RETRIEVE DATA FROM MULTIPLE TABLES

This topic is a very important with respect to databases which have been designed to adhere to E.F Codd's rules of normalization. As you have learnt during the first semester it is nowadays common practice to normalise databases. The process of normalization results in the creation of a database which is made up of a number of tables which are linked together through relationships. Having data related to the same subject stored in different tables poses a bit of a problem while retrieving data. For this reason the use of **JOINS** is paramount.

When data related data is stored in different tables, the relationship between the tables is commonly created with the help of foreign keys. For this reason in order to obtain the data a JOIN operation between the two tables is required – this operation usually requires the use of the primary key in one table and the foreign key in the other table.

Below is a typical example, where this type of operation is required. Given the employees and departments table, we would like to display the employee_id, department_id and department_name in our result.

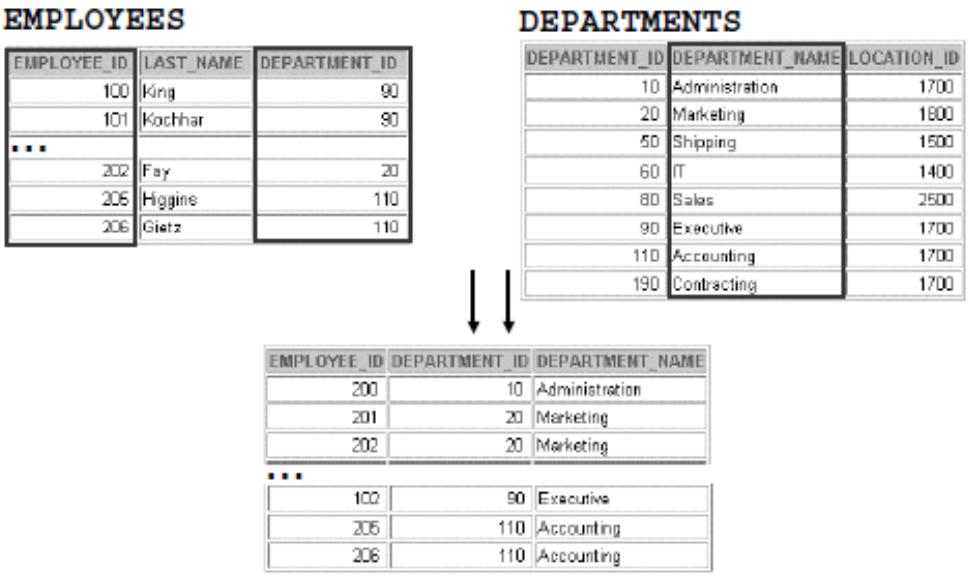


Figure 1 - Employees and Departments table joined together

In order to be able to perform different type of operations while obtaining data from different tables there are different type of JOINS:

- Inner Join – Equijoin, non-equijoin, self join
- Outer Join – Left, Right, Full
- Cross Joins

The general syntax for the use of JOIN operations is:

```
SELECT table1.column, table2.column, . . .
FROM table 1
        [[INNER] JOIN ON (table1.column = table2.column)]
        [LEFT | RIGHT | FULL OUTER JOIN ON (table1.column = tale2.column)]
        [CROSS JOIN table2]
```

where:

- **table1.column** denotes table and column from which data is to be retrieved
- **JOIN *table* ON *table1.column*** performs an equijoin based on the condition in the **ON** clause **= *table2.column***
- **LEFT | RIGHT | FULL OUTER** is used to perform outer joins
- **CROSS JOIN** returns a Cartesian product from the two tables

Inner Joins

This is the first type of Join operation that we will be considering in this topic. This operation requires the use of a comparison operator such as '=' or '<>'. The specified operator is used to compare rows from different tables based on the values in the specified columns. There are a number of different classifications for inner joins:

i) EQUIJOINS

This type of inner join will return a result which includes only those rows in the two tables which have a value which is the same for the columns which are specified within the ON clause. This means that if the columns which are being compared are not equal or contain a NULL value, they will not be returned. The examples below give a better demonstration of these type of joins.

NOTE:

- Any statement which makes use of JOINS can still make use of other clauses such as WHERE, ORDER BY and others.

Example 1: Write a query that will display the name, surname and department name of all the employees. Note that the first two columns are to be obtained from the employees table and the last column is obtained from the departments table

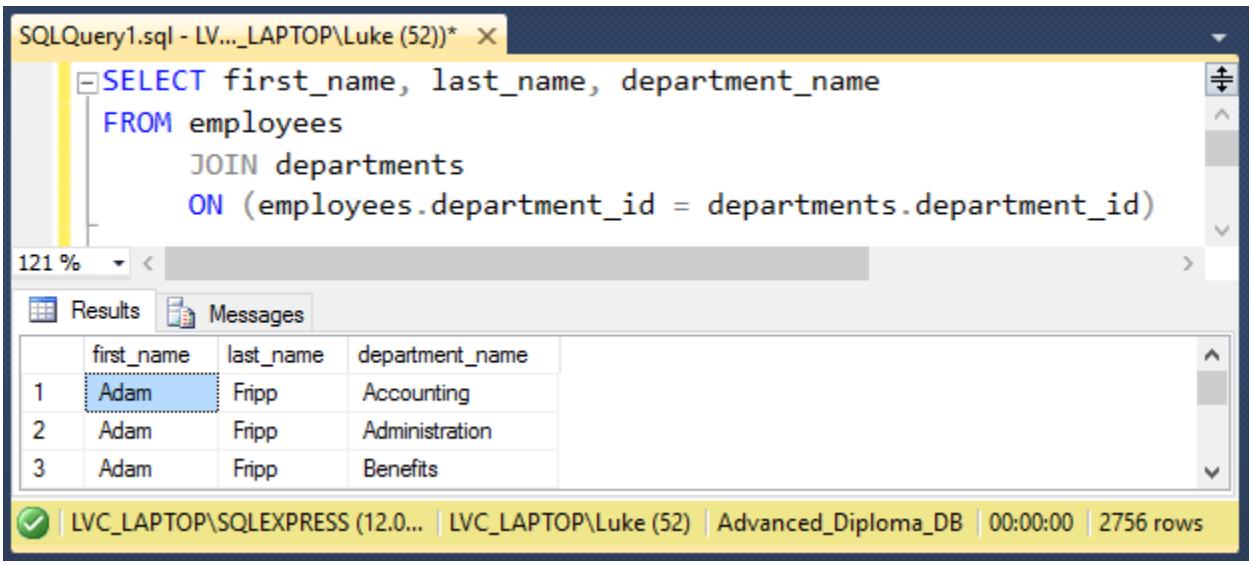


Figure 2 - Result of example 2: Equijoin using the employees and departments table

