SQL 201

(We will be using AdventureWorks database)

1. Views:
   1. This creates a window where the viewer can see selected data without actually making any changes to the main database/table.
   2. Views can be made of one or more than one tables.
   3. Query:

**create view myfirstview as select [Name], [CountryRegionCode] from [Sales].[SalesTerritory] where [Group] like 'North America'**

This creates a view by the name myfirstview. To see contents of this view, user query:

**select \* from myfirstview**

* 1. Combining view from 2 tables:

**create view mysecondview as select [Name], [Group], [SalesQuota],[Bonus] from [Sales].[SalesTerritory] A inner join [Sales].[SalesPerson] B on A.TerritoryID = B.TerritoryID where [Group] like 'North America'**

**select \* from mysecondview**

1. Trigger:
   1. Trigger sends a message or executes a function when a change (insert, update, delete) is attempted to be made to the table.
   2. Example:

Trigger to prevent Inserting value in the table:

--We will create trigger which cancels any addition to the preexisting table

**create trigger trigger1 on [HumanResources].[Shift] after insert as begin print 'insert is not allowed' rollback transaction end**

**go**

--For database level trigger, to prevent creation of table

**create trigger trigger2 on database after create\_table as begin print 'Table creation is not allowed' rollback transaction end**

**go**

1. Computed Column:
   1. These columns are computed based on values of other columns
   2. Example:

Crate Table (eg. with firstname & lastname)

Go to Column Properties

Go to Computed Column Specification

Select the computed column row above

In front of (formula), enter the formula (eg. firstname + “ “ + lastname)

Save the table.

To use this table, right click on table and select edit option & enter values in firstname and lastname. Upon hitting tab on fullname, it will show the concatenated value of first 2 columns.

1. Stored Procedure:
   1. It is a sequence of SQL queries stored inside a variable. Instead of carrying out the entire sequence of SQL statements, you just have to execute the variable.
   2. Parameterised Stored Procedure:

User passes in parameter and the Stored Procedure returns parameter-specific information.

* 1. Stored Procedure can be used to insert, update and delete rows in the table.
  2. Example:

Simple Stored Proceure

**-- Stored Procedures**

**create procedure sample\_stored\_procedure as set nocount on**

**select \* from [HumanResources].[Shift]**

**-- To call stored procedure**

**execute sample\_stored\_procedure**

“set nocount on” doesn’t give the number of rows affected

“set nocount off” gives the number of rows affected

To delete the created procedures, use:

**drop proc sample\_stored\_procedure**

Parameterised Stored Proceure

Type1

**-- Parameterised Stored Procedure**

**create procedure parameterised\_store\_proc @parameter\_name varchar(50)**

**as set nocount on**

**select \* from [HumanResources].[Shift] where name = @parameter\_name**

**--To call the stored procedure with parameter**

**execute parameterised\_store\_proc @parameter\_name = 'Day'**

Type2 – with default parameter given

**-- Parameterised Stored Procedure**

**create procedure parameterised\_store\_proc**

**@parameter\_name varchar(50) = 'Day'**

**as set nocount on**

**select \* from [HumanResources].[Shift] where name = @parameter\_name**

**--To call the stored procedure with parameter**

**execute parameterised\_store\_proc @parameter\_name = 'Night'**

**--To call the stored procedure without parameter (gives data for Day data)**

**execute parameterised\_store\_proc**

**--Returning value from Stored Procedures**

**create procedure returning\_stored\_procedures**

**as return 12**

**declare @variable\_name int**

**execute @variable\_name = returning\_stored\_procedures**

**select @variable\_name**

1. User Defined Functions:
   1. There are 3 types of functions:
      1. Scalar functions – returns a single value
      2. Table-Valued Functions – returns table
      3. System Functions – functions provided by SQL server which can be used for a variety of operations. They cannot be modified.

**--Scalar type function**

**create function ytdslaes()**

**returns money**

**as**

**begin**

**declare @ytdsales money**

**select @ytdsales = sum(SalesYTD) from sales.SalesTerritory**

**return @ytdsales**

**end**

**-- Executing the Function**

**declare @ytdresults as money**

**select @ytdresults = dbo.ytdslaes()**

**print @ytdresults**

**-- Parameterised Function**

**create function ytdgroup (@group varchar(50))**

**returns money**

**as**

**begin**

**declare @ytdsales money**

**select @ytdsales = sum(SalesYTD) from sales.SalesTerritory**

**where [Group] = @group**

**return @ytdsales**

**end**

**-- Executing the Function**

**declare @results money**

**select @results = dbo.ytdgroup ('north america')**

**print @results**

**--Table Valued Function (i.e. returning Table)**

**create function table\_function (@territory\_id int)**

**returns table**

**as return select \* from Sales.SalesTerritory**

**where TerritoryID = @territory\_id**

**--Execute Function**

**select \* from dbo.table\_function(7)**

1. Transactions and Error Handling
   1. ACID properties: Atomicity, Consistency, Isolation, Durability
      1. Atomicity: If the transaction is successful, only then change the databases. Otherwise rollback the transaction. All or nothing.
      2. Consistency: Maintains the database consistency (i.e. keeps it in valid state)
      3. Isolation: Ensures multiple transactions happen serially
      4. Durability: If the transaction has been committed, the changes made in the transaction will remain in the database even in the event of power-loss, crashes or errors.
   2. Examples:

**--Transaction handling**

**begin transaction**

**update [Sales].[SalesTerritory]**

**set CostYTD = 1.00 where [Group] = 'Pacific'**

**commit transaction**

**--Error handling**

**-- error\_num = 0 means success**

**-- error\_num > 0 means error**

**declare @errorresults varchar(50)**

**begin transaction**

**insert into [HumanResources].[Shift] (Name, StartTime, EndTime, ModifiedDate)**

**values ('afternoon', '13:00:00', '21:00:00', '2009-04-01 00:00:00')**

**set @errorresults = @@ERROR**

**if (@errorresults = 0)**

**begin**

**print 'success'**

**commit transaction**

**end**

**else**

**begin**

**print 'failure'**

**rollback transaction**

**end**

**-- Custom Error Message (raise error, gives the error message in red and )**

**declare @errorresults varchar(50)**

**begin transaction**

**insert into [HumanResources].[Shift] (Name, StartTime, EndTime, ModifiedDate)**

**values ('afternoon', '13:00:00', '21:00:00', '2009-04-01 00:00:00')**

**set @errorresults = @@ERROR**

**if (@errorresults = 0)**

**begin**

**print 'success'**

**commit transaction**

**end**

**else**

**begin**

**--raiserror takes 3 parameters: error message, level number and state number**

**RAISERROR ('entry already exists', 11, 8)**

**rollback transaction**

**end**

**--Try and Handle**

**begin try**

**begin transaction**

**insert into [HumanResources].[Shift] (Name, StartTime, EndTime, ModifiedDate)**

**values ('afternoon', '13:00:00', '21:00:00', '2009-04-01 00:00:00')**

**commit transaction**

**end try**

**begin catch**

**print 'catch process entered'**

**rollback transaction**

**end catch**

1. SQL Grouping Functions and Ranking Functions
   1. Common Table Expression – they start with ‘with’ keyword

**--Common Table Expression**

**with cte as (select name, countryregioncode from [Sales].[SalesTerritory])**

**select \* from cte where name like 'north%'**

* 1. Grouping Sets

**--Grouping Sets**

**select \* from [Sales].[SalesTerritory]**

**select sum(salesytd), [group] from [Sales].[SalesTerritory] group by [group]**

**select name, sum(salesytd), [group] from [Sales].[SalesTerritory] group by name, [group]**

**--to avoid adding one column at a time, you use:**

**--Grouping Sets**

**select name, countryregioncode, sum(salesytd), [group] from [Sales].[SalesTerritory]**

**group by grouping sets**

**( (name),(name, countryregioncode), (name, countryregioncode, [group]))**

**--Rollup**

**select name, countryregioncode, sum(salesytd), [group] from [Sales].[SalesTerritory]**

**group by rollup**

**((name, countryregioncode, [group]))**

**--Cube: gives table combination of all column names**

**select name, countryregioncode, sum(salesytd), [group] from [Sales].[SalesTerritory]**

**group by cube**

**( (name),(name, countryregioncode), (name, countryregioncode, [group]))**

* 1. Ranking Method
     1. Types:
        1. Row Number: rank will be 1,2,3,4,5,6,7,8,9,10,11,12
        2. Rank: rank will be 1,2,2,2,5,6,7,8,8,8,8,12
        3. Dense Rank: rank will be 1,2,2,2,3,4,5,6,6,6,6,7
        4. Ntile: rank is split in groups based on the number of units in every group.
     2. Example:

**--Ranking**

**select postalcode,**

**row\_number() over (order by postalcode) as 'row\_number',**

**rank() over (order by postalcode) as 'rank\_number',**

**dense\_rank() over (order by postalcode) as 'dense\_row\_number',**

**ntile(10) over (order by postalcode) as 'ntile\_number'**

**from [Person].[Address] where postalcode in ('98011','98020','98055','94010')**

1. XML – Extensible Markup Language
   1. It is data represented between tags.
   2. Examples:

**--Working with XML**

**--1. Creating xml from table. It returns a hyperlink for xml**

**--auto**

**select \* from [Sales].[SalesTerritory]**

**for xml auto, elements, root ('SalesTerritory')**

**--raw**

**select \* from [Sales].[SalesTerritory]**

**for xml raw, elements, root ('SalesTerritory')**

**--2.Creating table from xml**

**declare @xmlhandle int**

**declare @xmldocument xml**

**set @xmldocument = (select \* from [Sales].[SalesTerritory] for xml auto, elements, root ('SalesTerritory'))**

**execute sp\_xml\_preparedocument @xmlhandle output, @xmldocument**

**select \* from openxml (@xmlhandle, '/SalesTerritory/Sales.SalesTerritory',2) with (TerritoryID int, SalesYTD money)**

1. SQL Partition:
   1. Splitting table into partitions depending on data-size. User can define the partition condition.
   2. Not possible from SQL express. It is possible only via SQL enterprise edition.’
2. Dynamic Queries:
   1. Used when names of columns is not known. Usually used when pivot table is in consideration.
   2. In the query part, you can keep some part constant and only edit the remaining part.
   3. Example:

**--Pivot & Dynamic Queries**

**declare @sqlstring varchar(500)**

**set @sqlstring = 'select countryregioncode, [group], '**

**set @sqlstring = @sqlstring + 'salesytd from [Sales].[SalesTerritory]'**

**print @sqlstring**

**execute (@sqlstring)**

1. SQL Filestream and Free Text Search
2. SQL Geography and Geometry datatype
   1. Geometric: Helps plot 2D geometric shapes based on coordinates
   2. Geographic: Helps plot 3D geometric shapes based on coordinates on map.