Advanced Operators, ANSI Null, Coalesce, Null If

ISM 6218

Due on October 8th

The Avengers Team

"We will avenge every problem on our way"

Aitemir Yeskenov (Team Lead)

Nagarjuna Kanneganti

Sai Suraj Argula

Vinay Kumar Reddy Baradi

Table of Contents

Business Process Supported

For this assignment, we used two databases to experiment with queries using such operators as **EXISTS**, **IN, ANY, HAVING**, and play around with **IF NULL, ANSI NULL** as well as the **Coalesce**. The results include not only different data sets, but also execution comparison. We worked with CTEs and subqueries and had user stories for each scenario. We used **Employee** DB that we created ourselves and the **AdventureWorks** database provided by the instructor.

Requirements Described

Apply to your database of choice:

1. Recreate Coalesce Walkthrough.
2. Create a user story to compare Exists, IN, ANY, Having.
3. Create a user story to use WITH as a sub-query alias and as a CTE.

- You must define multiple CTEs for your User Story

4. Demostrate use of Null IF.
5. Create a walkthrough of ANSI NULL example.

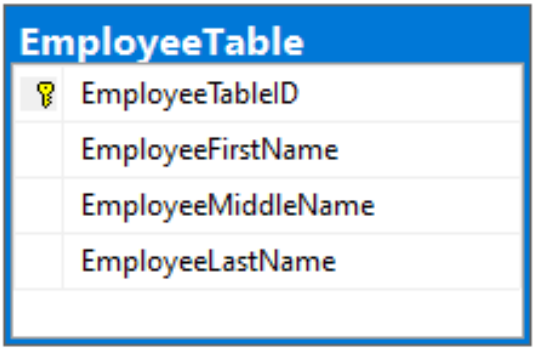
1) Recreate Coalesce Walkthrough.

User Story:

As an employer I want to send an Annual salary statement letter to all the employees. In particular we want, all the names, addresses, and calculated salaries over the year.

SQL Coalesce in a string concatenation operation

Below is the Employee Name Table structure and details in the table.



EmployeeTableID	EmployeeFirstName	EmployeeMiddleName	EmployeeLastName
1	Wade	Levi	Rosario
2	Dane	NULL	Levine
3	NULL	Cullen	Franklin
4	George	Benjamin	NULL
5	NULL	NULL	NULL
6	NULL	NULL	Duran
7	Nathan	NULL	NULL
8	NULL	Kevin	NULL
9	Kevin	Jenette	Harris
10	Amir	Britanney	Campos
11	Dalton	Tobias	Castro

Since I want to send the letters to their address, they need their full names. However, from the above dataset, we found some nulls in Employee Table. We want to concatenate the names to get the full names.

We have created Concatenation of the names to satisfy the requirement.

```
SELECT EmployeeFirstName +' '+ EmployeeMiddleName +' '+ EmployeeLastName FullName FROM [dbo].[EmployeeTable]
```

But from the result Table, we found if any of the value in the Names is Null the concatenation is resulting Null.

	FullName
1	Wade Levi Rosario
2	NULL
3	NULL
4	NULL
5	NULL
6	NULL
7	NULL
8	NULL
9	Kevin Jenette Harris
10	Amir Britanney Campos
11	Dalton Tobias Castro

We need to fix the above Result, we found coalesce operator to fix this one.

```
SELECT COALESCE(EmployeeFirstName,'') +' '+COALESCE(EmployeeMiddleName,'') +' '+ COALESCE(EmployeeLastName,'')
as FullName FROM [dbo].[EmployeeTable]
```

Below is the result which removes all the null values and concatenates the names.

