```
/****************/
/* SQL SERVER   */
/*    INDEX     */
/****************/
```

--96-- Logon triggers in sql server

In this video we will discuss Logon triggers in SQL Server.

As the name implies Logon triggers fire in response to a LOGON event.
Logon triggers fire after the authentication phase of logging in finishes, but before the user session is actually established.

Logon triggers can be used for
1. Tracking login activity
2. Restricting logins to SQL Server
3. Limiting the number of sessions for a specific login

Logon trigger example : The following trigger limits the maximum number of open connections for a user to 3.

```
CREATE TRIGGER tr_LogonAuditTriggers
ON ALL SERVER
FOR LOGON
```

```
AS
BEGIN
   DECLARE @LoginName NVARCHAR(100)

   Set @LoginName = ORIGINAL_LOGIN()

   IF (SELECT COUNT(*) FROM sys.dm_exec_sessions
       WHERE is_user_process = 1
       AND original_login_name = @LoginName) > 3
   BEGIN
       Print 'Fourth connection of ' + @LoginName + ' blocked'
       ROLLBACK
   END
END
```

An attempt to make a fourth connection, will be blocked.
logon triggers in sql server

The trigger error message will be written to the error log. Execute the following command to read the error log.
Execute sp_readerrorlog

logon trigger example sql server

--97-- Select into in sql server

We will be using the following 2 tables for the examples.
departments table   employees table

SQL Script to create Departments and Employees tables

```
Create table Departments
(
   DepartmentId int primary key,
   DepartmentName nvarchar(50)
)
Go

Insert into Departments values (1, 'IT')
Insert into Departments values (2, 'HR')
Insert into Departments values (3, 'Payroll')
Go

Create table Employees
(
   Id int primary key,
   Name nvarchar(100),
   Gender nvarchar(10),
   Salary int,
    DeptId int foreign key references Departments(DepartmentId)
)
Go

Insert into Employees values (1, 'Mark', 'Male', 50000, 1)
Insert into Employees values (2, 'Sara', 'Female', 65000, 2)
Insert into Employees values (3, 'Mike', 'Male', 48000, 3)
Insert into Employees values (4, 'Pam', 'Female', 70000, 1)
Insert into Employees values (5, 'John', 'Male', 55000, 2)
Go
```

The SELECT INTO statement in SQL Server, selects data from one table and inserts it into a new table.

SELECT INTO statement in SQL Server can do the following
1. Copy all rows and columns from an existing table into a new table.
This is extremely useful when you want to make a backup copy of the existing table.

```
SELECT * INTO EmployeesBackup FROM Employees
```

2. Copy all rows and columns from an existing table into a new table in an external database.

```
SELECT * INTO HRDB.dbo.EmployeesBackup FROM Employees
```

3. Copy only selected columns into a new table

```
SELECT Id, Name, Gender INTO EmployeesBackup FROM Employees
```

4. Copy only selected rows into a new table

```
SELECT * INTO EmployeesBackup FROM Employees WHERE DeptId = 1
```

5. Copy columns from 2 or more table into a new table

```
SELECT * INTO EmployeesBackup
FROM Employees
INNER JOIN Departments
ON Employees.DeptId = Departments.DepartmentId
```

6. Create a new table whose columns and datatypes match with an existing table.

```
SELECT * INTO EmployeesBackup FROM Employees WHERE 1 <> 1
```

7. Copy all rows and columns from an existing table into a new table on a different
SQL Server instance. For this, create a linked server and use the 4 part naming convention

```
SELECT * INTO TargetTable
FROM [SourceServer].[SourceDB].[dbo].[SourceTable]
```

Please note : You cannot use SELECT INTO statement to select data into an existing table.
For this you will have to use INSERT INTO statement.

```
INSERT INTO ExistingTable (ColumnList)
SELECT ColumnList FROM SourceTable
```

--98-- Difference between where and having in sql server

In this video we will discuss the difference between where and having clauses in SQL Server.

Let us understand the difference with an example. For the examples in this video we will use the following Sales table.
sales table

SQL Script to create and populate Sales table with test data

```
Create table Sales
(
   Product nvarchar(50),
   SaleAmount int
)
Go

Insert into Sales values ('iPhone', 500)
Insert into Sales values ('Laptop', 800)
Insert into Sales values ('iPhone', 1000)
Insert into Sales values ('Speakers', 400)
Insert into Sales values ('Laptop', 600)
Go
```

To calculate total sales by product, we would write a GROUP BY query as shown below

```
SELECT Product, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY Product
```

The above query produces the following result
sql server group by example

Now if we want to find only those products where the total sales amount is greater than $1000, we will use HAVING clause to filter products

```
SELECT Product, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY Product
HAVING SUM(SaleAmount) > 1000
```

Result :
sql server group by having example

If we use WHERE clause instead of HAVING clause, we will get a syntax error.
This is because the WHERE clause doesn’t work with aggregate functions like sum, min, max, avg, etc.

```
SELECT Product, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY Product
WHERE SUM(SaleAmount) > 1000
```

So in short, the difference is WHERE clause cannot be used with aggregates where as HAVING can.

However, there are other differences as well that we need to keep in mind when using WHERE

and HAVING clauses. WHERE clause filters rows before aggregate calculations are performed
where as HAVING clause filters rows after aggregate calculations are performed.
Let us understand this with an example.

Total sales of iPhone and Speakers can be calculated by using either WHERE or HAVING clause

Calculate Total sales of iPhone and Speakers using WHERE clause : In this example the WHERE
clause retrieves only iPhone and Speaker products and then performs the sum.

```
SELECT Product, SUM(SaleAmount) AS TotalSales
FROM Sales
WHERE Product in ('iPhone', 'Speakers')
GROUP BY Product
```

Result :
sql server group by with where clause

Calculate Total sales of iPhone and Speakers using HAVING clause : This example retrieves
all rows from Sales table, performs the sum and then removes all products except
iPhone and Speakers.

```
SELECT Product, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY Product
HAVING Product in ('iPhone', 'Speakers')
```

Result :
sql server group by with having

So from a performance standpoint, HAVING is slower than WHERE
and should be avoided when possible.

Another difference is WHERE comes before GROUP BY and HAVING
comes after GROUP BY.

Difference between WHERE and Having
1. WHERE clause cannot be used with aggregates where as HAVING can. This means WHERE clause
is used for filtering individual rows where as HAVING clause is used to filter groups.

2. WHERE comes before GROUP BY. This means WHERE clause filters rows before aggregate
calculations are performed. HAVING comes after GROUP BY. This means HAVING clause filters
rows after aggregate calculations are performed. So from a performance standpoint,
HAVING is slower than WHERE and should be avoided when possible.

3. WHERE and HAVING can be used together in a SELECT query. In this case WHERE clause is
applied first to filter individual rows. The rows are then grouped and aggregate calculations
are performed, and then the HAVING clause filters the groups.

--99-- Table valued parameters in SQL Server

In this video we will discuss table valued parameters in SQL Server.

Table Valued Parameter is a new feature introduced in SQL SERVER 2008. Table Valued Parameter allows a table
(i.e multiple rows of data) to be passed as a parameter to a stored procedure from T-SQL code or from an
application. Prior to SQL SERVER 2008, it is not possible to pass a table variable as a parameter to a stored procedure.

