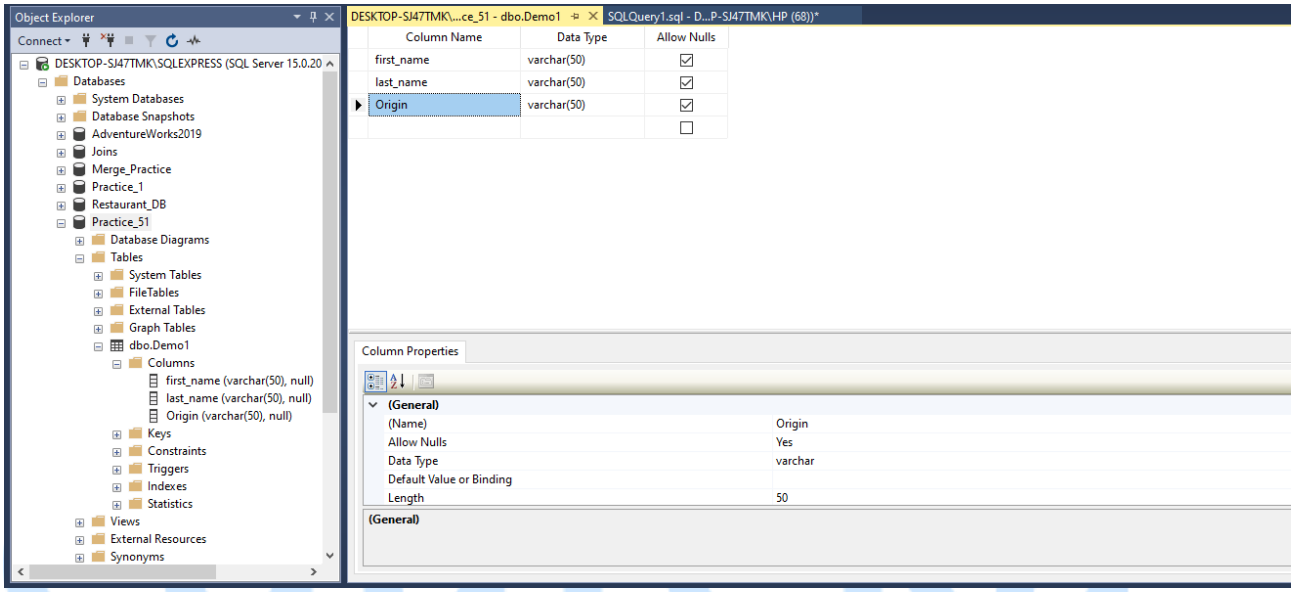
Advanced SQL - III

Demo - Date, Time, and Ranking Functions

Ranking Functions

Problem Statement: Use Ranking functions on the following sample database to assign a ranking to each row in the table.

Step 1: Create a new database named **Practice_51** with a table **Demo1** consisting of the following design: 3 attributes called 'first_name,' 'last_name,' and 'Origin.'