Example 2: Write a query that will display the department id, department name, and city of all the departments. Note that the first two columns are to be extracted from the departments table and the last column is obtained from the locations table

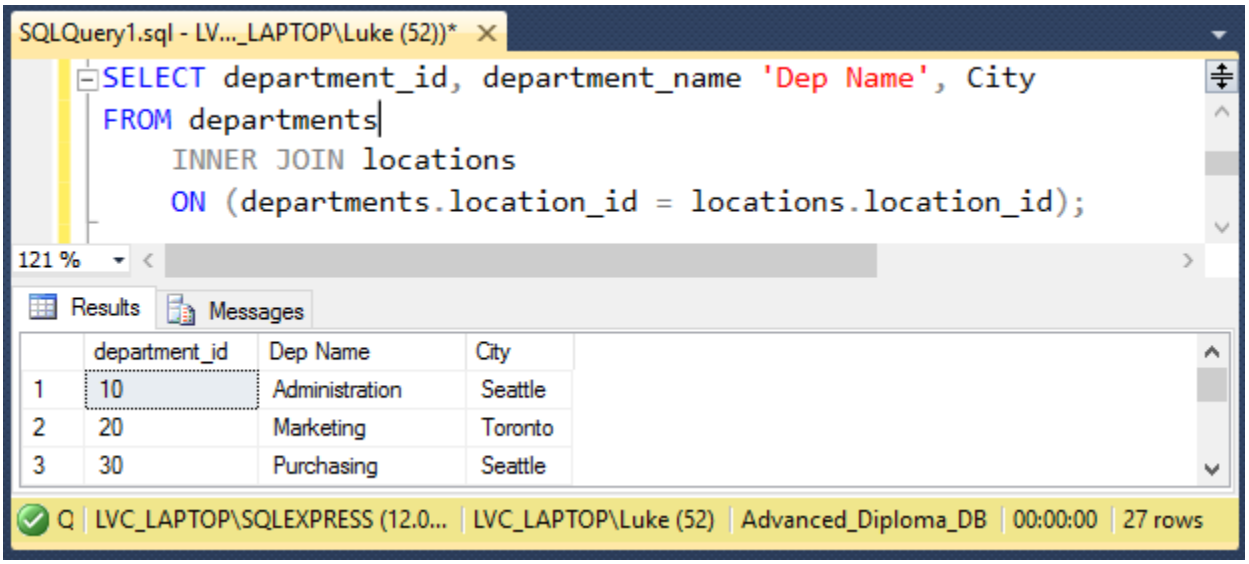


Figure 3 - Result of example 2: Equijoin to extract information from the departments and locations table

Example 3: Write a query that will display the department id, department name, and city of the departments with an id of 20 and 50. Note that the first two columns are to be extracted from the departments table and the last column is obtained from the locations table

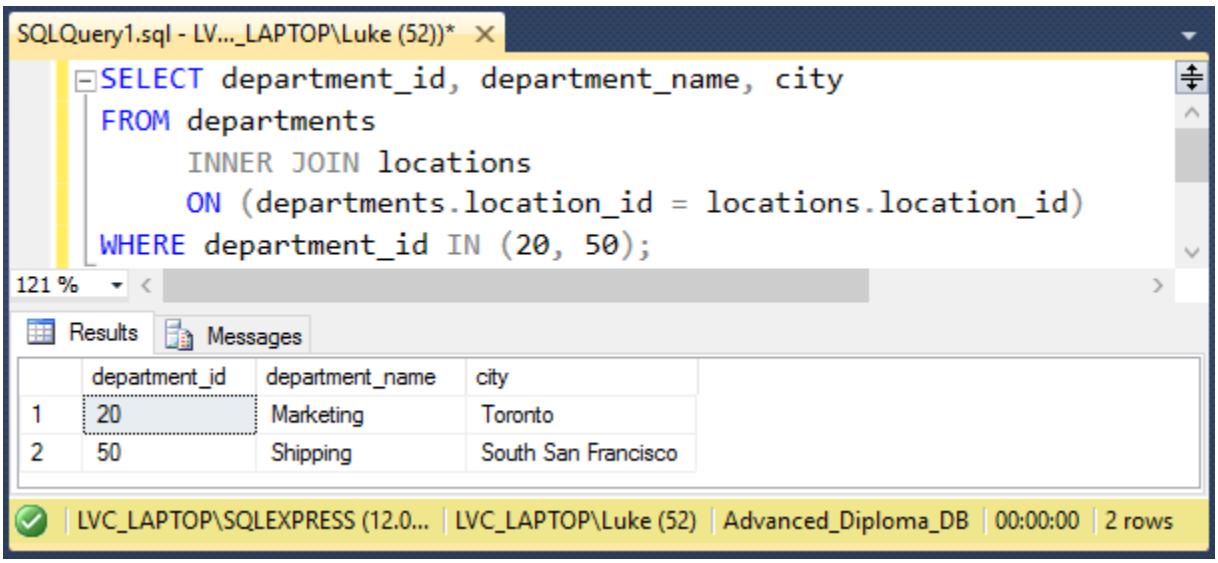


Figure 4 - Result of example 3: Retrieving information from multiple tables and restricting rows

Example 4: Write a query that will display the name, surname and job_title of all employees whose name starts and ends with an 'a'. The first two columns are to be retrieved from the employees table and the last column from the jobs table. The final result should be sorted using the name in ascending order.

