Limiting Rows of data

One of the major operations that can be carried out while using the SELECT statement is SELECTION – the ability to choose which rows should appear in the result.

Pid	Product Name	Product Price	SupplierId
1	Orange	0.50	10
2	Apple	0.40	10
3	Banana	0.60	20

Selecting particular rows from a table can be done using the WHERE clause within the SELECT statement. When selection is required the default syntax for the SELECT statement will change to the below:

SELECT * | {[DISTINCT] column|expression [alias], . . .}

FROM table

[WHERE condition/s]

What does the WHERE clause in the above syntax mean? This is the only way available in SQL, to be able to restrict the number of rows that are returned. The restriction related to the rows that are returned is depended on the condition/s specified after the WHERE clause. The condition is usually composed of column names, expressions, constants and comparison operators. In the following sections we will be discussing the different type of operators that can be used.

NOTE:

- The WHERE clause can compare values, in columns, literals, arithmetic expressions and functions. It consists of three elements:
  - column name,
  - comparison condition,
  - column name, constant, or list of values
- The WHERE clause is an optional component in any SELECT statement and it should always follow the FROM clause.
- Each WHERE clause should have a condition and if this is met then the row that satisfies it will be returned by the query result.
- Column aliases cannot be used with the WHERE clause.

Comparison Operators

As explained in the previous page, the WHERE clause requires a condition which in turn needs a comparison operator. There is a variety of different comparison operators that can be applied to these conditions:

OPERATOR	Meaning
=	is equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
<> , !=	not equal to
BETWEEN . . . AND . . .	between two values inclusive
IN (set)	match any values in a list
LIKE	match a character pattern
IS NULL	is a null value

NOTE: The syntax used with these comparison operators is **WHERE *expr condition value***

Example 1: Write a query that restricts the output to those employees who work in department 90 only.

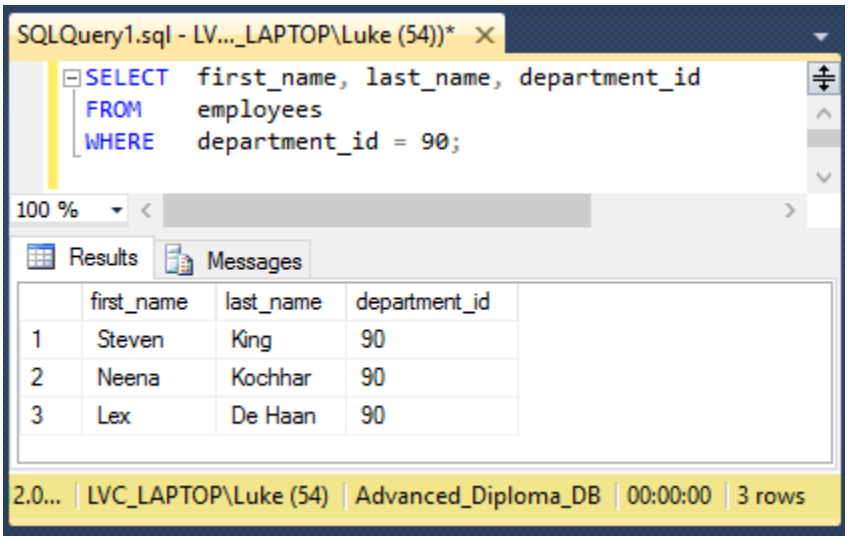


Figure 1 - Result of Example 1: restricting rows for people in department 90

Numbers, Character strings and dates in WHERE conditions

While adding conditions to the WHERE clause, you need to be aware of the data type of the column. The data type used, will affect how the condition is to be written.

- In the case of columns which include *numerical* values, the WHERE clause is simply in the following format: ***expr condition value*** (Eg: salary = 12)
- In the case of columns which include *character* values, the WHERE clause should be in the following format: ***expr condition 'value'*** (Eg: last_name = 'Sammut'). It is important to notice that string values are enclosed in single quotation marks
- In the case of columns which include *date* values, the WHERE clause should be in the following format: ***expr condition 'yyyymmdd'***. Once again notice that the date value is enclosed in single quotes. Also take not that while searching for particular date values the date needs to be in the *yyyymmdd* format (Eg: hire_date = '20050319')

Example 2: Write a query that will display all the employees whose surname is Whalen

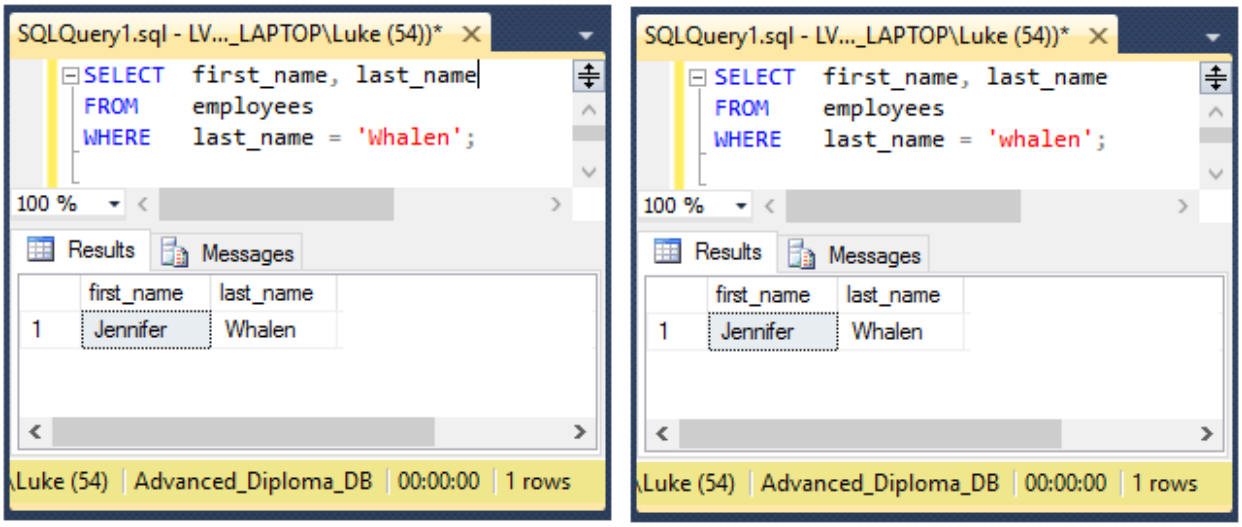


Figure 2 - Result of Example 2: restricting rows for people whose surname is Whalen

NOTE: From Figure 2, it is evident that conditions which handle character values (strings), are not case sensitive. As a matter of fact writing 'Whalen' or 'whalen' will return the same result.

Example 3: Write a statement that will return all the employees which were hired on the 17th of February 1996

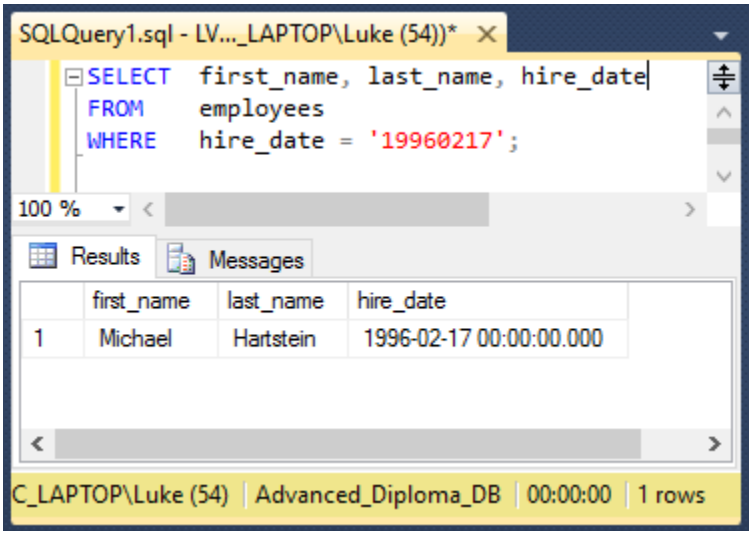


Figure 3 - Result of Example 3 – restricting rows for people who were born on the 17th of February 1996

Example 4: Write a query that will display the name, surname and salary who earn 10000 or more.

