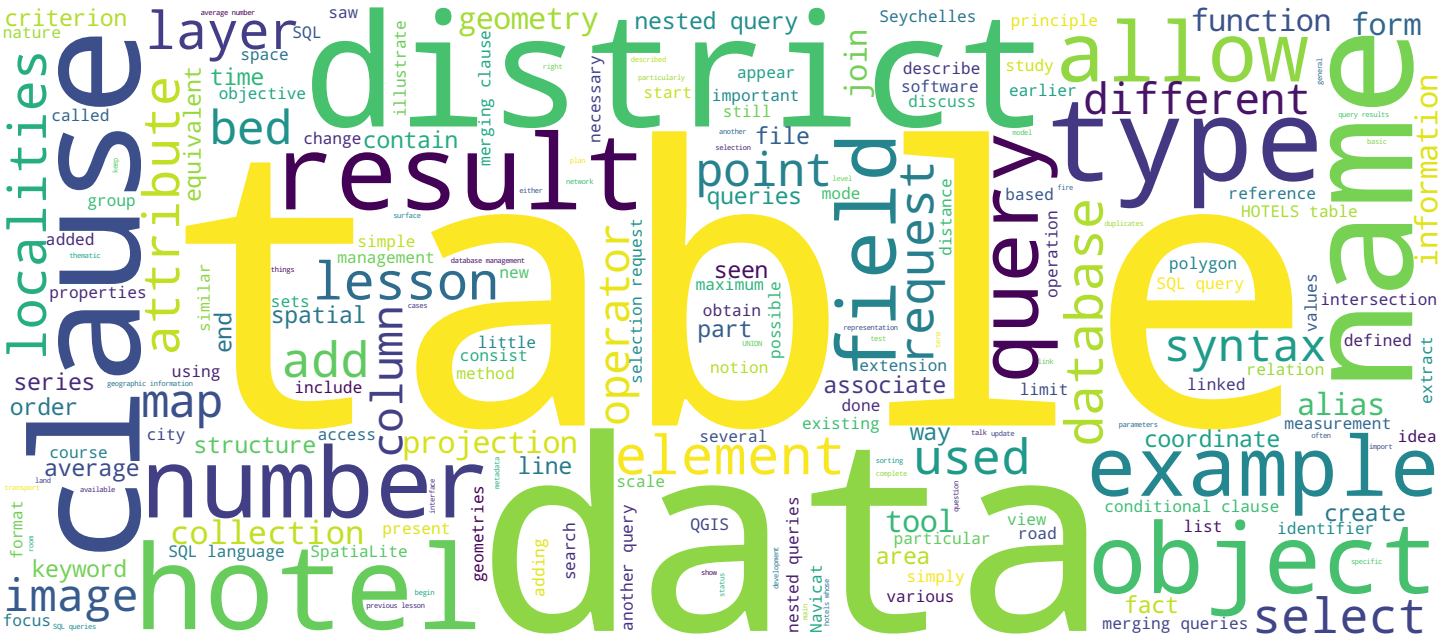
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