Let us understand how to pass multiple rows to a stored procedure using Table Valued Parameter with an example.
We want to insert multiple rows into the following Employees table. At the moment this table does not have any rows.
employees table

SQL Script to create the Employees table

```
Create Table Employees
(
   Id int primary key,
   Name nvarchar(50),
   Gender nvarchar(10)
)
Go
```

Step 1 : Create User-defined Table Type

```
CREATE TYPE EmpTableType AS TABLE
(
   Id INT PRIMARY KEY,
   Name NVARCHAR(50),
   Gender NVARCHAR(10)
)
Go
```

Step 2 : Use the User-defined Table Type as a parameter in the stored procedure. Table valued parameters must be passed
as read-only to stored procedures, functions etc. This means you cannot perform DML operations like INSERT, UPDATE or DELETE on a table-valued parameter in the body of a function, stored procedure etc.

```
CREATE PROCEDURE spInsertEmployees
@EmpTableType EmpTableType READONLY
AS
BEGIN
   INSERT INTO Employees
   SELECT * FROM @EmpTableType
END
```

Step 3 : Declare a table variable, insert the data and then pass the table variable as a parameter to the stored procedure.

```
DECLARE @EmployeeTableType EmpTableType

INSERT INTO @EmployeeTableType VALUES (1, 'Mark', 'Male')
INSERT INTO @EmployeeTableType VALUES (2, 'Mary', 'Female')
INSERT INTO @EmployeeTableType VALUES (3, 'John', 'Male')
INSERT INTO @EmployeeTableType VALUES (4, 'Sara', 'Female')
INSERT INTO @EmployeeTableType VALUES (5, 'Rob', 'Male')

EXECUTE spInsertEmployees @EmployeeTableType
```

Thats it. Now select the data from Employees table and notice that all the rows of the table variable are inserted
into the Employees table. table valued parameters example

In our next video, we will discuss how to pass table as a parameter to the stored procedure from an ADO.NET application

--100-- Send datatable as parameter to stored procedure

In this video we will discuss how to send datatable as parameter to stored procedure.
This is continuation to Part 99. Please watch Part 99 from SQL Server tutorial before proceeding.

In Part 99, we discussed creating a stored procedure that accepts a table as a parameter.
In this video we will discuss how to pass a datatable from a web application to the SQL Server stored procedure.

Here is what we want to do.
1. Design a webform that looks as shown below. This form allows us to insert 5 employees at a time into the database table.
Send datatable as parameter to stored procedure

2. When "Insert Employees" button is clicked, retrieve the from data into a datatabe and then pass the datatable
as a parameter to the stored procedure.

3. The stored procedure will then insert all the rows into the Employees table in the database.

Here are the steps to achieve this.
Step 1 : Create new asp.net web application project. Name it Demo.

Step 2 : Include a connection string in the web.config file to your database.
```
<add name="DBCS"
     connectionString="server=.;database=SampleDB;integrated security=SSPI"/>
```

Step 3 : Copy and paste the following HTML in WebForm1.aspx
```
<asp:Button ID="btnFillDummyData" runat="server" Text="Fill Dummy Data"
   OnClick="btnFillDummyData_Click" />
```

```
<br /><br />
<table>
   <tr>
      <td>
         ID : <asp:TextBox ID="txtId1" runat="server"></asp:TextBox>
      </td>
      <td>
         Name : <asp:TextBox ID="txtName1" runat="server"></asp:TextBox>
      </td>
      <td>
         Gender : <asp:TextBox ID="txtGender1" runat="server"></asp:TextBox>
      </td>
   </tr>
   <tr>
      <td>
         ID : <asp:TextBox ID="txtId2" runat="server"></asp:TextBox>
      </td>
      <td>
         Name : <asp:TextBox ID="txtName2" runat="server"></asp:TextBox>
      </td>
      <td>
         Gender : <asp:TextBox ID="txtGender2" runat="server"></asp:TextBox>
      </td>
   </tr>
   <tr>
      <td>
         ID : <asp:TextBox ID="txtId3" runat="server"></asp:TextBox>
      </td>
      <td>
         Name : <asp:TextBox ID="txtName3" runat="server"></asp:TextBox>
      </td>
      <td>
         Gender : <asp:TextBox ID="txtGender3" runat="server"></asp:TextBox>
      </td>
   </tr>
   <tr>
      <td>
         ID : <asp:TextBox ID="txtId4" runat="server"></asp:TextBox>
      </td>
      <td>
         Name : <asp:TextBox ID="txtName4" runat="server"></asp:TextBox>
      </td>
      <td>
         Gender : <asp:TextBox ID="txtGender4" runat="server"></asp:TextBox>
      </td>
   </tr>
   <tr>
      <td>
         ID : <asp:TextBox ID="txtId5" runat="server"></asp:TextBox>
```

```
        </td>
        <td>
            Name : <asp:TextBox ID="txtName5" runat="server"></asp:TextBox>
        </td>
        <td>
            Gender : <asp:TextBox ID="txtGender5" runat="server"></asp:TextBox>
        </td>
    </tr>
</table>
<br />
<asp:Button ID="btnInsert" runat="server" Text="Insert Employees"
    OnClick="btnInsert_Click" />
```

Step 4 : Copy and paste the following code in the code-behind file

```
using System;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;

namespace Demo
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        { }

        private DataTable GetEmployeeData()
        {
            DataTable dt = new DataTable();
            dt.Columns.Add("Id");
            dt.Columns.Add("Name");
            dt.Columns.Add("Gender");

            dt.Rows.Add(txtId1.Text, txtName1.Text, txtGender1.Text);
            dt.Rows.Add(txtId2.Text, txtName2.Text, txtGender2.Text);
            dt.Rows.Add(txtId3.Text, txtName3.Text, txtGender3.Text);
            dt.Rows.Add(txtId4.Text, txtName4.Text, txtGender4.Text);
            dt.Rows.Add(txtId5.Text, txtName5.Text, txtGender5.Text);

            return dt;
        }

        protected void btnInsert_Click(object sender, EventArgs e)
        {
            string cs = ConfigurationManager.ConnectionStrings["DBCS"].ConnectionString;
            using (SqlConnection con = new SqlConnection(cs))
            {
                SqlCommand cmd = new SqlCommand("spInsertEmployees", con);
                cmd.CommandType = CommandType.StoredProcedure;
                SqlParameter paramTVP = new SqlParameter()
                {
                    ParameterName = "@EmpTableType",
                    Value = GetEmployeeData()
                };
                cmd.Parameters.Add(paramTVP);

                con.Open();
                cmd.ExecuteNonQuery();
                con.Close();
            }
        }

        protected void btnFillDummyData_Click(object sender, EventArgs e)
        {
            txtId1.Text = "1";
            txtId2.Text = "2";
            txtId3.Text = "3";
            txtId4.Text = "4";
            txtId5.Text = "5";

            txtName1.Text = "John";
            txtName2.Text = "Mike";
            txtName3.Text = "Sara";
            txtName4.Text = "Pam";
            txtName5.Text = "Todd";

            txtGender1.Text = "Male";
            txtGender2.Text = "Male";
            txtGender3.Text = "Female";
            txtGender4.Text = "Female";
            txtGender5.Text = "Male";
        }
    }
}
```

--101-- Grouping Sets in SQL Server

Grouping sets is a new feature introduced in SQL Server 2008. Let us understand Grouping sets with an example.

We will be using the following Employees table for the examples in this video.
grouping sets examples in sql server

SQL Script to create and populate Employees table

```
Create Table Employees
(
    Id int primary key,
    Name nvarchar(50),
    Gender nvarchar(10),
    Salary int,
    Country nvarchar(10)
)
Go

Insert Into Employees Values (1, 'Mark', 'Male', 5000, 'USA')
Insert Into Employees Values (2, 'John', 'Male', 4500, 'India')
Insert Into Employees Values (3, 'Pam', 'Female', 5500, 'USA')
Insert Into Employees Values (4, 'Sara', 'Female', 4000, 'India')
Insert Into Employees Values (5, 'Todd', 'Male', 3500, 'India')
Insert Into Employees Values (6, 'Mary', 'Female', 5000, 'UK')
Insert Into Employees Values (7, 'Ben', 'Male', 6500, 'UK')
Insert Into Employees Values (8, 'Elizabeth', 'Female', 7000, 'USA')
Insert Into Employees Values (9, 'Tom', 'Male', 5500, 'UK')
Insert Into Employees Values (10, 'Ron', 'Male', 5000, 'USA')
Go
```

We want to calculate Sum of Salary by Country and Gender. The result should be as shown below.
microsoft sql server group by example

We can very easily achieve this using a Group By query as shown below

```
Select Country, Gender, Sum(Salary) as TotalSalary
From Employees
Group By Country, Gender
```