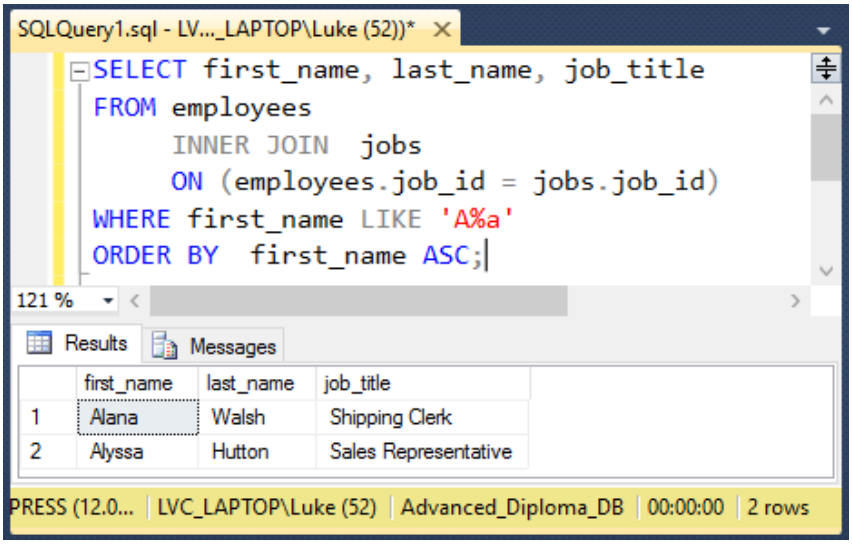


Figure 5 - Result of example 4: Data from multiple tables with WHERE and ORDER BY clauses

NOTE: Qualify Ambiguous Column Names

- A very important point while retrieving data from multiple tables is that of stating the table name from where the columns are to be retrieved. In all the previous examples this has not been done but it will help in improving the query's performance
- By using ambiguous column names, you will be able to distinguish the source table for columns with identical names
- This technique requires the use of *table aliases*:
    - This simplifies the complexity of code (code becomes shorter, hence less memory). It will therefore improve performance
    - Table aliases should have the following characteristics: should be meaningful and should not be too long

Example 5: Write a query that will display the employee number, surname, department no (from the employees and departments tables) and the department name. Notice that this example is using Column Ambiguous Names, where the employees table is renamed 'e' and the departments table is renamed to 'd'. Also notice that all the columns have this 'new name' included in front of them.

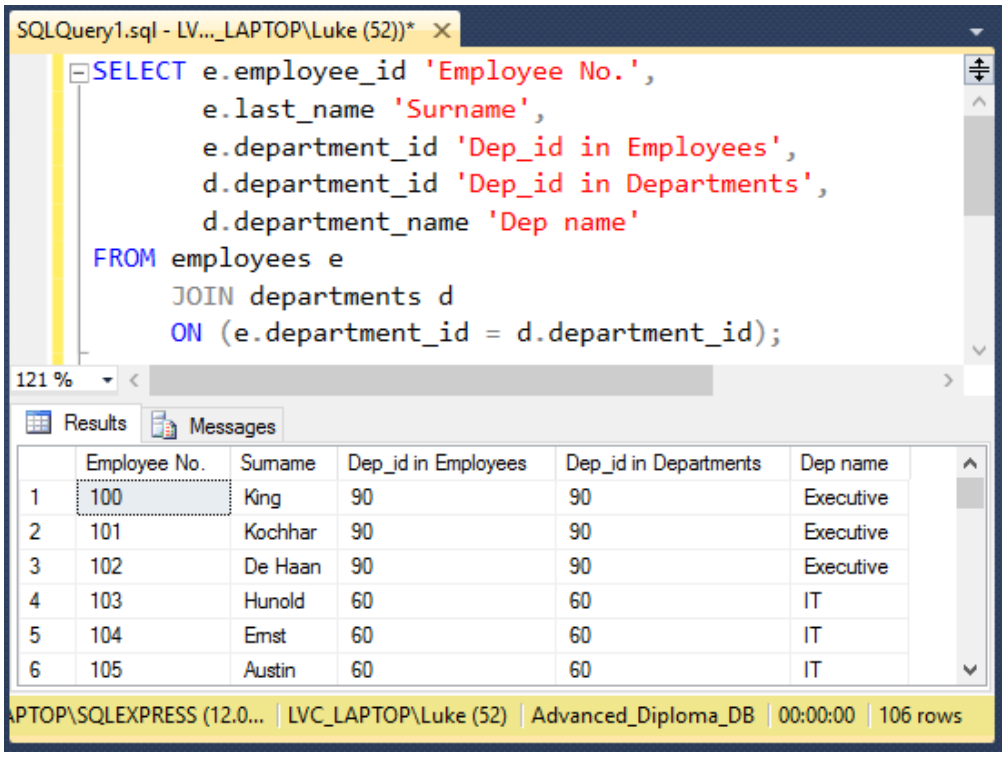
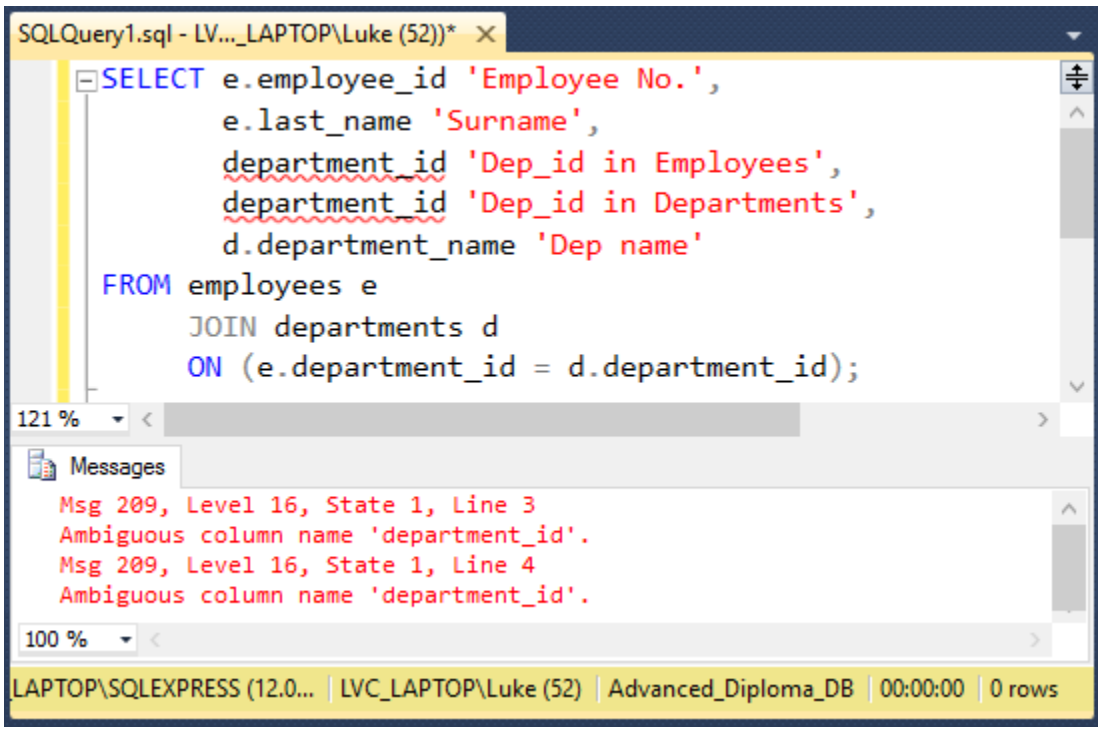


