customers
Inner Join Orders on Customers.CustomerID = Orders.CustomerID
where Orders.EmployeeID = 5
;
-- Inner Join with 3 Tables
select Customers.CompanyName, Orders.OrderID, Orders.OrderDate, Orders.EmployeeID,
  Concat(Employees.FirstName,' ',Employees.LastName) as "Employee Name"
from Customers
Inner Join Orders on Customers.CustomerID = Orders.CustomerID
Inner Join Employees on Orders.EmployeeID = Employees.EmployeeID
where Orders.EmployeeID = 5
;
-- Outer Join (Looking for Any Customers that have not placed orders!)...all data from left table, and data from right if matches.
select Customers.CompanyName, Customers.ContactName, Customers.Phone, Orders.OrderID
from Customers
Left Join Orders on Customers.CustomerID = Orders.CustomerID -- This will give us all customers + an Order ID if one exists, otherwise it will be null
where Orders.OrderID IS NULL
;
```

```
-- Notes on joining 3 or more tables:
-- You might notice parentheses used in SQL Joins for three or more tables. This is
  because you would want to first join, for example, Table1 and Table2 to produce a
  temporary table. The combined data in that temporary table would then be joined to
  Table3.

-- Joining 3 Tables with Inner Join with Aggregate Function
-- Create a report that shows the order amount totals by customer before any discounts
   were applied.  Sort by the order totals in descending order.
select Customers.CompanyName, sum("Order Details".Quantity * "Order
  Details".UnitPrice) as "Order Amount"
from Customers
Inner Join Orders on Customers.CustomerID = Orders.CustomerID
Inner Join "Order Details" on Orders.OrderID = "Order Details".OrderID
group by Customers.CompanyName
Order by "Order Amount" DESC
;
-- Inner Joined 4 Tables (Show the Company, the Order ID, and the Product that was
  sold to each Company with an order. Sort by Order ID)
select Customers.CompanyName, Orders.OrderID, Products.ProductName
from Customers
Inner Join Orders on Customers.CustomerID = Orders.CustomerID
Inner Join "Order Details" on Orders.OrderID = "Order Details".OrderID
Inner Join Products on "Order Details".ProductID = Products.ProductID
order by orders.OrderID ASC
;

-- ALIASING TABLES! You can Alias Tables to make things more usable.
-- eg. can alias customers table as "c" or orders table as "o".  Very common to use
  one letter.
select c.CompanyName, o.OrderID, o.OrderDate, o.EmployeeID
from Customers as c
Inner Join Orders as o on Customers.CustomerID = Orders.CustomerID

-- UNION JOINS! (Stacks one query's results on top of another's query results) - AS
  LONG AS DATA TYPES MATCH COLUMN FOR COLUMN AND SAME NUMBER OF COLUMNS PER QUERY
-- essentially combines two query results into one result set
-- Example uses Customers table on left and Suppliers table on right.
-- an example of this usage using these two tables would be the creation of a phone
  book of all company names whether they be suppliers or customers.
select CompanyName, ContactName, City, Country, Phone, 'Customers' as "Contact Type"
  -- Last part adds column called Contact Type so you can see if sup or cust
from Customers
Union ALL -- Use if you don't want duplicates.  Leave off "All" if you don't care
  about duplicates.
select CompanyName, ContactName, City, Country, Phone, 'Suppliers' as "Contact Type"
  -- Last part adds column called Contact Type so you can see if sup or cust
from Suppliers
;
-- SELF JOIN (Every now and then you may have to do a self join, that's when you join
  a table to itself to get the results you want)
/* An example of a self join would be when you have an employee table and you're
```

```
  seeing a 'reports to' column, but there's a number, and that number corresponds to
  the EmployeeID, and instead of having to keep scrolling back and forth, you can do a
   self join. You do this by aliasing the table with 2 names*/
select e1.FirstName, e1.LastName, e2.FirstName as Supervisor
from Employees as E1
left join Employees as E2
on E1.EmployeeID = E2.EmployeeID