Within the same result set we also want Sum of Salary just by Country.
The Result should be as shown below. Notice that Gender column within the resultset is
NULL as we are grouping only by Country column
group by union all sql server

To achieve the above result we could combine 2 Group By queries using UNION ALL as shown below.

```
Select Country, Gender, Sum(Salary) as TotalSalary
From Employees
Group By Country, Gender

UNION ALL

Select Country, NULL, Sum(Salary) as TotalSalary
From Employees
Group By Country
```

Within the same result set we also want Sum of Salary just by Gender. The Result should be as shown below.
Notice that the Country column within the resultset is NULL as we are grouping only by Gender column.
sql server union group by sum

We can achieve this by combining 3 Group By queries using UNION ALL as shown below

```
Select Country, Gender, Sum(Salary) as TotalSalary
From Employees
Group By Country, Gender

UNION ALL

Select Country, NULL, Sum(Salary) as TotalSalary
From Employees
Group By Country

UNION ALL

Select NULL, Gender, Sum(Salary) as TotalSalary
From Employees
Group By Gender
```

Finally we also want the grand total of Salary. In this case we are not grouping on any particular column.
So both Country and Gender columns will be NULL in the resultset.
sql server 2008 union group by

To achieve this we will have to combine the fourth query using UNION ALL as shown below.

```
Select Country, Gender, Sum(Salary) as TotalSalary
From Employees
Group By Country, Gender

UNION ALL

Select Country, NULL, Sum(Salary) as TotalSalary
From Employees
Group By Country

UNION ALL

Select NULL, Gender, Sum(Salary) as TotalSalary
From Employees
Group By Gender

UNION ALL

Select NULL, NULL, Sum(Salary) as TotalSalary
From Employees
```

There are 2 problems with the above approach.
1. The query is huge as we have combined different Group By queries using UNION ALL operator.

This can grow even more if we start to add more groups
2. The Employees table has to be accessed 4 times, once for every query.

If we use Grouping Sets feature introduced in SQL Server 2008, the amount of T-SQL code that you have to write will be greatly reduced. The following Grouping Sets query produce the same result as the above UNION ALL query.

```
Select Country, Gender, Sum(Salary) TotalSalary
From Employees
Group BY
    GROUPING SETS
    (
        (Country, Gender), -- Sum of Salary by Country and Gender
        (Country),          -- Sum of Salary by Country
        (Gender) ,          -- Sum of Salary by Gender
        ()                  -- Grand Total
    )
```

Output of the above query
sql server 2008 grouping sets example

The order of the rows in the result set is not the same as in the case of UNION ALL query.
To control the order use order by as shown below.

```
Select Country, Gender, Sum(Salary) TotalSalary
From Employees
Group BY
    GROUPING SETS
    (
        (Country, Gender), -- Sum of Salary by Country and Gender
        (Country),          -- Sum of Salary by Country
        (Gender) ,          -- Sum of Salary by Gender
        ()                  -- Grand Total
    )
Order By Grouping(Country), Grouping(Gender), Gender
```

Output of the above query
sql server grouping sets order by grouping

--102-- ROLLUP IN SQL SERVER

ROLLUP in SQL Server is used to do aggregate operation on multiple levels in hierarchy.

Let us understand Rollup in SQL Server with examples. We will use the following Employees table for the examples in this video.
Employees Table

Retrieve Salary by country along with grand total
sql server group by with rollup

There are several ways to achieve this. The easiest way is by using Rollup with Group By.

```
SELECT Country, SUM(Salary) AS TotalSalary
FROM Employees

GROUP BY ROLLUP(Country)
```

The above query can also be rewritten as shown below

```
SELECT Country, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Country WITH ROLLUP
```

We can also use UNION ALL operator along with GROUP BY

```
SELECT Country, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Country

UNION ALL

SELECT NULL, SUM(Salary) AS TotalSalary
FROM Employees
```

We can also use Grouping Sets to achieve the same result

```
SELECT Country, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY GROUPING SETS
(
   (Country),
   ()
)
```

Lets look at another example.

Group Salary by Country and Gender. Also compute the Subtotal for Country level and Grand Total as shown below.
sql server 2008 group by with rollup

Using ROLLUP with GROUP BY

```
SELECT Country, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY ROLLUP(Country, Gender)

--OR

SELECT Country, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Country, Gender WITH ROLLUP
```

Using UNION ALL with GROUP BY

```
SELECT Country, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Country, Gender

UNION ALL

SELECT Country, NULL, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Country

UNION ALL

SELECT NULL, NULL, SUM(Salary) AS TotalSalary
FROM Employees
```

Using GROUPING SETS

```
SELECT Country, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY GROUPING SETS
(
    (Country, Gender),
    (Country),
    ()
)
```

--103-- Cube in SQL Server

Cube() in SQL Server produces the result set by generating all combinations of columns specified in GROUP BY CUBE().

Let us understand Cube() in SQL Server with examples. We will use the following Employees table for the examples in this video.
Employees Table

Write a query to retrieve Sum of Salary grouped by all combinations of the following 2 columns as well as Grand Total.
Country,
Gender

The output of the query should be as shown below
sql server group by with cube

Using Cube with Group By

```
SELECT Country, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Cube(Country, Gender)
```

```
--OR

SELECT Country, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Country, Gender with Cube
```

The above query is equivalent to the following Grouping Sets query

```
SELECT Country, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY
   GROUPING SETS
   (
       (Country, Gender),
       (Country),
       (Gender),
       ()
   )
```

The above query is equivalent to the following UNION ALL query. While the data in the result set is the same, the ordering is not. Use ORDER BY to control the ordering of rows in the result set.

```
SELECT Country, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Country, Gender

UNION ALL

SELECT Country, NULL, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Country

UNION ALL

SELECT NULL, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Gender

UNION ALL

SELECT NULL, NULL, SUM(Salary) AS TotalSalary
FROM Employees
```

--104-- Difference Between Cube and Rollup

In this video we will discuss the difference between cube and rollup in SQL Server.

CUBE generates a result set that shows aggregates for all combinations of values in

the selected columns, where as ROLLUP generates a result set that shows aggregates
for a hierarchy of values in the selected columns.

Let us understand this difference with an example. Consider the following Sales table.
difference between cube and rollup in SQL Server

SQL Script to create and populate Sales table

```
Create table Sales
(
   Id int primary key identity,
   Continent nvarchar(50),
   Country nvarchar(50),
   City nvarchar(50),
   SaleAmount int
)
Go

Insert into Sales values('Asia','India','Bangalore',1000)
Insert into Sales values('Asia','India','Chennai',2000)
Insert into Sales values('Asia','Japan','Tokyo',4000)
Insert into Sales values('Asia','Japan','Hiroshima',5000)
Insert into Sales values('Europe','United Kingdom','London',1000)
Insert into Sales values('Europe','United Kingdom','Manchester',2000)
Insert into Sales values('Europe','France','Paris',4000)
Insert into Sales values('Europe','France','Cannes',5000)
Go
```

ROLLUP(Continent, Country, City) produces Sum of Salary for the following hierarchy
Continent, Country, City
Continent, Country,
Continent
()

CUBE(Continent, Country, City) produces Sum of Salary for all the following column combinations
Continent, Country, City
Continent, Country,
Continent, City
Continent
Country, City
Country,
City
()

```
SELECT Continent, Country, City, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
```

difference between cube and rollup in sql server 2005

```
SELECT Continent, Country, City, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY CUBE(Continent, Country, City)
```

difference between cube and rollup in sql server 2008

You wont see any difference when you use ROLLUP and CUBE on a single column. Both the following queries produces the same output.

```
SELECT Continent, Sum(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ROLLUP(Continent)

-- OR

SELECT Continent, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY CUBE(Continent)
```

sql server rollup vs cube

--105-- Grouping Funcion in SQL Server

This is continuation to Part 104. Please watch Part 104 from SQL Server tutorial before proceeding.
We will use the following Sales table for this example.
sql server rollup grouping

What is Grouping function
Grouping(Column) indicates whether the column in a GROUP BY list is aggregated or not. Grouping returns 1 for
aggregated or 0 for not aggregated in the result set.