Results | Messages

	FullName
1	Wade Levi Rosario
2	Dane Levine
3	Cullen Franklin
4	George Benjamin
5	
6	Duran
7	Nathan
8	Kevin
9	Kevin Jenette Harris
10	Amir Britanney Campos
11	Dalton Tobias Castro

SQL Coalesce function and pivoting

Employer wants to send letters to employees of the Michigan state first and for that he wants to identify all the employees of the Michigan first and group them. Below is the table of the

EmployeeCityState
EmployeeID
EmployeeCity
EmployeeState

EmployeeCityStateID	EmployeeCity	EmployeeState
1	Chicago	Illinois
2	Bellevue	Nebraska
3	Detroit	Michigan
4	Warren	Michigan
5	San Francisco	California
6	Cleveland	Ohio
7	Worcester	Massachusetts
8	Indianapolis	Indiana
9	Cincinnati	Ohio
10	Salt Lake City	Utah
11	Georgia	Georgia

We used the below statement to identify all the cities the letters to be sent.

```
DECLARE @CityState nvarchar(MAX);
SELECT @CityState = COALESCE(@CityState,'') +''''+[dbo].[EmployeeCityState].[EmployeeCity] +''''+ ','
FROM [dbo].[EmployeeCityState] WHERE  [EmployeeState] = 'Michigan';

--SELECT substring(@CityState,1,len(@CityState)-1)
SELECT '('+substring(@CityState,1,len(@CityState)-1)+')' as identifiedcities
```

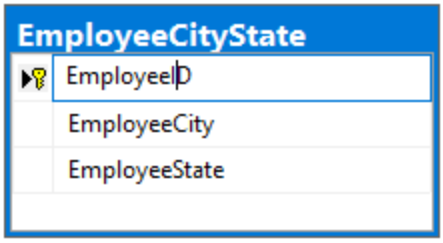
Results | Messages

	citiestosend
1	('Detroit','Warren','Lansing','Detroit','Warren')

Scalar user-defined function and SQL Coalesce function

User Story:

As an employer I want to send letters to all the employees by their state so that we are able to post them whenever we need and to do that, we need to group the letter by state.



EmployeeCityStateID	EmployeeCity	EmployeeState
1	Chicago	Illinois
2	Bellevue	Nebraska
3	Detroit	Michigan
4	Warren	Michigan
5	San Francisco	California
6	Cleveland	Ohio
7	Worcester	Massachusetts
8	Indianapolis	Indiana
9	Cincinnati	Ohio
10	Salt Lake City	Utah
11	Georgia	Georgia

Below is the SQL code where we did and used both Coalesce and User-defined function.

```sql
Alter FUNCTION dbo.CityState
(
  @CityState varchar(100)
)
RETURNS NVARCHAR(MAX)
AS
BEGIN
  DECLARE @str NVARCHAR(MAX);

  SELECT @str = COALESCE(@str + ', ', '') + [EmployeeCity]
    FROM [dbo].[EmployeeCityState]
    WHERE [EmployeeState] = @CityState
    ORDER BY [EmployeeState];

  RETURN (@str);
END
GO

SELECT [EmployeeState], city = dbo.CityState([EmployeeState])
  FROM [dbo].[EmployeeCityState]
  GROUP BY [EmployeeState]
  ORDER BY [EmployeeState];
```

Below is the result set, we can send letters to any state of our choice on any date we want:

Results | Messages

	EmployeeState	city
1	Alabama	Huntsville
2	Alaska	Ketchikan, Anchorage, Anchorage
3	Arizona	Mesa, Mesa, Chandler
4	Arkansas	Jonesboro
5	California	San Francisco, San Diego, San Francisco, San Francisco, Sacramento
6	Colorado	Fort Collins
7	Connecticut	Hartford, Bridgeport
8	Delaware	Bear, Newark, Newark, Pike Creek, Bear, Bear, Bear
9	Florida	Jacksonville, Orlando, Tampa, Tallahassee, Jacksonville, Miami
10	Georgia	Georgia, Columbus, Athens, Atlanta
11	Hawaii	Kailua
12	Idaho	Nampa, Meridian, Nampa, Nampa
13	Illinois	Chicago
14	Indiana	Indianapolis, Indianapolis
15	Iowa	Iowa City, Des Moines, Des Moines, Davenport
16	Kansas	Overland Park
17	Kentucky	Louisville
18	Louisiana	Baton Rouge, New Orleans, Shreveport, New Orleans
19	Maine	Lewiston

SQL Coalesce and Computed columns

User story:

We want to compute the Annual Salaries of the employees. Employee annual salary is the combination of the fixed amount and incentive upon the number of sales of the employee.