RESCIF
Réseau d'excellence des sciences de l'ingénieur de la Francophonie

An Introduction to Geographic Information Systems

Merging queries, embedded queries

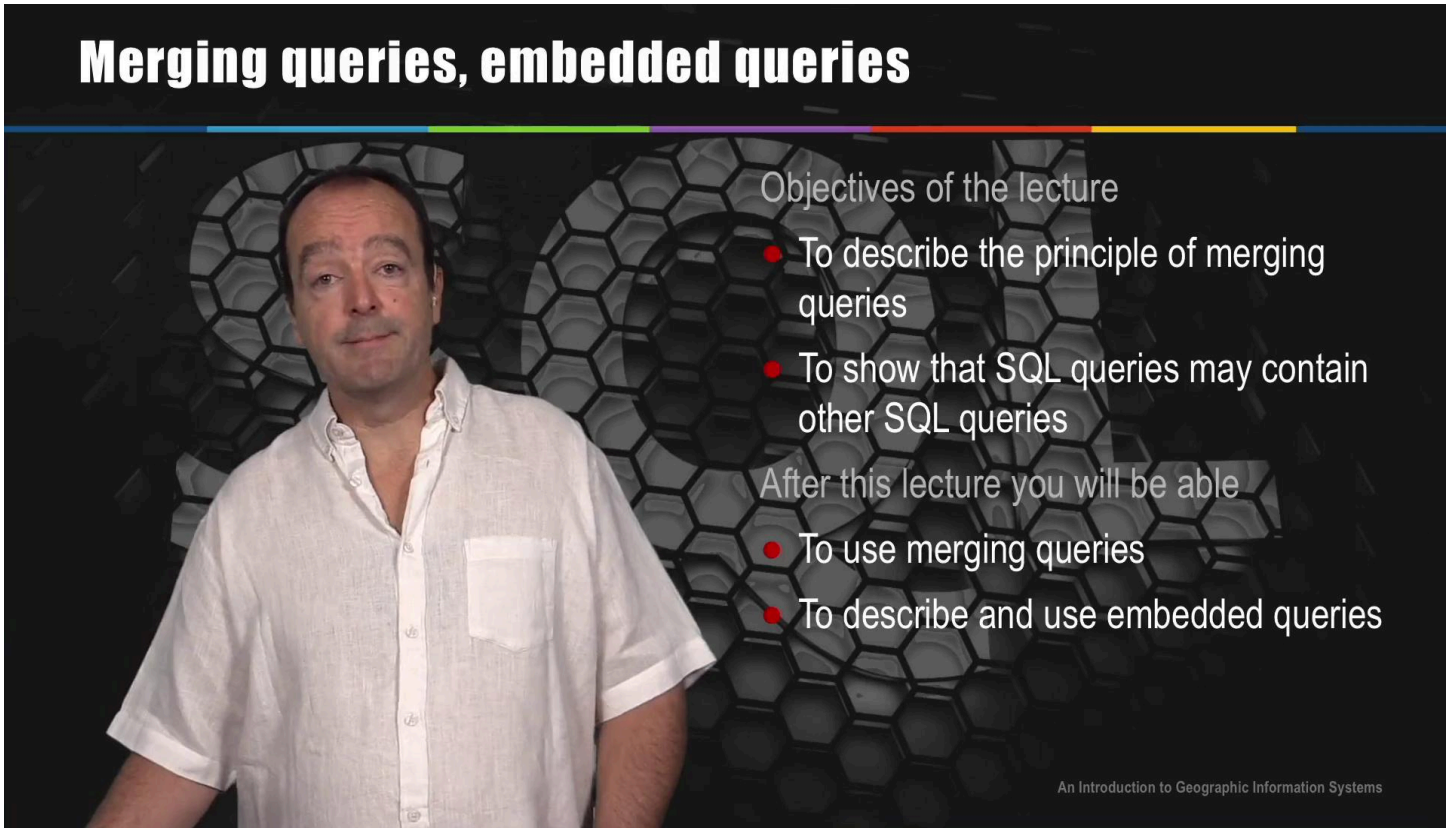
Stéphane Joost, Marc Soutter, Fernand Kouamé, Amadou Sall