The following query returns 1 for aggregated or 0 for not aggregated in the result set

```
SELECT   Continent, Country, City, SUM(SaleAmount) AS TotalSales,
         GROUPING(Continent) AS GP_Continent,
         GROUPING(Country) AS GP_Country,
         GROUPING(City) AS GP_City
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
```

Result :
Grouping function in SQL Server

What is the use of Grouping function in real world
When a column is aggregated in the result set, the column will have a NULL value.
If you want to replace NULL with All then this GROUPING function is very handy.

```
SELECT
```

```
   CASE WHEN
       GROUPING(Continent) = 1 THEN 'All' ELSE ISNULL(Continent, 'Unknown')
   END AS Continent,
   CASE WHEN
       GROUPING(Country) = 1 THEN 'All' ELSE ISNULL(Country, 'Unknown')
   END AS Country,
   CASE
       WHEN GROUPING(City) = 1 THEN 'All' ELSE ISNULL(City, 'Unknown')
   END AS City,
   SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
```

Result :
sql server rollup replace null

Cant I use ISNULL function instead as shown below

```
SELECT   ISNULL(Continent, 'All') AS Continent,
         ISNULL(Country, 'All') AS Country,
         ISNULL(City, 'All') AS City,
         SUM(SaleAmount) AS TotalSales
FROM Sales

GROUP BY ROLLUP(Continent, Country, City)
```

Well, you can, but only if your data does not contain NULL values.
Let me explain what I mean.

At the moment the raw data in our Sales has no NULL values.
Lets introduce a NULL value in the City column of the row where Id = 1

```
Update Sales Set City = NULL where Id = 1
```

Now execute the following query with ISNULL function

```
SELECT   ISNULL(Continent, 'All') AS Continent,
         ISNULL(Country, 'All') AS Country,
         ISNULL(City, 'All') AS City,
         SUM(SaleAmount) AS TotalSales
FROM Sales

GROUP BY ROLLUP(Continent, Country, City)
```

Result : Notice that the actuall NULL value in the raw data is also replaced with the word 'All', which is incorrect. Hence the need for Grouping function.
sql server rollup grouping replace null

Please note : Grouping function can be used with Rollup, Cube and Grouping Sets

--106-- GROUPING ID FUNCION IN SQL SERVER

In this video we will discuss
1. GROUPING_ID function in SQL Server
2. Difference between GROUPING and GROUPING_ID functions
3. Use of GROUPING_ID function

GROUPING_ID function computes the level of grouping.

Difference between GROUPING and GROUPING_ID

Syntax : GROUPING function is used on single column, where as the column list for GROUPING_ID function must match with GROUP BY column list.

```
GROUPING(Col1)
GROUPING_ID(Col1, Col2, Col3,...)
```

GROUPING indicates whether the column in a GROUP BY list is aggregated or not.
Grouping returns 1 for aggregated or 0 for not aggregated in the result set.

GROUPING_ID() function concatenates all the GOUPING() functions, perform the binary to decimal conversion,
and returns the equivalent integer. In short
GROUPING_ID(A, B, C) = GROUPING(A) + GROUPING(B) + GROUPING(C)

Let us understand this with an example.

```
SELECT   Continent, Country, City, SUM(SaleAmount) AS TotalSales,
        CAST(GROUPING(Continent) AS NVARCHAR(1)) +
        CAST(GROUPING(Country) AS NVARCHAR(1)) +
        CAST(GROUPING(City) AS NVARCHAR(1)) AS Groupings,
        GROUPING_ID(Continent, Country, City) AS GPID
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
```

Query result :

Row Number 1 : Since the data is not aggregated by any column GROUPING(Continent), GROUPING(Country) and GROUPING(City) return 0
and as result we get a binar string with all ZEROS (000). When this converted to decimal we get 0 which is displayed in GPID column.

Row Number 7 : The data is aggregated for Country and City columns, so GROUPING(Country) and GROUPING(City) return 1 where as
GROUPING(Continent) return 0. As result we get a binar string (011). When this converted to decimal we get 10 which is displayed in
GPID column.

Row Number 15 : This is the Grand total row. Notice in this row the data is aggregated by all the 3 columns. Hence all the 3 GROUPING
functions return 1. So we get a binary string with all ONES (111). When this converted to decimal we get 7 which is displayed in GPID column.

Use of GROUPING_ID function : GROUPING_ID function is very handy if you want to sort and filter by level of grouping.

Sorting by level of grouping :

```
SELECT   Continent, Country, City, SUM(SaleAmount) AS TotalSales,
         GROUPING_ID(Continent, Country, City) AS GPID
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
ORDER BY GPID
```

Result :

Filter by level of grouping : The following query retrieves only continent level aggregated data

```
SELECT   Continent, Country, City, SUM(SaleAmount) AS TotalSales,
         GROUPING_ID(Continent, Country, City) AS GPID
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
HAVING GROUPING_ID(Continent, Country, City) = 3
```

Result :

--107-- Debugging SQL Server Stored Procedures

In this video we will discuss how to debug stored procedures in SQL Server.

Setting up the Debugger in SSMS : If you have connected to SQL Server using (local) or . (period), and when you start the debugger you will get the following error
Unable to start T-SQL Debugging. Could not connect to computer.
unable to start t-sql debugging. could not connect to computer

To fix this error, use the computer name to connect to the SQL Server instead of using (local) or .
debugging in ssms

For the examples in this video we will be using the following stored procedure.

```
Create procedure spPrintEvenNumbers
@Target int
as
Begin
    Declare @StartNumber int
    Set @StartNumber = 1

    while(@StartNumber < @Target)
    Begin
        If(@StartNumber%2 = 0)
        Begin
            Print @StartNumber
        End
        Set @StartNumber = @StartNumber + 1
    End
    Print 'Finished printing even numbers till ' + RTRIM(@Target)
End
```

Connect to SQL Server using your computer name, and then execute the above code to create the stored procedure. At this point, open a New Query window. Copy and paste the following T-SQL code to execute the stored procedure.

```
DECLARE @TargetNumber INT
SET @TargetNumber = 10
EXECUTE spPrintEvenNumbers @TargetNumber
Print 'Done'
```

Starting the Debugger in SSMS : There are 2 ways to start the debugger
1. In SSMS, click on the Debug Menu and select Start Debugging
start debugging in sql server 2008

2. Use the keyboard shortcut ALT + F5

At this point you should have the debugger running. The line that is about to be executed is marked with an yellow arrow debugging sql queries

Step Over, Step into and Step Out in SSMS : You can find the keyboard shortcuts in the Debug menu in SSMS.
difference between step into step over and step out

Let us understand what Step Over, Step into and Step Out does when debugging the following piece of code
difference between step into and step over in debugging

1. There is no difference when you STEP INTO (F11) or STEP OVER (F10) the code on LINE 2

2. On LINE 3, we are calling a Stored Procedure. On this statement if we press F10 (STEP OVER), it wont give us the opportunity to debug the stored procedure code. To be able to debug the stored procedure code you will have to STEP INTO it by pressing F11.

3. If the debugger is in the stored procedure, and you dont want to debug line by line with in that stored procedure, you can STEP OUT of it by pressing SHIFT + F11. When you do this, the debugger completes the execution of the stored procedure and waits on the next line in the main query, i.e on LINE 4 in this example.