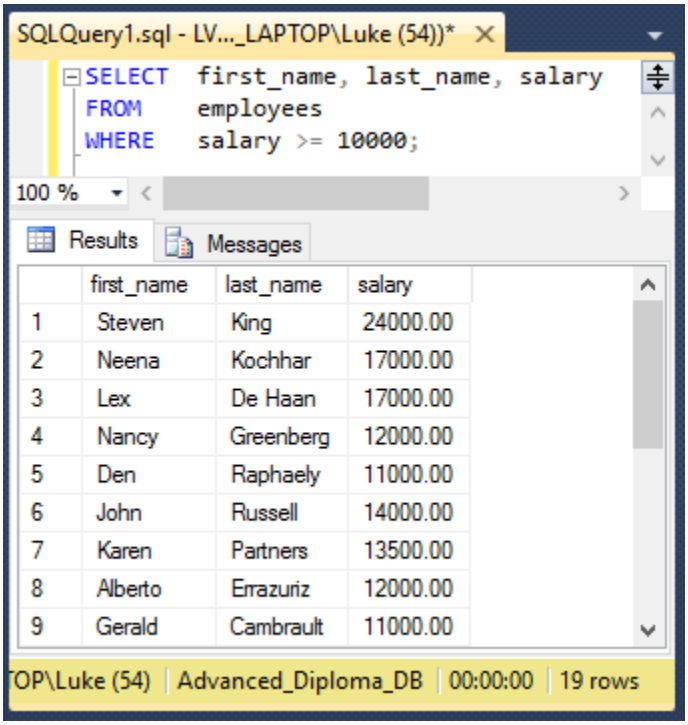


Figure 4 - Result of Example 4: all people who earn 10000 or more

Range Conditions using BETWEEN operator

There are instances where, the number of rows returned by a query is determined via conditions which are based on a range of values. In such cases the BETWEEN operator is an ideal choice. This operator should have a lower and upper limit which are specified and whenever the query is executed these two values are said to be inclusive (this means that if there are rows in the table which have the same value as the upper and/or lower limit, they will still be returned).

The BETWEEN operator can be used with number, character strings and date data types. It is sometimes referred to as the alternative to >= and <= operators.

Example 5: Write a query that will display all the surnames which fall in the range which is covered by King and Smith.

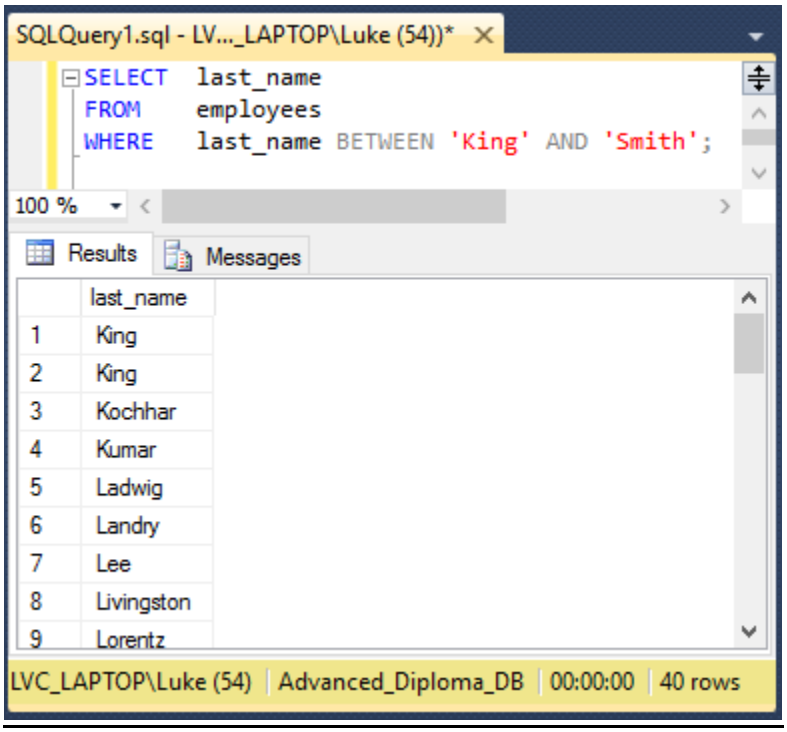


Figure 5 -Result of Example 5: people whose surname is between King and Smith (both inclusive)

NOTE:

- The surnames which are used in the BETWEEN conditions will be included in the result.
- Whenever the BETWEEN operator is used with character strings, the alphabetical order is taken into consideration.

Example 6: Write a query that will return all the employees who earn more than 9999 but less than 23001

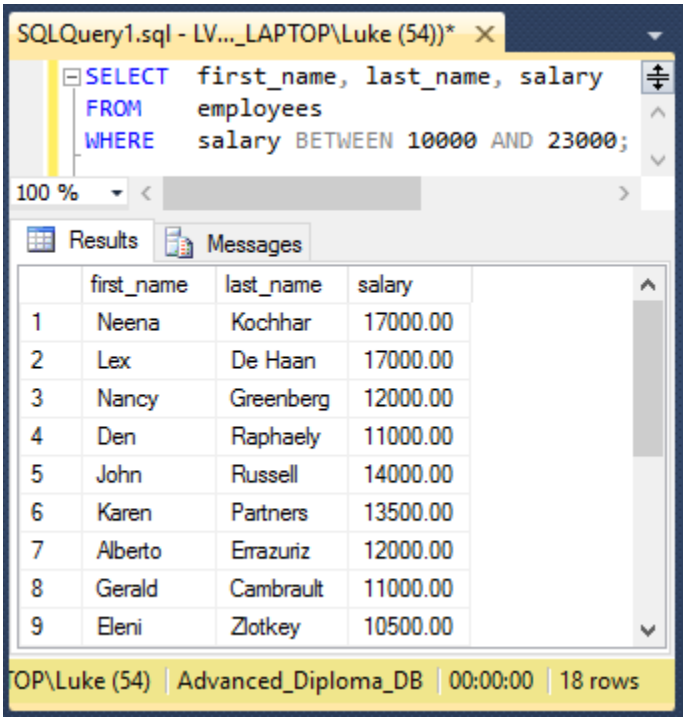


Figure 6 - Result of Example 5: employees who earn more than 9999 but less than 23001

Example 7: Write a query that will return all the employees who have a salary which is less than 10000 and/or greater than 23000.

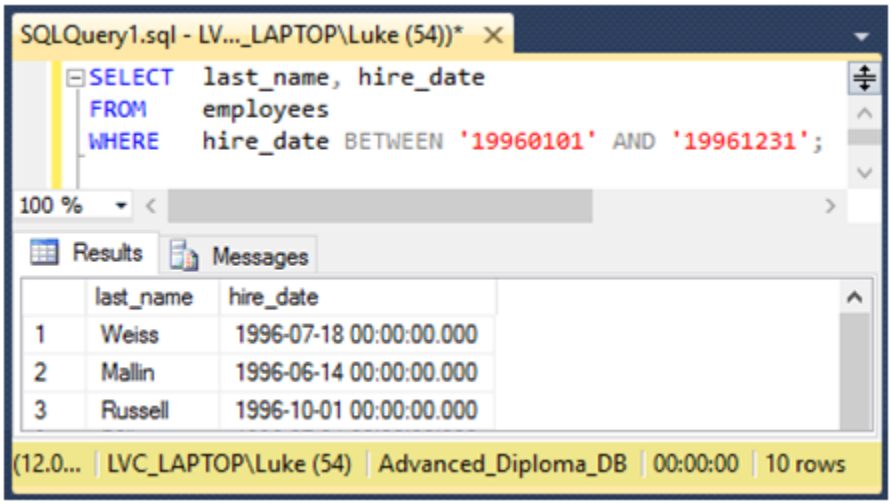


Figure 7 - Result of Example 7: all employees hired in 1996

Example 8: Write a query that will display all the employees who have a salary which does not fall between the 10000 and 23000 range.