-- to find the N'th highest salary using a sub-query
-- To find 2nd highest, replace 'n' with 2, for 3rd highest, replace n with 3.
SELECT TOP 1 salary
FROM
    (SELECT DISTINCT TOP n salary
    FROM employees
    ORDER BY salary DESC) as RESULT -- <-- Result = alias for subquery
ORDER BY SALARY ASC

-- ANOTHER WAY TO FIND THE N'TH HIGHEST SALARY USING CTE:
-- CTE (essentialy a small subquery that you can decalare and call)
with findNthHighest
as (select distinct top 3 extension, FirstName
    from Employees order by extension desc)
select top 1 extension from findNthHighest order by Extension
;

-- CTE (COMMON TABLE E PRESSIONS) i.e. temporary result set
with my_cte as
    (select * from Customers where fax IS NULL) --<-- CTE creates a sub query
      temporary result...
Select * from my_cte --<-- ...That can be called here
order by CompanyName

-- VIEWS (Virtual Table based on a SQL statement that has bene given a name, can be
  from one or more tables with aggregates etc)
-- Why would you want to create a view   Hides complex sql.  Saves on work joining
  tables etc. Can set permissions on views.
create view "Top 10 Orders" as --<-- Keywords are "Create View [viewname] as"
with top10_cte as
    (select top 10 orderID, convert(varchar(12), orderdate, 113) as [Order Date],
      Customerid from Orders)
Select * from top10_cte;

-- To call the view from the query window, just call the view name
select *
from [last 10 orders] --<-- View name is [Last 10 Orders]. It is now dynamic so new
  info will always be presented here

-- Another Example of Creation of View and how to call it: creates view of
  discontinued products
create view [Discontinued Product List] as
select ProductID, ProductName
from Products
```

```
where Discontinued = 1;
-- Calling this view:
select * from [Discontinued Product List];

-- CREATING A VIEW WITH SCHEMA BINDING/SCHEMABINDING (You're protecting the source table that the view is based off of)
-- So the source table cannot be modified in any way that will affect the view.  The reason is that you may have a material view
-- but if a change is made to the source table of that view, then the view might not work or be accurate.
-- if you do want to modify the source table, the view must first be modified or dropped.
-- Example:
create view Test
with schemabinding as --<-- This is the material part
select ProductID, QuantityPerUnit
from dbo.Products;
-- Doing this will lock the source table so that if you alter the table like this:
alter table Products
alter column Quantityperunit varchar(60)
-- Then you will get the error message:
    -- Msg 5074, Level 16, State 1, Line 1
    -- The object 'Test' is dependent on column 'Quantityperunit'.
    -- Msg 4922, Level 16, State 9, Line 1
    -- ALTER TABLE ALTER COLUMN Quantityperunit failed because one or more objects access this column.

-- Then to fix this error, you need to enter:
Drop view test; --<-- This drops the view so you can alter the table.

-- ALTERING OR MODIFYING A VIEW (i.e. add in a couple of fields, add sort, add aggregate etc)
-- Use keywords "Alter View [view name] as" and then just restructure view the way you want.
Alter view [discontinued product list] as
select productID, Productname, SupplierID
from Products;

-- DROPPING VIEWS
drop view [discontinued product list]
-- or drop multiple views:
drop view dbo.[discontinued product list], dbo.Test

-- STORED PROCEDURES
-- Allows you to name and store sql scripts that you use often (and can be more than one query)
-- you can also pass parameters through them, to change your results on the fly.
-- "CREATE PROCEDURE (or create proc) [Name of Procedure]  Param as [SQL Script]"
-- To Call or Execute a stored procedure, use "Execute [Name of Procedure]  Param
create procedure CustomerOrders  CustomerID nchar(5) as -- Note nchar(5) is just a data type, I'm specifying 5 characters
select customers.CustomerID, orders.OrderID, orders.OrderDate
```

```
from Customers
inner join orders
on customers.CustomerID = orders.CustomerID
where Customers.CustomerID =  CustomerID; --< this is the filter on the where clause using the parameter you set up at the top

-- To execute this, you would just query:
Execute CustomerOrders 'Queen' --<-- Where 'Queen' is the parameter (will return customer with customer id Queen)
-- Note that if you have multiple parameters, you can just separate the paramters with a comma when you execute.