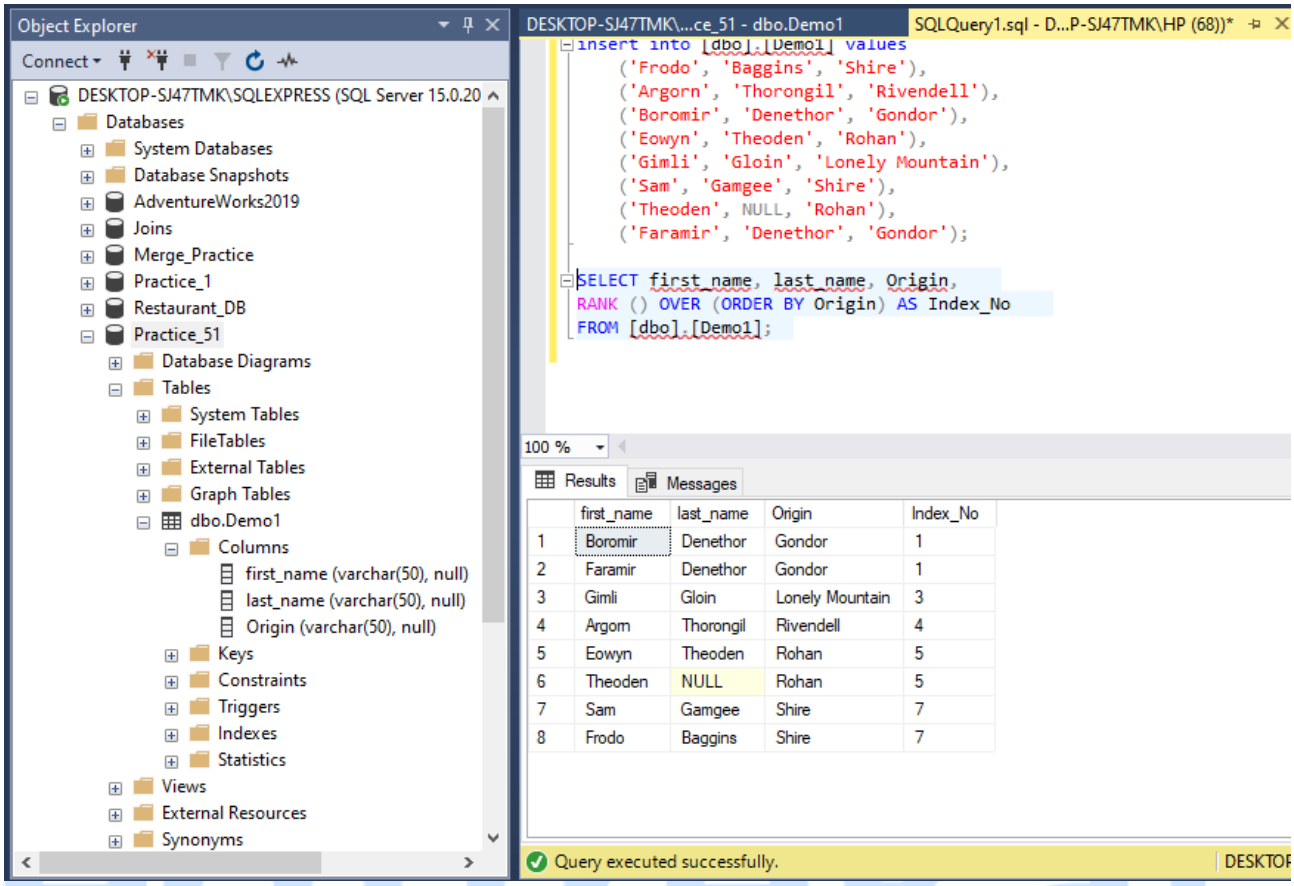


Step 2: Insert the following records into the database in the Demo1 table:

```
insert into [dbo].[Demo1] values
 ('Frodo', 'Baggins', 'Shire'),
 ('Argorn', 'Thorongil', 'Rivendell'),
 ('Boromir', 'Denethor', 'Gondor'),
 ('Eowyn', 'Theoden', 'Rohan'),
 ('Gimli', 'Gloin', 'Lonely Mountain'),
 ('Sam', 'Gamgee', 'Shire'),
 ('Theoden', NULL, 'Rohan'),
 ('Faramir', 'Denethor', 'Gondor');
```

Step 3: Use the `RANK()` function to determine the rank for each row in the result set.

```
SELECT first_name, last_name, Origin,
RANK () OVER (ORDER BY city) AS Rank_No
FROM Demo1;
```



In this output, we can see that some of the rows get the same rank because they have the same value in the Origin column. And the following number in the ranking will be its previous rank plus several duplicate numbers.

Step 4: For the next step of the demo, use a transaction to drop the 'Index_No' column. Use `select * from Demo1;` to keep track of all the changes made over the course of this demo.