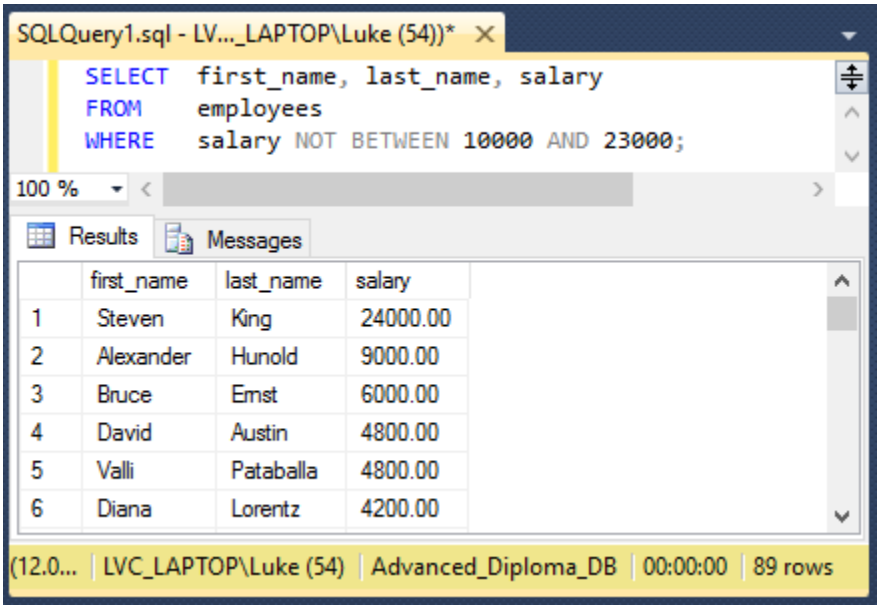


Figure 8 - Result of Example 8: employees who earn less than 10000 and more than 23000

NOTE: In example 8, the ***NOT*** keyword was introduced in front of the BETWEEN keyword. The NOT keyword is also known as negation (opposite of something). In this case we do not want the employees who have a salary which falls between 10000 and 23000, hence the use of the NOT keyword.

The IN operator

In certain cases the values in a table would need to be compared to a list of values and if they match to the values in the list then they are returned. In order to carry out such an operation, one can use many different possibilities but one of the most efficient is the use of the IN operator.

The IN operator, tests for values in a specific set of values and returns all the rows that match. This operator can be used with any data type (numeric, character strings and dates). It is important to keep in mind that when this operator is used with date and character data types, the values need to be enclosed in single quotes.

As in the case of the BETWEEN operator, the IN operator can be used together with the NOT operator. This will negate the condition and includes all the rows which do not equate to the values in the list provided (Take a look at Example 11).

Example 9: Write a query to display the name, surname and department of all the employees which work in departments 30, 60 and 90

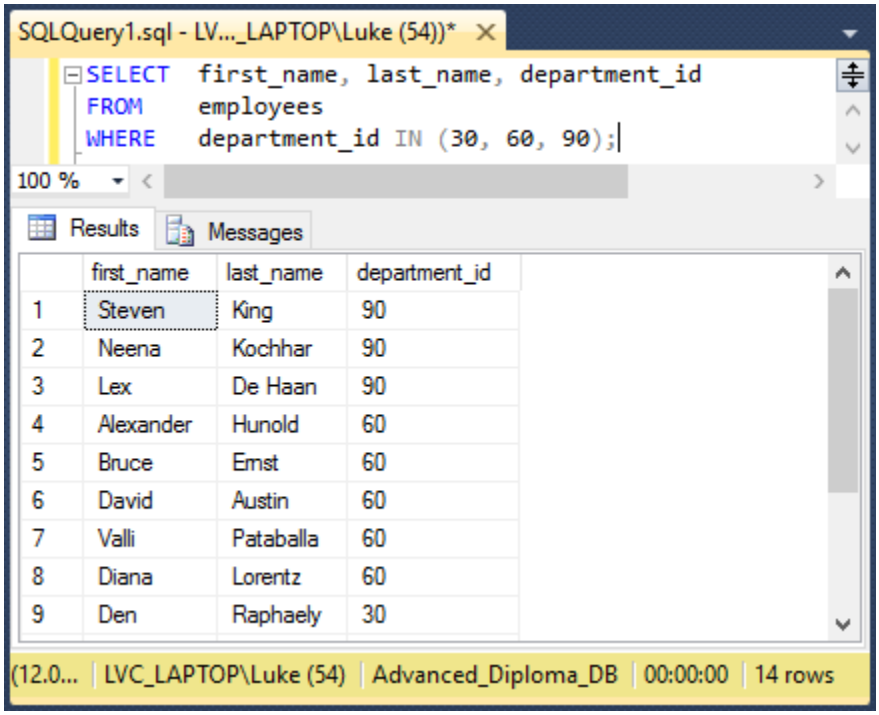


Figure 9 - Result of example 9: IN operator used to return all the employees in departments 30, 60 and 90

Example 10: Write a query to display the employee number, surname, manger no and department number of all the employees whose surname is equivalent to Hartstein and Vargas.

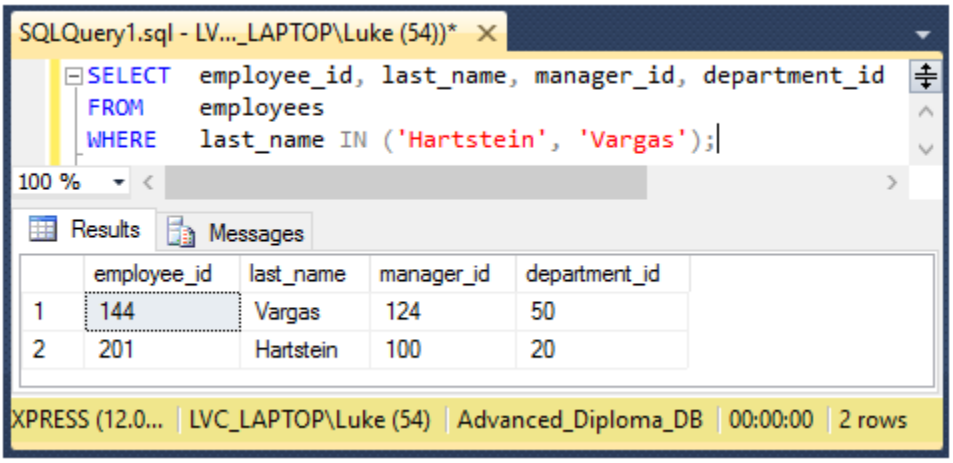


Figure 10 - Result of Example 10: All rows for people with surname Hartstein and Vargas

Example 11: Write a query that will return the name and surname of all the employees who do not have a King and Whalen as their surname.