Search MOOC

Video

EPFL

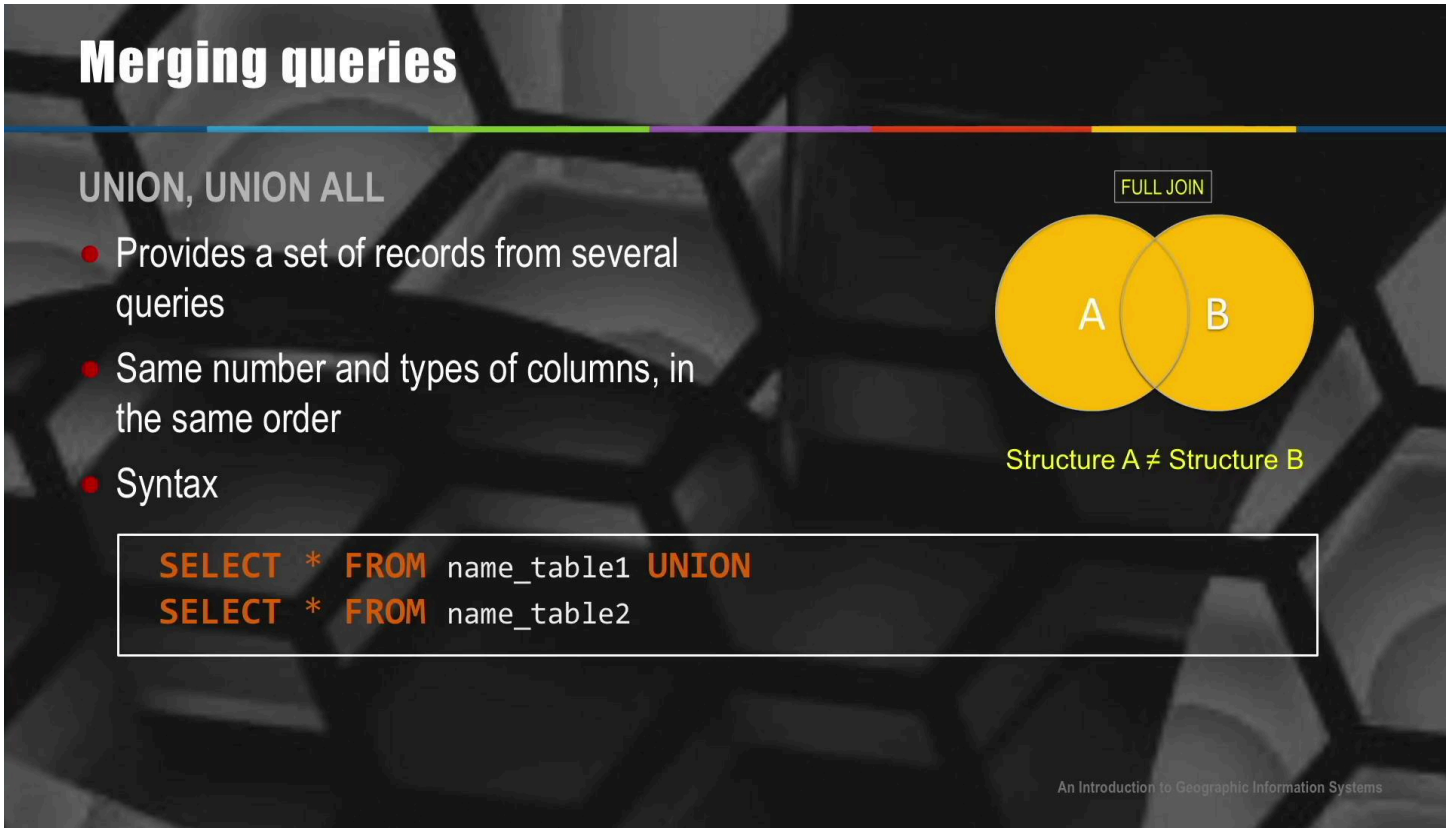


Hello. This lesson will focus on the merging of queries which we use when we want to associate the results of two or more queries to extract the common elements or to remove from all the query results the group of elements that would be present in the results of another query. We will also discuss the use of query results as part of another query in the case where we nest a query into another query, what we do for example when we would like to use an aggregation function as the average number of employees of a series of companies as a criterion to select the SMEs which would have fewer employees than the average. The objective of this lesson is to describe the principle of merging queries and to show that SQL queries can contain other SQL queries, so that at the end of the lesson you are able to use merging queries to assemble results, to associate the results of multiple queries and to write nested queries.