Below is the raw table of the employee fixed salary commission per sale and number of sales the employee has.

Results | Messages

	EMPNO	ENAME	HOURLYWAGE	SALARY	COMMISSION	NUMSALES
1	1	sai	NULL	8000.00	NULL	20.00
2	2	Vinay	NULL	5600.00	7000.00	4.00
3	3	Aitemir	40.00	1250.00	5000.00	10.00

We used the select statement with Coalesce operator to calculate the salary. So that when there is null value for commission / hourly wage we be able to get exact calculation of total salary instead of Nulls.

```
SELECT EMPNO,ENAME,CAST(COALESCE(HOURLYWAGE * 40 * 52,
    salary,
    Salary+(COMMISSION * NUMSALES)) AS decimal(10,2)) AS TotalSalary
FROM dbo.EMP
ORDER BY TotalSalary;
```

The result by using the above SQL statement.

Results | Messages

	EMPNO	ENAME	TotalSalary
1	2	Vinay	5600.00
2	1	sai	8000.00
3	3	Aitemir	83200.00

SQL COALESCE and CASE expression

User story:

As an employee I want to send letter to other employees and wants to include phone numbers in the address. Below is the table he has

Results | Messages

empid	firstname	homephone	cellphone
1	sai	NULL	8145
2	vinay	815165	NULL
3	aitmer	81356454	81454456

Using **Coalesce:**

```sql
SELECT
  empid , firstname,
  COALESCE(homephone,cellphone)phone FROM
  dbo.tb_Contact

select empid,firstname,
  CASE
    WHEN homephone is NOT NULL Then homephone
    WHEN cellphone is NOT NULL Then cellphone
   ELSE 'NA'
   END
   ContactNumber
FROM
```

Results | Messages

empid	firstname	phone
1	sai	8145
2	vinay	815165
3	aitmer	81356454

Using **Case Expression**:

```sql
select empid,firstname,
  CASE
    WHEN homephone is NOT NULL Then homephone
    WHEN cellphone is NOT NULL Then cellphone
   ELSE 'NA'
   END
   ContactNumber
FROM
  dbo.tb_Contact
```

Results | Messages

	empid	firstname	ContactNumber
1	1	sai	8145
2	2	vinay	815165
3	3	aitmer	81356454

We can observe how both Coalesce and Case give same results.

2) User story for EXISTS, IN, ANY, HAVING

EXISTS:

We want to retrieve a list of employees that are production technicians (WC60 specialization). In particular, we are interested in their full names and person types. In order to do that, we wrote a statement with a subquery that retrieves the records based on the job titles from the **employee** table:

```sql
-- requrement 2

SELECT BusinessEntityId, FirstName, LastName, MiddleName, PersonType
FROM Person.Person AS p
WHERE EXISTS (SELECT BusinessEntityID FROM HumanResources.Employee AS e
              WHERE e.BusinessEntityID = p.BusinessEntityID
              AND e.JobTitle = 'Production Technician - WC60');

SELECT BusinessEntityId, FirstName, LastName, MiddleName, PersonType
```

150 %

Results | Messages

	BusinessEntityId	FirstName	LastName	MiddleName	PersonType
1	70	David	Ortiz	J	EM
2	69	Steve	Masters	F	EM
3	68	Charles	Fitzgerald	B	EM
4	67	Jay	Adams	G	EM
5	66	Karan	Khanna	R	EM
6	65	Randy	Reeves	T	EM
7	64	Michael	Zwilling	J	EM
8	63	Maciej	Dusza	W	EM
9	46	Eugene	Kogan	O	EM
10	45	Thomas	Michaels	R	EM
11	44	Simon	Rapier	D	EM
12	43	Nancy	Anderson	A	EM
13	42	James	Kramer	D	EM
14	41	Bryan	Baker	NULL	EM
15	39	Ed	Dudenhoefer	R	EM
16	38	Kim	Abercrombie	B	EM
17	37	Chris	Okelberry	O	EM
18	36	Jose	Lugo	R	EM
19	35	Brandon	Heidepriem	G	EM
20	34	Suchitra	Mohan	O	EM
21	33	Annik	Stahl	O	EM
22	32	Rebecca	Laszlo	A	EM
23	31	Margie	Shoop	W	EM
24	30	Britta	Simon	L	EM
25	29	Mark	McArthur	K	EM
26	28	Guy	Gilbert	R	EM

As shown above, the records are retrieved successfully. We have 26 records. For the example above, we used the **EXISTS** operator that helps users to get the result set of the exact values expected (in other words if the records of a condition specified in subquery exist, then the result set would retrieve those records).

