As you have learnt during the first semester (in IICT4001 – Database Design Concepts), data type selection is one of the most important decisions taken while creating table. Data type selection is also important in other advanced topics such as procedures and functions as everything is built upon this decision.

Given that T-SQL has a lot of built-in functions to be able to work with data, it is important to select the correct data type in its initial stages. All these in-built functions work by accepting an input, work on it and returning an output. The operations that can be carried out are highly dependent on the supplied input (in most cases this is dependent on the column's data type).

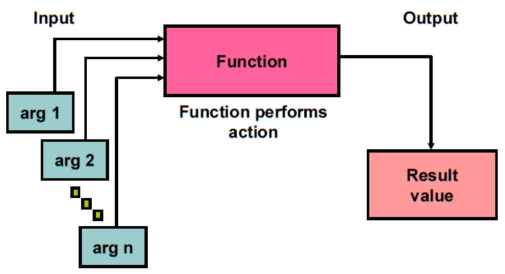
The built-in functions which are available in T-SQL are categorised into 4 different types

- *Scalar Functions*: also known as single row functions (covered in this topic)
- *Aggregate Functions*: also known as multiple row functions (covered in topic 5)
- *Rowset Functions*: this category of functions will return an object can be used like table references in an SQL statement
- *Ranking Functions*: this type of function will return a value for each row in a partition which is related to the ranking of the row.

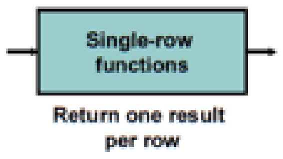
It is important to note that in this subject only the first two types of built-in functions will be considered. In both cases, there will not be an exhaustive coverage of all the functions as the list of functions is quite lengthy.

What is a ***Scalar Function***?

Functions are very powerful tools that are used in the SQL language to very important tasks. Functions typically accept one or more arguments as inputs and return a value as a reply.



In this topic we will be taking a closer look at different scalar functions which accept a single argument and produce a single output.



## Scalar Functions

As mentioned in the previous page these type of functions operate on a single row of data at a given point in time and return only one result per row. T-SQL offers a vast range of scalar functions which includes: *configuration functions, conversion functions, cursor functions, date functions, JSON functions, Logical functions, logical functions, mathematical functions, metadata functions, security functions, string functions, system functions, system statistical functions* and *Text/Image functions.*

In this topic we will be covering only the following type of functions:

- String Functions
- Mathematical Functions
- Date Functions
- Conversion Functions
- General Functions

### A. String Functions

As the name of this section suggests, these type of functions expect a string or a character based input. These functions will work upon the input and return either a string or a numerical value. These functions are said to be deterministic as they will always return the same value when a specific value is inputted.

String functions are typically split into two different categories:

i. *CASE Manipulation*: include functions which are specifically designed to handle the case of the letter of particular data.

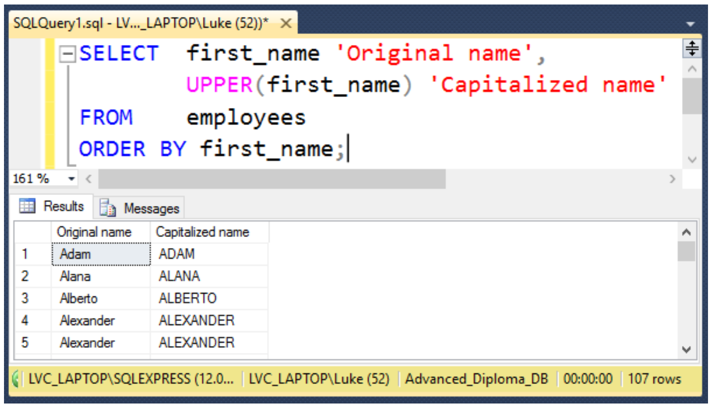
    - *LOWER Function:* [LOWER (character_expression)]

      This converts mixed-case or uppercase character strings to lowercase

    - *UPPER Function:* [UPPER (character_expression)]

      This converts all alpha character values within a particular string to lowercase.

*Example 1*: Write a query that will access the name in the employees table and display a result which includes all two columns – one having the original name and the other having the same name which is converted to capital letters. The smallest alphabetical name is displayed first



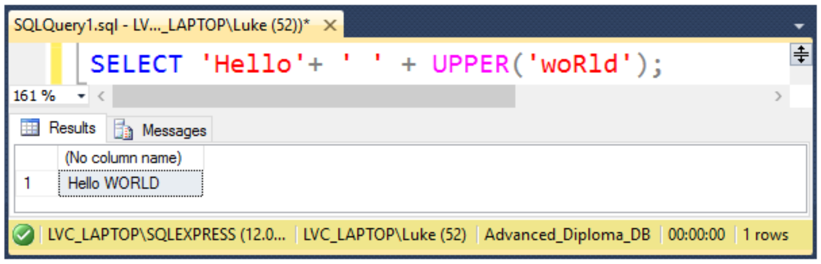
```sql
SELECT  first_name 'Original name',
        UPPER(first_name) 'Capitalized name'
FROM    employees
ORDER BY first_name;
```

| | Original name | Capitalized name |
|---|---|---|
| 1 | Adam | ADAM |
| 2 | Alana | ALANA |
| 3 | Alberto | ALBERTO |
| 4 | Alexander | ALEXANDER |
| 5 | Alexander | ALEXANDER |

```
LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 107 rows
```

*Figure 1 - Result of Example 1: second column capitalized*

*Example 2:* Write a query that will display the text Hello WORLD if you have two separate strings with the following values 'Hello' and 'woRld'.



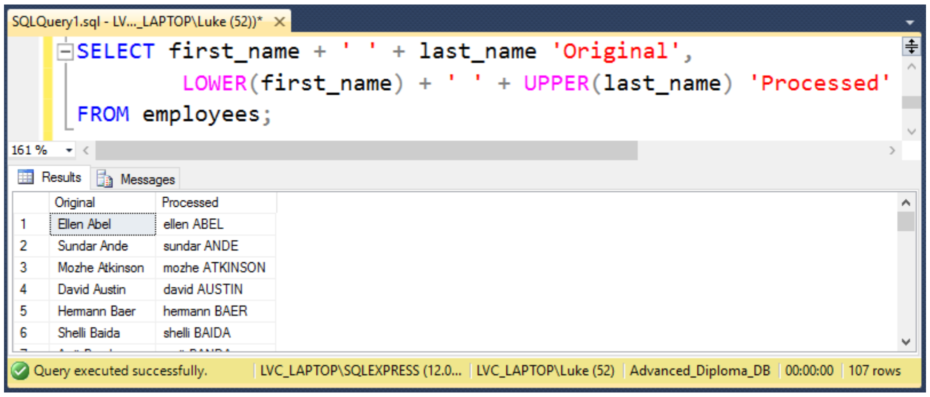
```sql
SELECT 'Hello'+ ' ' + UPPER('woRld');
```

| | (No column name) |
|---|---|
| 1 | Hello WORLD |

```
LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 1 rows
```

*Figure 2 -Result of example 2: two strings not in database one using a CASE function*

*Example 3*: Write a query that displays two columns 'Original' and 'Processed'. The first column includes the name and surname separated by a space (as found in the table employees) and the second column should have the name all in small letters and the surname all in capital letters.



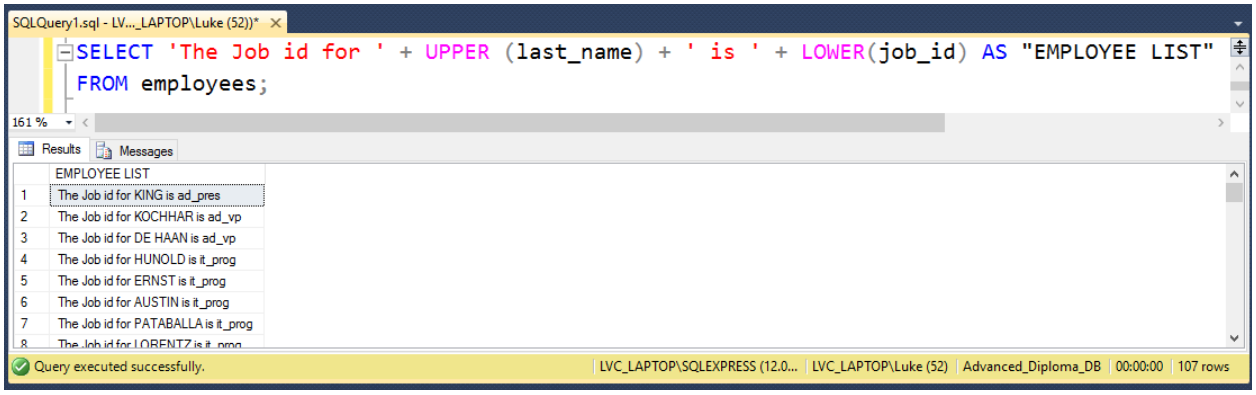
```sql
SELECT first_name + ' ' + last_name 'Original',
        LOWER(first_name) + ' ' + UPPER(last_name) 'Processed'
FROM employees;
```

| | Original | Processed |
|---|---|---|
| 1 | Ellen Abel | ellen ABEL |
| 2 | Sundar Ande | sundar ANDE |
| 3 | Mozhe Atkinson | mozhe ATKINSON |
| 4 | David Austin | david AUSTIN |
| 5 | Hermann Baer | hermann BAER |
| 6 | Shelli Baida | shelli BAIDA |

```
Query executed successfully. | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 107 rows
```

*Figure 3 - Result of example 3: two columns one with original data the other using functions*

*Example 4:* Write a query that will show one single column named 'Employee List'. This column should have the following format: **The job id for** *surname* **is** *job_id.* The surname should be written all in capital letters and the job_id should have only small case letters.



```
SELECT 'The Job id for ' + UPPER (last_name) + ' is ' + LOWER(job_id) AS "EMPLOYEE LIST"
FROM employees;
```

```
   EMPLOYEE LIST
1  The Job id for KING is ad_pres
2  The Job id for KOCHHAR is ad_vp
3  The Job id for DE HAAN is ad_vp
4  The Job id for HUNOLD is it_prog
5  The Job id for ERNST is it_prog
6  The Job id for AUSTIN is it_prog
7  The Job id for PATABALLA is it_prog
8  The Job id for LORENTZ is it_prog
```

Query executed successfully. | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 107 rows

*Figure 4 - Result of example 4: one single column with literal strings and table columns using functions*

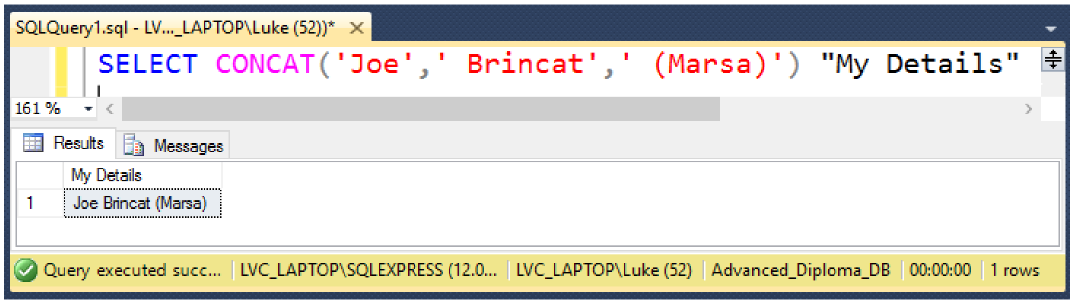
ii. *CHARACTER Manipulation:* the functions that are listed in this category are capable of manipulating characters found in input row. These functions can be used to handle/manipulate any part of some given data

- *CONCAT Function* [CONCAT (string_value1, string_value2, string_valueN)]

  This function is used to concatenate/join the listed value together. It equivalent to the + operator when applied to string values. It requires a minimum of two string values to be concatenated and any NULL values are automatically converted to an empty string.

  Eg: CONCAT('Hello', 'World') -> HelloWorld

*Example 5:* Write a query that will display your name, surname and town of residence (enclosed in brackets) in one single column named 'My Details'



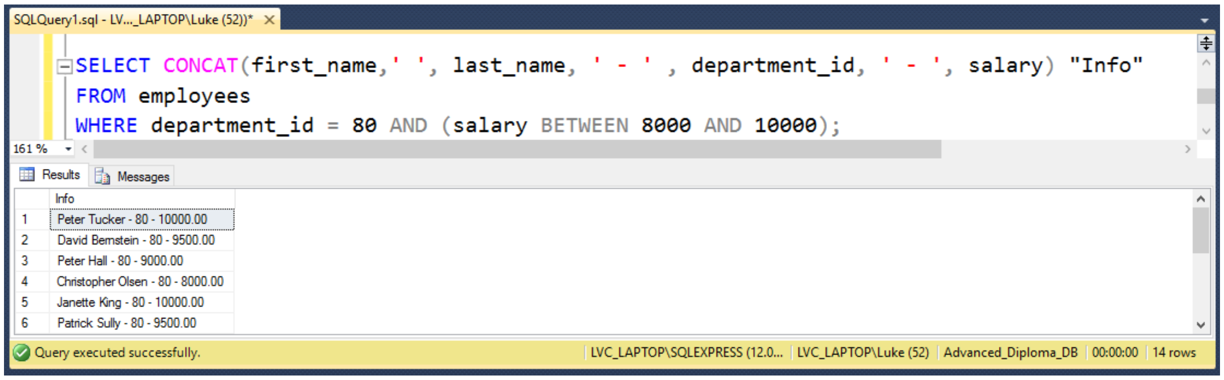
```
SELECT CONCAT('Joe',' Brincat',' (Marsa)') "My Details"
```

```
   My Details
1  Joe Brincat (Marsa)
```

Query executed succ... | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 1 rows

*Figure 5 - Result of example 5: CONCAT function to display personal details*

*Example 6*: Write a query that will display all the employees who work in department 80 but earn between 8000 and 10000. You are to include all the information in one single column named 'Info' using the CONCAT function.