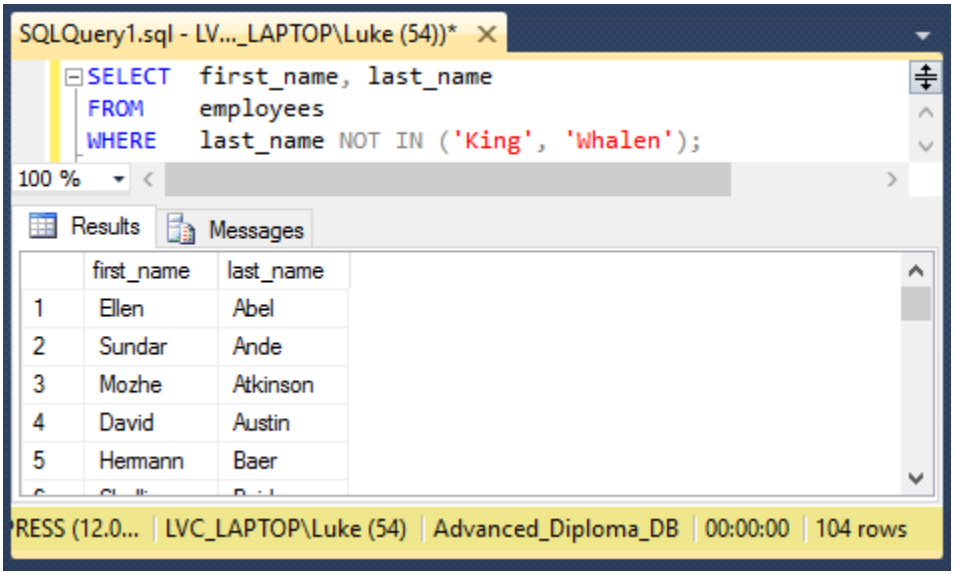


Figure 11 - Result of Exercise 11: all employees who do not match the list provided

Pattern matching operations via the LIKE operator

The LIKE operator is a very important operator while retrieving information from a database. This operator is used to check whether a specific character string matches a specified pattern. Whenever wildcard searching and pattern matching is required then the ideal operator is LIKE. This operator is used a lot when you would like to search for something particular but you are not 100% sure what to include as a searching parameter. Using the LIKE operator may be considered much more flexible than using = or != operators.

The LIKE operator has a number of wildcards. These wildcards are listed in the table below:

Wildcard Character	Description
%	Any string of 0 or more characters
_ (underscore)	Any single character
[]	Any single character within the specified range (Eg: [a-f] or [abcdef])
[^]	Any single character not within the specified range (Eg: [^a-f] or [^abcdef])
ESCAPE	Indicates that the above wildcard characters should be interpreted as a regular character and not a wildcard

NOTE: Wildcards can be combined together for more complex searching

Example 12: Write a query that returns all the employees whose Surname starts with a K.

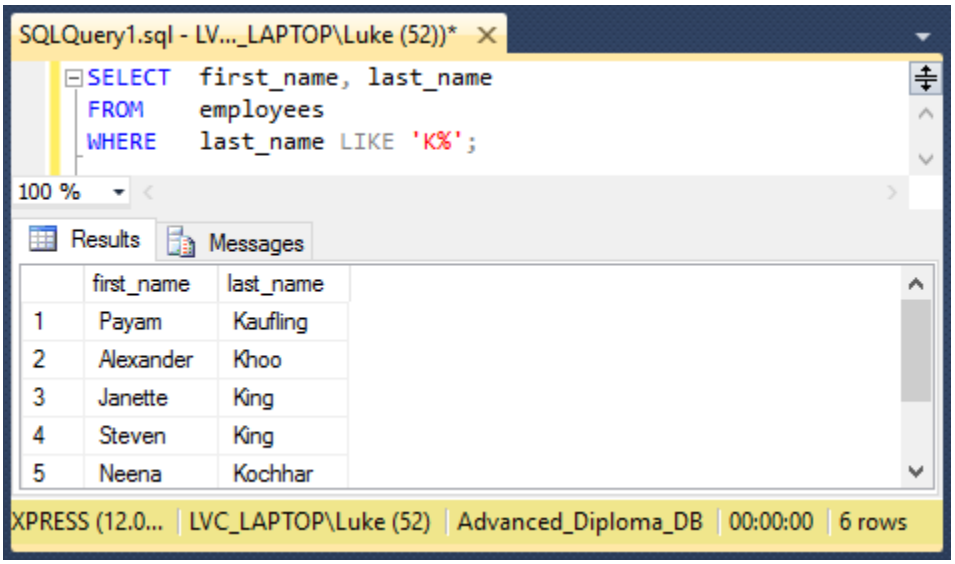


Figure 12 - Result of Example 12: surnames that start with a K

Example 13: Write a query that returns all the people who have the string 'Hi' somewhere in their surname

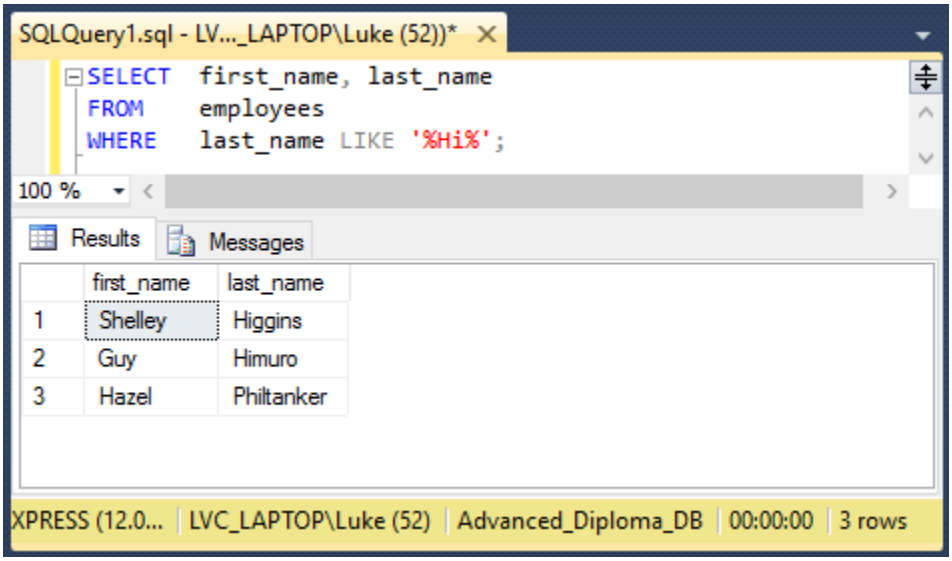


Figure 13 - Result of Example 13: surnames that contain Hi

Example 14: Write a query that returns all the employees whose surname is 4 characters long and end with 2 consecutive letter 'l'

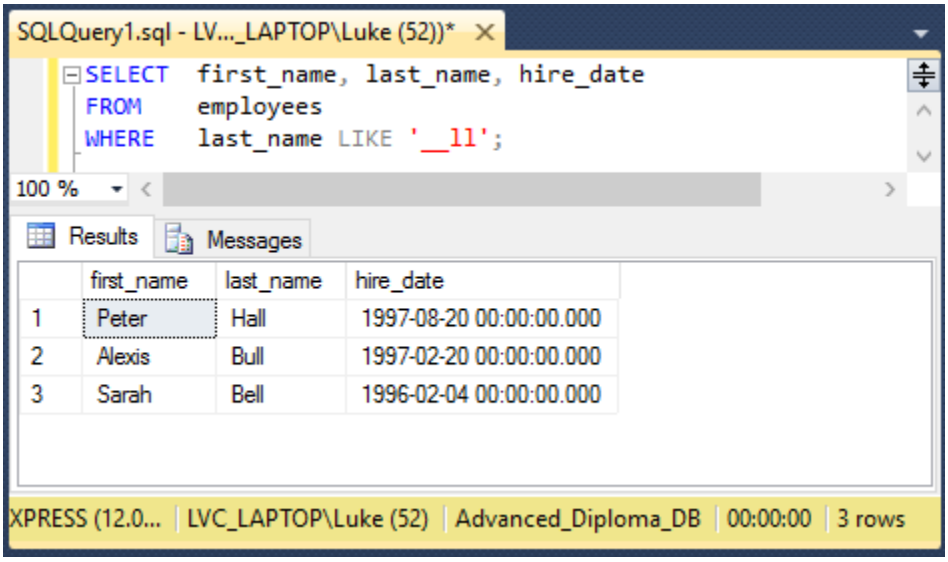


Figure 14 – Result of Example 14: employees who have a 4 letter surname which ends with a double letter 'l'

Example 15: Write a query which returns all the surnames that start with the letters A, B or C