For the sake of experiment, using the same user story, we also implemented same query with other operators.

IN:

```
SELECT BusinessEntityId, FirstName, LastName, MiddleName, PersonType
FROM Person.Person AS p
WHERE BusinessEntityID IN (SELECT BusinessEntityID FROM HumanResources.Employe
                WHERE e.BusinessEntityID = p.BusinessEntityID
                AND e.JobTitle = 'Production Technician - WC60');
```

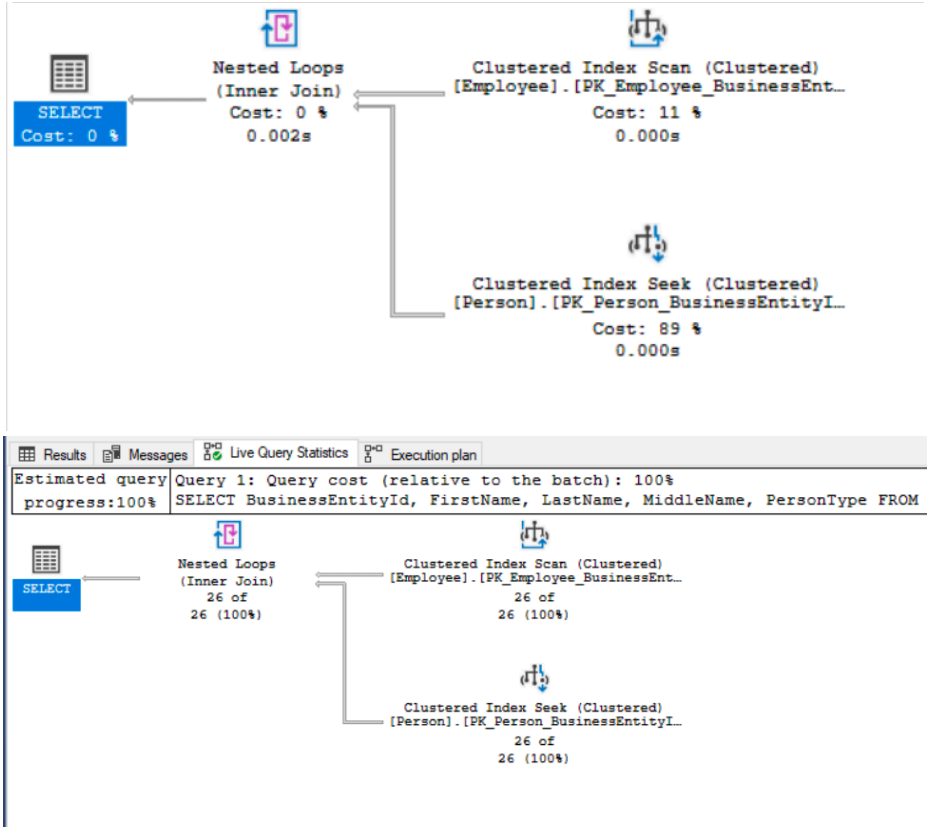
ANY:

```
SELECT BusinessEntityId, FirstName, LastName, MiddleName, PersonType
FROM Person.Person AS p
WHERE BusinessEntityID = ANY (SELECT BusinessEntityID FROM HumanResources.Empl
                WHERE e.BusinessEntityID = p.BusinessEntityID
                AND e.JobTitle = 'Production Technician - WC60');
```

All the queries retrieve same result sets – exactly 26 records. We then compared the execution plans:

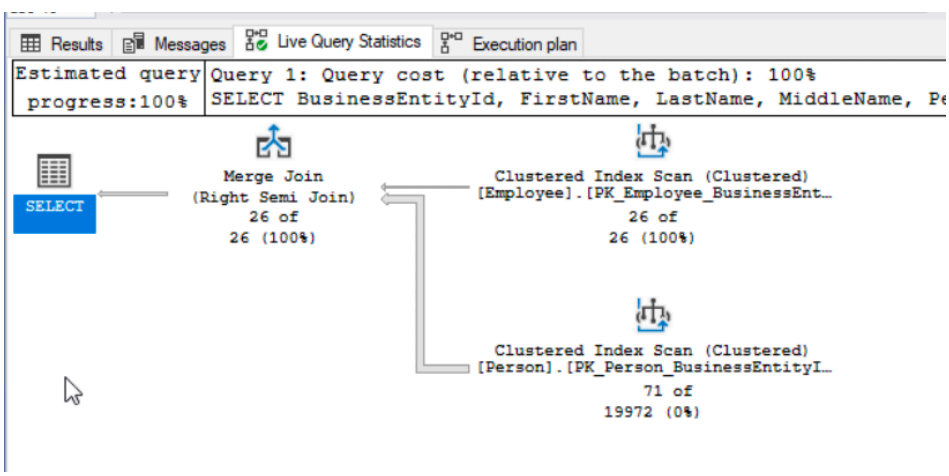
EXISTS:

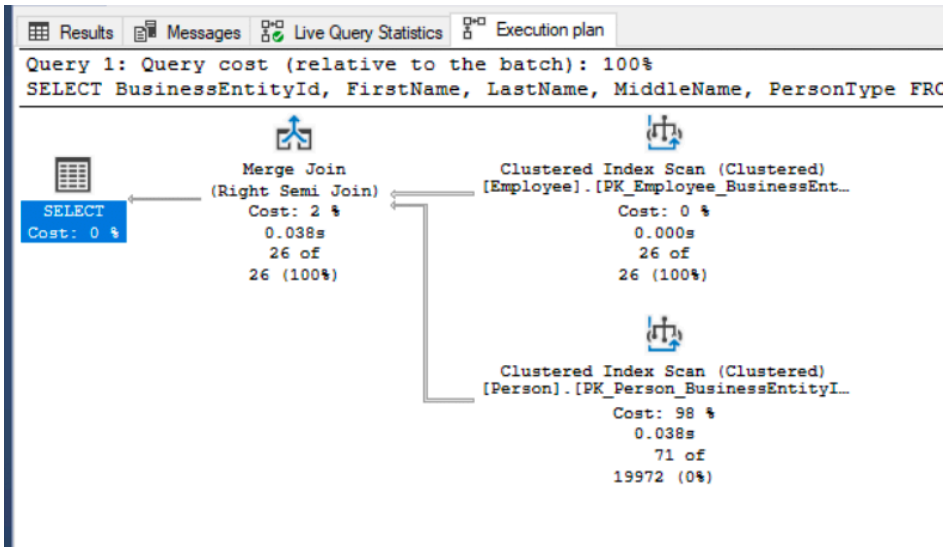
```
   CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 145 ms.
```



IN:

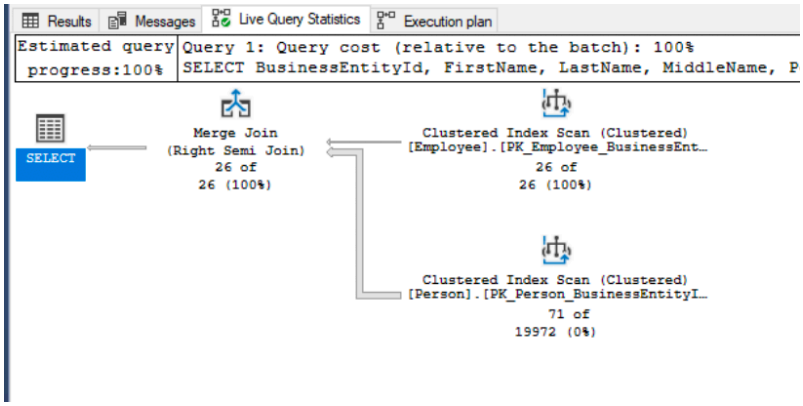
```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 16 ms.

(26 rows affected)
```