-- ALTERING STORED PROCEDURES
-- Example using above query (we're going to alter it)
alter procedure CustomerOrders  CustomerID nchar(5),  EmployeeID int as --< added new Param as int for EmployeeID
select customers.CustomerID, orders.OrderID, orders.OrderDate, orders.EmployeeID --<-- Added new Orders.Employee Column
from Customers
inner join orders
on customers.CustomerID = orders.CustomerID
where Customers.CustomerID =  CustomerID
and orders.EmployeeID =  EmployeeID --<-- Added EmployeeID filter via parameter EmployeeID
;
Execute CustomerOrders 'Queen', 4 --<-- This will execute the altered procedure (Note two parameters are sep'd by comma)

-- DROPPING STORED PROCEDURES
drop procedure StoredProcedureName

-- "BEGIN TRAN" "Commit" or "Rollback" Commit means if something fails, don't apply updates.  If you use rollback, then regardless of whether all tests pass, it still rolls back.
-- Note, it takes a lot of time to roll back because it applies all changes as it goes through, and it has to undo all those.

-- LOCKING LOCK (not sure how long it does)
-- I'm busy with this record, no one else can touch it right now.  You might put locks on some records while you work on a transaction.  But you may cause problems (i.e. people waiting).
-- Know but we don't typically use.

-- INDE ING
-- For every table, if you have 20m records, you don't want to have to read them all, so you have an index on customer number which just points to the record number of the customer.  If it can't find an index to use, then it will go through all records, very very slow.  Speeds things up.
-- Records are stored either in arival sequence, or a clustered index in a clustered sequence.
-- Unclustered indexes are just all the other indexes.
-- System looks after indexes automatically after you've defined the indexes.
```

```
-- CREATING TABLES
-- Syntax: (Note, constraints are optional but Column and Datatype are manditory)
CREATE TABLE TableName
(Column DataType Contraint,
Column DataType Constraint,
Column DataType Constraint);
-- Data Types (so many that can be used. PDF was attached from lecture.  A datatype
  tells columns what to expect for memory etc.
-- Constraints (Rules for the data in the columns in your table, eg. is the column a
  required field, primary key field, foreign key field , you can do this at the create
   or the alter stage)
-- Constraint examples ("Not Null" is also a contraint, also "Unique" makes sure each
  row is unique, "Check" is used to validate data, i.e. check that the value in a
  column is what it is supposed to be, e.g. check that only certain countries or dates
   or emploeyes are in the column.  "Default" can be used to specify the default value
   for a column.

-- IDENTIFY KEYWORD (Auto Incrementing Primary Keys)  When you want SQL to generate a
  unique number for a field whenever a new record is inserted into a table, you will
  use the IDENTITY keyword. The IDENTITY keyword auto-increments a number that you
  select, by the value that you select. In the example below, the starting value for
  IDENTITY is 1, and it will increment by 1 for each new record:
CREATE TABLE InventoryList
(
ID INT IDENTITY(1, 1) PRIMARY KEY,
ItemName NVARCHAR(100) NOT NULL,
PurchaseDate DATETIME,
);
-- This this example, the starting value for IDENTITY is 100, and it will increment by
   2 for each new record:
CREATE TABLE InventoryList
(
ID INT IDENTITY(100, 2) PRIMARY KEY,
ItemName NVARCHAR(100) NOT NULL,
PurchaseDate DATETIME,
);
```