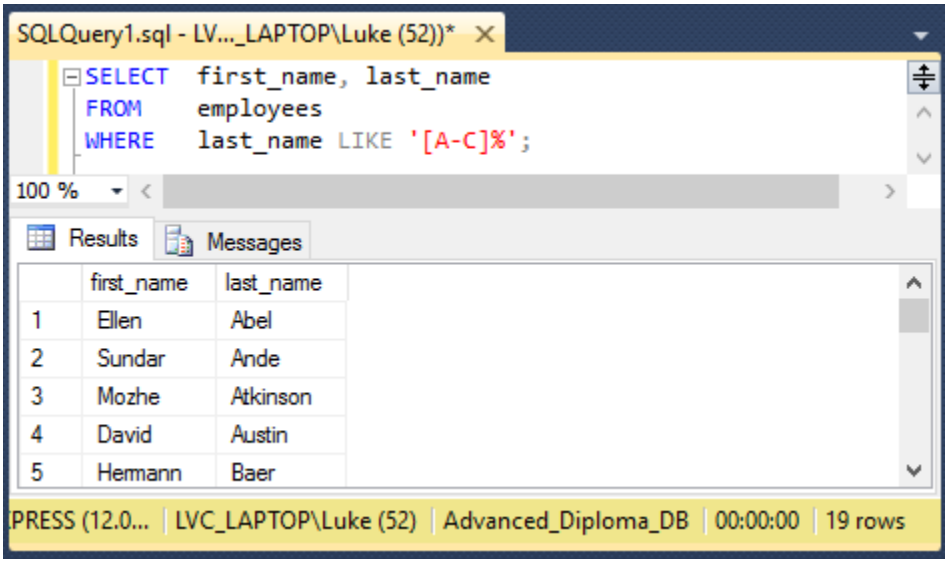


Figure 15 - Result of example 15: all the surnames that start with either A, B or C

Example 16: Write a query that will return all the employee surnames whose second letter is the letter 'o'

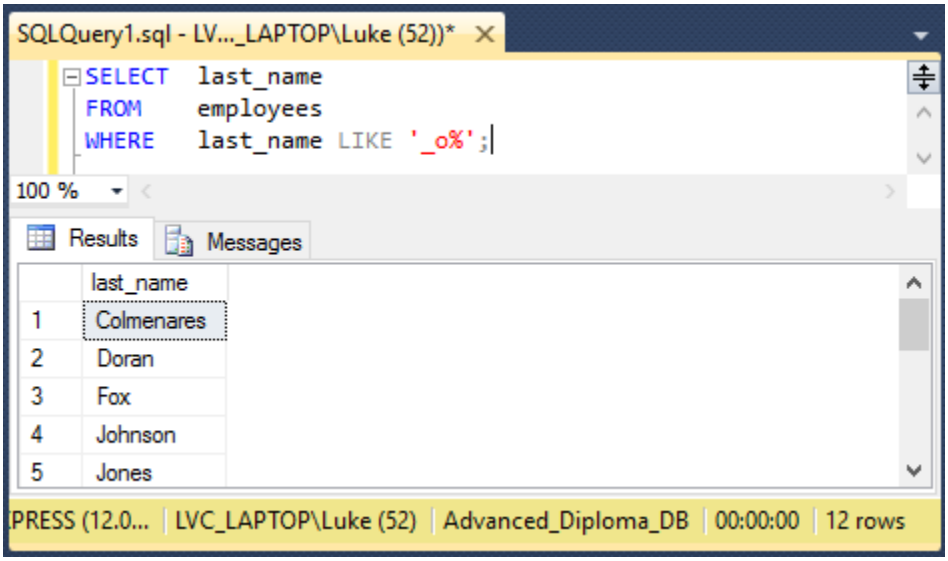


Figure 16 -Result of Example 16: all surnames whose second letter is 'o'

Example 17: Write a query that returns all the employees whose surname does not start with the letters A, B or C

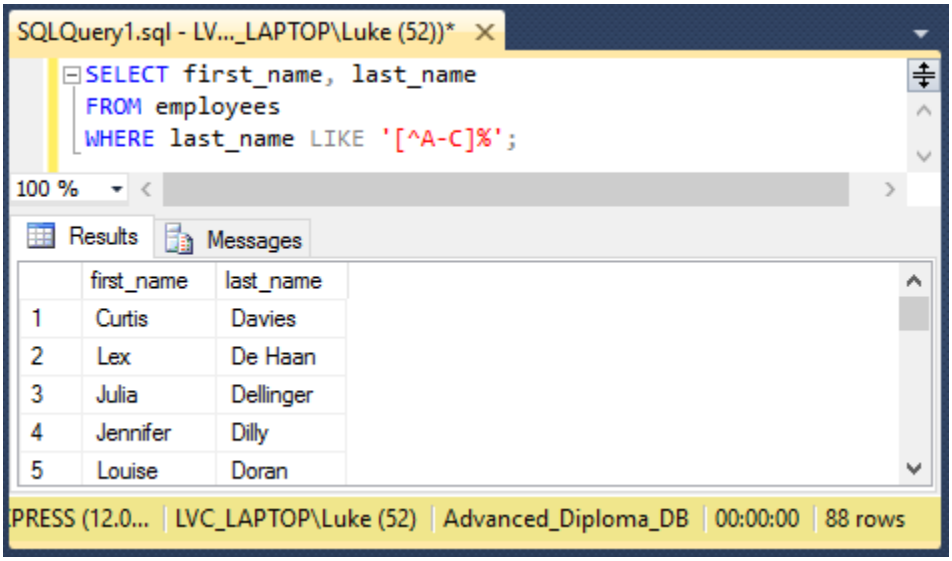


Figure 17 - Result of Example 17: all employees whose surname does not start with the letters A, B, C

Example 18: Write a query that returns all the employees who have a job_id which starts with SA_

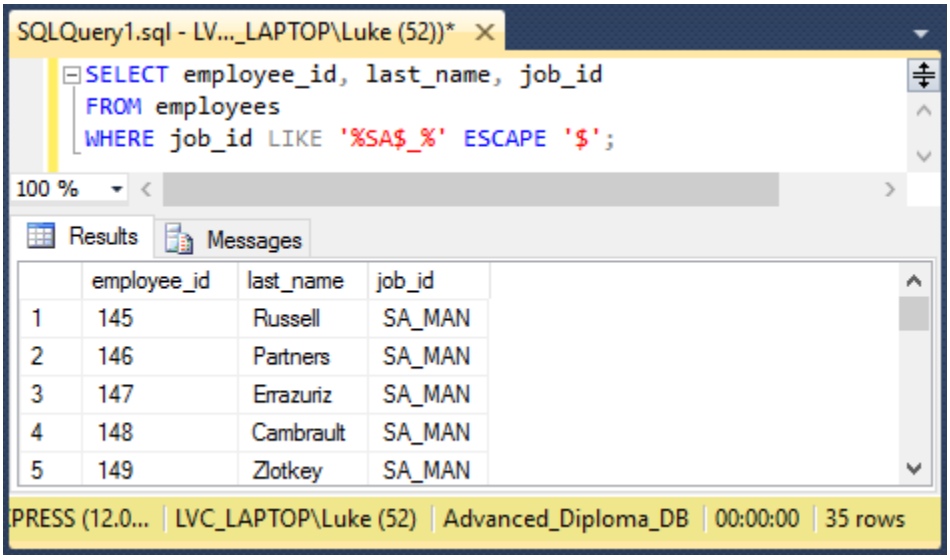


*Figure 18 - Result of Example 18: all employees whose job_id starts with SA_*

Conditional and Logical operators

In this document we have already gone through a number of logical operators such as the IN, BETWEEN and others. It is commonly the case that while writing queries conditional operators will be required. The table below introduces the three conditional operators that will be discussed in this section.

Operator	Description
AND	Returns true if both component conditions are true
OR	Returns true if either component condition is true
NOT	Returns true if the condition is false

The AND/OR operators are typically used and beneficial when more than one condition needs to be tested in the WHERE clause. Without these operators, it is not possible to test multiple conditions.