Figure 6 - Result of example 5: Example using column ambiguous names

Example 6: Two problematic statements due to the lack of column ambiguous names. In the first screenshot below, table aliases are included but they are not used with the columns in the SELECT statement. This does not allow SQL Server to determine from where it is to retrieve the department_id columns. The same applies to the second screenshot which does not specify from which table the location_id column is to be retrieved



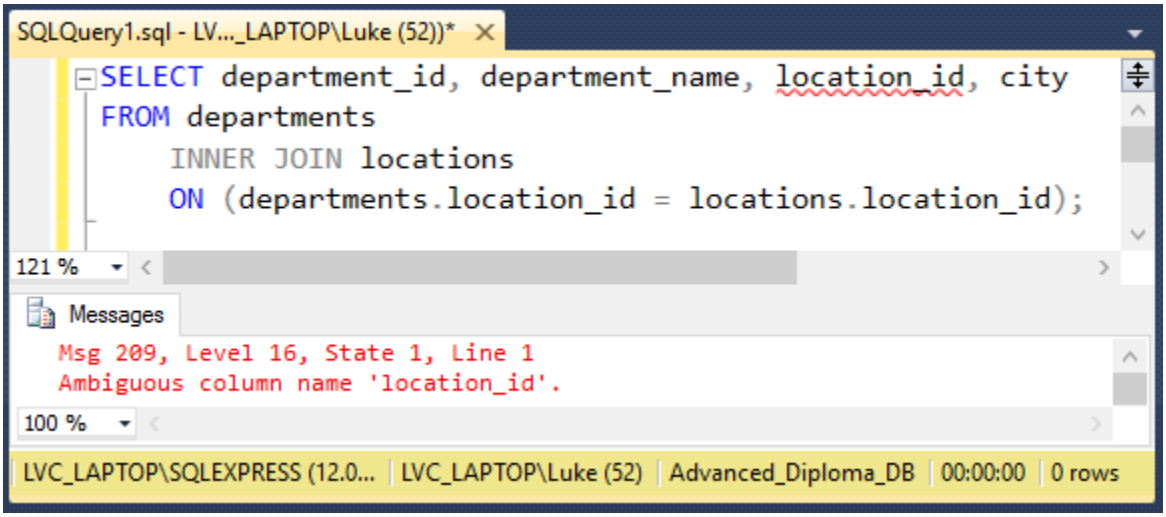


Figure 7 - Result of example 5: Example using column ambiguous names

ii) NON-EQUIJOINS

All the previous example included the joining of data from different tables whenever an equality operation is available. These type of inner joins, include a condition that contains an operator which is *not* an equality operator.

This kind of inner join can be best explained by considering the ***employees*** and ***job_grades*** tables. These two tables do not have a physical relationship (PK and FK relationship) but there exists a logical relationship, that of the *salary* column (in employees) and the *lowest_salary* and *higest_salary* (in job_grades). This relationship can be computed via a non-equal comparison.

EMPLOYEES

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Ernst	6000
Lorentz	4200
Mourgos	5800
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500
Zlotkey	10500
Abel	11000
Taylor	8600

JOB_GRADES

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

Example 7: Write a query that will display the surname salary and grade level of all the employees. The grade level can be obtained by determining in which range the salary falls in. The >= and <= can be used instead of the BETWEEN operator.

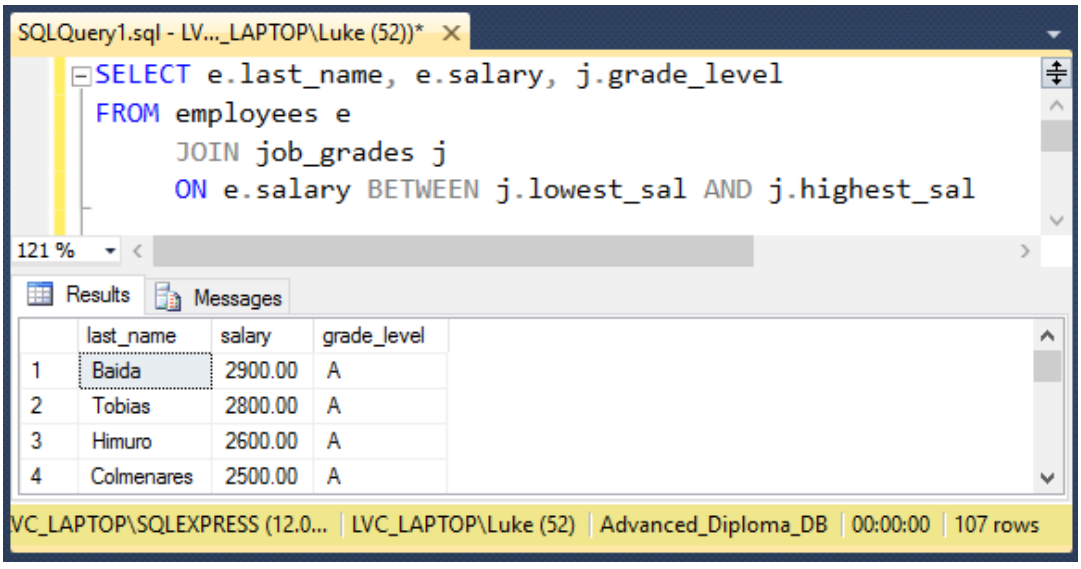


Figure 8 - Result of example7: an example of a non-equijoin

iii) SELF JOIN

The SELF Join operation is not as commonly used as the equijoin. This is due to the fact that this operation requires a recursive relationship. This type of relationship is not as common as normal 1-1 or 1-many relationships and therefore SELF joins are also not so common. A SELF join is a join operation which joins a table to it-self (this means that the table is to be considered twice within one single statement).

In order to explain the concept of SELF joins, the employees table will be considered. Within the employees table, the employee_id and manager_id columns are related together as they have a relationship (recursive) between them.

EMPLOYEES (WORKER)

EMPLOYEE_ID	LAST_NAME	MANAGER_ID
100	King	
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103
107	Lorentz	103
124	Mourgos	100

...

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
124	Mourgos

...

The above table previews show the data that is found in the employees table. The manager_id column refers to the employee_id column. If we had to find the surname of Lorentz's manager, we would have to follow the below steps.