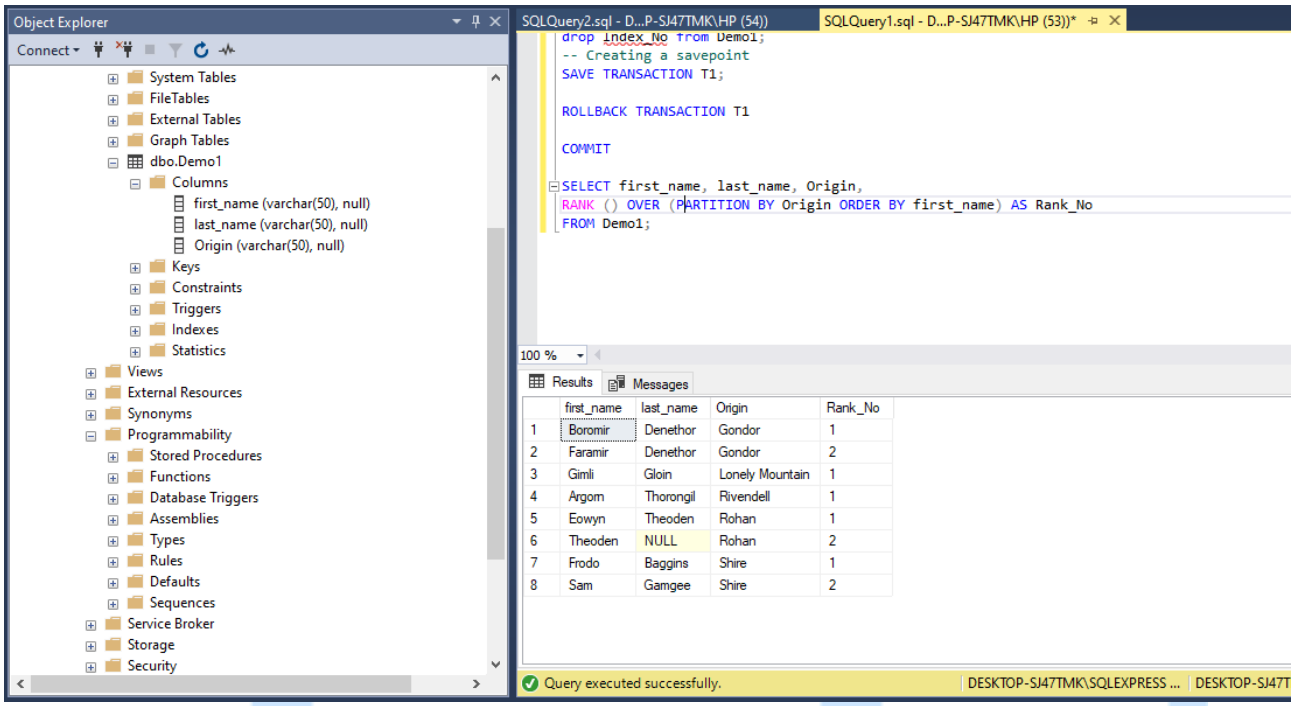
```
Begin TRANSACTION
drop Index_No from Demo1;
-- Creating a savepoint
SAVE TRANSACTION T1;

ROLLBACK TRANSACTION T1

COMMIT
```

The following statement is another example where we are going to use a partition by a clause that will divide the rows based on the Origin column and assign a ranking to each row within a partition. The order of the output is based on the first_name

Step 5: Use the statement with `RANK()` function with partition

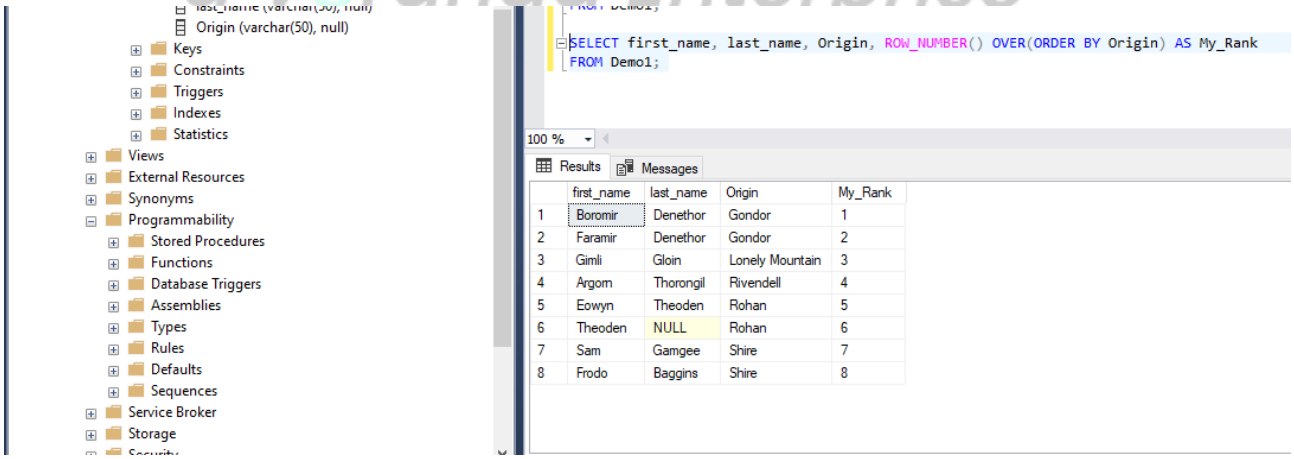
```
SELECT first_name, last_name, Origin,
RANK () OVER (PARTITION BY Origin ORDER BY
```

```
first_name) AS Rank_No
FROM Demo1;
```