Example 19: Write a query that will display all those employees whose Job title contains the 'MAN' string and who earn 10,000 or more

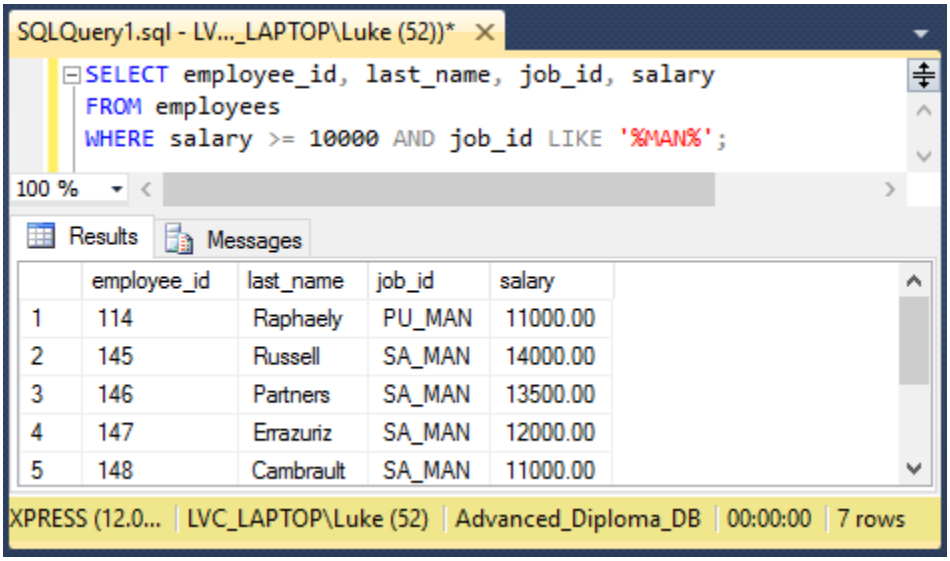


Figure 19 - Result of example 19: all employees with MAN in job id and a salary greater than 10000

Example 20: Write a query that will display all those employees who has a Job title containing the 'MAN' string OR who earn 10,000 or more

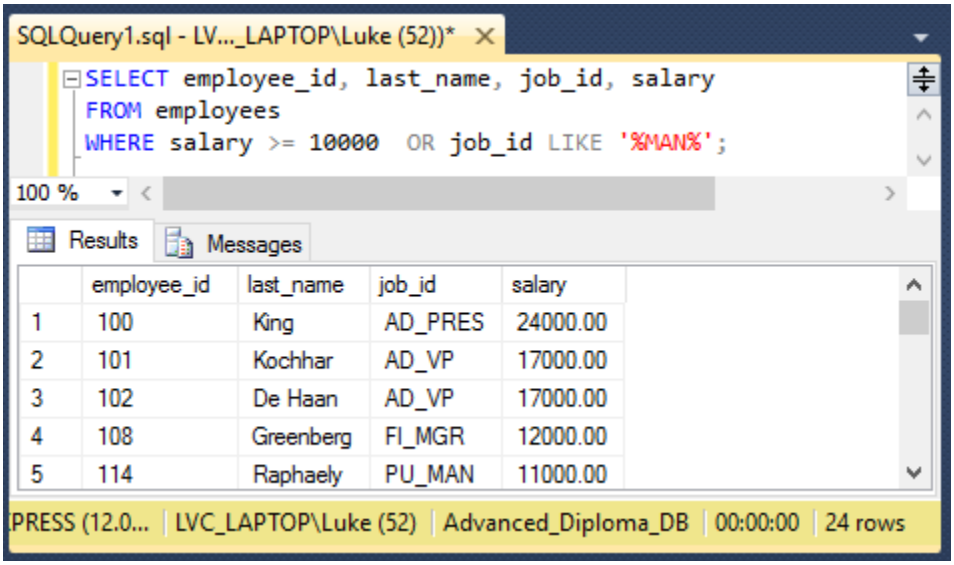


Figure 20 - Result of example 20: all employees who either have MAN in their job id or else earn 10000 or more

Example 21: Write a query that will display the surname and job_id of all the employees who are not IT_PROG, ST_CLERK and SA_REP

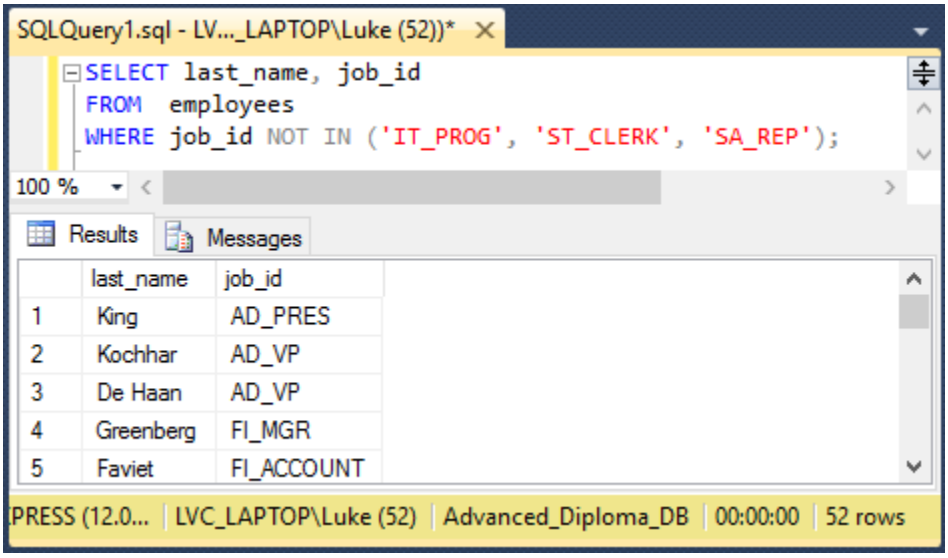


Figure 21 - Result of Example 21: all employees who are not IT_PROG, ST_CLERK and SA_REP

Precedence Rules

These rules are very important when the expression in the WHERE clause are made up of multiple operators. This rule determines which expression is to be executed before the other (sequence of execution). It is important to know the sequence in which expressions are being executed as this can affect the final result in a significant way.

The table below shows the precedence levels used by SQL Server. It is important to note that the operators at the higher levels are evaluated before the lower operators.

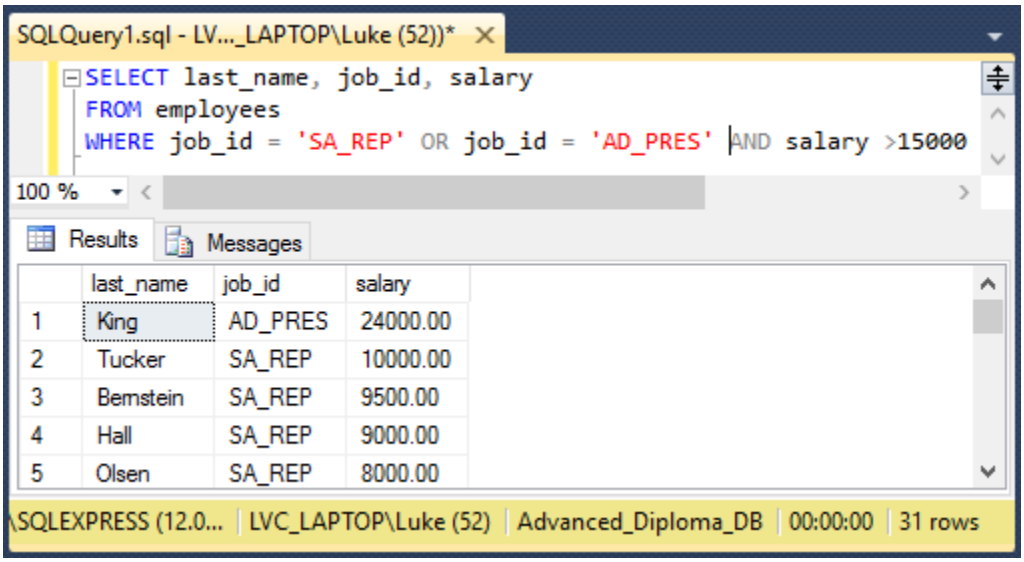
Level	Operator
1	~ (Bitwise NOT)
2	* (Multiply), / (Division), % (Modulo)
3	+ (Positive), - (Negative), & (Bitwise AND), ^ (Bitwise Exclusive OR), | (Bitwise OR)
4	=, > ,< ,>=, <=, <>, !=, !>, !<
5	NOT
6	AND
7	ALL, ANY, BETWEEN, IN, LIKE, OR, SOME
8	=

The precedence order can be superseded with the use of round brackets.