Notes

Summary

0m 05s



In this lesson, we will discuss successively the topic of the merging queries then we will see the nested queries in the WHERE clause before talking about nested queries in the FROM clause and we will end with the use of the IN and NOT IN operators in merging queries. By using this table which summarizes the basic syntax elements of the SQL language, we find all the selection, conditional filtering, aggregation and sorting clauses that we saw in previous lessons. We see that we still have to discuss these merging clauses with the UNION, INTERSECT and EXCEPT keywords. The UNION clause of the SQL language allows us to put end to end the results of several queries which themselves use the SELECT command. It is therefore a command that allows to concatenate the results of two queries or more. To use this merging clause it is important that both requests that we seek to combine are structured in the same way, so they have the same number and the same type of columns and these columns appear in the same order in the two tables that are linked by the union query. From there, the syntax is simple it is two SELECT clauses which are simply connected by the UNION keyword.

Notes

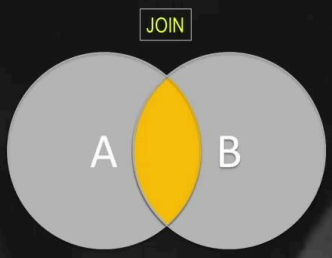
Summary

1m 38s

Merging queries

INTERSECT

- Provides the intersection of two sets of query results (record belonging to both)
- Same number and types of columns, in the same order
- Syntax



Structure A ≠ Structure B

```
SELECT * FROM name_table1 INTERSECT
SELECT * FROM name_table2
```


In the spirit it is an operation which is similar to that of the FULL JOIN that we have seen in previous lessons, except that in the join the structure of the two sets that we associate can be completely different. We have fields that are totally different whereas here, when we unite the two tables of two queries it is really necessary that the structure is the same in both cases. The particularity of the UNION ALL clause consists of not eliminating the duplicates so if we have elements that are present both in the first and in the second table they both appear whereas in the simple union query the duplicates are eliminated. Second type of merging query, the intersection queries with the INTERSECT clause which allows to obtain the intersection of the results of two queries. As in the case of the union it is important, and even essential, that the two tables which we use have the same number and the same type of columns and that these columns are listed in the same order. From there, the syntax is of the same nature with two classic selection queries connected by the INTERSECT keyword, by the INTERSECT clause.

Notes

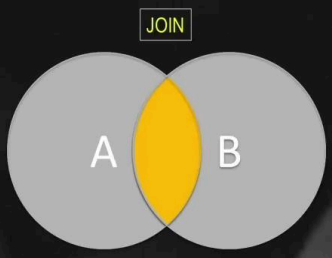
Summary

3m 08s

Merging queries

INTERSECT

- Provides the intersection of two sets of query results (record belonging to both)
- Same number and types of columns, in the same order
- Syntax



Structure A ≠ Structure B

```
SELECT * FROM name_table1 INTERSECT
SELECT * FROM name_table2
```


As before we can link to the idea of join where we had the simple join which consisted in linking the elements of the two sets, of the two queries, through a common field. The difference here is that the intersection between two requests in a merging clause implies that the structure of the two sets is the same and not that we associate two sets with rather variable attributes.

Notes

Summary

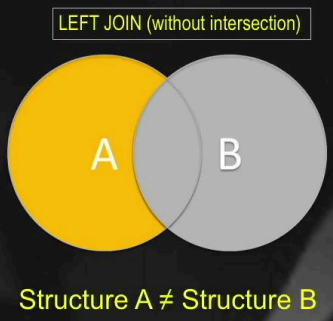
4m 27s

Merging queries

EXCEPT (MINUS)

- Provides the records returned by a query that do not belong to the set of records returned by another query
- Same number and types of columns, in the same order
- Syntax

```
SELECT * FROM name_table1 EXCEPT
SELECT * FROM name_table2
```



An Introduction to Geographic Information Systems

Third type of merging request that allows to extract from the query results those which would not be part of a second query. With the EXCEPT or MINUS clause, in some DBMS, especially in MySQL, the MINUS clause must be used. Again, we must have the same number and the same type of columns in the same order in the 2 tables with a syntax always of the same nature, two selection queries connected by the merging clause. This type of fusion resembles a left join without intersection. In set language with this representation, except that in the case of a join the structures of the tables are not the same.