Step 6: Using the row number function to return the unique sequential number for each row within its partition. After using the transaction to drop the 'Rank_No' use the transaction statement or a simple `drop()` function to delete the column.

```
SELECT first_name, last_name, Origin,
ROW_NUMBER() OVER(ORDER BY Origin) AS My_Rank
FROM Demo1;
```



The row numbering begins at one and increases by one until the partition's total number of rows is reached. It will return the different ranks for the row having similar values that make it different from the `RANK()` function. It will assign the ranking for the table as per their city. Here we can see that it assigns different ranks for the row which has the same Origin values.

Step 7: Understanding the `DENSE_RANK()` function.

```
SELECT first_name, last_name, Origin,
DENSE_RANK() OVER(ORDER BY Origin DESC) AS
Ranking
FROM Demo1;
```

This function assigns a unique rank for each row within a partition as per the specified column value without any gaps. It always specifies ranking in consecutive order. If we get a duplicate value, this function will assign it the same rank, the next rank being the next sequential number. This characteristic differs DENSE_RANK() function from the RANK() function.

Step 8: Updating the following query

```
Update Demo1 set first_name = 'Diego' WHERE
Origin = 'Gondor'

SELECT first_name, last_name, Origin,
DENSE_RANK() OVER(ORDER BY Origin DESC) AS
Ranking
FROM Demo1;
```

Step 9: `NTILE(N)` function is used to distribute rows of an ordered partition into a pre-defined number (N) of approximately equal groups. Each row group gets its rank based on the defined condition and starts numbering from one group.

```
--NTILE(N)
SELECT first_name, last_name, Origin, NTILE(3)
OVER(ORDER BY Origin) AS Ranking
FROM Demo1;
```

The specified table has eight records. Therefore, the NTILE(3) tells that the result set must have a group of three records.

Date & Time Functions

The SQL Server has many date and time functions with different functionalities some examples are shown here.

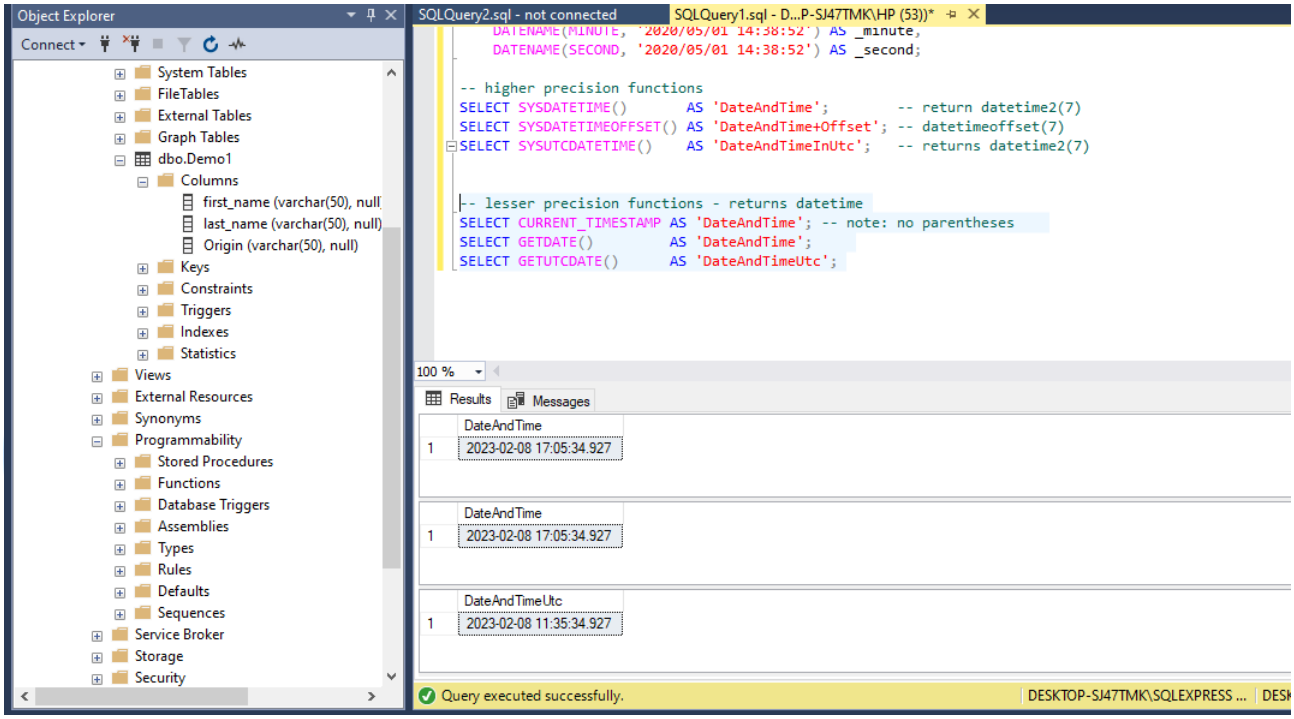
Step 10: Some Examples of date and time functions with Higher Precision Functions

```
-- higher precision functions
SELECT SYSDATETIME()        AS 'DateAndTime';        --
return datetime2(7)
SELECT SYSDATETIMEOFFSET() AS 'DateAndTime+Offset'; --
```

```
datetimeoffset(7)
SELECT SYSUTCDATETIME()     AS 'DateAndTimeInUtc';    --
returns datetime2(7)
```