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```
SELECT CONCAT(first_name,' ', last_name, ' - ' , department_id, ' - ', salary) "Info"
FROM employees
WHERE department_id = 80 AND (salary BETWEEN 8000 AND 10000);
```

161 %

Results | Messages

|   | Info |
|---|---|
| 1 | Peter Tucker - 80 - 10000.00 |
| 2 | David Bernstein - 80 - 9500.00 |
| 3 | Peter Hall - 80 - 9000.00 |
| 4 | Christopher Olsen - 80 - 8000.00 |
| 5 | Janette King - 80 - 10000.00 |
| 6 | Patrick Sully - 80 - 9500.00 |

Query executed successfully. | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 14 rows

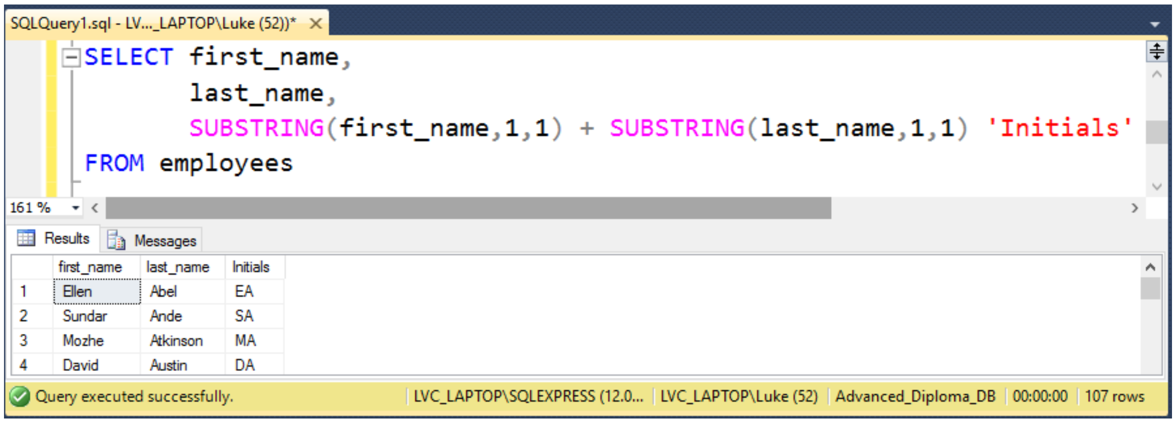
*Figure 6 -Result of example 6: CONCAT function used to join strings together*

- *SUBSTRING Function* [SUBSTRING(expression, s, l)]

  This function returns the specified characters for the expression, starting at character position s, l characters long.

  Eg: SUBSTRING('Hello World',1,5) -> Hello
  SUBSTRING('Hello World',2,2) -> el
  SUBSTRING('Hello World', 6,6) -> World

*Example 7:* Write a query that will list the name, surname and the initials of all the employees in three different columns. The third column should be named 'Initials'



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```
SELECT first_name,
       last_name,
       SUBSTRING(first_name,1,1) + SUBSTRING(last_name,1,1) 'Initials'
FROM employees
```

161 %

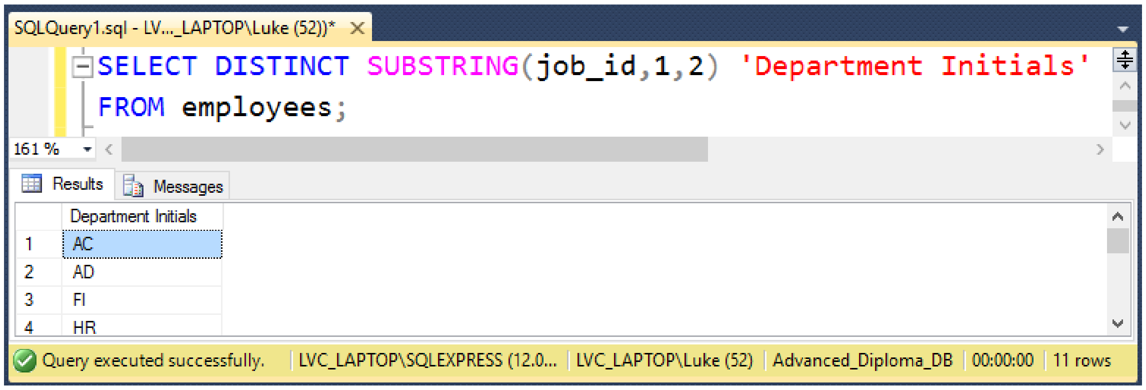
Results | Messages

|   | first_name | last_name | Initials |
|---|---|---|---|
| 1 | Ellen | Abel | EA |
| 2 | Sundar | Ande | SA |
| 3 | Mozhe | Atkinson | MA |
| 4 | David | Austin | DA |

Query executed successfully. | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 107 rows

*Figure 7 - Result of example 7: initials of name and surname through SUBSTRING*

*Example 8:* Write a query that will display the department initials using the job_id in the employees table. Any duplicate initials should be removed



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```sql
SELECT DISTINCT SUBSTRING(job_id,1,2) 'Department Initials'
FROM employees;
```

|  | Department Initials |
|---|---|
| 1 | AC |
| 2 | AD |
| 3 | FI |
| 4 | HR |

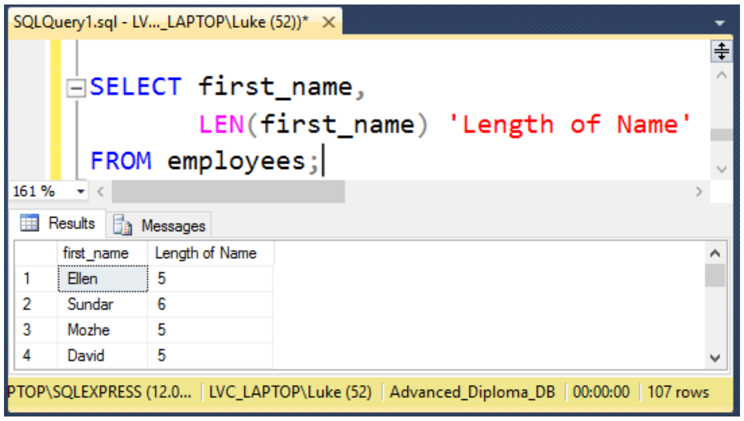
Query executed successfully. | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 11 rows

*Figure 8 - Result of example 8: unique initials obtained from part of the job_id*

- *LEN Function* [LEN(string_expression)]

  This function will return the number of characters found in the string expression.

*Example 9:* Write a query that will display the name of all employees and the number of characters used in the particular name



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```sql
SELECT first_name,
        LEN(first_name) 'Length of Name'
FROM employees;
```

|  | first_name | Length of Name |
|---|---|---|
| 1 | Ellen | 5 |
| 2 | Sundar | 6 |
| 3 | Mozhe | 5 |
| 4 | David | 5 |

PTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 107 rows

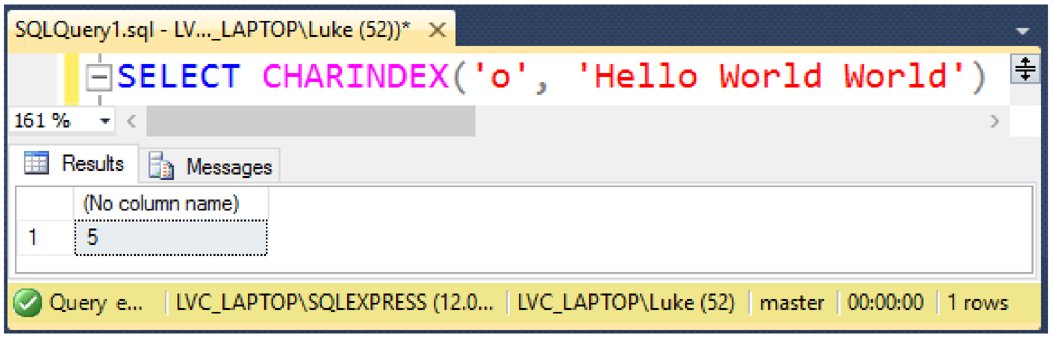
*Figure 9 - Result of example 9: number of characters in the used for each name*

- *CHARINDEX Function*
  [CHARINDEX(expressionToSearchFor, expressionToSearchIn[,start_location])]

  This function searches for an expression within another expression and returns its starting position if found.

  Eg: CHARINDEX( 'W', 'Hello World) -> 7

*Example 10:* Write a query that will find the position of the letter 'o' in the expression 'Hello World'.



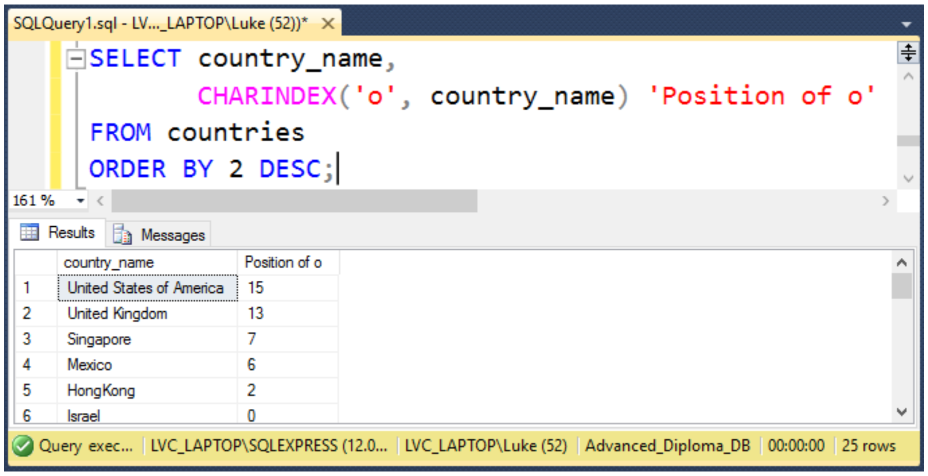
```
SELECT CHARINDEX('o', 'Hello World World')
```

```
   (No column name)
1  5
```

Query e... | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | master | 00:00:00 | 1 rows

*Figure 10 - Result of example 10: position of letter 'o'*

*Example 11:* Write a query that will find the position of the letter 'o' in country name. You are to order the result such that the country name which has the furthest 'o' in the name is displayed first.



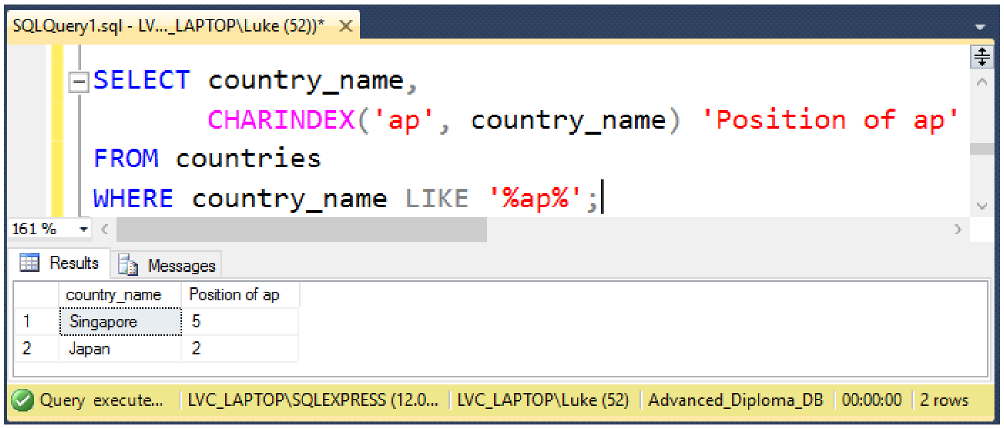
```
SELECT country_name,
       CHARINDEX('o', country_name) 'Position of o'
FROM countries
ORDER BY 2 DESC;
```

```
   country_name              Position of o
1  United States of America  15
2  United Kingdom            13
3  Singapore                 7
4  Mexico                    6
5  HongKong                  2
6  Israel                    0
```

Query exec... | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 25 rows

*Figure 11 - Result of example 11: position of 'o' in country name*

*Example 12:* Write a query that will search for all the countries who contain ‘ap’ and return the country name and the position where ‘ap’ appears in the country_name.


SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```
SELECT country_name,
       CHARINDEX('ap', country_name) 'Position of ap'
FROM countries
WHERE country_name LIKE '%ap%';
```

161 %

Results | Messages

|   | country_name | Position of ap |
|---|---|---|
| 1 | Singapore | 5 |
| 2 | Japan | 2 |

Query execute... | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 2 rows

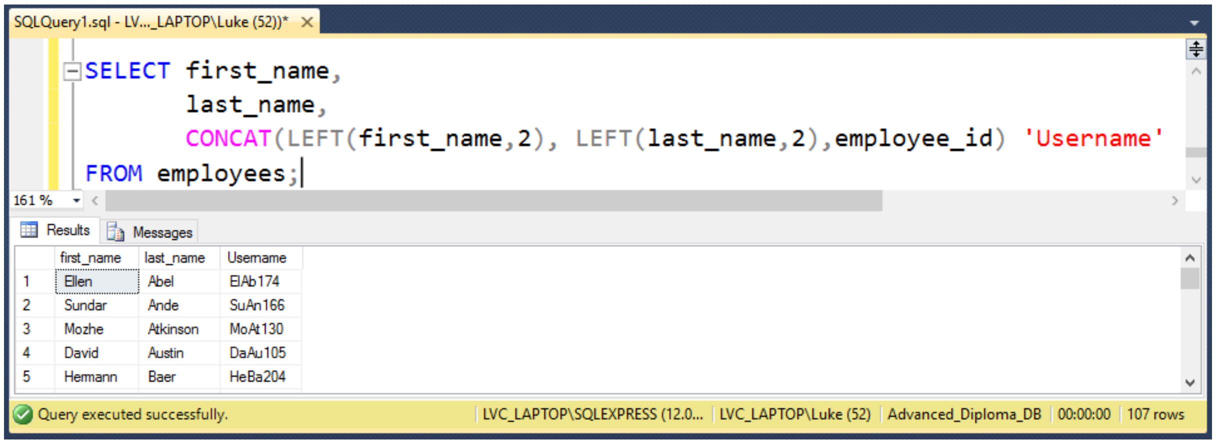
*Figure 5 - Result of example 12: the position of ‘ap’ in the country name for countries which have ‘ap’ in them*

- *LEFT Function* [LEFT(character_expression, integer_expression)]

  This function the left part of the character string with the specified number of characters in the integer_expression part.

  Eg: LEFT( ‘abcdef’, ‘2) -> ab

*Example 13:* Write a query that will return three rows (name, surname and new username). The name and surname are obtained directly from the employees table but the new username is obtained by concatenating the first two letters of the name and surname together with the employee_id.


SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```
SELECT first_name,
       last_name,
       CONCAT(LEFT(first_name,2), LEFT(last_name,2),employee_id) 'Username'
FROM employees;
```

161 %

Results | Messages

|   | first_name | last_name | Usemame |
|---|---|---|---|
| 1 | Ellen | Abel | ElAb174 |
| 2 | Sundar | Ande | SuAn166 |
| 3 | Mozhe | Atkinson | MoAt130 |
| 4 | David | Austin | DaAu105 |
| 5 | Hermann | Baer | HeBa204 |

Query executed successfully. | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 107 rows

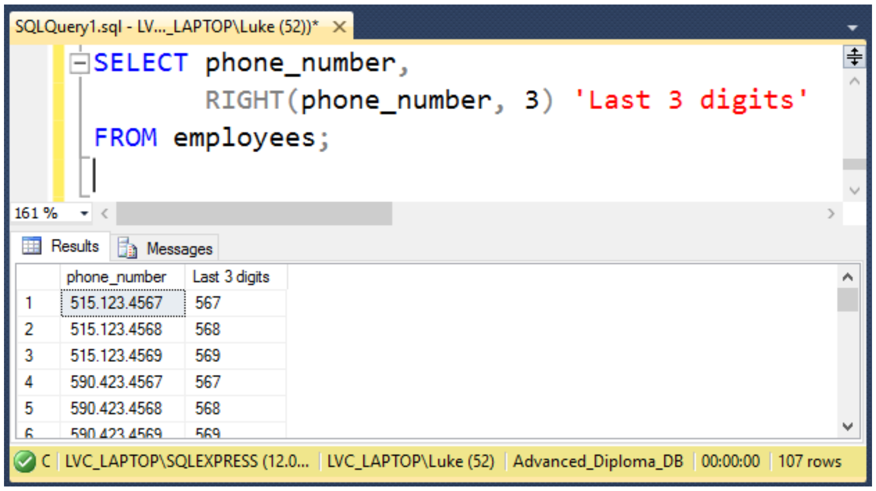
*Figure 63 – Result of example 13: new username calculated*

- *RIGHT Function* [RIGHT(character_expression, integer_expression)]

  This function is the opposite of the previous function. It returns the right part of a string with the specified number of characters.