- Search for 'Lorentz' in the last_name column of the employees table
- Get to know the manager_id value for Lorentz
- Find the surname of the person who has an employee_id which is equal to the manager_id of Lorentz. In this case Hunold would be the manager of Lorentz

Example 8: Write a query that will display the employee's surname and his manager's surname.

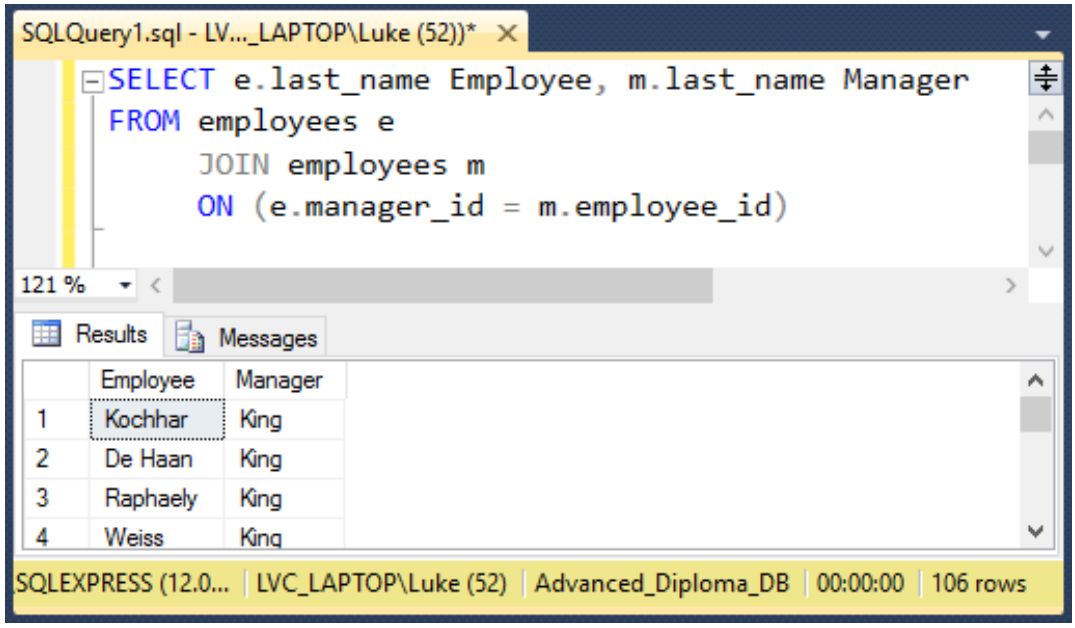


Figure 9 - Result of example 8: SELF join to obtain manager

Example 9: Write a query that will display the employee's name and surname and his manager's name and surname. Make sure to sort your answer using the manager column first and the the employee column

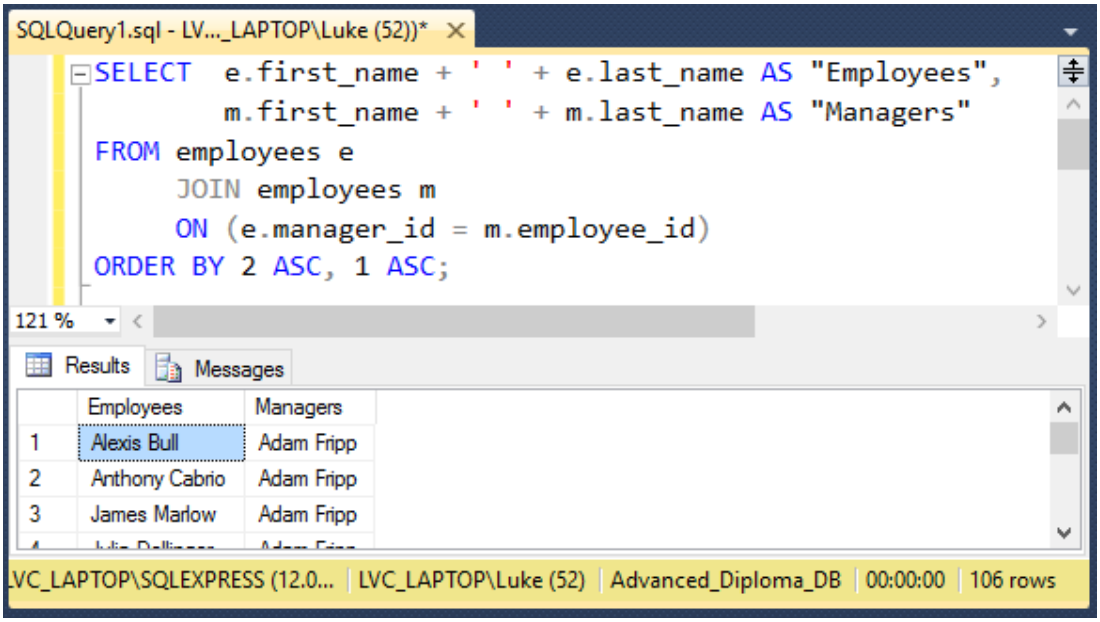


Figure 10 - Result of example 9: SELF join with concatenation

iv) Multi-way joins

In all the previous examples, data has been retrieved from only two tables. Sometimes we will be required to obtain data from 3 or more tables. In such cases multi-way joins are to be implemented.

Example 10: Write a query that will display the employee's name and surname (in one single column), department name and job title. You are to obtain the content of the first column from the employees table, that of the second column from the departments table and that of the third column from the jobs table. Your result should be sorted using the department name, followed by the job title and then the first columns.