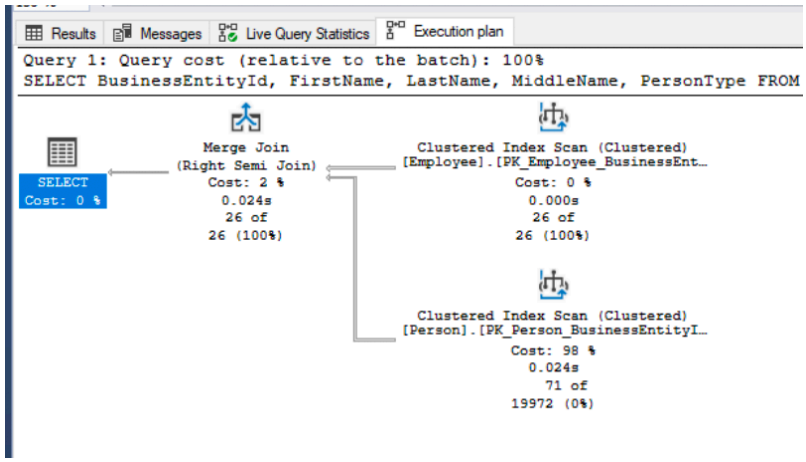




ANY:

```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 10 ms.
```





Execution plan comparison:

The comparison showed quite interesting results:

EXISTS – elapsted time **145ms**, cost **0.002s**

IN – elapsed time **16ms**, cost **0.038s**

ANY – elapsed time **10ms**, cost **0.024s**

You can see that ANY showed the best record for the elapsed time, yet the cost time is better in exists. Overall, the comparison gives a good idea on what could be more efficient for the future experiments.

3) Use of WITH:

CTE example:

We wanted to retrieve a list of records of people from the **person.person** table that are currently employed and have data under the **humanresources.employee** table. In order to do that, we ran

a CTE query. We used CTE query since this type of query is not going to be used any time in the future, so the data is stored within CTE temporary:

```
with cteEmployedPeople (BusinessEntityId, FirstName, LastName,
MiddleName, PersonType)
as
(
Select p.BusinessEntityID, FirstName,
LastName, MiddleName, PersonType from person.person as p
inner join humanresources.Employee as e
on p.BusinessEntityID = e.BusinessEntityID
)
select * from cteEmployedPeople;
```

150 %

Results | Messages

	BusinessEntityId	FirstName	LastName	MiddleName	PersonType
1	263	Jean	Trenary	E	EM
2	78	Reuben	D'sa	H	EM
3	242	Deborah	Poe	E	EM
4	125	Matthias	Berndt	T	EM
5	278	Garrett	Vargas	R	SP
6	239	Mindy	Martin	C	EM
7	184	John	Chen	Y	EM
8	87	Cristian	Petculescu	K	EM
9	174	Benjamin	Martin	R	EM
10	284	Tete	Mensa-Annan	A	SP
11	211	Hazem	Abolrous	E	EM
12	115	Angela	Barbariol	W	EM
13	241	David	Liu	J	EM
14	7	Dylan	Miller	A	EM
15	222	A. Scott	Wright	NULL	EM
16	232	Pat	Coleman	H	EM
17	280	Pamela	Ansman-Wolfe	O	SP
18	287	Amy	Alberts	E	SP
19	51	Jeffrey	Ford	L	EM
20	1	Ken	Sánchez	J	EM
21	197	Rajesh	Patel	M	EM
22	266	Peter	Connelly	I	EM
23	33	Annik	Stahl	O	EM
24	113	Linda	Moschell	K	EM

As you can see on this screen, we successfully got the result set with the list of employees – we have their businessEntityId, full name, and the person type.