  Eg: RIGHT( ‘abcdef’, ‘2) -> ef

*__Example 14__*: Write a query that will display the original telephone number of all the employees and the last three digits of the same telephone number. It is important that the second column is named ‘Last 3 digits’



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```
SELECT phone_number,
       RIGHT(phone_number, 3) 'Last 3 digits'
FROM employees;
```

161 %

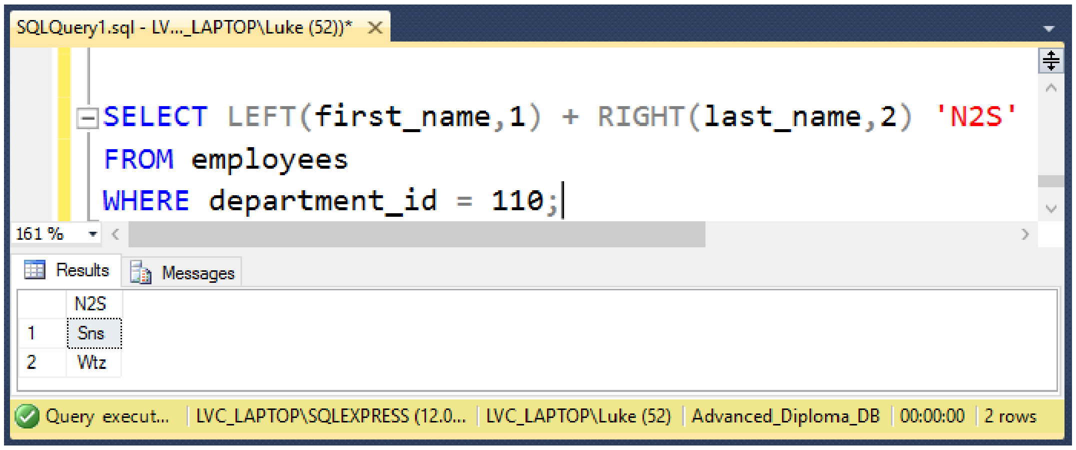
Results | Messages

```
   phone_number   Last 3 digits
1  515.123.4567   567
2  515.123.4568   568
3  515.123.4569   569
4  590.423.4567   567
5  590.423.4568   568
6  590.423.4569   569
```

C | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 107 rows

*Figure 14 - Result of example 14: last 3 digits of the telephone number*

*__Example 15:__* Write a query that will display the first letter of the name and the last two letters of the surname in one single column named ‘N2S’ for all those employees working in department 110.



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```
SELECT LEFT(first_name,1) + RIGHT(last_name,2) 'N2S'
FROM employees
WHERE department_id = 110;
```

161 %

Results | Messages

```
   N2S
1  Sns
2  Wtz
```

Query execut... | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 2 rows

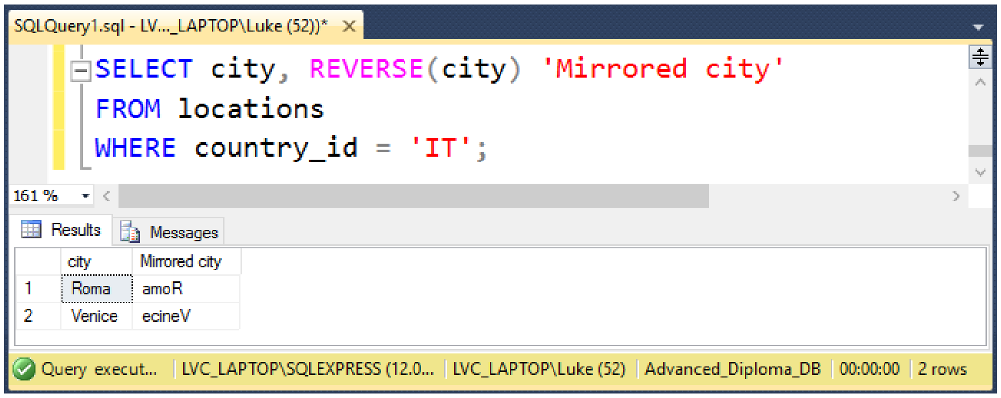
*Figure 15: Result of example 15: N2S contains the first letter of the name and the last 2 letters of the surname*

- *REVERSE Function* [REVERSE(string_expression)]

  This function will return the inputted string in reverse order

  Eg: REVERSE('Luke') -> ekuL

*Example 16:* Write a query that will return all cities which are found in Italy within the locations table. The query should return two columns – the original city name and the 'Mirrored City' which is the original name in reverse.



```sql
SELECT city, REVERSE(city) 'Mirrored city'
FROM locations
WHERE country_id = 'IT';
```

|   | city | Mirrored city |
|---|---|---|
| 1 | Roma | amoR |
| 2 | Venice | ecineV |

```
Query execut... | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 2 rows
```

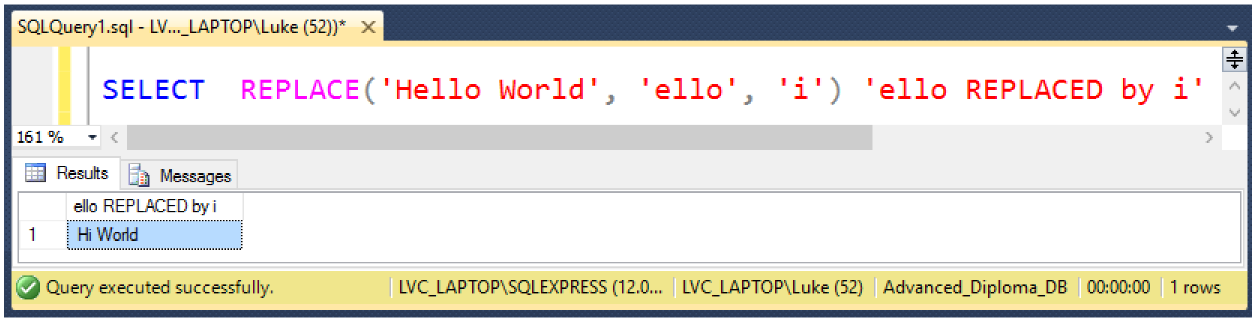
*Figure 16 – Result of example 16: the city name which is using the REVERSE keyword*

- *REPLACE Function* [REPLACE(string_expression, string_pattern, string_replacement)]

  This function will search for a particular pattern in a string and replaces by anther string

  Eg: REPLACE('JACK and JUE', 'J', 'BL') -> BLACK and BLUE

*Example 17:* Write a query that replaces the string 'ello' with 'i' in the intial string 'Hello World'



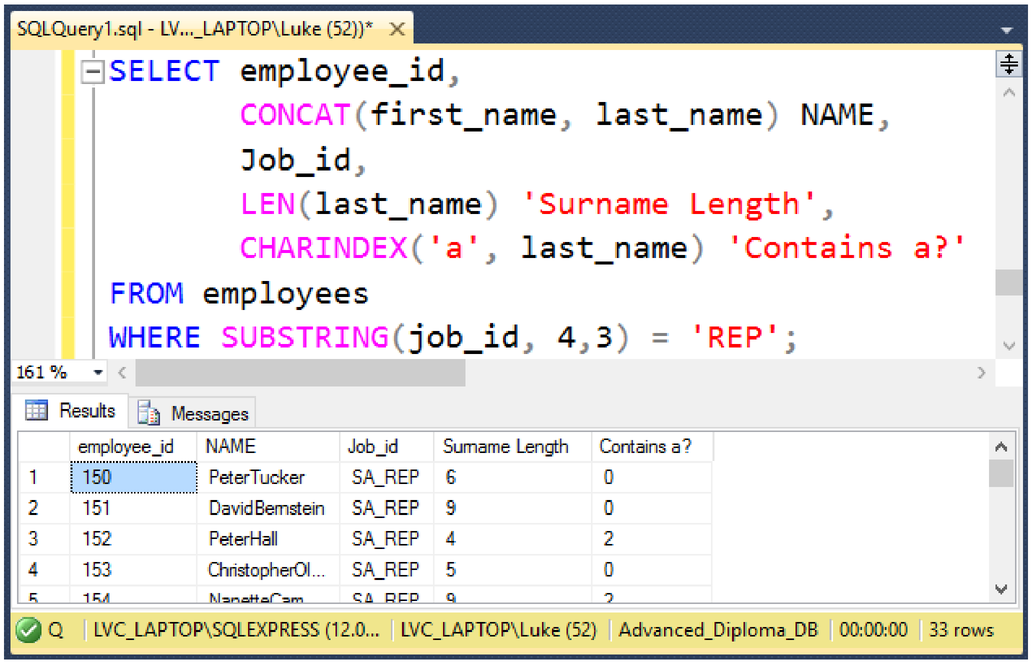
```sql
SELECT  REPLACE('Hello World', 'ello', 'i') 'ello REPLACED by i'
```

|   | ello REPLACED by i |
|---|---|
| 1 | Hi World |

```
Query executed successfully. | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 1 rows
```

*Figure 17 – Result of example 17: ello is being replaced with i*

*Example 18:* Write a query that will display the employee number, employee first name and last name joined together, the job_id, followed by the length of the employee last name and the numeric position of the letter a in the employee last name for all employees who have the string REP contained in the job_ID starting at the fourth position.


SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```
SELECT employee_id,
       CONCAT(first_name, last_name) NAME,
       Job_id,
       LEN(last_name) 'Surname Length',
       CHARINDEX('a', last_name) 'Contains a?'
FROM employees
WHERE SUBSTRING(job_id, 4,3) = 'REP';
```

161 %

Results | Messages

```
   employee_id  NAME            Job_id  Surname Length  Contains a?
1  150          PeterTucker     SA_REP  6               0
2  151          DavidBemstein   SA_REP  9               0
3  152          PeterHall       SA_REP  4               2
4  153          ChristopherOl...SA_REP  5               0
5  154          NanetteCam      SA_REP  9               2
```

Q | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 33 rows

*Figure 18 – Result of example 18: complex query returned*

*Test Question:* Modify the above so that the query will display the data for those employees whose last names end with the letter *n*. Make use of one of the functions explained in section A

## B. Mathematical Functions

Mathematical functions are used with functions which are numeric in nature. These functions accept a numeric input and return a numeric value in return. T-SQL has a number of mathematical function but in the subject we will be using only some of the most common ones.

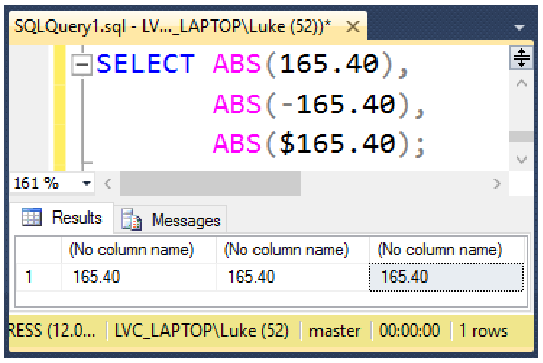
Some of the mathematical functions available will keep the same data type of the input for the output (Eg: ABS, CEILING, DEGREES, FLOOR, POWER, RADIANSE, SIGN) while trigonometric functions change the input value to float and return a float value (Eg: EXP, LOG, LOG10, SQUAR, SQRT).

In the next page you will find the list of mathematical functions which are to be covered in this topic for the scope of the assignment:

- *ABS Function* [ABS(numeric_expression)]

  This function will change any negative numerical inputs to positive values

*Example 19:* Write a query that will take the following numerical input (165.40, -165.40, $165.40) and return the positive equivalent value



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```
SELECT ABS(165.40),
       ABS(-165.40),
       ABS($165.40);
```

161 %

Results | Messages

|   | (No column name) | (No column name) | (No column name) |
|---|---|---|---|
| 1 | 165.40 | 165.40 | 165.40 |

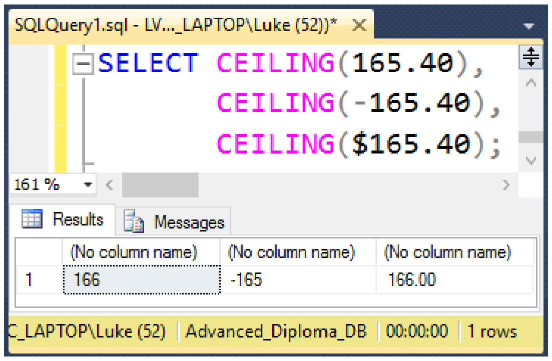
RESS (12.0... | LVC_LAPTOP\Luke (52) | master | 00:00:00 | 1 rows

*Figure 19 – Result of example19: Returning positive values*

- *CEILING Function* [CEILING(numeric_expression)]

  This function will return the smallest integer which is bigger than or equal to the specified numeric expression

*Example 20:* Write a query that will take the following numerical input (165.40, -165.40, $165.40) and return the smallest integer that is bigger than or equal to the given input



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```
SELECT CEILING(165.40),
       CEILING(-165.40),
       CEILING($165.40);
```

161 %

Results | Messages

|   | (No column name) | (No column name) | (No column name) |
|---|---|---|---|
| 1 | 166 | -165 | 166.00 |

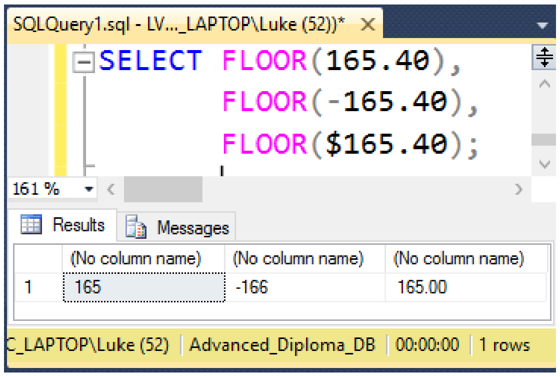
C_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 1 rows

*Figure 20 – Result of example 20: returns the ceiling of a numerical value*

- *FLOOR Function* [FLOOR(numeric_expression]

  This function will return the largest integer which is smaller than or equal to the specified numeric expression

*Example 21:* Write a query that will take the following numerical input (165.40, -165.40, $165.40) and return the largest integer that is smaller than or equal to the given input


SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```
SELECT FLOOR(165.40),
       FLOOR(-165.40),
       FLOOR($165.40);
```

161 %

Results | Messages

|   | (No column name) | (No column name) | (No column name) |
|---|---|---|---|
| 1 | 165 | -166 | 165.00 |

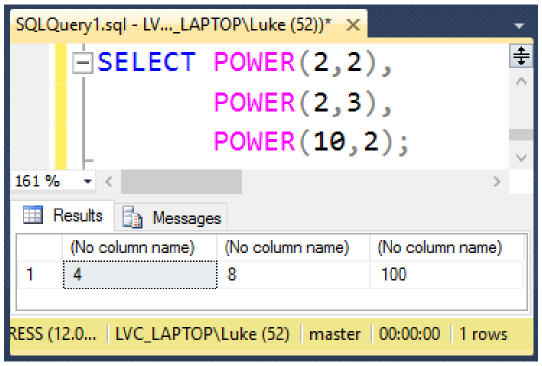
C_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 1 rows

*Figure 21- Result of example 21: returns the floor value of the given input*

- *POWER Function* [POWER(float_expression, y)]

  This function will return the value that is included in the float_expression part to the power of y.