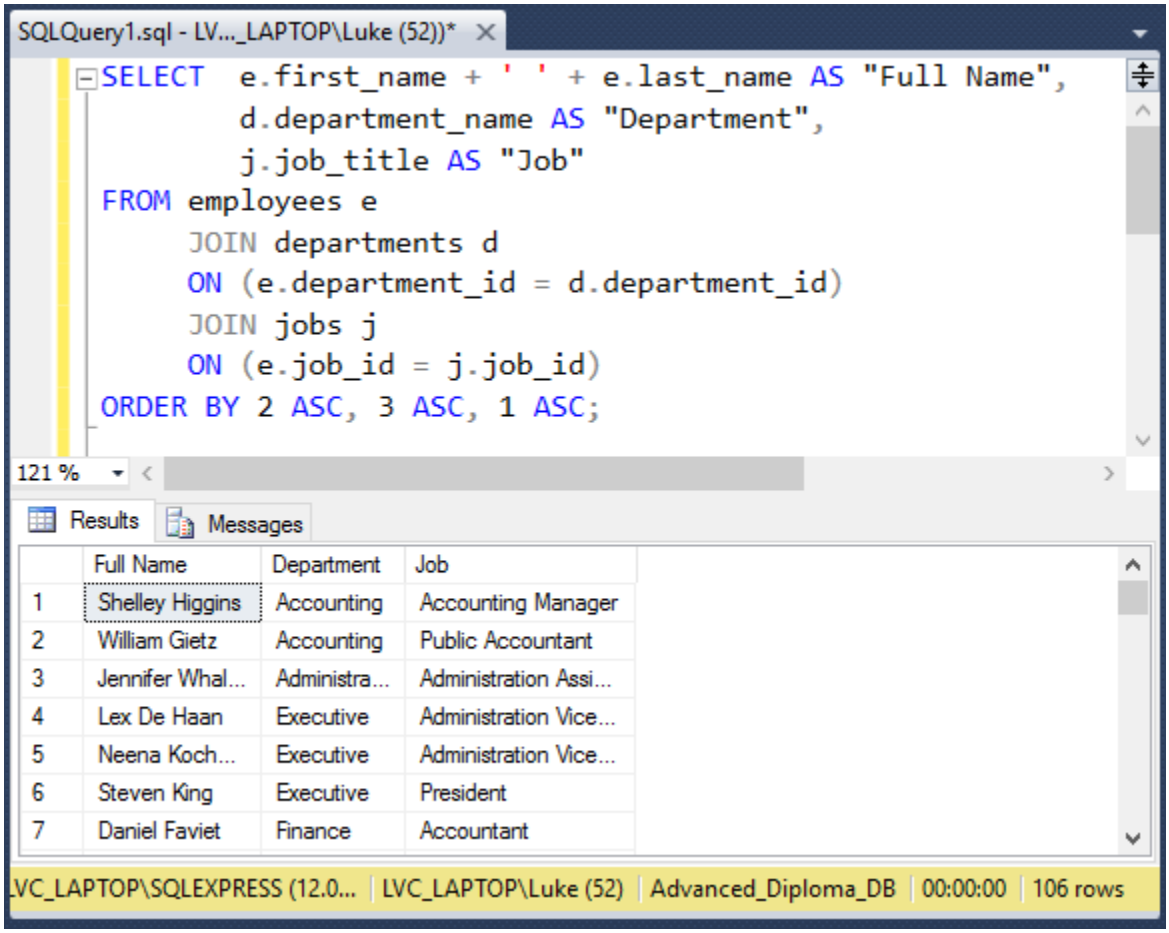


Figure 11 - Result of example 10: 3 way join involving the employees, departments and jobs tables

NOTE: Given that data is obtained from 3 different tables we require two join conditions. You are to take note of the importance of the conditions in the ON clauses.

Example 11: Write a query that will display the employee number, city and department name for each employee in the employees table. You will be using the employees, departments and locations tables.

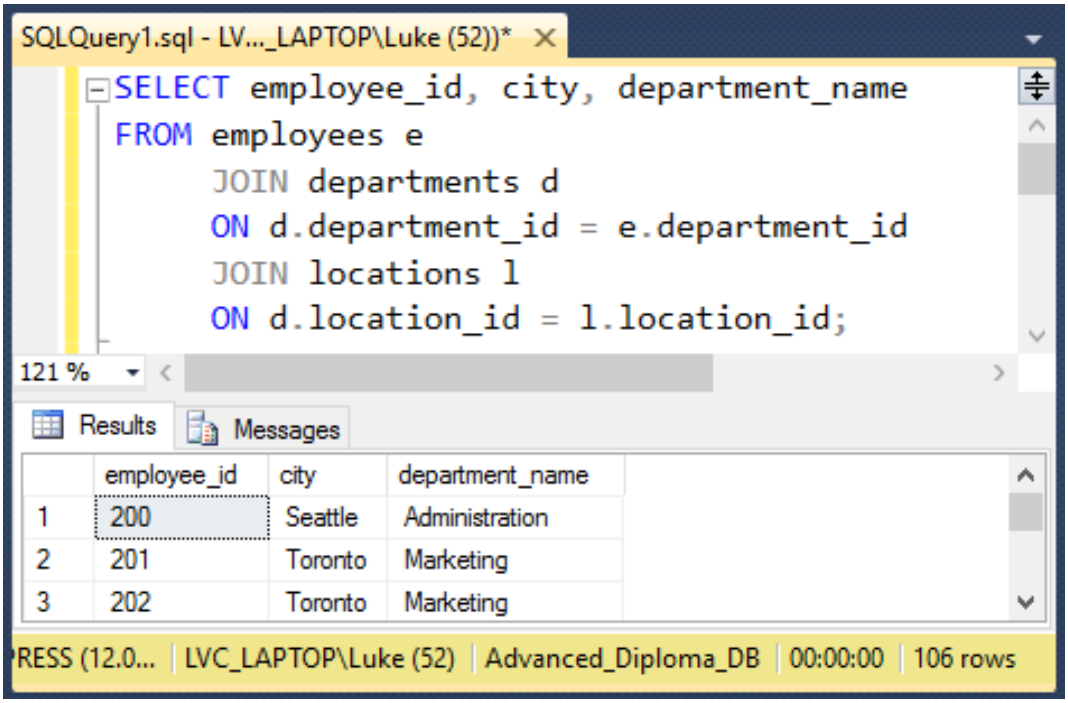


Figure 12 - Result of example 11: 3-way join using the employees, departments and location table

Outer Join

This category of joins is important when you wish to handle data which does not meet a particular join condition. As you might have notices, any row of data which does not satisfy the joining condition will not appear in the query result.