Step 11: SQL Server Lesser Precision Data and Time Functions have a scale of 3 and are:

```
-- lesser precision functions - returns datetime
SELECT CURRENT_TIMESTAMP AS 'DateAndTime'; --
note: no parentheses
SELECT GETDATE()            AS 'DateAndTime';
SELECT GETUTCDATE()         AS 'DateAndTimeUtc';
```



Step 12: Executing `DATENAME()` Function

```
--DATENAME Function
SELECT DATENAME(YEAR, GETDATE())          AS 'Year';
SELECT DATENAME(QUARTER, GETDATE())       AS 'Quarter';
SELECT DATENAME(MONTH, GETDATE())         AS 'Month
Name';
SELECT DATENAME(DAYOFYEAR, GETDATE())     AS 'DayOfYear';
SELECT DATENAME(DAY, GETDATE())           AS 'Day';
SELECT DATENAME(WEEK, GETDATE())          AS 'Week';
SELECT DATENAME(WEEKDAY, GETDATE())       AS 'Day of the
Week';
SELECT DATENAME(HOUR, GETDATE())          AS 'Hour';
SELECT DATENAME(MINUTE, GETDATE())        AS 'Minute';
SELECT DATENAME(SECOND, GETDATE())        AS 'Second';
SELECT DATENAME(MILLISECOND, GETDATE()) AS
'MilliSecond';
SELECT DATENAME(MICROSECOND, GETDATE()) AS
```

```
'MicroSecond';
SELECT DATENAME(NANOSECOND, GETDATE())  AS
'NanoSecond';
SELECT DATENAME(ISO_WEEK, GETDATE()) AS 'Week';
```

Step 13: Executing DATEPART() Function

```
--DATEPART() Function
SELECT DATEPART(YEAR, GETDATE()) AS
'Year';
SELECT DATEPART(QUARTER, GETDATE()) AS
'Quarter';
SELECT DATEPART(MONTH, GETDATE()) AS
'Month';
SELECT DATEPART(DAYOFYEAR, GETDATE()) AS
'DayOfYear';
SELECT DATEPART(DAY, GETDATE()) AS
'Day';
SELECT DATEPART(WEEK, GETDATE()) AS
'Week';
SELECT DATEPART(WEEKDAY, GETDATE()) AS
'WeekDay';
SELECT DATEPART(HOUR, GETDATE()) AS
'Hour';
SELECT DATEPART(MINUTE, GETDATE()) AS
'Minute';
SELECT DATEPART(SECOND, GETDATE()) AS
'Second';
SELECT DATEPART(MILLISECOND, GETDATE()) AS
'MilliSecond';
SELECT DATEPART(MICROSECOND, GETDATE()) AS
'MicroSecond';
SELECT DATEPART(NANOSECOND, GETDATE()) AS
'NanoSecond';
SELECT DATEPART(ISO_WEEK, GETDATE()) AS
'Week';
```

Step 14: Executing DATEDIFF() Function

```
--Date and Time Difference
SELECT DATEDIFF(DAY, 2019-31-01, 2019-01-01)
AS 'DateDif'    -- returns int
SELECT DATEDIFF_BIG(DAY, 2019-31-01, 2019-01-01)
AS 'DateDifBig' -- returns bigint
```