*Example 22:* Write a query that will take the a value of 2 and raise it to the power of 2 and 3 and then raise the value 10 to the power of 2


SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```
SELECT POWER(2,2),
       POWER(2,3),
       POWER(10,2);
```

161 %

Results | Messages

|   | (No column name) | (No column name) | (No column name) |
|---|---|---|---|
| 1 | 4 | 8 | 100 |

RESS (12.0... | LVC_LAPTOP\Luke (52) | master | 00:00:00 | 1 rows

*Figure 22 – Result of example 22: values raised to a power*

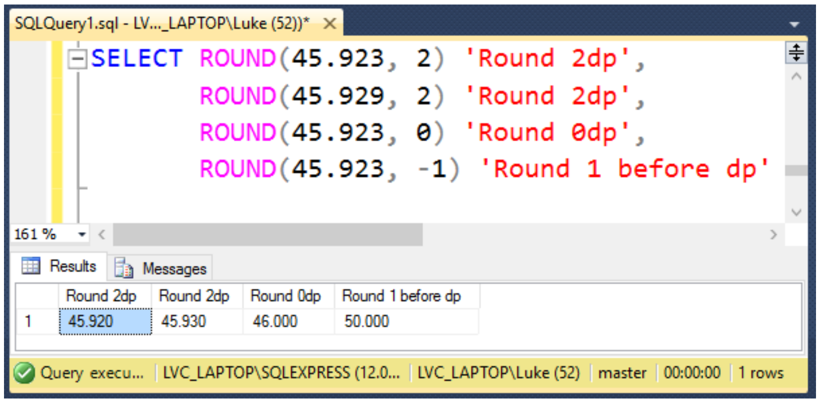
- *ROUND Function* [ROUND(numeric_expression, length [,function])]

  This function will take a numerical input and round/truncates the result to a specific length or precision.

  When the length value is positive the input is rounded to the number of decimal positions specified. When the length value is negative the input is rounded on the left side of the decimal point.

  The function value specifies the operation to be performed. If this value is omitted, the default value of 0 is taken – meaning that rounding is performed. If the value differs from 0 the operation will change to truncate.

*Example 23:* Write a query that will round the following inputs (45.923, 45.929, 45.923, 45.923) using the following decimal points (2, 2, 0, -1)

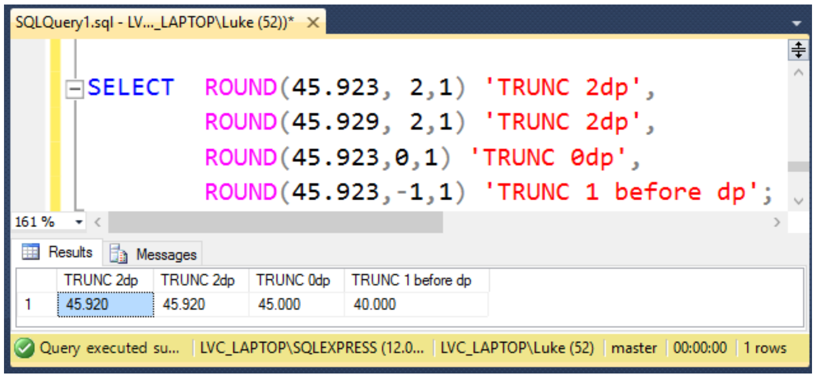


```
SELECT ROUND(45.923, 2) 'Round 2dp',
       ROUND(45.929, 2) 'Round 2dp',
       ROUND(45.923, 0) 'Round 0dp',
       ROUND(45.923, -1) 'Round 1 before dp'
```

```
   Round 2dp  Round 2dp  Round 0dp  Round 1 before dp
1  45.920     45.930     46.000     50.000
```

Figure 23 – Result of example 23: Values rounded to given decimal place

*Example 24:* Write a query that will truncate the following inputs (45.923, 45.929, 45.923, 45.923) using the following decimal points (2, 2, 0, -1)



```
SELECT  ROUND(45.923, 2,1) 'TRUNC 2dp',
        ROUND(45.929, 2,1) 'TRUNC 2dp',
        ROUND(45.923,0,1) 'TRUNC 0dp',
        ROUND(45.923,-1,1) 'TRUNC 1 before dp';
```

```
   TRUNC 2dp  TRUNC 2dp  TRUNC 0dp  TRUNC 1 before dp
1  45.920     45.920     45.000     40.000
```

Figure 24: Result of example 24: values truncated to given decimal place

NOTE: The difference between round and truncate. Round gives a value to the nearest, while truncate removes/deleted the number from a particular position.

## C. Date Functions

Date functions can be applied to data which falls under one of the many different type of date and time data types. The date and time data types found in SQL server include *time, date, smalldatetime, datetime, datetime2,* and *datetimeoffset.* For more details related to any of these data types you are to refer to Microsoft's online documentation.

Date functions are categorised in many different sub-sections. In this topic we will take a closer look at some of the widely used functions in this category:

i. *System Date and Time Functions*: The data and time values that are stored in the database are derived from the operating system which hosts the SQL Server installation. SQL Server makes use of an in-built API (GetSystemTimeAsFileTim()) to return date and time values. The accuracy of these values depends on the hardware and operating system version that SQL Server is working on.

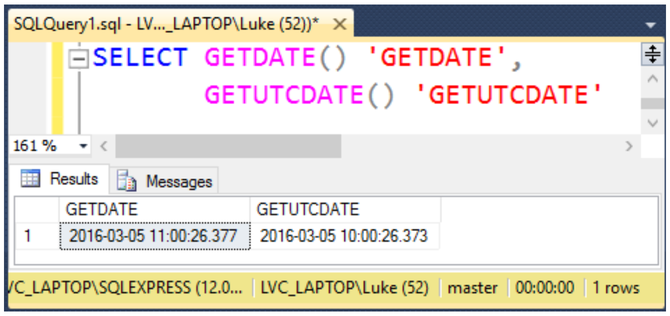
- *GETDATE Function* [GETDATE()]

  This function returns the date and time of the computer on which the SQL Server instance is working. TimeZone is not included.

- *GETUTCDATE Function* [GETUTCDATE()]

  This function returns the date and time of the computer on which the SQL Server instance is working. The output of this function includes a UTC time (Coordinated Universal Time). This function returns the date and time which is independent of Time Zones

*Example 25:* Write a query that will display the current date of your location, together with the date and time in UTC format. Notice that the UTC time is 1hr less as the server is running on a GMT+1 timezone.


SQLQuery1.sql - LV..._LAPTOP\Luke (52))*
161 %
Results
Messages
VC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | master | 00:00:00 | 1 rows

```
SELECT GETDATE() 'GETDATE',
       GETUTCDATE() 'GETUTCDATE'
```

| | GETDATE | GETUTCDATE |
|---|---|---|
| 1 | 2016-03-05 11:00:26.377 | 2016-03-05 10:00:26.373 |

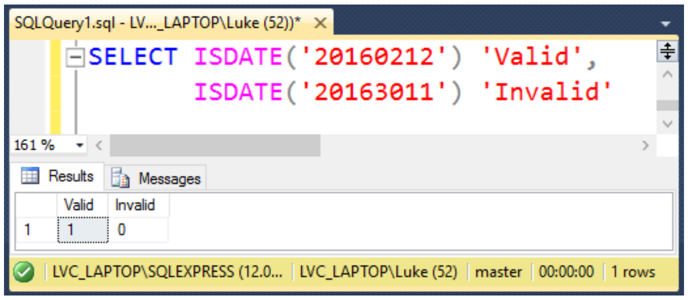
*Figure 25 – Result of example 25: Return the current time on server and UTC time.*

ii. *Date and Time Validation Function:* these is only one function in this category and its function is really important to make sure that a date is correctly formatted.

- *ISDATE Function* [ISDATE(expression)]

  This function will return a value of 1 if the expression contains a valid date, time or datetime value. A value of 0 will be returned if the expression is an invalid datetime value or a datetime2 value.

*<u>Example 26:</u>* Write a query that will check if a date is valid one or not. The first date entered is correct but the second date is invalid as there is no month with a value of 30



```
SELECT ISDATE('20160212') 'Valid',
       ISDATE('20163011') 'Invalid'
```

```
   Valid  Invalid
1  1      0
```

SQLQuery1.sql - LV..._LAPTOP\Luke (52))*
161 %
Results
Messages
LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | master | 00:00:00 | 1 rows

*Figure 26 – Result of example 26: ISDATE function checking date validity*

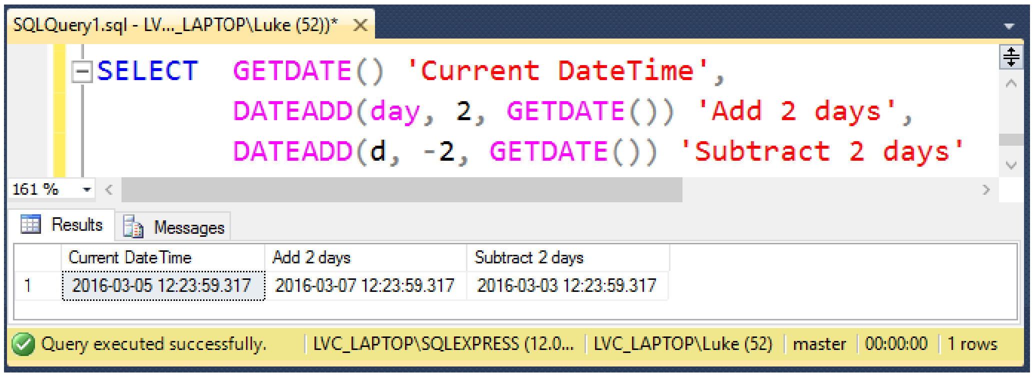
iii. *Arithmetic with dates:* this category of functions widely used as it helps in manipulating date and time values using arithmetic function.

- *DATEADD Function* [DATEADD(datepart, number, date)]
  The DATEADD function will return a date which has the value included in the number parameter added to the datepart within the specified date. The number parameter is a signed integer (can be a positive or negative integer)

  The datepart parameter can have many different arguments. While using this function one can use either the full name of the argument or else an abbreviation. The following list includes all the valid *datepart* arguments [fullname (abbreviations)]: year (yy, yyyy), quarter (qq, q), month (mm, m), dayofyear (dy, y), day (dd, d), week (wk, ww), weekday (dw, w), hour (hh), minute (mi, n), second (ss, s), millisecond (ms), microsecond (mcs), nanosecond (ns)

  NOTE: If you attempt to add a month and the number of days in the returned month exceeds the maximum number of days in that month, then the last day of the month is returned (Eg: DATEADD(month, 1, ‘20140831’ -> 20140930)

*Example 27*: Write a query that will take the current date and time and adds 2 days in a new column and then subtracts 2 days in the next column. The column names should read 'Current DateTime', 'Add 2 days' and 'Subtract 2 days'.



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```
SELECT  GETDATE() 'Current DateTime',
        DATEADD(day, 2, GETDATE()) 'Add 2 days',
        DATEADD(d, -2, GETDATE()) 'Subtract 2 days'
```

|   | Current DateTime | Add 2 days | Subtract 2 days |
|---|---|---|---|
| 1 | 2016-03-05 12:23:59.317 | 2016-03-07 12:23:59.317 | 2016-03-03 12:23:59.317 |

Query executed successfully. | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | master | 00:00:00 | 1 rows

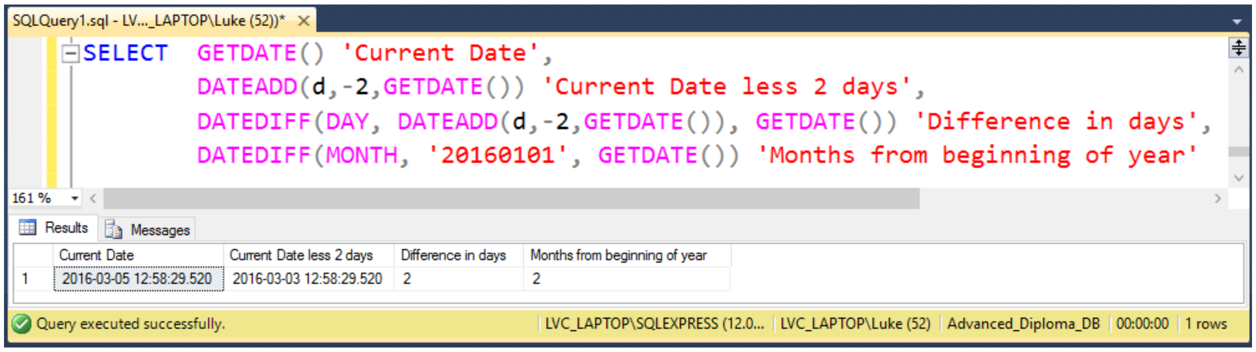
*Figure 27: Result of example 27: Current date and time, added by 2 days and subtracted by 2 days*

- *DATEDIFF Function* [DATEDIFF(datepart, startdate, enddate)]

  This function will return a signed integer for the difference between the startdate and endate of a particular datepart. In the case of very large differences the DATEDIFF_BIG function is to be applied.

  This function can be used in the SLECT, WHERE, HAVING, GROUP BY and ORDER BY clauses.

*Example 28:* Write a query that will display the current date, the same date subtracted by 2 days, the difference in days between the first two columns and the difference in months from the current date to the first day of the same year



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```
SELECT  GETDATE() 'Current Date',
        DATEADD(d,-2,GETDATE()) 'Current Date less 2 days',
        DATEDIFF(DAY, DATEADD(d,-2,GETDATE()), GETDATE()) 'Difference in days',
        DATEDIFF(MONTH, '20160101', GETDATE()) 'Months from beginning of year'
```

|   | Current Date | Current Date less 2 days | Difference in days | Months from beginning of year |
|---|---|---|---|---|
| 1 | 2016-03-05 12:58:29.520 | 2016-03-03 12:58:29.520 | 2 | 2 |

Query executed successfully. | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 1 rows

*Figure 28 – Result of example 28: Difference in dates (DATEDIFF) is used to subtract dates*

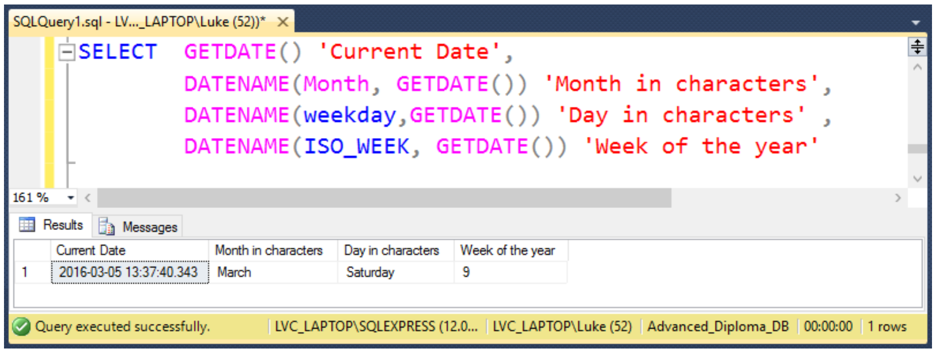
iv. *Date and Time retrieval functions:* these are function which can be used to extract information from a given date

- *DATENAME Function* [DATENAME(datepart, date)]

  This function will return a string which includes the specified datepart for the given date

  The datepart parameter can have many different arguments as listed below: year (yy, yyyy,yy), quarter (qq, q), month (mm, m), dayofyear (dy, y), day (dd, d), week (wk, ww), weekday (dw), hour (hh), minute (n), second (ss, s), millisecond (ms), microsecond (mcs), nanosecond (ns), ISO_WEEK (this is the week number in the year)

*<u>Example 29:</u>* Write a query that displays the current date, followed by the month and day in string format. Also list the week number of the current date



```
SELECT  GETDATE() 'Current Date',
        DATENAME(Month, GETDATE()) 'Month in characters',
        DATENAME(weekday,GETDATE()) 'Day in characters' ,
        DATENAME(ISO_WEEK, GETDATE()) 'Week of the year'
```

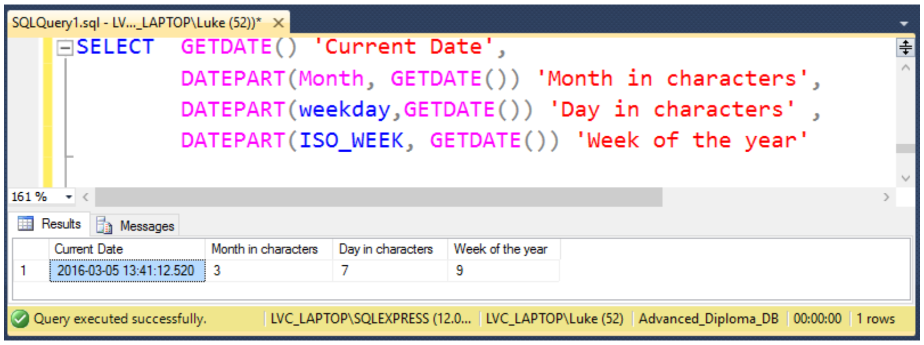
|   | Current Date | Month in characters | Day in characters | Week of the year |
|---|---|---|---|---|
| 1 | 2016-03-05 13:37:40.343 | March | Saturday | 9 |

Query executed successfully. | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 1 rows

*Figure 29 – Result of example 29: extracting important string information from a given date*

- *DATEPART Function* [DATEPART(datepart, date)]

  DATEPART will return an integer value which represents the specified datepart of the inputted date. This is the equivalent of DATENAME but returns a numerical value.