Sub-query example:

We received a new requirement to generate a report of employees with the info of logins, job titles, and current status. In addition, we were told to skip the employees with IDs between 100 and 250. To do that, we ran a query with **WITH** clause and concatenated two result sets from two queries with different conditions:

```sql
with EmployedPeople (BusinessEntityID,
LoginId, JobTitle, CurrentFlag)
AS
(
    SELECT BusinessEntityID, LoginId, JobTitle, CurrentFlag
    FROM HumanResources.Employee
    WHERE BusinessEntityID < 100
    UNION
    SELECT BusinessEntityID, LoginId, JobTitle, CurrentFlag
    FROM HumanResources.Employee
    WHERE BusinessEntityID > 250
)
SELECT BusinessEntityID, LoginId, JobTitle, CurrentFlag as Active
FROM EmployedPeople;
```

Results | Messages

	BusinessEntityID	LoginId	JobTitle	Active
1	1	adventure-works\ken0	Chief Executive Officer	1
2	2	adventure-works\terri0	Vice President of Engineering	1
3	3	adventure-works\roberto0	Engineering Manager	1
4	4	adventure-works\rob0	Senior Tool Designer	1
5	5	adventure-works\gail0	Design Engineer	1
6	6	adventure-works\jossef0	Design Engineer	1
7	7	adventure-works\dylan0	Research and Development Manager	1
8	8	adventure-works\diane1	Research and Development Engineer	1
9	9	adventure-works\gigi0	Research and Development Engineer	1
10	10	adventure-works\michael6	Research and Development Manager	1
11	11	adventure-works\ovidiu0	Senior Tool Designer	1
12	12	adventure-works\thierry0	Tool Designer	1
13	13	adventure-works\janice0	Tool Designer	1
14	14	adventure-works\michael8	Senior Design Engineer	1
15	15	adventure-works\sharon0	Design Engineer	1
16	16	adventure-works\david0	Marketing Manager	1

MULTIPLE CTES example:

We wanted to retrieve a big result set with all the people in the database and make sure that we have a specific data portion with the employed people (records that are part of the **humanresources.employee** table). In order to do that, we created a CTE with the people that are employed, then created another CTE with the list of people that are not employees and ran a query to concatenate two results sets into one by adding UNION ALL. Since as part of the requirement there is no specification about having distinct or non-distinct values, we used UNION ALL:

```sql
with cteEmployedPeople (BusinessEntityId, FirstName, LastName, MiddleName, Per
as
(
    Select p.BusinessEntityID, FirstName,
    LastName, MiddleName, PersonType from person.person as p
    inner join humanresources.Employee as e
    on p.BusinessEntityID = e.BusinessEntityID
)
,
cteNonEmployedPeople (BusinessEntityId, FirstName, LastName, MiddleName, Perso
as
(
    Select p.BusinessEntityID, FirstName,
    LastName, MiddleName, PersonType from person.person as p
    inner join humanresources.Employee as e
    on p.BusinessEntityID <> e.BusinessEntityID
)
Select BusinessEntityId, FirstName, LastName, MiddleName, PersonType
from cteEmployedPeople
UNION ALL
Select BusinessEntityId, FirstName, LastName, MiddleName, PersonType
from cteNonEmployedPeople;
```

150 %

Results | Messages

	BusinessEntityId	FirstName	LastName	MiddleName	PersonType
1	263	Jean	Trenary	E	EM
2	78	Reuben	D'sa	H	EM
3	242	Deborah	Poe	E	EM
4	125	Matthias	Berndt	T	EM
5	278	Garrett	Vargas	R	SP
6	239	Mindy	Martin	C	EM
7	184	John	Chen	Y	EM

Activate Windows
Go to Settings to activate

Query executed successfully. | DESKTOP-7NG8FDI (14.0 RTM) | DESKTOP-7NG8FDI\Aitemi... | AdventureWorks2017 | 00:01:09 | 5,791,880 rows

Results | Messages

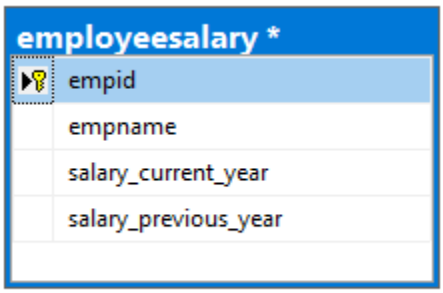
```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 8 ms.

(5791880 rows affected)

 SQL Server Execution Times:
   CPU time = 3890 ms,  elapsed time = 69345 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

Completion time: 2019-10-08T17:24:26.7168759-04:00
```

As a result, 5,791,880 rows are affected.