To stop debugging : There are 2 ways to stop debugging
1. In SSMS, click on the Debug Menu and select Stop Debugging
2. Use the keyboard shortcut SHIFT + F5

Show Next Statement shows the next statement that the debugger is about to execute.
Run to Cursor command executes all the statements in a batch up to the current cursor position
run to cursor in ssms

Locals Window in SSMS : Displays the current values of variables and parameters
Locals window in SSMS

If you cannot see the locals window or if you have closed it and if you want to open it,
you can do so using the following menu option. Locals window is only available if you are in DEBUG mode.
view locals window in ssms

Watch Window in SSMS : Just like Locals window, Watch window is used to watch the values of variables.
You can add and remove variables from the watch window. To add a variable to the Watch Window, right click on the variable and select "Add Watch" option from the context menu.
Watch Window in SSMS

Call Stack Window in SSMS : Allows you to navigate up and down the call stack to see what values your
application is storing at different levels. Its an invaluable tool for determining why your code is doing what its doing.
Call Stack Window in SSMS

Immediate Window in SSMS : Very helpful during debugging to evaluate expressions, and print variable values.
To clear immediate window type >cls and press enter.
Immediate Window in SSMS

Breakpoints in SSMS : There are 2 ways to set a breakpoint in SSMS.
1. By clicking on the grey margin on the left hand side in SSMS (to remove click again)
2. By pressing F9 (to remove press F9 again)

Enable, Disable or Delete all breakpoints : There are 2 ways to Enable, Disable or Delete all breakpoints

1. From the Debug menu
disable all breakpoints in ssms

2. From the Breakpoints window. To view Breakpoints window select Debug => Windows => Breakpoints or
use the keyboard shortcut ALT + CTRL + B
view breakpoints window in ssms

Conditional Breakpoint : Conditional Breakpoints are hit only when the specified condition is met.

These are extremely useful when you have some kind of a loop and you want to break, only when the loop
variable has a specific value (For example loop varible = 100).

How to set a conditional break point in SSMS :
1. Right click on the Breakpoint and select Condition from the context menu
how to set conditional breakpoint in ssms

2. In the Breakpoint window specify the condition
setting a conditional breakpoint ssms

--108-- Over Clause in SQL Server

In this video we will discuss the power and use of Over clause in SQL Server.

The OVER clause combined with PARTITION BY is used to break up data into partitions.
Syntax : function (...) OVER (PARTITION BY col1, Col2, ...)

The specified function operates for each partition.

For example :
COUNT(Gender) OVER (PARTITION BY Gender) will partition the data by GENDER i.e there will 2 partitions (Male and Female) and then the COUNT() function is applied over each partition.

Any of the following functions can be used. Please note this is not the complete list.
COUNT(), AVG(), SUM(), MIN(), MAX(), ROW_NUMBER(), RANK(), DENSE_RANK() etc.

Example : We will use the following Employees table for the examples in this video.
over clause in sql server

SQl Script to create Employees table
```
Create Table Employees
(
    Id int primary key,
    Name nvarchar(50),
    Gender nvarchar(10),
    Salary int
)
Go

Insert Into Employees Values (1, 'Mark', 'Male', 5000)
Insert Into Employees Values (2, 'John', 'Male', 4500)
Insert Into Employees Values (3, 'Pam', 'Female', 5500)
Insert Into Employees Values (4, 'Sara', 'Female', 4000)
Insert Into Employees Values (5, 'Todd', 'Male', 3500)
Insert Into Employees Values (6, 'Mary', 'Female', 5000)
Insert Into Employees Values (7, 'Ben', 'Male', 6500)
Insert Into Employees Values (8, 'Jodi', 'Female', 7000)
Insert Into Employees Values (9, 'Tom', 'Male', 5500)
Insert Into Employees Values (10, 'Ron', 'Male', 5000)
Go
```

Write a query to retrieve total count of employees by Gender. Also in the result we want Average, Minimum and Maximum salary by Gender. The result of the query should be as shown below.
sql server group by min max

This can be very easily achieved using a simple GROUP BY query as show below.

```
SELECT Gender, COUNT(*) AS GenderTotal, AVG(Salary) AS AvgSal,
       MIN(Salary) AS MinSal, MAX(Salary) AS MaxSal
FROM Employees
GROUP BY Gender
```

What if we want non-aggregated values (like employee Name and Salary) in result set along with aggregated values
non-aggregate columns in a group by query

You cannot include non-aggregated columns in the GROUP BY query.

```
SELECT Name, Salary, Gender, COUNT(*) AS GenderTotal, AVG(Salary) AS AvgSal,
       MIN(Salary) AS MinSal, MAX(Salary) AS MaxSal
FROM Employees
GROUP BY Gender
```

The above query will result in the following error
Column 'Employees.Name' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause

One way to achieve this is by including the aggregations in a subquery and then JOINING it with the main query as shown in the example below.
Look at the amount of T-SQL code we have to write.

```
SELECT Name, Salary, Employees.Gender, Genders.GenderTotals,
       Genders.AvgSal, Genders.MinSal, Genders.MaxSal
FROM Employees

INNER JOIN
(SELECT Gender, COUNT(*) AS GenderTotals,
        AVG(Salary) AS AvgSal,
        MIN(Salary) AS MinSal, MAX(Salary) AS MaxSal
FROM Employees
GROUP BY Gender) AS Genders

ON Genders.Gender = Employees.Gender
```

Better way of doing this is by using the OVER clause combined with PARTITION BY

```
SELECT Name, Salary, Gender,
       COUNT(Gender) OVER(PARTITION BY Gender) AS GenderTotals,
       AVG(Salary) OVER(PARTITION BY Gender) AS AvgSal,
       MIN(Salary) OVER(PARTITION BY Gender) AS MinSal,
       MAX(Salary) OVER(PARTITION BY Gender) AS MaxSal
FROM Employees
```

--109-- RowNumber Function in SQL Server

In this video we will discuss ROW_NUMBER function in SQL Server.
This is continuation to Part 108. Please watch Part 108 from SQL Server tutorial before proceeding.

ROW_NUMBER function
Introduced in SQL Server 2005
Returns the sequential number of a row starting at 1
ORDER BY clause is required
PARTITION BY clause is optional
When the data is partitioned, row number is reset to 1 when the partition changes
Syntax : ROW_NUMBER() OVER (ORDER BY Col1, Col2)

Row_Number function without PARTITION BY : In this example, data is not partitioned,
so ROW_NUMBER will provide a consecutive numbering for all the rows in the
table based on the order of rows imposed by the ORDER BY clause.

```
SELECT Name, Gender, Salary,
       ROW_NUMBER() OVER (ORDER BY Gender) AS RowNumber
FROM Employees
```

sql server row_number example

Please note : If ORDER BY clause is not specified you will get the following error
The function 'ROW_NUMBER' must have an OVER clause with ORDER BY

Row_Number function with PARTITION BY : In this example, data is partitioned by Gender,
so ROW_NUMBER will provide a consecutive numbering only for the rows with in a parttion.

When the partition changes the row number is reset to 1.

```
SELECT Name, Gender, Salary,
       ROW_NUMBER() OVER (PARTITION BY Gender ORDER BY Gender) AS RowNumber
FROM Employees
```

sql server row_number example

Use case for Row_Number function : Deleting all duplicate rows except one from a sql server table.

--110-- RANK and DENSE_RANK in SQL Server

In this video we will discuss Rank and Dense_Rank functions in SQL Server

Rank and Dense_Rank functions
Introduced in SQL Server 2005

Returns a rank starting at 1 based on the ordering of rows imposed by the ORDER BY clause
ORDER BY clause is required
PARTITION BY clause is optional
When the data is partitioned, rank is reset to 1 when the partition changes
Difference between Rank and Dense_Rank functions
Rank function skips ranking(s) if there is a tie where as Dense_Rank will not.

For example : If you have 2 rows at rank 1 and you have 5 rows in total.
RANK() returns - 1, 1, 3, 4, 5
DENSE_RANK returns - 1, 1, 2, 3, 4

Syntax :
RANK() OVER (ORDER BY Col1, Col2, ...)
DENSE_RANK() OVER (ORDER BY Col1, Col2, ...)