```
SELECT  GETDATE() 'Current Date',
        DATEPART(Month, GETDATE()) 'Month in characters',
        DATEPART(weekday,GETDATE()) 'Day in characters' ,
        DATEPART(ISO_WEEK, GETDATE()) 'Week of the year'
```

|   | Current Date | Month in characters | Day in characters | Week of the year |
|---|---|---|---|---|
| 1 | 2016-03-05 13:41:12.520 | 3 | 7 | 9 |

Query executed successfully. | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 1 rows

*Figure 30 – Same question as in example 29 but using the DATEPART function instead*

- *DAY Function* [DAY(date)]

  The day in the inputted date will be returned in an integer format

  Eg: DAY('19990910') -> 10

- *MONTH Function* [MONTH(date)]

  The numerical value of the month in the inputted date will be returned

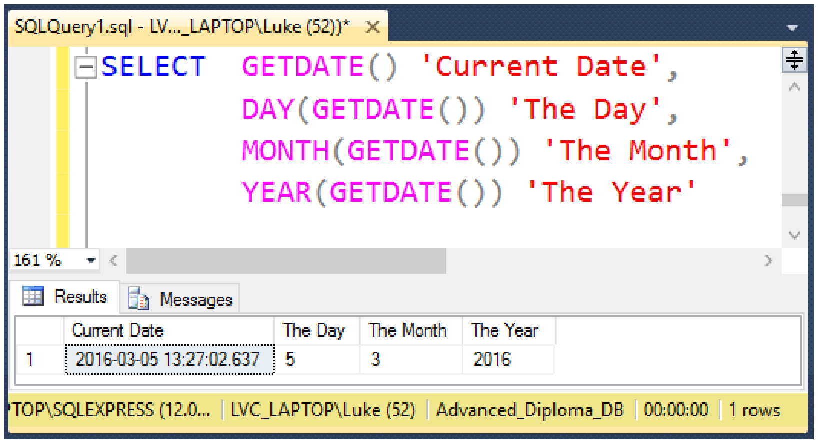
  Eg: MONTH('20160116') -> 1

- *Year Function* [YEAR(date)]

  The integer representing the year in a given date will be returned

  Eg: YEAR('20140404') -> 2014

*__Example 30:__* Write a query that displays the current date and time and then returns the day, month and year in the next three columns. The columns should be named 'Current date', ' The Day', 'The Month' and 'The Year' respectively.



```
SELECT  GETDATE() 'Current Date',
        DAY(GETDATE()) 'The Day',
        MONTH(GETDATE()) 'The Month',
        YEAR(GETDATE()) 'The Year'
```

|   | Current Date | The Day | The Month | The Year |
|---|---|---|---|---|
| 1 | 2016-03-05 13:27:02.637 | 5 | 3 | 2016 |

TOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 1 rows

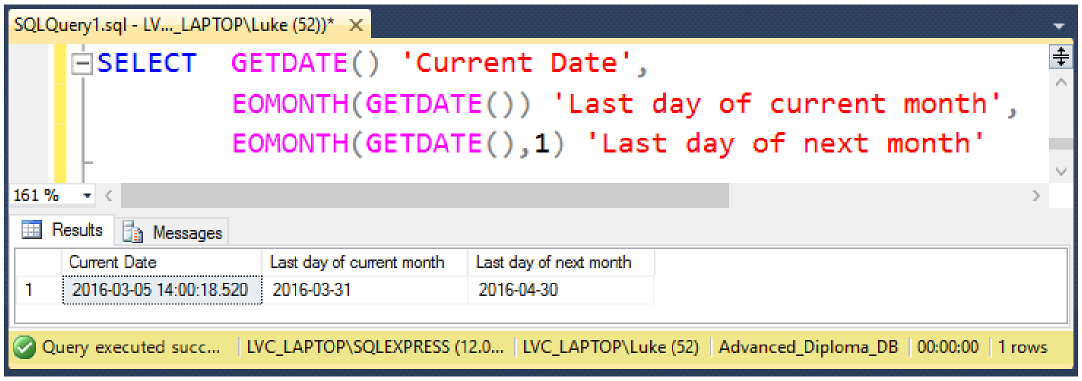
*Figure 31 – Result of example 30: The numerical day, month and year values for the current date were outputted*

v. *Date and time modification functions:*

- *EOMONTH Function* [EOMONTH(start_date [,month_to_add])]

  This function will the last day of the month of the specified date. It is also possible to add a number of months to the given date and the last day of the moth for the new date will be returned

*Example 31:* Write a query that will return the current date and the last day of the month of the current month and next month



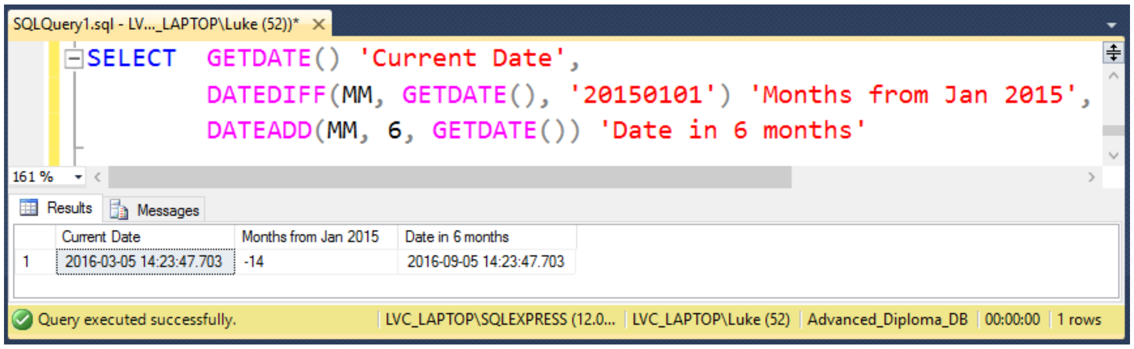
```
SELECT  GETDATE() 'Current Date',
        EOMONTH(GETDATE()) 'Last day of current month',
        EOMONTH(GETDATE(),1) 'Last day of next month'
```

|   | Current Date | Last day of current month | Last day of next month |
|---|---|---|---|
| 1 | 2016-03-05 14:00:18.520 | 2016-03-31 | 2016-04-30 |

Query executed succ... | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 1 rows

*Figure 32 – Result of example 31: end of month date*

*Example 32:* Write a query that will display the current date, the number of months between the 1st January 2015 and the current date, the date in 6 months' time.



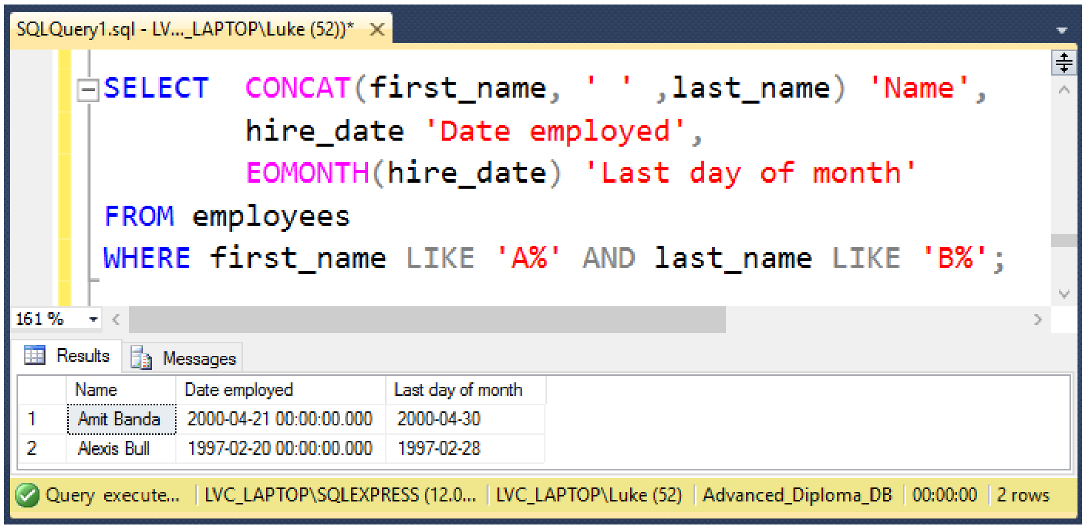
```
SELECT  GETDATE() 'Current Date',
        DATEDIFF(MM, GETDATE(), '20150101') 'Months from Jan 2015',
        DATEADD(MM, 6, GETDATE()) 'Date in 6 months'
```

|   | Current Date | Months from Jan 2015 | Date in 6 months |
|---|---|---|---|
| 1 | 2016-03-05 14:23:47.703 | -14 | 2016-09-05 14:23:47.703 |

Query executed successfully. | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 1 rows

*Figure 33 – Result of example 32: current date calculations*

*Example 33:* Write a query that will return the name and surname in one single column, followed by the date employeed and the last day of the month for each hire date. You are to include only the employees whose name starts with an 'A' and surname starts with a 'B'



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```
SELECT  CONCAT(first_name, ' ' ,last_name) 'Name',
        hire_date 'Date employed',
        EOMONTH(hire_date) 'Last day of month'
FROM employees
WHERE first_name LIKE 'A%' AND last_name LIKE 'B%';
```

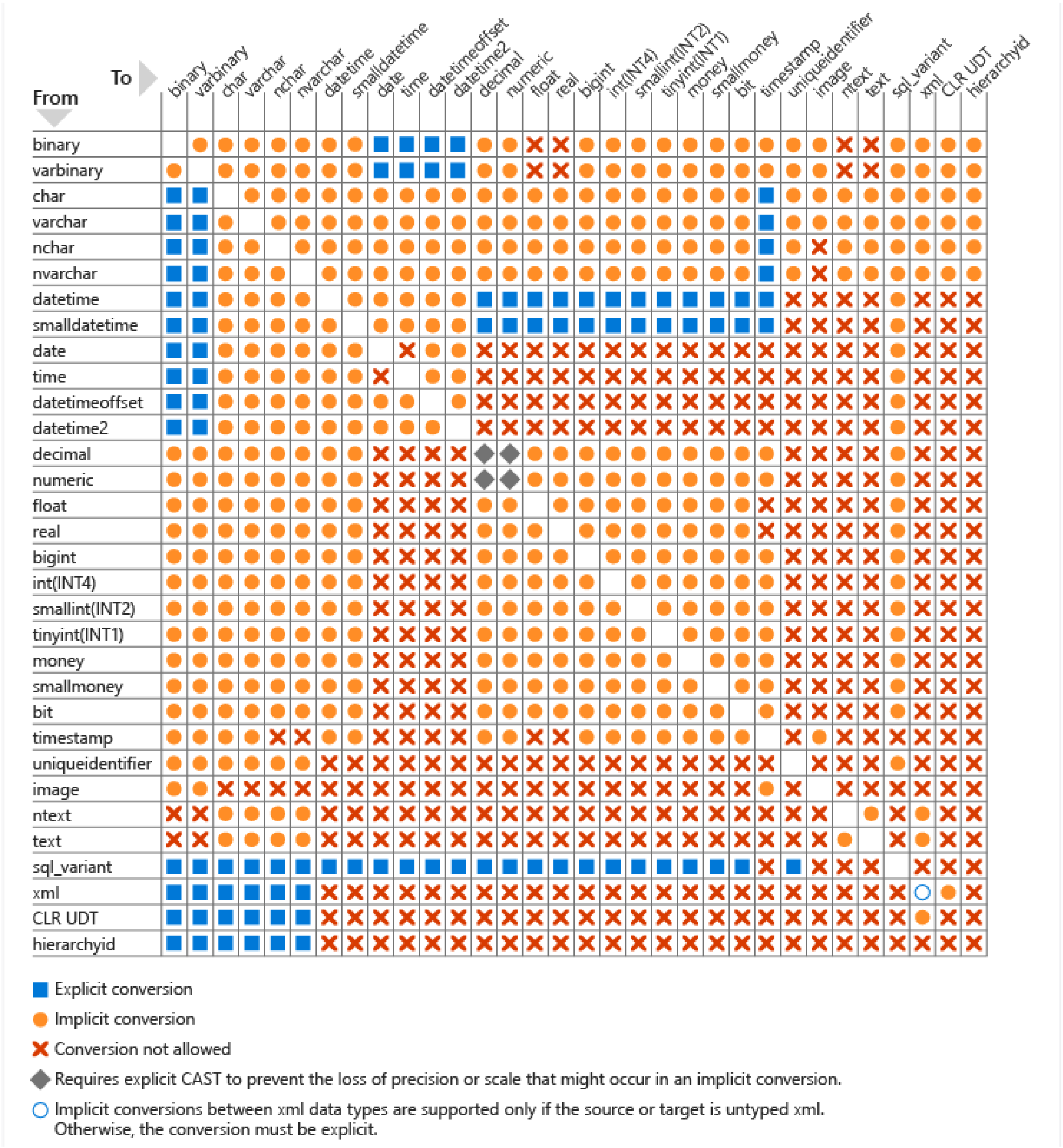
```
   Name         Date employed             Last day of month
1  Amit Banda   2000-04-21 00:00:00.000   2000-04-30
2  Alexis Bull  1997-02-20 00:00:00.000   1997-02-28
```

Query execute... | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 2 rows

*Figure 34 – Result of example 33: the last day of the month for a given hiredate*

## D. Conversion Functions

Conversions functions are really important whenever data is moved, compared or combined. These three operations usually require the need to convert data from one data type to another. These functions are really important as if they do not exist, many other functions will become unusable.