4) Use of NULL IF

I want to know if my employees' salaries are equal in both the current year current year and last year and return flag 'null' if they are equal and flag 'null' if employees joined the organization this year. If the salary increased - we return the last year salary.

Below is the salary table of the employees and salaries from last year and current year:



Results | Messages

empid	empname	salary_current_year	salary_previous_year
1	sai	100000.00	150000.00
2	aitmer	NULL	300000.00
3	vinay	0.00	100000.00
4	nag	NULL	150000.00
5	sourabh	300000.00	300000.00

Demonstration of **NULL IF** operator:

```sql
SELECT empname, NULLIF(salary_current_year ,
  salary_previous_year) AS changeinsalary
FROM employeesalary;
```

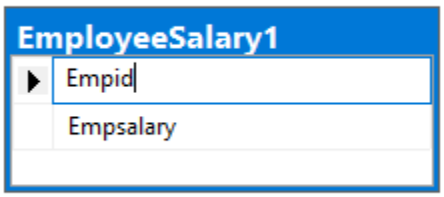
110 %

Results | Messages

	empname	changeinsalary
1	sai	100000.00
2	aitmer	NULL
3	vinay	0.00
4	nag	NULL
5	sourabh	NULL

5) ANSI NULL walkthrough:

Here is the employee salary table that we work with:



Results | Messages

Empid	Empsalary
1	NULL
2	10000.00
3	11111.00

After experimenting with numerous operators, we decided to run an experiment using the **ANSI NULL** operator so as to improve the future queries and find out if it is valuable for our working

processes, especially when it comes to retrieving with nulls. First of all, we ran the queries without the ANSI null:

When **ANSI Null** is turned off:

```
SET ANSI_NULLS off;
GO
DECLARE @varname int;
SET @varname = NULL

SELECT empid,Empsalary
FROM dbo.EmployeeSalary1
WHERE Empsalary = @varname;

SELECT empid,Empsalary
FROM dbo.EmployeeSalary1
WHERE Empsalary <> @varname;

SELECT empid,Empsalary
FROM dbo.EmployeeSalary1
WHERE Empsalary IS NULL;
GO
```

10 %

Results | Messages

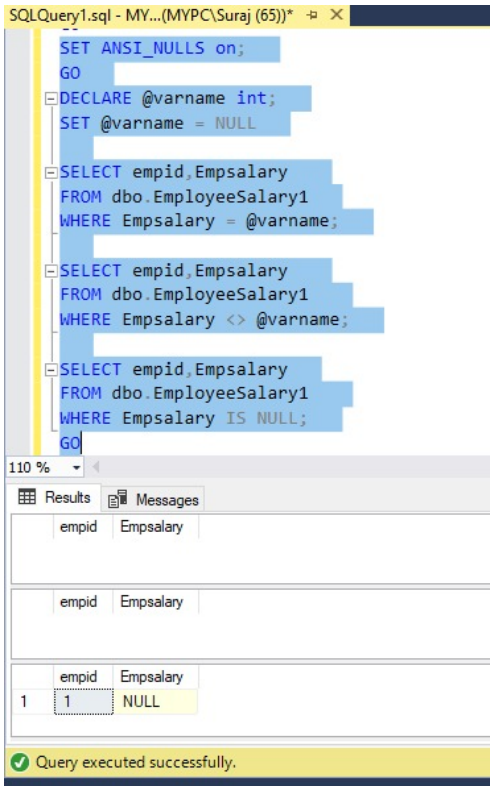
	empid	Empsalary
1	1	NULL

	empid	Empsalary
1	2	10000.00
2	3	11111.00

	empid	Empsalary
1	1	NULL

Query executed successfully.

When **ANSI Null** is turned on:



When SET ANSI_NULLS is ON, a SELECT statement that uses WHERE salary= NULL returns zero rows and if there are null values in Salary and SELECT statement that uses WHERE Salary <> NULL returns zero rows, and even if there are non-null values in Salary, which is opposite to when we turn off the ANSI Null. - i.e it returned null (when salary =null) and non-null rows (when salary is non null).