Example : We will use the following Employees table for the examples in this video
rank and dense_rank example

SQl Script to create Employees table

```
Create Table Employees
(
    Id int primary key,
    Name nvarchar(50),
    Gender nvarchar(10),
    Salary int
)
Go

Insert Into Employees Values (1, 'Mark', 'Male', 8000)
Insert Into Employees Values (2, 'John', 'Male', 8000)
Insert Into Employees Values (3, 'Pam', 'Female', 5000)
Insert Into Employees Values (4, 'Sara', 'Female', 4000)
Insert Into Employees Values (5, 'Todd', 'Male', 3500)
Insert Into Employees Values (6, 'Mary', 'Female', 6000)
Insert Into Employees Values (7, 'Ben', 'Male', 6500)
Insert Into Employees Values (8, 'Jodi', 'Female', 4500)
Insert Into Employees Values (9, 'Tom', 'Male', 7000)
Insert Into Employees Values (10, 'Ron', 'Male', 6800)
Go
```

RANK() and DENSE_RANK() functions without PARTITION BY clause :
In this example, data is not partitioned, so RANK() function provides a consecutive numbering except when there is a tie. Rank 2 is skipped as there are 2 rows at rank 1. The third row gets rank 3.

DENSE_RANK() on the other hand will not skip ranks if there is a tie.
The first 2 rows get rank 1. Third row gets rank 2.

```
SELECT Name, Salary, Gender,
RANK() OVER (ORDER BY Salary DESC) AS [Rank],
DENSE_RANK() OVER (ORDER BY Salary DESC) AS DenseRank
FROM Employees
```

difference between rank and dense_rank with example

RANK() and DENSE_RANK() functions with PARTITION BY clause :
Notice when the partition changes from Female to Male Rank is reset to 1

```
SELECT Name, Salary, Gender,
RANK() OVER (PARTITION BY Gender ORDER BY Salary DESC) AS [Rank],
DENSE_RANK() OVER (PARTITION BY Gender ORDER BY Salary DESC)
AS DenseRank
FROM Employees
```

rank vs dense_rank in sql server 2008

Use case for RANK and DENSE_RANK functions : Both these functions can be used to find Nth highest salary.
However, which function to use depends on what you want to do when there is a tie. Let me explain with an example.

If there are 2 employees with the FIRST highest salary, there are 2 different business cases
If your business case is, not to produce any result for the SECOND highest salary, then use RANK function
If your business case is to return the next Salary after the tied rows as the SECOND highest Salary, then use DENSE_RANK function
Since we have 2 Employees with the FIRST highest salary. Rank() function will not return any rows for the SECOND highest Salary.

```
WITH Result AS
(
   SELECT Salary, RANK() OVER (ORDER BY Salary DESC) AS Salary_Rank
   FROM Employees
)
SELECT TOP 1 Salary FROM Result WHERE Salary_Rank = 2
```

Though we have 2 Employees with the FIRST highest salary.
Dense_Rank() function returns, the next Salary after the tied rows
as the SECOND highest Salary

```
WITH Result AS
(
   SELECT Salary, DENSE_RANK() OVER (ORDER BY Salary DESC) AS Salary_Rank
   FROM Employees
)
SELECT TOP 1 Salary FROM Result WHERE Salary_Rank = 2
```

You can also use RANK and DENSE_RANK functions to find the Nth highest Salary among Male or

Female employee groups. The following query finds the 3rd highest salary amount paid among the Female employees group

```
WITH Result AS
(
   SELECT Salary, Gender,
          DENSE_RANK() OVER (PARTITION BY Gender ORDER BY Salary DESC)
          AS Salary_Rank
   FROM Employees
)
SELECT TOP 1 Salary FROM Result WHERE Salary_Rank = 3
AND Gender = 'Female'
```

--111-- Difference Between Rank, Dense Rank and Row Number

In this video we will discuss the similarities and difference between RANK, DENSE_RANK and ROW_NUMBER functions in SQL Server.

Similarities between RANK, DENSE_RANK and ROW_NUMBER functions
Returns an increasing integer value starting at 1 based on the ordering of rows imposed by the ORDER BY clause (if there are no ties)
ORDER BY clause is required
PARTITION BY clause is optional
When the data is partitioned, the integer value is reset to 1 when the partition changes

We will use the following Employees table for the examples in this video
rank dense_rank row_number in sql server

SQL Script to create the Employees table
```
Create Table Employees
(
    Id int primary key,
    Name nvarchar(50),
    Gender nvarchar(10),
    Salary int
)
Go

Insert Into Employees Values (1, 'Mark', 'Male', 6000)
Insert Into Employees Values (2, 'John', 'Male', 8000)
Insert Into Employees Values (3, 'Pam', 'Female', 4000)
Insert Into Employees Values (4, 'Sara', 'Female', 5000)
Insert Into Employees Values (5, 'Todd', 'Male', 3000)
```

Notice that no two employees in the table have the same salary.
So all the 3 functions RANK, DENSE_RANK and ROW_NUMBER produce the same increasing integer value when ordered by Salary column.

```
SELECT Name, Salary, Gender,
ROW_NUMBER() OVER (ORDER BY Salary DESC) AS RowNumber,
RANK() OVER (ORDER BY Salary DESC) AS [Rank],
DENSE_RANK() OVER (ORDER BY Salary DESC) AS DenseRank
FROM Employees
```

row_number vs rank vs dense_rank in sql server

You will only see the difference when there ties
(duplicate values in the column used in the ORDER BY clause).

Now lets include duplicate values for Salary column.

To do this
First delete existing data from the Employees table

```
DELETE FROM Employees
```

Insert new rows with duplicate valuse for Salary column

```
Insert Into Employees Values (1, 'Mark', 'Male', 8000)
Insert Into Employees Values (2, 'John', 'Male', 8000)
Insert Into Employees Values (3, 'Pam', 'Female', 8000)
Insert Into Employees Values (4, 'Sara', 'Female', 4000)
Insert Into Employees Values (5, 'Todd', 'Male', 3500)
```

At this point data in the Employees table should be as shown below
Employees table with duplicate salary column

Notice 3 employees have the same salary 8000. When you execute the following query
you can clearly see the difference between RANK, DENSE_RANK and ROW_NUMBER functions.

```
SELECT Name, Salary, Gender,
ROW_NUMBER() OVER (ORDER BY Salary DESC) AS RowNumber,
RANK() OVER (ORDER BY Salary DESC) AS [Rank],
DENSE_RANK() OVER (ORDER BY Salary DESC) AS DenseRank
FROM Employees
```

dense_rank vs rank vs row_number sql server

Difference between RANK, DENSE_RANK and ROW_NUMBER functions

ROW_NUMBER : Returns an increasing unique number for each row starting at 1,
even if there are duplicates.

RANK : Returns an increasing unique number for each row starting at 1.
When there are duplicates, same rank is assigned to all the duplicate rows,
but the next row after the duplicate rows will have the rank it would have been
assigned if there had been no duplicates.
So RANK function skips rankings if there are duplicates.

DENSE_RANK : Returns an increasing unique number for each row starting at 1.

When there are duplicates, same rank is assigned to all the duplicate rows but
the DENSE_RANK function will not skip any ranks. This means the next row after
the duplicate rows will have the next rank in the sequence.

--112-- Calculate Running Total

In this video we will discuss how to calculate running total in SQL Server 2012 and later versions.