Example 22: The below statements show the importance behind the precedence rule. Although the two statements seem to be the same, in reality they are not. This fact shown even more clearly from the results obtained.

The first query returns 31 rows of data as, first the expression `job_id = 'AD_PRES' AND salary >15000` is evaluated and then the second expression `job_id = 'SA_REP` is evaluated.

In the second query the first expression changed to `(job_id = 'SA_REP' OR job_id = 'AD_PRES')` due to the use of brackets and the second expression changed to `salary >15000`.



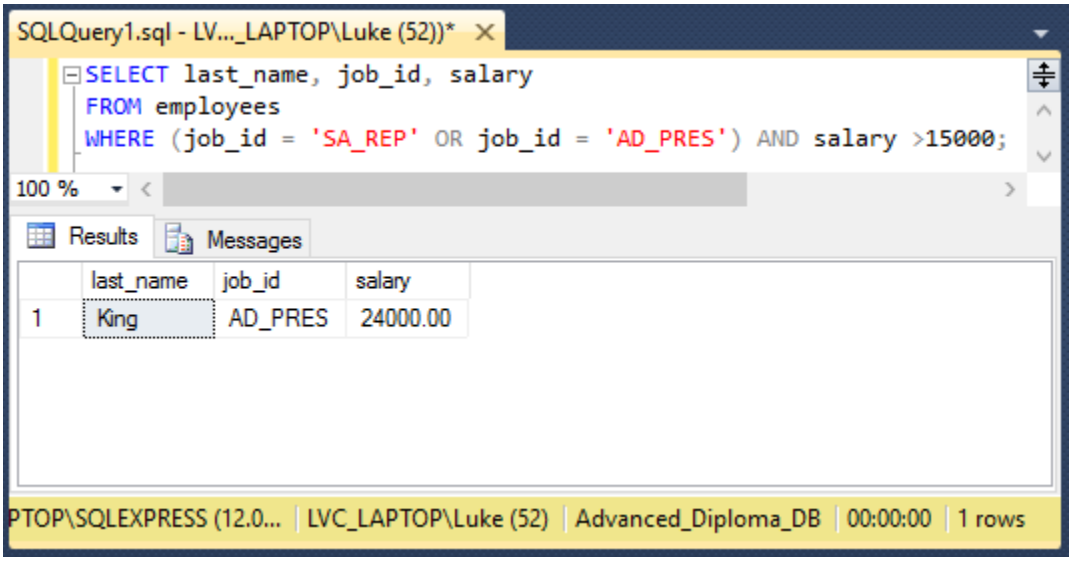


Figure 22 - Result of two different queries due to the precedence rule

NULL values

In any database entity, if a column is not assigned a NOT NULL constraint, then the column can contain null values. We have already discussed null values in the first topic of this subject (refer to the section named – Defining NULL Values).

Always keep in mind that in order to check if column contains null values, you are to use the **IS NULL** operator. Values which are set to null will not be returned if the condition uses the equals (=) operator. The reason behind this is related to the fact that null values cannot be equal or unequal to a value.

Example 23: Write a query that obtains all employees who have not been assigned a department

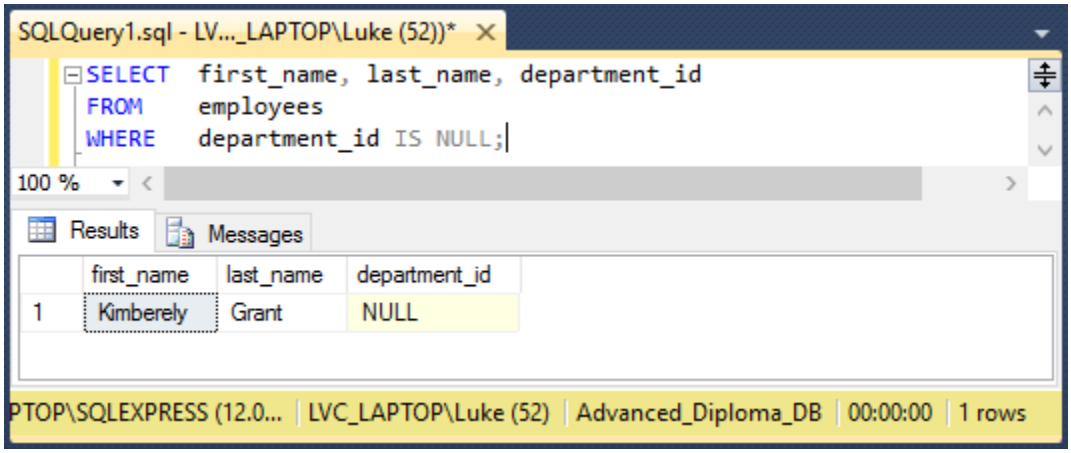


Figure 23 - Result of example 23: Employees which are not assigned a department

Example 24: Write a query that obtains all the employees who have a manager

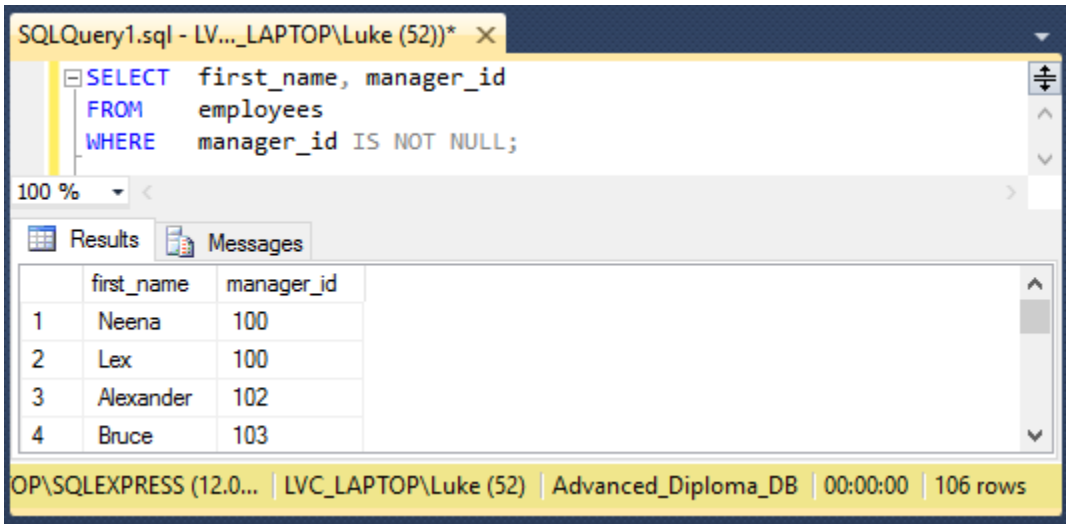


Figure 24 - Result of Example 24: All employees who have been assigned a manager

Sorting rows of Data

In the previous sections we have seen different operators which help us in selecting the rows that apply to our needs. With the help of these operators we can return a subset of the rows that are available in the entity. Unfortunately the use of the WHERE clause does not allow us to organise/sort the rows in a particular order.

In order to be able to organise/sort rows in, the ORDER BY clause needs to be applied.