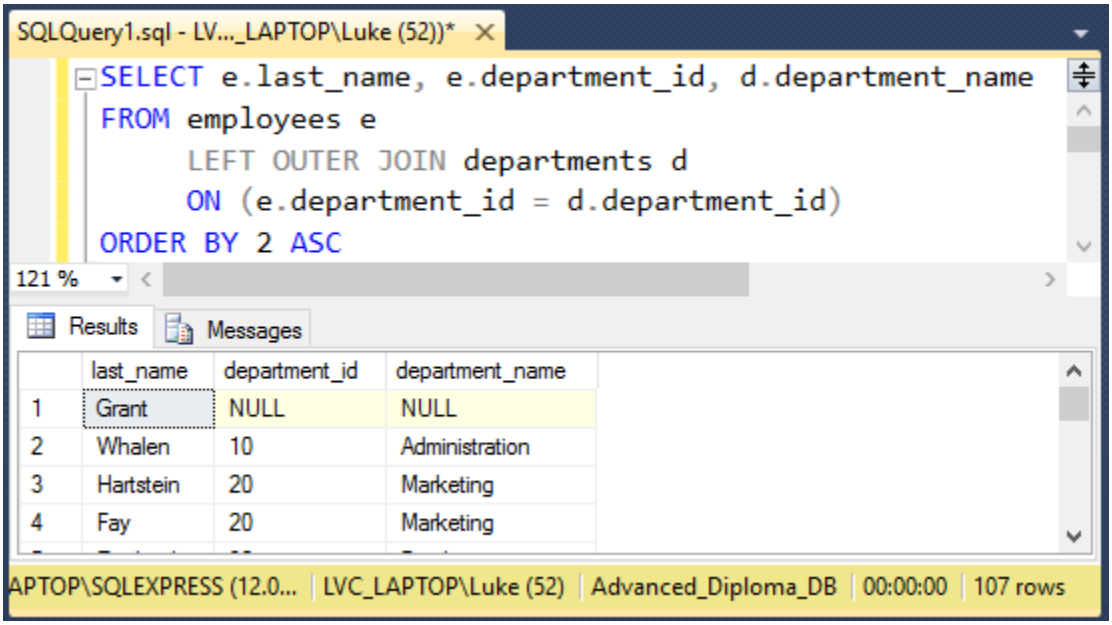
Difference between Inner and Outer Joins:

- Inner Join: this is a join which will return only the rows which satisfy the joining condition
- Outer Join: this will return the same result of an inner join together with any rows who do not satisfy/match the joining condition

In this section we will take a closer look at the three different type of outer joins, which are:

i) *LEFT Outer Join*: it returns all matching results and unmatched rows from the LEFT table
ii) *RIGHT Outer Join*: it returns all matching results and unmatched rows from the RIGHT table
iii) *Full Outer Join*: it returns all matching results and unmatched rows from both the LEFT and RIGHT table

Example 12: Write a query that will display the surname, department number and department name of all the employees both if they are assigned a department and even if they are not. In the first screenshot LEFT OUTER join is used as employees table is placed before the JOIN clause, hence the LEFT table. The second screenshot returns the same result but this time a RIGHT OUTER join is used as the employees table was placed after the JOIN clause, hence RIGHT table.



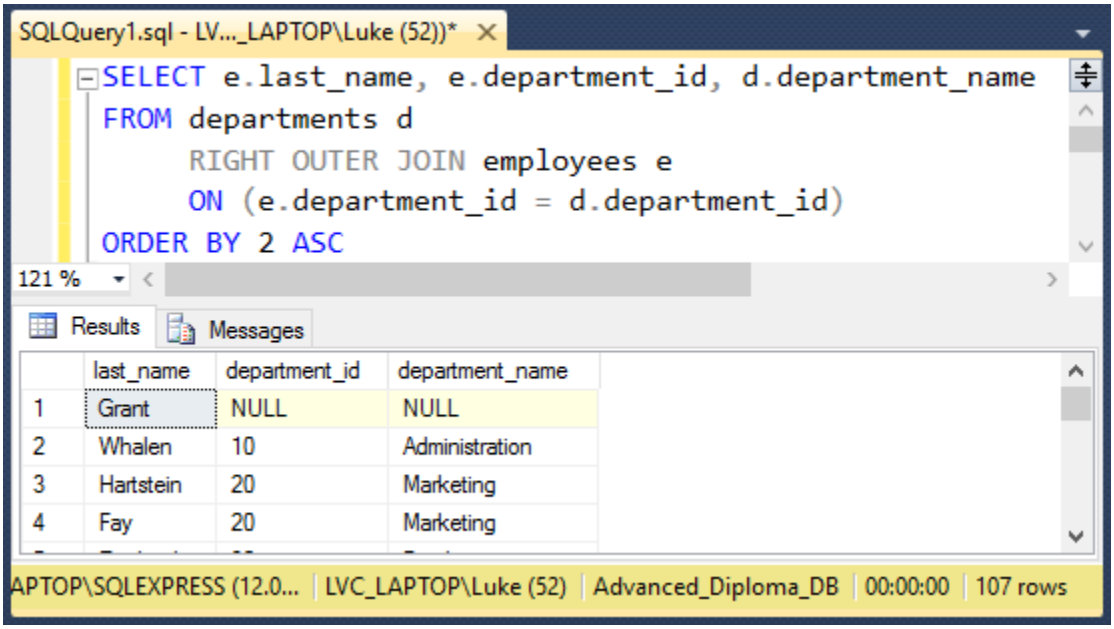


Figure 13 - Result of example 12: Two statements which return the same result

Example 13: Write a query that will display the surname, department number and department name of all the departments even if they do not have any employees assigned.

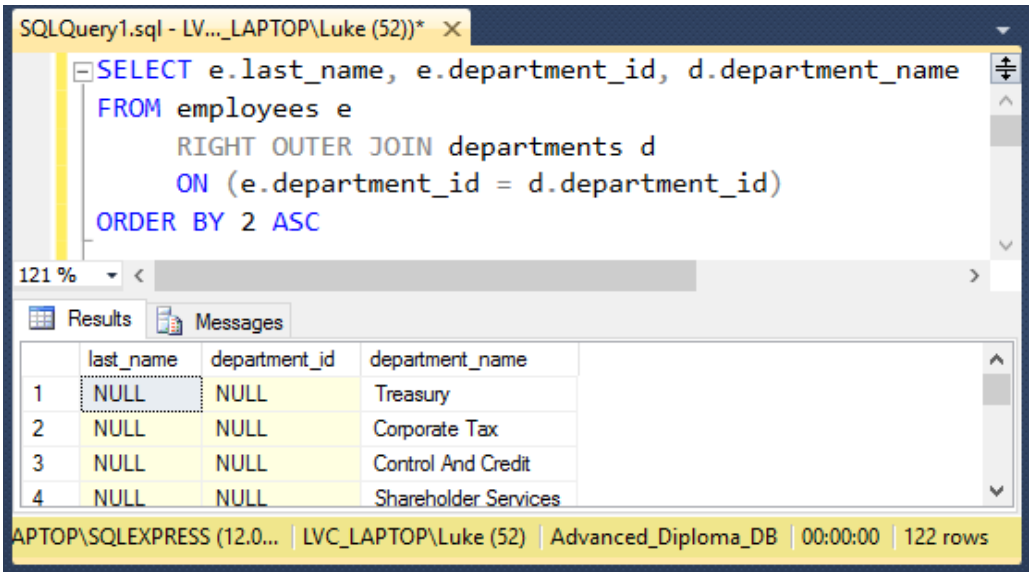


Figure 14 -Result of example 13: obtaining all departments even if they are not assigned any employees