| From \ To | binary | varbinary | char | varchar | nchar | nvarchar | datetime | smalldatetime | date | time | datetimeoffset | datetime2 | decimal | numeric | float | real | bigint | int(INT4) | smallint(INT2) | tinyint(INT1) | money | smallmoney | bit | timestamp | uniqueidentifier | image | ntext | text | sql_variant | xml | CLR UDT | hierarchyid |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| binary |  | ● | ● | ● | ● | ● | ● | ● | ■ | ■ | ■ | ■ | ● | ● | ✖ | ✖ | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ● | ● | ● | ● |
| varbinary | ● |  | ● | ● | ● | ● | ● | ● | ■ | ■ | ■ | ■ | ● | ● | ✖ | ✖ | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ● | ● | ● | ● |
| char | ■ | ■ |  | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ■ | ● | ● | ● | ● | ● | ● | ● | ● |
| varchar | ■ | ■ | ● |  | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ■ | ● | ● | ● | ● | ● | ● | ● | ● |
| nchar | ■ | ■ | ● | ● |  | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ■ | ● | ✖ | ● | ● | ● | ● | ● | ● |
| nvarchar | ■ | ■ | ● | ● | ● |  | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ■ | ● | ✖ | ● | ● | ● | ● | ● | ● |
| datetime | ■ | ■ | ● | ● | ● | ● |  | ● | ● | ● | ● | ● | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ✖ | ✖ | ✖ | ✖ | ● | ✖ | ✖ | ✖ |
| smalldatetime | ■ | ■ | ● | ● | ● | ● | ● |  | ● | ● | ● | ● | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ✖ | ✖ | ✖ | ✖ | ● | ✖ | ✖ | ✖ |
| date | ■ | ■ | ● | ● | ● | ● | ● | ● |  | ✖ | ● | ● | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ● | ✖ | ✖ | ✖ |
| time | ■ | ■ | ● | ● | ● | ● | ● | ● | ✖ |  | ● | ● | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ● | ✖ | ✖ | ✖ |
| datetimeoffset | ■ | ■ | ● | ● | ● | ● | ● | ● | ● | ● |  | ● | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ● | ✖ | ✖ | ✖ |
| datetime2 | ■ | ■ | ● | ● | ● | ● | ● | ● | ● | ● | ● |  | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ● | ✖ | ✖ | ✖ |
| decimal | ● | ● | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ◆ | ◆ | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ● | ✖ | ✖ | ✖ |
| numeric | ● | ● | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ◆ | ◆ | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ● | ✖ | ✖ | ✖ |
| float | ● | ● | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ● | ● |  | ● | ● | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ✖ | ● | ✖ | ✖ | ✖ |
| real | ● | ● | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ● | ● | ● |  | ● | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ✖ | ● | ✖ | ✖ | ✖ |
| bigint | ● | ● | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ● | ● | ● | ● |  | ● | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ● | ✖ | ✖ | ✖ |
| int(INT4) | ● | ● | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ● | ● | ● | ● | ● |  | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ● | ✖ | ✖ | ✖ |
| smallint(INT2) | ● | ● | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ● | ● | ● | ● | ● | ● |  | ● | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ● | ✖ | ✖ | ✖ |
| tinyint(INT1) | ● | ● | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ● | ● | ● | ● | ● | ● | ● |  | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ● | ✖ | ✖ | ✖ |
| money | ● | ● | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ● | ● | ● | ● | ● | ● | ● | ● |  | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ● | ✖ | ✖ | ✖ |
| smallmoney | ● | ● | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ● | ● | ● | ● | ● | ● | ● | ● | ● |  | ● | ● | ✖ | ✖ | ✖ | ✖ | ● | ✖ | ✖ | ✖ |
| bit | ● | ● | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |  | ● | ✖ | ✖ | ✖ | ✖ | ● | ✖ | ✖ | ✖ |
| timestamp | ● | ● | ● | ● | ✖ | ✖ | ● | ● | ✖ | ✖ | ✖ | ✖ | ● | ● | ✖ | ✖ | ● | ● | ● | ● | ● | ● | ● |  | ✖ | ● | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ |
| uniqueidentifier | ● | ● | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ |  | ✖ | ✖ | ✖ | ● | ✖ | ✖ | ✖ |
| image | ● | ● | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ● | ✖ |  | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ |
| ntext | ✖ | ✖ | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ |  | ● | ✖ | ● | ✖ | ✖ |
| text | ✖ | ✖ | ● | ● | ● | ● | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ● |  | ✖ | ● | ✖ | ✖ |
| sql_variant | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ✖ | ■ | ✖ | ✖ | ✖ |  | ✖ | ✖ | ✖ |
| xml | ■ | ■ | ■ | ■ | ■ | ■ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ○ | ● | ✖ |
| CLR UDT | ■ | ■ | ■ | ■ | ■ | ■ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ● | ✖ | ✖ |
| hierarchyid | ■ | ■ | ■ | ■ | ■ | ■ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ | ✖ |

■ Explicit conversion

● Implicit conversion

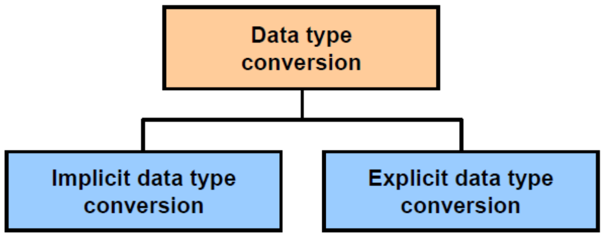
✖ Conversion not allowed

◆ Requires explicit CAST to prevent the loss of precision or scale that might occur in an implicit conversion.

○ Implicit conversions between xml data types are supported only if the source or target is untyped xml. Otherwise, the conversion must be explicit.

*Figure 35 – The above image was obtained from the Onlne documentation found in MSDN. It shows all the datatype available in SQL Server and the possible data type conversions*

Conversion functions are usually categorised in two: IMPLICIT and EXPLICIT functions



i) *Implicit functions*: these are functions which are automatically carried out by the server and hence the user is not required to specify them in the statement. The server recognizes the need for the conversion and automatically converts the data from one data type to another.
Eg: salary = '17000' results in an implicit conversion of string '17000' to number 17000.

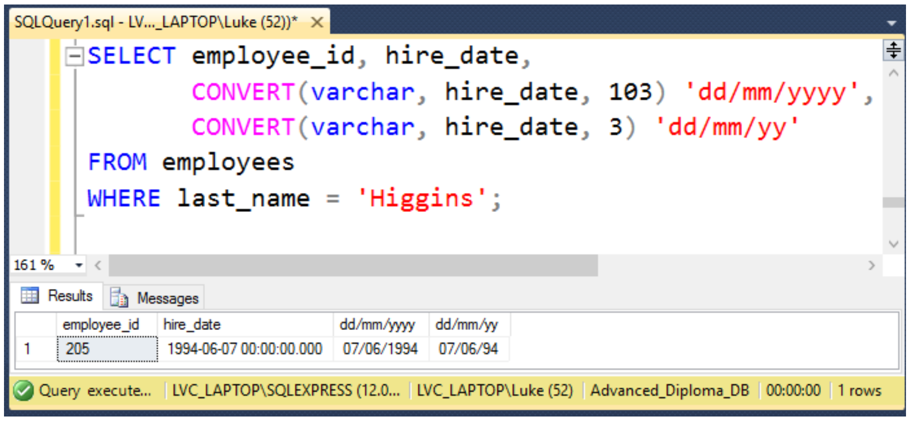
ii) *Explicit functions*: these function are not done automatically by the server and are therefore required to be done by the user. It is recommended that when in doubt you perform the conversion to ensure reliability while executing statements. The most popular functions in this section are:

- *CAST Function:* [CAST(expression AS data_type [(length)])]
- *CONVERT Function* [CONVERT (data_type [(length]), expression [,style])]

Both functions above have the same objective – that of converting an expression from one data type to another. The *expression* in the syntax is the value that is to be converted, the *data_type* is the target datatype, length is optional and this is used to specify the *length* of the target data type. In the case of the CONVERT function we have the *style* parameter which is an integer that specifies how the function is to translate expression. To take a look at the different styles available take a look at:

https://msdn.microsoft.com/en-us/library/ms187928.aspx

*Example 34:* Write a query that will return the employee number, employement date as stored in the database and the employment date in the following two formats dd//mm/yyyy and dd/mm/yy for employees whose surname is 'Higgins'.



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

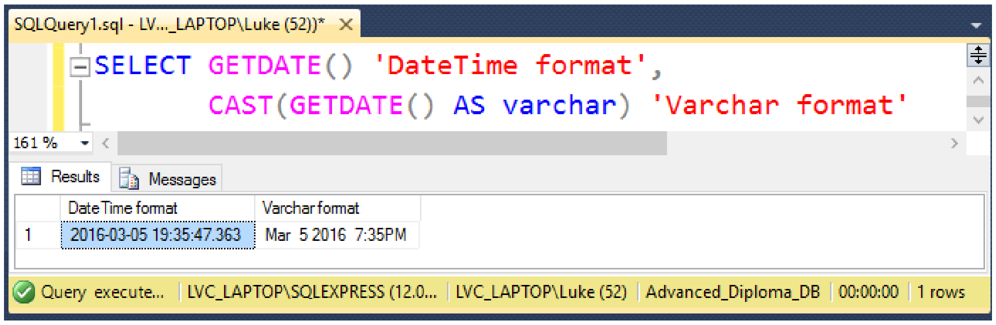
```
SELECT employee_id, hire_date,
       CONVERT(varchar, hire_date, 103) 'dd/mm/yyyy',
       CONVERT(varchar, hire_date, 3) 'dd/mm/yy'
FROM employees
WHERE last_name = 'Higgins';
```

```
  employee_id  hire_date                dd/mm/yyyy  dd/mm/yy
1 205          1994-06-07 00:00:00.000  07/06/1994  07/06/94
```

Query execute... | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 1 rows

*Figure 36 – Result of example 34 – changing the format of the displayed date through conversion functions.*

*Example 35:* Write a query that will display the current date and time in DateTime format and, the same value converted to varchar.



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

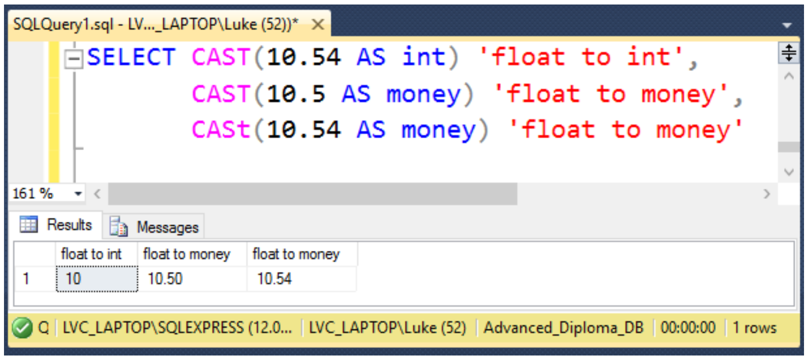
```
SELECT GETDATE() 'DateTime format',
       CAST(GETDATE() AS varchar) 'Varchar format'
```

```
  DateTime format          Varchar format
1 2016-03-05 19:35:47.363  Mar  5 2016  7:35PM
```

Query execute... | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 1 rows

*Figure 37 –Result of example 35: changing from a datetime to a varchar data type*

*Example 36:* Write a query that will change 10.54 to int, 10.5 and 10.54 to money. Note that when we convert to int the value is truncated.



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```
SELECT CAST(10.54 AS int) 'float to int',
       CAST(10.5 AS money) 'float to money',
       CASt(10.54 AS money) 'float to money'
```

```
  float to int  float to money  float to money
1 10            10.50           10.54
```

Q | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 1 rows

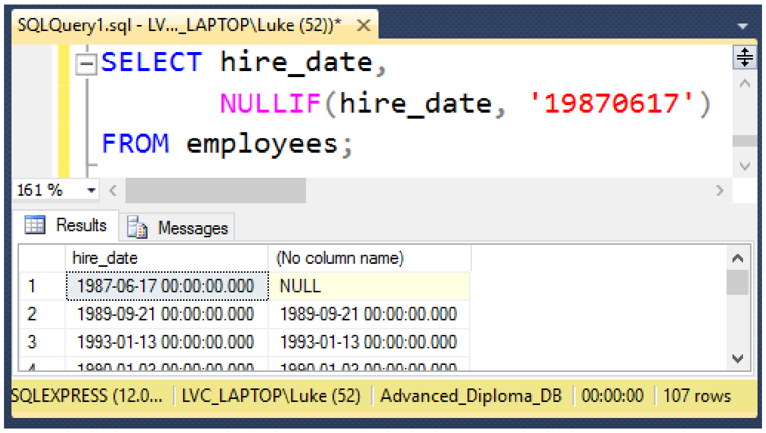
*Figure 38 – Result of exercise 36: casting value from one data type to another.*

## E. General Functions

The last category of functions that we will be considering in this topic includes a number of functions which are mostly related to the use of functions with values which include null values in the expression list.

- *NULLIF Function* [NULLIF (expression, expression)]
  This function will compare the two inputs and returns null if the two values are equal and if they are not equal the first expression is returned.

*__Example 37:__* Write a query that will compare the hire_date against 1987-06-17. When the hire_date is equal to the said value the result is a NULL value.


SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

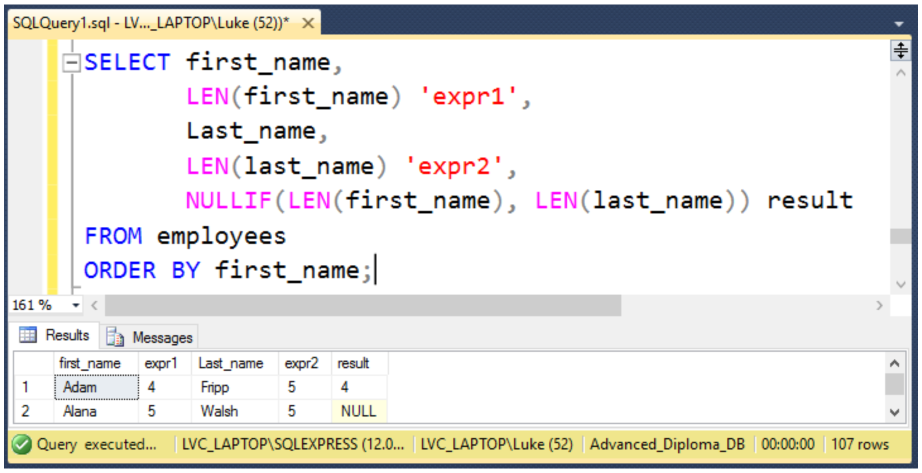
```
SELECT hire_date,
       NULLIF(hire_date, '19870617')
FROM employees;
```

```
   hire_date                 (No column name)
1  1987-06-17 00:00:00.000   NULL
2  1989-09-21 00:00:00.000   1989-09-21 00:00:00.000
3  1993-01-13 00:00:00.000   1993-01-13 00:00:00.000
```

SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 107 rows

*Figure 39 – Result of example 37: comparing the hire_date to 17-06-1987*

*__Example 38:__* Write a query where the length of the first_name and last_name are equal then the result of NULLIF is NULL, else expression 1 is outputted.


SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```
SELECT first_name,
       LEN(first_name) 'expr1',
       Last_name,
       LEN(last_name) 'expr2',
       NULLIF(LEN(first_name), LEN(last_name)) result
FROM employees
ORDER BY first_name;
```

```
   first_name  expr1  Last_name  expr2  result
1  Adam        4      Fripp      5      4
2  Alana       5      Walsh      5      NULL
```

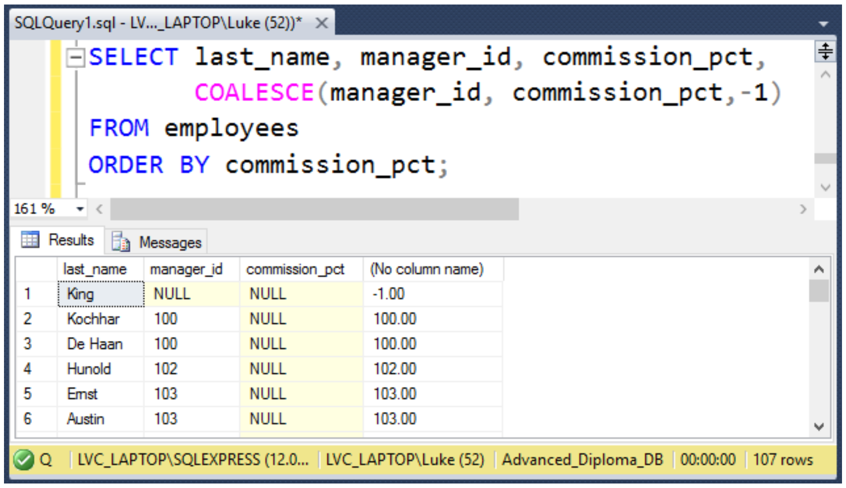
Query executed... | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 107 rows

*Figure 40 – Result of example 38: comparison of the lengths of the name and the surname*

- *COALESCE Function* [COALESCE(expression [, . . n])]

  This function compares the expressions one after the other and returns the first expression that does not evaluate to NULL

*Example 39:* Write a query that displays the surname, manager number, and commission percentage. In the last column if the manager_id is not null it is displayed, if the manager_id value is null, then the commission_pct is displayed. If the manager_id and commission_pct values are null, then the value -1 is displayed.