SELECT * | {[DISTINCT] column|expression [alias], . . .}

FROM table

[WHERE condition/s]

[ORDER BY {COLUMN, EXPR, NUMERIC_POSITION} {ASC|DESC}

Whenever the ORDER BY clause is used in an SQL statement, sorting can be done either in Ascending order (this is the default option - ASC) or in Descending order (DESC).

While sorting the results different techniques can be used after the ORDER BY clause:

1. Using the COLUMN NAME

Example 25: Write a query that will order the rows according to the date when employees were hired. Given the DESC keyword after the column, the results will start from the most recent date to the oldest one.

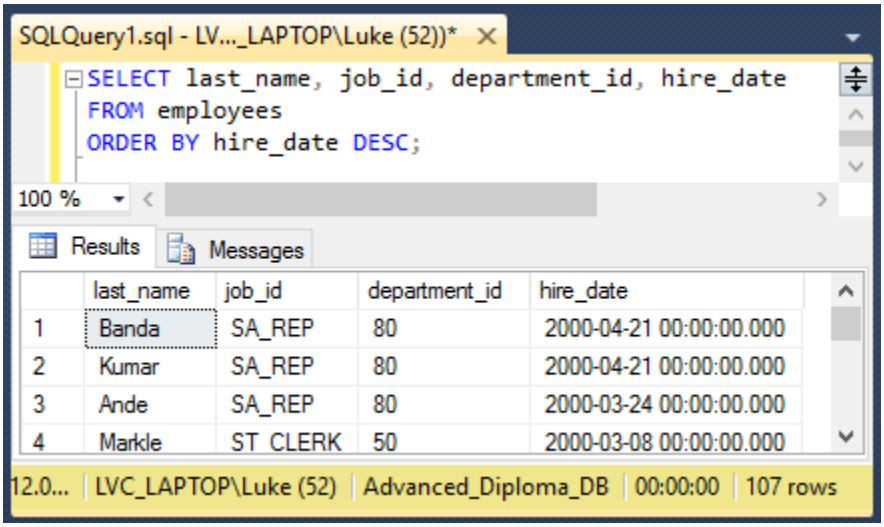


Figure 25 - Result of example 25: The returned rows are sorted using the hire_date in descending order

2. Using the COLUMN ALIASES

Example 26: Write a query sort the results using the column alias used in the SELECT statement.

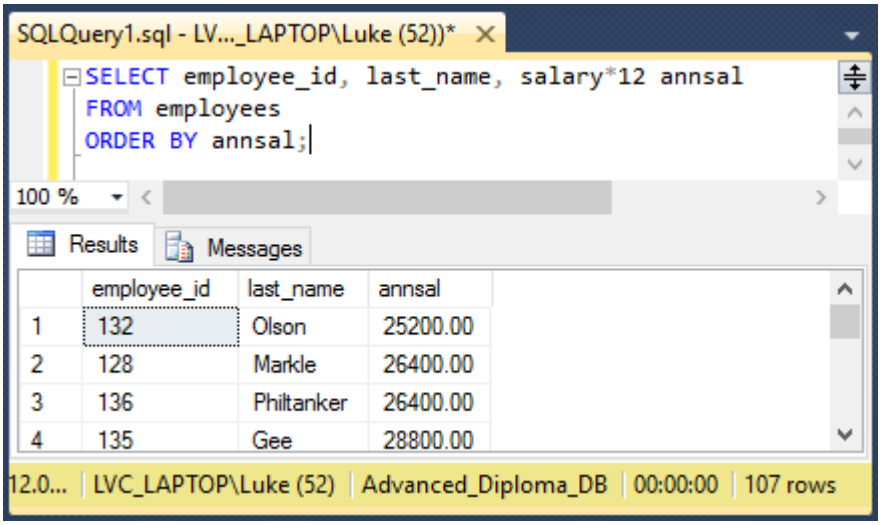


Figure 26 - Result of example 26: The result will be sorted according to the values in the last column

NOTE: The rows are ordered using the last column which is named annsal. The rows are automatically sorted in ascending order in fact the employee which earns the least is placed at the very top.

3. Using COLUMN NUMERIC POSITIONS

Example 27: Write a query that will order the rows obtained according to the third column, which happens to be the department_id

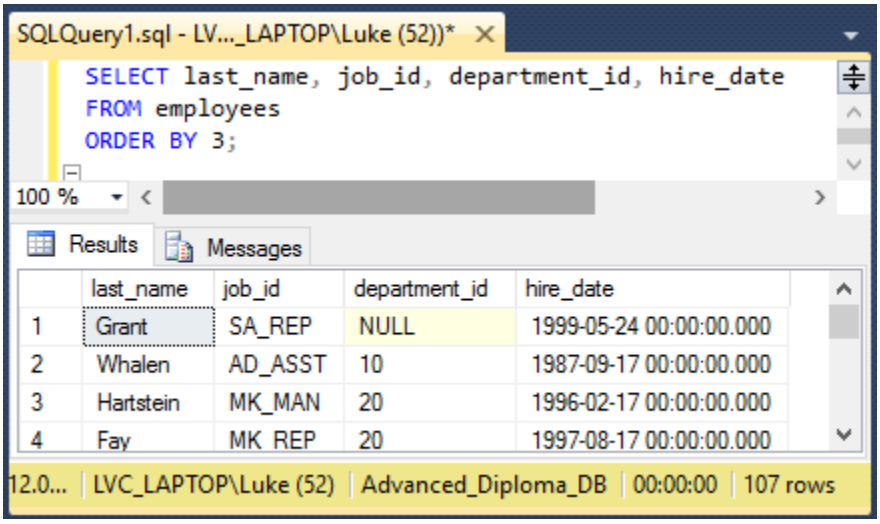


Figure 27 - Result of Example 27: sorting the rows using the position of the column as specified in the SELECT

Multiple column sorting

Sometimes you would be required to sort the data using multiple columns. This is the case when certain columns in the entity have the same value and hence other columns need to be used to sort the data even further. A typical example is when you are sorting people using the surname. If the entity contains people with the same surname, it would make sense to sort the data even further possibly using the name.

Example 28: Write a query that sorts the results first via the department_id is ascending order, then followed by the salary in descending order.

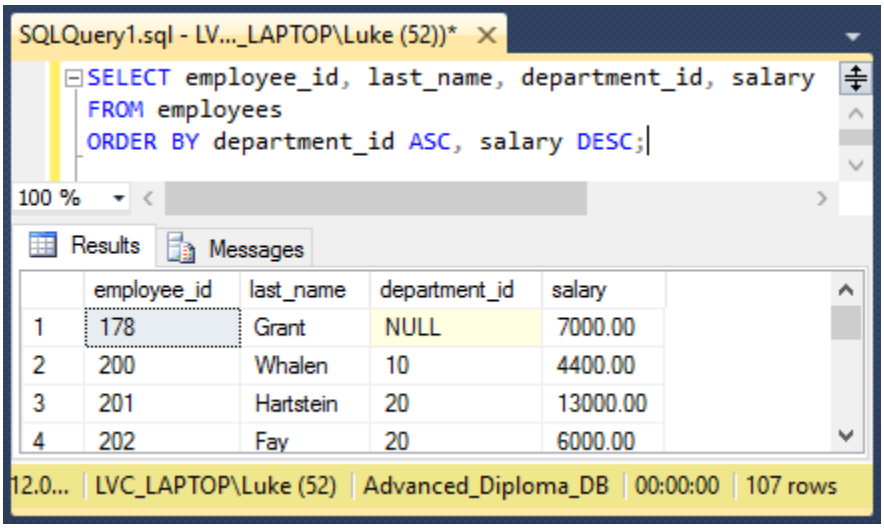


Figure 28 - Result of example 28: multiple column sorting is used.

NOTE: the above result works in this way. The results are first ordered using the department_id column in ascending order. In the case that more than one rows for a particular department_id exist, the second column in the ORDER BY is taken into consideration. In Figure 28 this is evident for employees who work in department 20.