We will use the following Employees table for the examples in this video.
running total sql server

SQL Script to create Employees table

```
Create Table Employees
(
    Id int primary key,
    Name nvarchar(50),
    Gender nvarchar(10),
    Salary int
)
Go

Insert Into Employees Values (1, 'Mark', 'Male', 5000)
Insert Into Employees Values (2, 'John', 'Male', 4500)
Insert Into Employees Values (3, 'Pam', 'Female', 5500)
Insert Into Employees Values (4, 'Sara', 'Female', 4000)
Insert Into Employees Values (5, 'Todd', 'Male', 3500)
Insert Into Employees Values (6, 'Mary', 'Female', 5000)
Insert Into Employees Values (7, 'Ben', 'Male', 6500)
Insert Into Employees Values (8, 'Jodi', 'Female', 7000)
Insert Into Employees Values (9, 'Tom', 'Male', 5500)
Insert Into Employees Values (10, 'Ron', 'Male', 5000)
Go
```

SQL Query to compute running total without partitions

```
SELECT Name, Gender, Salary,
       SUM(Salary) OVER (ORDER BY ID) AS RunningTotal
FROM Employees
```

calculate running total in SQL Server 2012

SQL Query to compute running total with partitions

```
SELECT Name, Gender, Salary,
       SUM(Salary) OVER (PARTITION BY Gender ORDER BY ID) AS RunningTotal
FROM Employees
```

running total column

What happens if I use order by on Salary column

If you have duplicate values in the Salary column, all the duplicate values
will be added to the running total at once. In the example below notice that
we have 5000 repeated 3 times. So 15000 (i.e 5000 + 5000 + 5000)
is added to the running total at once.

```
SELECT Name, Gender, Salary,
       SUM(Salary) OVER (ORDER BY Salary) AS RunningTotal
FROM Employees
```

So when computing running total, it is better to use a column that has unique data
in the ORDER BY clause.

--113-- NTILE function in SQL Server

In this video we will discuss NTILE function in SQL Server

NTILE function
Introduced in SQL Server 2005
ORDER BY Clause is required
PARTITION BY clause is optional
Distributes the rows into a specified number of groups
If the number of rows is not divisible by number of groups,
you may have groups of two different sizes.
Larger groups come before smaller groups
For example

NTILE(2) of 10 rows divides the rows in 2 Groups (5 in each group)
NTILE(3) of 10 rows divides the rows in 3 Groups (4 in first group, 3 in 2nd & 3rd group)
Syntax : NTILE (Number_of_Groups) OVER (ORDER BY Col1, Col2, ...)

We will use the following Employees table for the examples in this video.
Employees Table

SQL Script to create Employees table

```
Create Table Employees
(
   Id int primary key,
   Name nvarchar(50),
   Gender nvarchar(10),
   Salary int
)
Go

Insert Into Employees Values (1, 'Mark', 'Male', 5000)
Insert Into Employees Values (2, 'John', 'Male', 4500)
Insert Into Employees Values (3, 'Pam', 'Female', 5500)
Insert Into Employees Values (4, 'Sara', 'Female', 4000)
Insert Into Employees Values (5, 'Todd', 'Male', 3500)
Insert Into Employees Values (6, 'Mary', 'Female', 5000)
Insert Into Employees Values (7, 'Ben', 'Male', 6500)
Insert Into Employees Values (8, 'Jodi', 'Female', 7000)
Insert Into Employees Values (9, 'Tom', 'Male', 5500)
Insert Into Employees Values (10, 'Ron', 'Male', 5000)
Go
```

NTILE function without PARTITION BY clause : Divides the 10 rows into 3 groups.
4 rows in first group, 3 rows in the 2nd & 3rd group.

```
SELECT Name, Gender, Salary,
NTILE(3) OVER (ORDER BY Salary) AS [Ntile]
FROM Employees
```

ntile function in sql server 2008 with example

What if the specified number of groups is GREATER THAN the number of rows
NTILE function will try to create as many groups as possible with one row in each group.

With 10 rows in the table, NTILE(11) will create 10 groups with 1 row in each group.

```
SELECT Name, Gender, Salary,
NTILE(11) OVER (ORDER BY Salary) AS [Ntile]
FROM Employees
```

sql server ntile example

NTILE function with PARTITION BY clause : When the data is partitioned,
NTILE function creates the specified number of groups with in each partition.

The following query partitions the data into 2 partitions (Male & Female).
NTILE(3) creates 3 groups in each of the partitions.

```
SELECT Name, Gender, Salary,
NTILE(3) OVER (PARTITION BY GENDER ORDER BY Salary) AS [Ntile]
FROM Employees
```

sql server ntile function with partition by clause

--114-- Lead and Lag Functions

In this video we will discuss about Lead and Lag functions.

Lead and Lag functions
Introduced in SQL Server 2012
Lead function is used to access subsequent row data along with current row data
Lag function is used to access previous row data along with current row data
ORDER BY clause is required
PARTITION BY clause is optional

Syntax
LEAD(Column_Name, Offset, Default_Value) OVER (ORDER BY Col1, Col2, ...)
LAG(Column_Name, Offset, Default_Value) OVER (ORDER BY Col1, Col2, ...)
Offset - Number of rows to lead or lag.
Default_Value - The default value to return if the number of rows to lead or
lag goes beyond first row or last row in a table or partition. If default value is not specified NULL is returned.

We will use the following Employees table for the examples in this video
lead lag function in sql

SQL Script to create the Employees table

```
Create Table Employees
(
    Id int primary key,
    Name nvarchar(50),
    Gender nvarchar(10),
    Salary int
)
Go

Insert Into Employees Values (1, 'Mark', 'Male', 1000)
Insert Into Employees Values (2, 'John', 'Male', 2000)
Insert Into Employees Values (3, 'Pam', 'Female', 3000)
Insert Into Employees Values (4, 'Sara', 'Female', 4000)
Insert Into Employees Values (5, 'Todd', 'Male', 5000)
Insert Into Employees Values (6, 'Mary', 'Female', 6000)
Insert Into Employees Values (7, 'Ben', 'Male', 7000)
Insert Into Employees Values (8, 'Jodi', 'Female', 8000)
Insert Into Employees Values (9, 'Tom', 'Male', 9000)
Insert Into Employees Values (10, 'Ron', 'Male', 9500)
Go
```

Lead and Lag functions example WITHOUT partitions :
This example Leads 2 rows and Lags 1 row from the current row.

When you are on the first row, LEAD(Salary, 2, -1) allows you to move forward 2 rows
and retrieve the salary from the 3rd row.
When you are on the first row, LAG(Salary, 1, -1) allows us to move backward 1 row.
Since there no rows beyond row 1, Lag function in this case returns the default value -1.
When you are on the last row, LEAD(Salary, 2, -1) allows you to move forward 2 rows.
Since there no rows beyond the last row 1, Lead function in this case returns the default value -1.
When you are on the last row, LAG(Salary, 1, -1) allows us to move backward 1 row
and retrieve the salary from the previous row.

```
SELECT Name, Gender, Salary,
      LEAD(Salary, 2, -1) OVER (ORDER BY Salary) AS Lead_2,
      LAG(Salary, 1, -1) OVER (ORDER BY Salary) AS Lag_1
```

```
FROM Employees
```

lead and lag functions in sql server

Lead and Lag functions example WITH partitions : Notice that in this example,
Lead and Lag functions return default value if the number of rows to lead or
lag goes beyond first row or last row in the partition.

```
SELECT Name, Gender, Salary,
       LEAD(Salary, 2, -1) OVER (PARTITION By Gender ORDER BY Salary) AS Lead_2,
       LAG(Salary, 1, -1) OVER (PARTITION By Gender ORDER BY Salary) AS Lag_1
FROM Employees
```

sql server 2012 lead lag

--115-- FIRST_VALUE function in SQL Server

In this video we will discuss FIRST_VALUE function in SQL Server

FIRST_VALUE function
Introduced in SQL Server 2012
Retrieves the first value from the specified column
ORDER BY clause is required
PARTITION BY clause is optional
Syntax : FIRST_VALUE(Column_Name) OVER (ORDER BY Col1, Col2, ...)