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```
SELECT last_name, manager_id, commission_pct,
           COALESCE(manager_id, commission_pct,-1)
FROM employees
ORDER BY commission_pct;
```

161 %

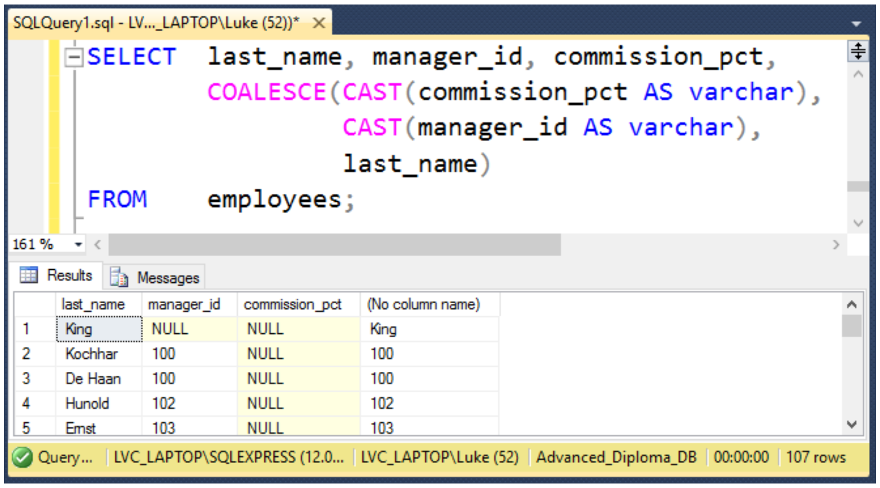
Results | Messages

| | last_name | manager_id | commission_pct | (No column name) |
|---|---|---|---|---|
| 1 | King | NULL | NULL | -1.00 |
| 2 | Kochhar | 100 | NULL | 100.00 |
| 3 | De Haan | 100 | NULL | 100.00 |
| 4 | Hunold | 102 | NULL | 102.00 |
| 5 | Ernst | 103 | NULL | 103.00 |
| 6 | Austin | 103 | NULL | 103.00 |

Q | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 107 rows

*Figure 41 – Result of example 39: return the first not null value*

*Example 40:* Write a query that will return the surname, manager number and commission percentage. In the last column the commission no, manager no and last name are compared and the first not null value from the three is returned



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```
SELECT  last_name, manager_id, commission_pct,
        COALESCE(CAST(commission_pct AS varchar),
                 CAST(manager_id AS varchar),
                 last_name)
FROM    employees;
```

161 %

Results | Messages

| | last_name | manager_id | commission_pct | (No column name) |
|---|---|---|---|---|
| 1 | King | NULL | NULL | King |
| 2 | Kochhar | 100 | NULL | 100 |
| 3 | De Haan | 100 | NULL | 100 |
| 4 | Hunold | 102 | NULL | 102 |
| 5 | Ernst | 103 | NULL | 103 |

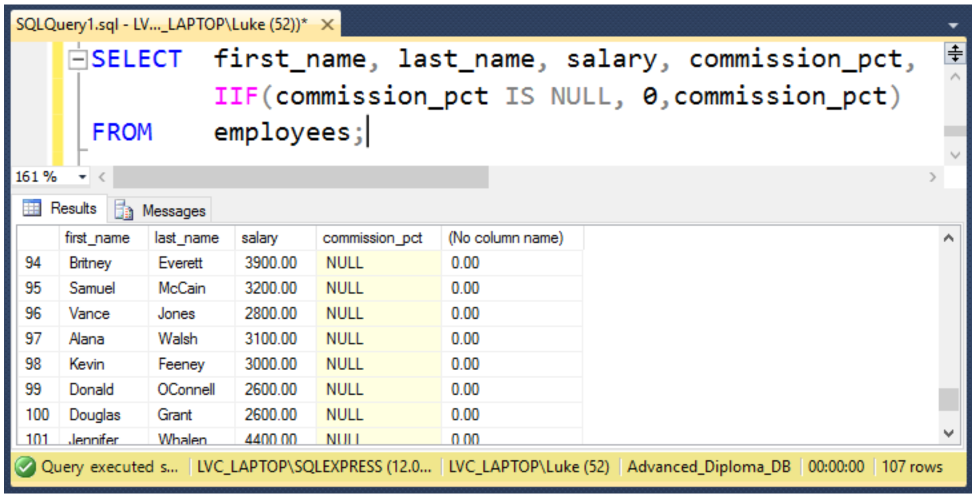
Query... | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 107 rows

*Figure 42 – Result of example 40: return the first not null value.*

- *IIF Function* [IIF(Boolean_expression, true_value, false_value)]

  This function will evaluate the Boolean expression and returns either the true_value or the false_value depending on the Boolean_expression.

*Example 41:* Write a query that will display the name, surname, salary and commission percentage. The last column checks whether the commission percentage is null or not. If the commission percentage is null then 0 is returned, otherwise the commission percentage is returned



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```sql
SELECT  first_name, last_name, salary, commission_pct,
        IIF(commission_pct IS NULL, 0,commission_pct)
FROM    employees;
```

161 %

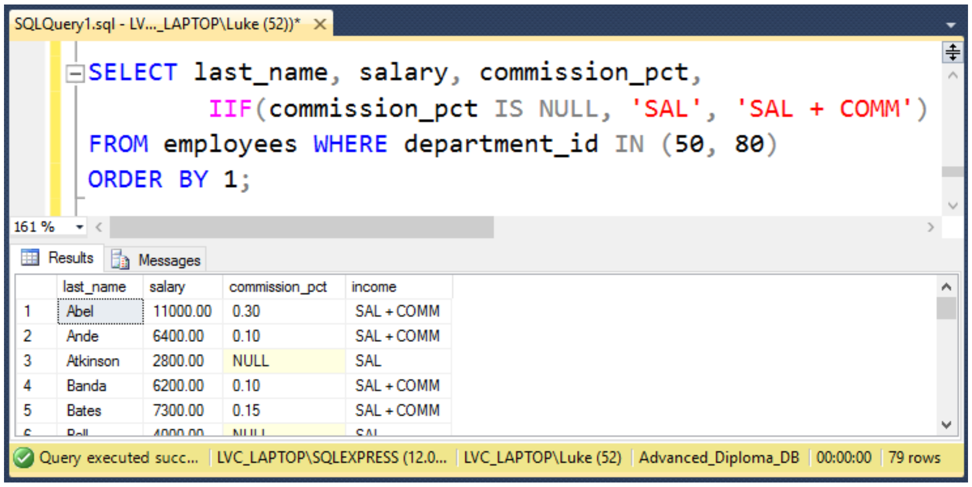
Results | Messages

| | first_name | last_name | salary | commission_pct | (No column name) |
|---|---|---|---|---|---|
| 94 | Britney | Everett | 3900.00 | NULL | 0.00 |
| 95 | Samuel | McCain | 3200.00 | NULL | 0.00 |
| 96 | Vance | Jones | 2800.00 | NULL | 0.00 |
| 97 | Alana | Walsh | 3100.00 | NULL | 0.00 |
| 98 | Kevin | Feeney | 3000.00 | NULL | 0.00 |
| 99 | Donald | OConnell | 2600.00 | NULL | 0.00 |
| 100 | Douglas | Grant | 2600.00 | NULL | 0.00 |
| 101 | Jennifer | Whalen | 4400.00 | NULL | 0.00 |

Query executed s... | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 107 rows

*Figure 43 – Result of example 41: check if commission percentage is null and returns 0 if null or the percentage if it has a value*

*Example 42:* Write a query that will list the surname, salary and commission percentage. The last column will return SAL if the person has no commission and SAL+COMM if the person has a commission. Consider all the employees in department 50 or 80



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```sql
SELECT last_name, salary, commission_pct,
        IIF(commission_pct IS NULL, 'SAL', 'SAL + COMM')
FROM employees WHERE department_id IN (50, 80)
ORDER BY 1;
```

161 %

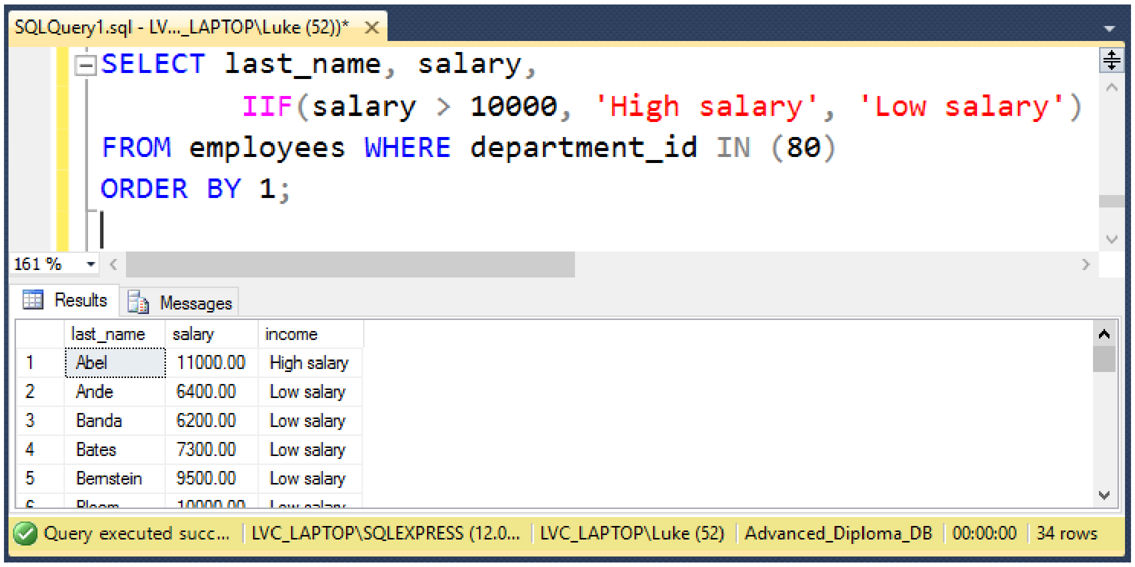
Results | Messages

| | last_name | salary | commission_pct | income |
|---|---|---|---|---|
| 1 | Abel | 11000.00 | 0.30 | SAL + COMM |
| 2 | Ande | 6400.00 | 0.10 | SAL + COMM |
| 3 | Atkinson | 2800.00 | NULL | SAL |
| 4 | Banda | 6200.00 | 0.10 | SAL + COMM |
| 5 | Bates | 7300.00 | 0.15 | SAL + COMM |
| 6 | Bell | 4000.00 | NULL | SAL |

Query executed succ... | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 79 rows

*Figure 44: Result of example 42: Checks if commission percentage is null or not and returns SAL or SAL+COMM*

*Example 43:* Write a query that will display the surname, salary and classification. The classification is calculated on the salary. If the salary is greater than 10000 this column should read 'High Salary', otherwise it should read 'Low salary'.



```
SELECT last_name, salary,
        IIF(salary > 10000, 'High salary', 'Low salary')
FROM employees WHERE department_id IN (80)
ORDER BY 1;
```

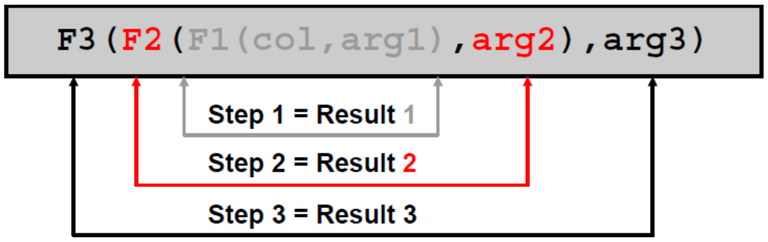
| | last_name | salary | income |
|---|---|---|---|
| 1 | Abel | 11000.00 | High salary |
| 2 | Ande | 6400.00 | Low salary |
| 3 | Banda | 6200.00 | Low salary |
| 4 | Bates | 7300.00 | Low salary |
| 5 | Bernstein | 9500.00 | Low salary |
| 6 | Bloom | 10000.00 | Low salary |

Query executed succ... | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 34 rows

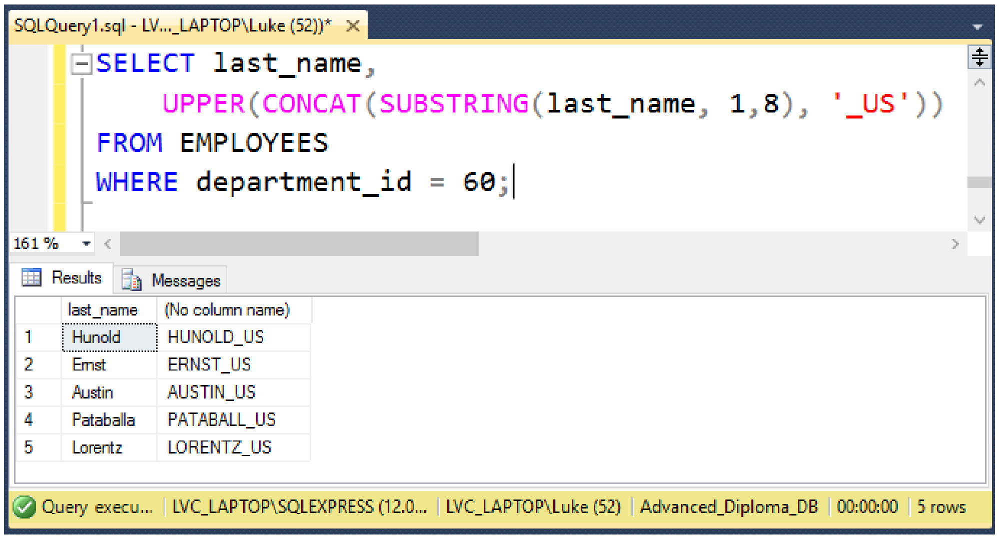
*Figure 45 – Result of Example 43: Checks if salary is greater than 10000 or not.*

## Nesting Functions

One of the most important operations in SQL is that of nesting functions. Nesting functions is an operation which involves functions within functions. Scalar functions can be nested to any level and these are evaluated starting from the deepest level to the least deep level.