Example 14: Write a query that will display the surname, department number and department name of all the employees both if they are assigned a department and even if they are not and also the departments even if they have been assigned an employee or not. Note that this example requires the use of a FULL OUTER JOIN as we need all the rows that match the ON condition, and all those rows in both the employees and departments table who do not match the condition in the ON clause

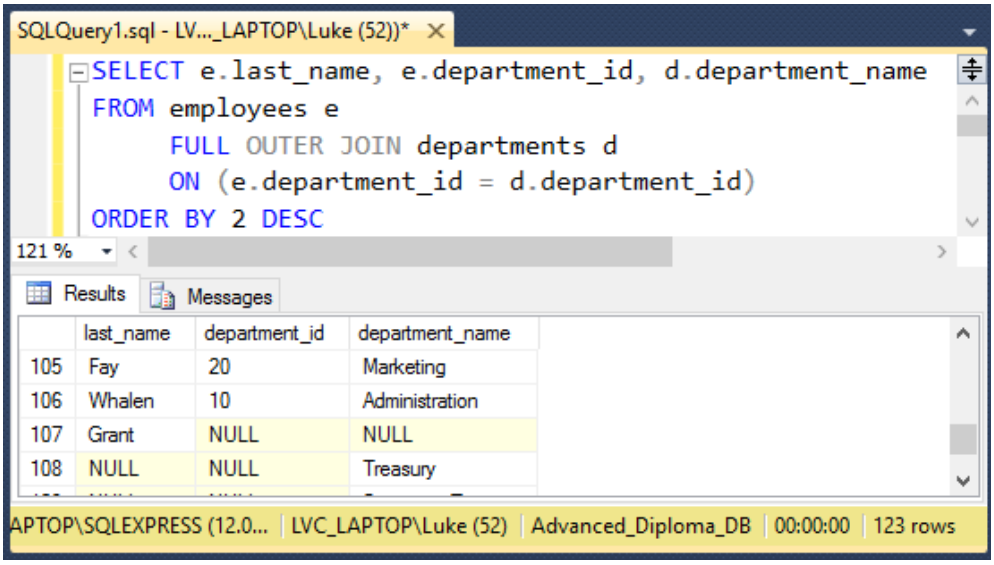
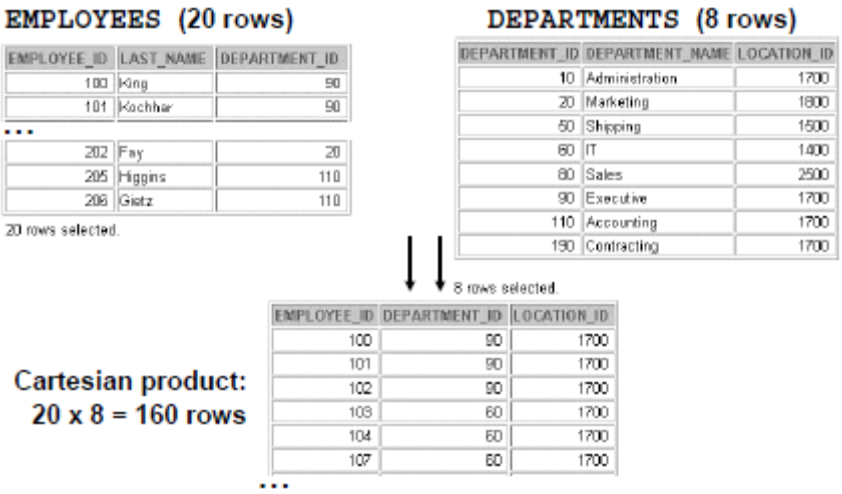


Figure 15 - Result of Example 14 - An example of a Full Outer Join

Cross Join

This type of join is also referred to as the Cartesian product. This operation tends to create a large number of rows of data which has very limited use. The CROSS JOIN operation is commonly used when large amounts of data are required for testing purposes. This operation produces the cross-product of the two tables – all the rows in the first table will be joined to all the rows in the second table.



Example 15: Write a query that will return the surname and department name columns in a Cartesian operation between the employees and departments tables

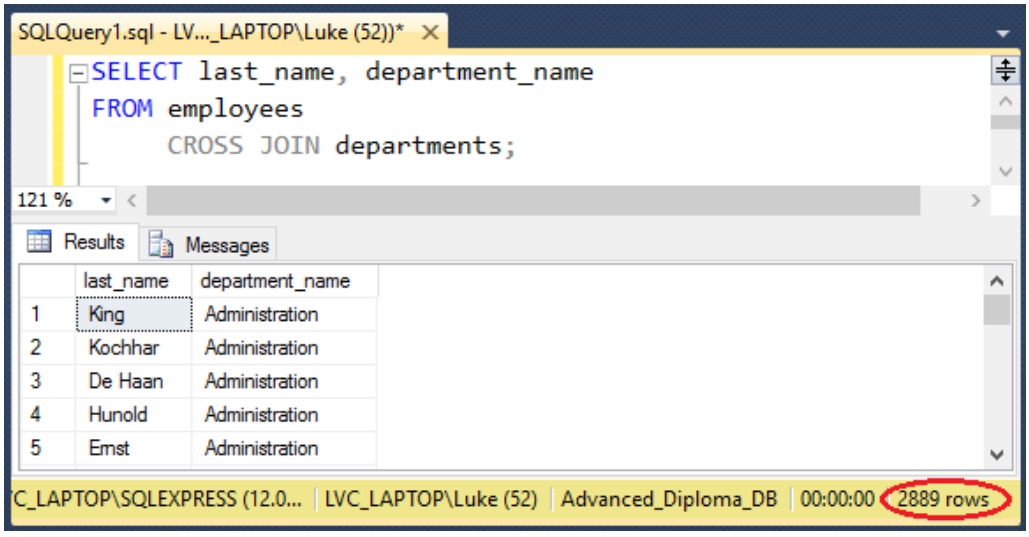


Figure 16 - Result of example 15: Cartesian product between the employees and departments tables