FIRST_VALUE function example WITHOUT partitions : In the following example,
FIRST_VALUE function returns the name of the lowest paid employee from the entire table.

```
SELECT Name, Gender, Salary,
FIRST_VALUE(Name) OVER (ORDER BY Salary) AS FirstValue
FROM Employees
```

first_value function example

FIRST_VALUE function example WITH partitions : In the following example,
FIRST_VALUE function returns the name of the lowest paid employee from the
respective partition.

```
SELECT Name, Gender, Salary,
FIRST_VALUE(Name) OVER (PARTITION BY Gender ORDER BY Salary) AS FirstValue
FROM Employees
```

sql server 2012 first_value function

--116-- Window Functions in SQL Server: ROWS and RANGE

In this video we will discuss window functions in SQL Server

In SQL Server we have different categories of window functions
Aggregate functions - AVG, SUM, COUNT, MIN, MAX etc..
Ranking functions - RANK, DENSE_RANK, ROW_NUMBER etc..
Analytic functions - LEAD, LAG, FIRST_VALUE, LAST_VALUE etc...
OVER Clause defines the partitioning and ordering of a rows (i.e a window) for the above functions to operate on.
Hence these functions are called window functions. The OVER clause accepts the following three arguments to define a
window for these functions to operate on.
ORDER BY : Defines the logical order of the rows
PARTITION BY : Divides the query result set into partitions. The window function is applied to each partition separately.
ROWSor RANGE clause : Further limits the rows within the partition by specifying start and end points within the partition.
The default for ROWS or RANGE clause is
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

Let us understand the use of ROWS or RANGE clause with an example.

Compute average salary and display it against every employee row as shown below.
sql server window function example

We might think the following query would do the job.

```
SELECT Name, Gender, Salary,
       AVG(Salary) OVER(ORDER BY Salary) AS Average
FROM Employees
```

As you can see from the result below, the above query does not produce the overall salary average.
It produces the average of the current row and the rows preceeding the current row.
This is because, the default value of ROWS or RANGE clause (RANGE BETWEEN UNBOUNDED PRECEDING
AND CURRENT ROW) is applied.
window function sql server example

To fix this, provide an explicit value for ROWS or RANGE clause as shown below.
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING tells the window function to operate on
the set of rows starting from the first row in the partition to the last row in the partition.

```
SELECT Name, Gender, Salary,
       AVG(Salary) OVER(ORDER BY Salary ROWS BETWEEN
       UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS Average
FROM Employees
```

sql server rows range clause

The same result can also be achieved by using RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING

What is the difference between ROWS and RANGE
We will discuss this in a later video

The following query can be used if you want to compute the average salary of
1. The current row
2. One row PRECEDING the current row and
3. One row FOLLOWING the current row

```
SELECT Name, Gender, Salary,
       AVG(Salary) OVER(ORDER BY Salary
       ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS Average
FROM Employees
```

sql server rows between 1 preceding and 1 following

--117-- Difference Betwen Rows and Range

In this video we will discuss the difference between rows and range in SQL Server.
This is continuation to Part 116. Please watch Part 116 from SQL Server tutorial before proceeding.

Let us understand the difference with an example. We will use the following Employees table in this demo.
range vs rows in sql server

SQL Script to create the Employees table

```
Create Table Employees
(
    Id int primary key,
    Name nvarchar(50),
    Salary int
)
Go

Insert Into Employees Values (1, 'Mark', 1000)
Insert Into Employees Values (2, 'John', 2000)
Insert Into Employees Values (3, 'Pam', 3000)
Insert Into Employees Values (4, 'Sara', 4000)
Insert Into Employees Values (5, 'Todd', 5000)
Go
```

Calculate the running total of Salary and display it against every employee row
sql server running total query

The following query calculates the running total.
We have not specified an explicit value for ROWS or RANGE clause.

```
SELECT Name, Salary,
       SUM(Salary) OVER(ORDER BY Salary) AS RunningTotal
FROM Employees
```

So the above query is using the default value which is
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

This means the above query can be re-written using an explicit
value for ROWS or RANGE clause as shown below.

```
SELECT Name, Salary,
       SUM(Salary) OVER(ORDER BY Salary
       RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunningTotal
FROM Employees
```

We can also achieve the same result, by replacing RANGE with ROWS

```
SELECT Name, Salary,
       SUM(Salary) OVER(ORDER BY Salary
       ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunningTotal
FROM Employees
```

What is the difference between ROWS and RANGE
To understand the difference we need some duplicate values
for the Salary column in the Employees table.

Execute the following UPDATE script to introduce duplicate values in the Salary column

```
Update Employees set Salary = 1000 where Id = 2
Update Employees set Salary = 3000 where Id = 4
Go
```

Now execute the following query. Notice that we get the running total as expected.

```
SELECT Name, Salary,
       SUM(Salary) OVER(ORDER BY Salary
       ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunningTotal
FROM Employees
```

running total example in sql server

The following query uses RANGE instead of ROWS

```
SELECT Name, Salary,
       SUM(Salary) OVER(ORDER BY Salary
       RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunningTotal
FROM Employees
```

You get the following result when you execute the above query. Notice we dont
get the running total as expected. range clause vs rows clause in sql server

So, the main difference between ROWS and RANGE is in the way duplicate rows are treated.
ROWS treat duplicates as distinct values, where as RANGE treats them as a single entity.

All together side by side. The following query shows how running total changes

1. When no value is specified for ROWS or RANGE clause
2. When RANGE clause is used explicitly with its default value
3. When ROWS clause is used instead of RANGE clause

```
SELECT Name, Salary,
       SUM(Salary) OVER(ORDER BY Salary) AS [Default],
       SUM(Salary) OVER(ORDER BY Salary
       RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS [Range],
       SUM(Salary) OVER(ORDER BY Salary
       ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS [Rows]
FROM Employees
```

calculate cumulative total in sql server

--118-- LASTVALUE function in SQL Server

In this video we will discuss LAST_VALUE function in SQL Server.

LAST_VALUE function
Introduced in SQL Server 2012
Retrieves the last value from the specified column
ORDER BY clause is required
PARTITION BY clause is optional
ROWS or RANGE clause is optional, but for it to work correctly you may have to explicitly specify a value
Syntax : LAST_VALUE(Column_Name) OVER (ORDER BY Col1, Col2, ...)

LAST_VALUE function not working as expected : In the following example, LAST_VALUE function does not return
the name of the highest paid employee. This is because we have not specified an explicit value for ROWS or RANGE clause.
As a result it is using its default value RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.

```
SELECT Name, Gender, Salary,
    LAST_VALUE(Name) OVER (ORDER BY Salary) AS LastValue
FROM Employees
```

sql server last_value returns incorrect data

LAST_VALUE function working as expected : In the following example, LAST_VALUE function returns the name of the highest
paid employee as expected. Notice we have set an explicit value for ROWS or RANGE clause to ROWS BETWEEN UNBOUNDED
PRECEDING AND UNBOUNDED FOLLOWING

This tells the LAST_VALUE function that its window starts at the first row and ends at the last row in the result set.

```
SELECT Name, Gender, Salary,
   LAST_VALUE(Name) OVER (ORDER BY Salary ROWS BETWEEN UNBOUNDED
PRECEDING AND UNBOUNDED FOLLOWING) AS LastValue
FROM Employees
```

sql server last_value function example

LAST_VALUE function example with partitions : In the following example, LAST_VALUE function returns the name of the
highest paid employee from the respective partition.

```
SELECT Name, Gender, Salary,
   LAST_VALUE(Name) OVER (PARTITION BY Gender ORDER BY Salary
   ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS
LastValue
FROM Employees
```

sql server last_value function with partition example

--119-- UPIVOT in SQL Server
In this video we will discuss UNPIVOT operator in SQL Server.

PIVOT operator turns ROWS into COLUMNS, where as UNPIVOT turns COLUMNS into ROWS.

We discussed PIVOT operator in Part 54 of SQL Server tutorial.
Please watch Part 54 before proceeding.

Let us understand UNPIVOT with an example. We will use the following tblProductSales table in this demo.

unpivot in sql server example

SQL Script to create tblProductSales table
```
Create Table tblProductSales
(
 SalesAgent nvarchar(50),
 India int,
 US int,
 UK int
)
Go

Insert into tblProductSales values ('David', 960, 520, 360)
Insert into tblProductSales values ('John', 970, 540, 800)
Go
```

Write a query to turn COLUMNS into ROWS. The result of the query should be as shown below.

sql server unpivot example

```
SELECT SalesAgent, Country, SalesAmount
FROM tblProductSales
UNPIVOT
(
      SalesAmount
      FOR Country IN (India, US , UK)
) AS UnpivotExample
```