*Example 44*: Write a query that lists the last_name, and in the next column you need to select the first 8 characters of the surname followed by _US.  This operations should be applied to all the employees who work in department 60



```
SQLQuery1.sql - LV..._LAPTOP\Luke (52))*
SELECT last_name,
    UPPER(CONCAT(SUBSTRING(last_name, 1,8), '_US'))
FROM EMPLOYEES
WHERE department_id = 60;
```

161 %

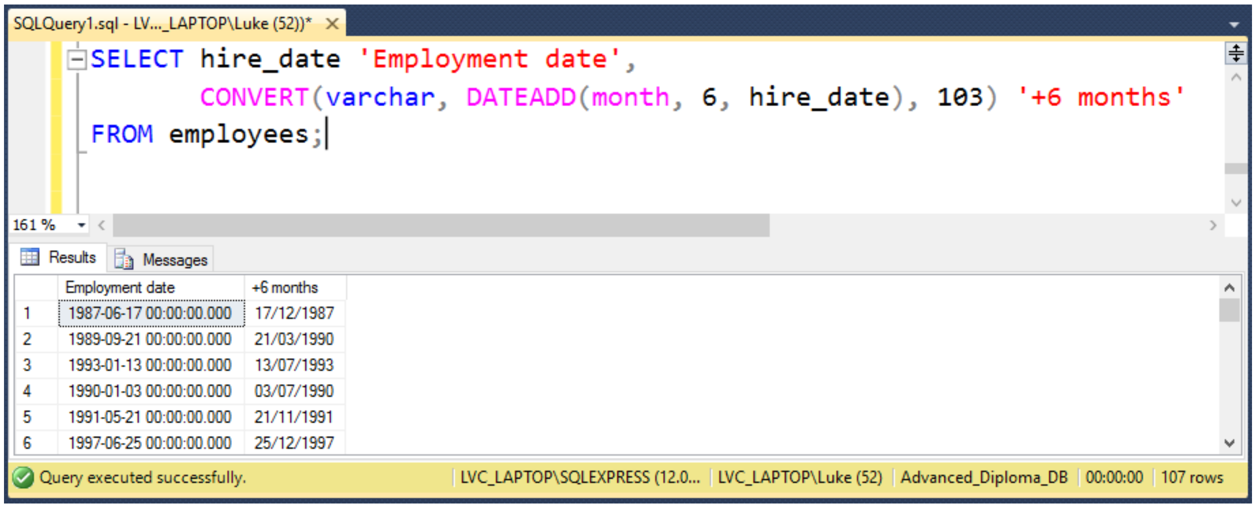
Results | Messages

| | last_name | (No column name) |
|---|---|---|
| 1 | Hunold | HUNOLD_US |
| 2 | Ernst | ERNST_US |
| 3 | Austin | AUSTIN_US |
| 4 | Pataballa | PATABALL_US |
| 5 | Lorentz | LORENTZ_US |

```
Query execu... | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 5 rows
```

*Figure 46 – Result of example 44: use nesting function*

*Example 45:* Write a query that will add 6 months to the employment date and displays the resulting date in the following format dd/mm/yy



```
SQLQuery1.sql - LV..._LAPTOP\Luke (52))*
SELECT hire_date 'Employment date',
       CONVERT(varchar, DATEADD(month, 6, hire_date), 103) '+6 months'
FROM employees;
```

161 %

Results | Messages

| | Employment date | +6 months |
|---|---|---|
| 1 | 1987-06-17 00:00:00.000 | 17/12/1987 |
| 2 | 1989-09-21 00:00:00.000 | 21/03/1990 |
| 3 | 1993-01-13 00:00:00.000 | 13/07/1993 |
| 4 | 1990-01-03 00:00:00.000 | 03/07/1990 |
| 5 | 1991-05-21 00:00:00.000 | 21/11/1991 |
| 6 | 1997-06-25 00:00:00.000 | 25/12/1997 |

```
Query executed successfully. | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 107 rows
```

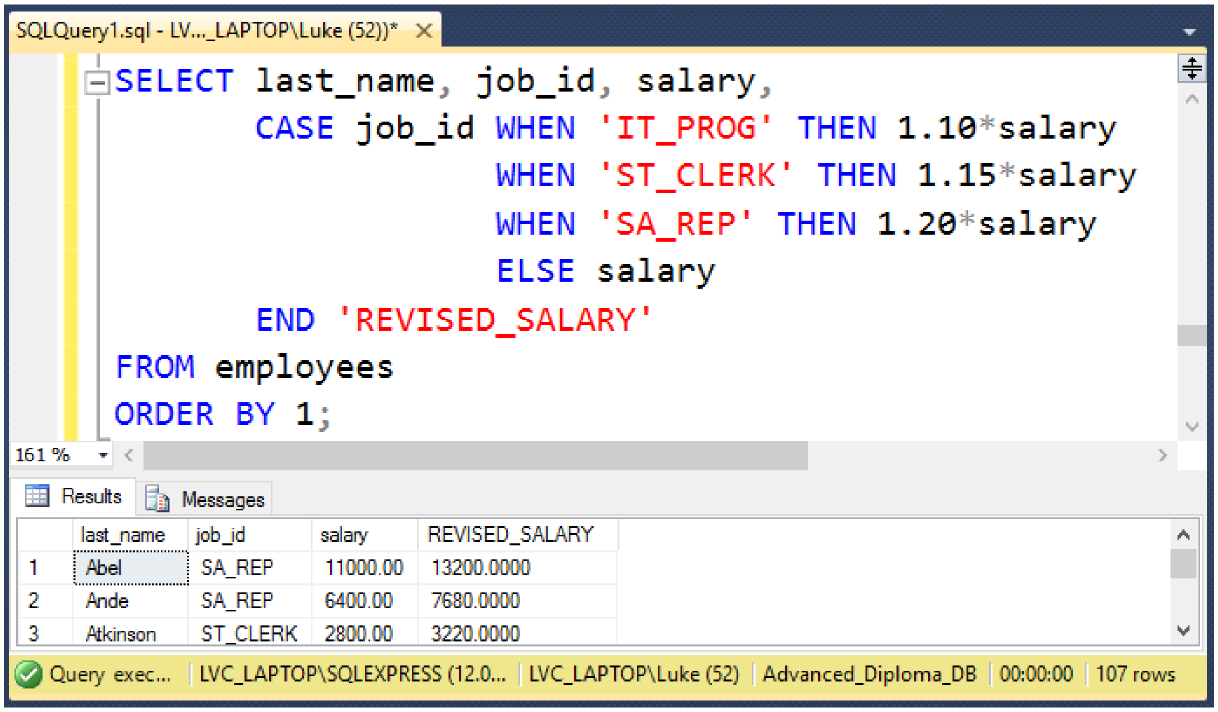
*Figure 47 – Result of example 45: use of nesting functions*

## Conditional CASE expression

This expression is used to determine which condition is to be used to display the final result. This conditional expression is the equivalent to an IF-THEN-ELSE statement. The case statement is made up of many expressions. For each and every expression, the first WHEN..THEN pair that is equal to the comparison_expr till return the return_expr. If none of the WHEN..THEN pairs meet the condition the else clause is executed if it exists

```
SYNTAX:     CASE [expr] WHEN comparison_expr1 THEN return_expr1
                       [WHEN comparison_expr2 THEN return_expr2
                        WHEN comparison_exprn THEN return_exprn]
                        ELSE else_expr
             END
```

*Example 46:* Write a query that will update the salary of people with a job of 'IT PROG' by 10%, an 'ST_CLERK' by 15% and 'SA_REP' by 20%. All the rest should have the same salary.



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

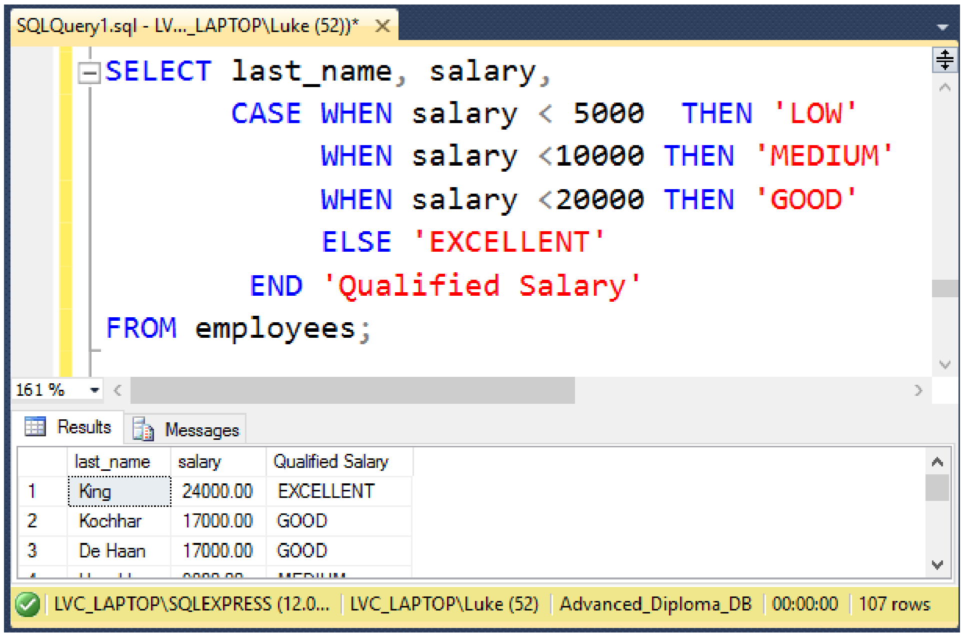
```
SELECT last_name, job_id, salary,
       CASE job_id WHEN 'IT_PROG' THEN 1.10*salary
                   WHEN 'ST_CLERK' THEN 1.15*salary
                   WHEN 'SA_REP' THEN 1.20*salary
                   ELSE salary
       END 'REVISED_SALARY'
FROM employees
ORDER BY 1;
```

| | last_name | job_id | salary | REVISED_SALARY |
|---|---|---|---|---|
| 1 | Abel | SA_REP | 11000.00 | 13200.0000 |
| 2 | Ande | SA_REP | 6400.00 | 7680.0000 |
| 3 | Atkinson | ST_CLERK | 2800.00 | 3220.0000 |

Query exec... | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 107 rows

*Figure 48 – Result of example 46: Revised salary of employees*

*Example 47:* Write a query that displays the surname, salary and salary description. The salary description should be LOW for salaries less than 5000, MEDIUM for salaries less than 10000, GOOD for salaries 20000 and EXCELLENT for all the rest



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```sql
SELECT last_name, salary,
         CASE WHEN salary < 5000  THEN 'LOW'
              WHEN salary <10000 THEN 'MEDIUM'
              WHEN salary <20000 THEN 'GOOD'
              ELSE 'EXCELLENT'
          END 'Qualified Salary'
FROM employees;
```

161 %

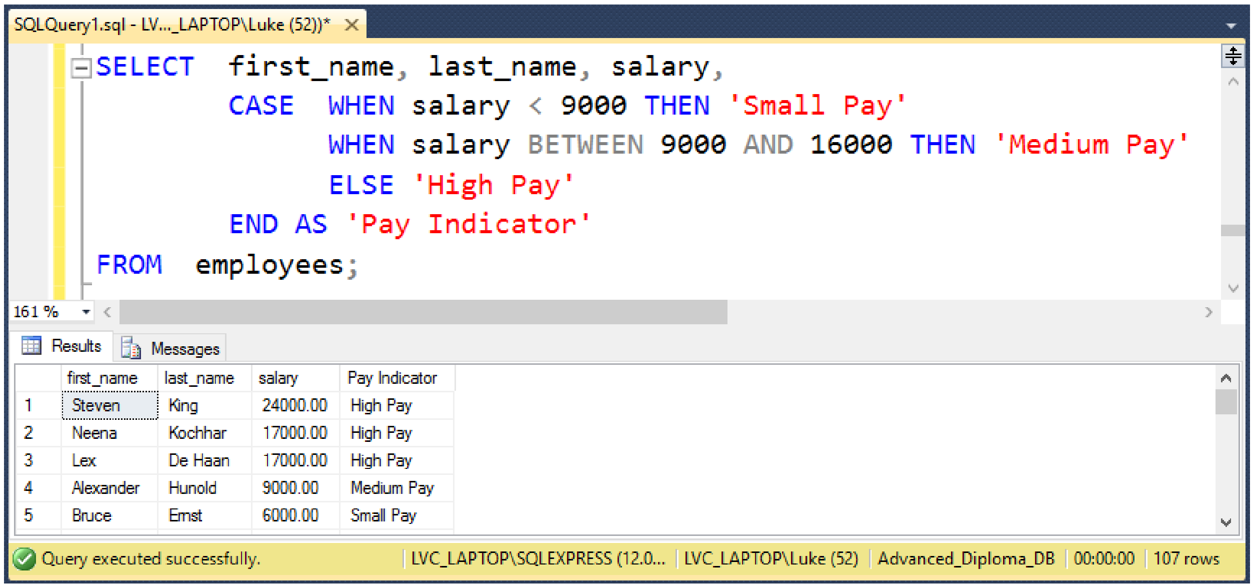
Results | Messages

| | last_name | salary | Qualified Salary |
|---|---|---|---|
| 1 | King | 24000.00 | EXCELLENT |
| 2 | Kochhar | 17000.00 | GOOD |
| 3 | De Haan | 17000.00 | GOOD |

LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 107 rows

*Figure 49 – Result of example 47: salary description via CASE*

*Example 48:* Write a query that displays the surname, salary and salary description. The salary description should be 'Small Pay' for salaries less than 8000, 'Medium Pay' for salaries between 9000 AND 16000, and 'High Pay' for all the rest



SQLQuery1.sql - LV..._LAPTOP\Luke (52))*

```sql
SELECT  first_name, last_name, salary,
        CASE  WHEN salary < 9000 THEN 'Small Pay'
              WHEN salary BETWEEN 9000 AND 16000 THEN 'Medium Pay'
              ELSE 'High Pay'
        END AS 'Pay Indicator'
FROM  employees;
```

161 %

Results | Messages

| | first_name | last_name | salary | Pay Indicator |
|---|---|---|---|---|
| 1 | Steven | King | 24000.00 | High Pay |
| 2 | Neena | Kochhar | 17000.00 | High Pay |
| 3 | Lex | De Haan | 17000.00 | High Pay |
| 4 | Alexander | Hunold | 9000.00 | Medium Pay |
| 5 | Bruce | Ernst | 6000.00 | Small Pay |

Query executed successfully. | LVC_LAPTOP\SQLEXPRESS (12.0... | LVC_LAPTOP\Luke (52) | Advanced_Diploma_DB | 00:00:00 | 107 rows

*Figure 50 – Result of example 48: salary description via CASE*