Notes

Summary

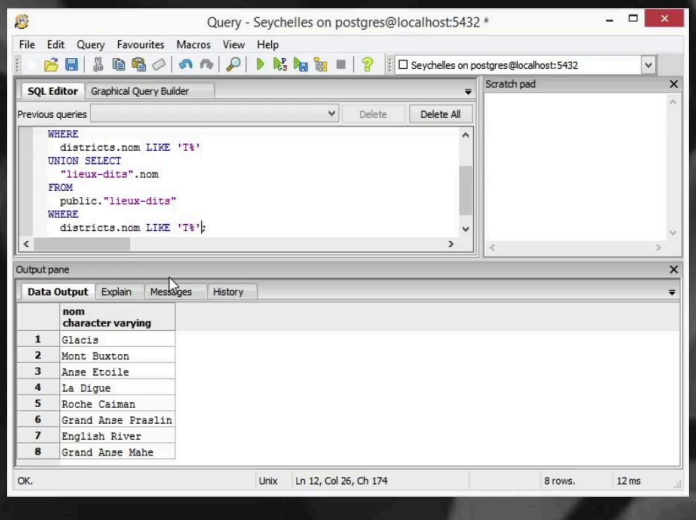
5m 02s

Merging queries

UNION, UNION ALL

INTERSECT

EXCEPT



To illustrate these merging queries we take the Seychelles database with the table of districts and localities and the fields that contain the names of these two tables. The query itself that would combine these two tables is written as you see it and we simply copy this query to reproduce it a second time before sorting it out by deleting in the first case the elements that are linked to the localities and in the second case the elements that are linked to the districts so as to obtain the table that associates the names of the districts and the names of the localities with 330 results. We can then simply add the ALL keyword to keep the duplicates and we see that we move to 371 results so there are about forty duplicates that appear. The intersection of the two sets this time gives us 17 results, so 17 localities that are similar in name to the districts. Finally, the subtraction of localities to all the district names gives 8 results, so 8 districts that do not have equivalents in the localities. We can refine the research a little by transforming the selection requests and by limiting the district research to those that begin with the letter T and then doing the same thing for the localities.

Notes

Summary

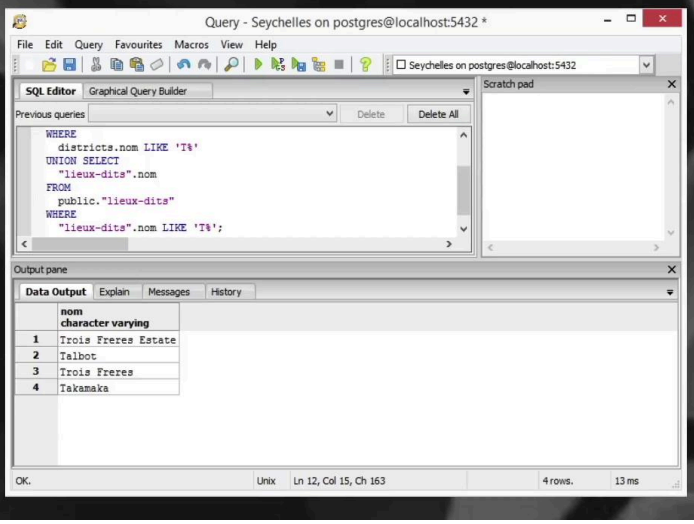
5m 56s

Merging queries

UNION, UNION ALL

INTERSECT

EXCEPT

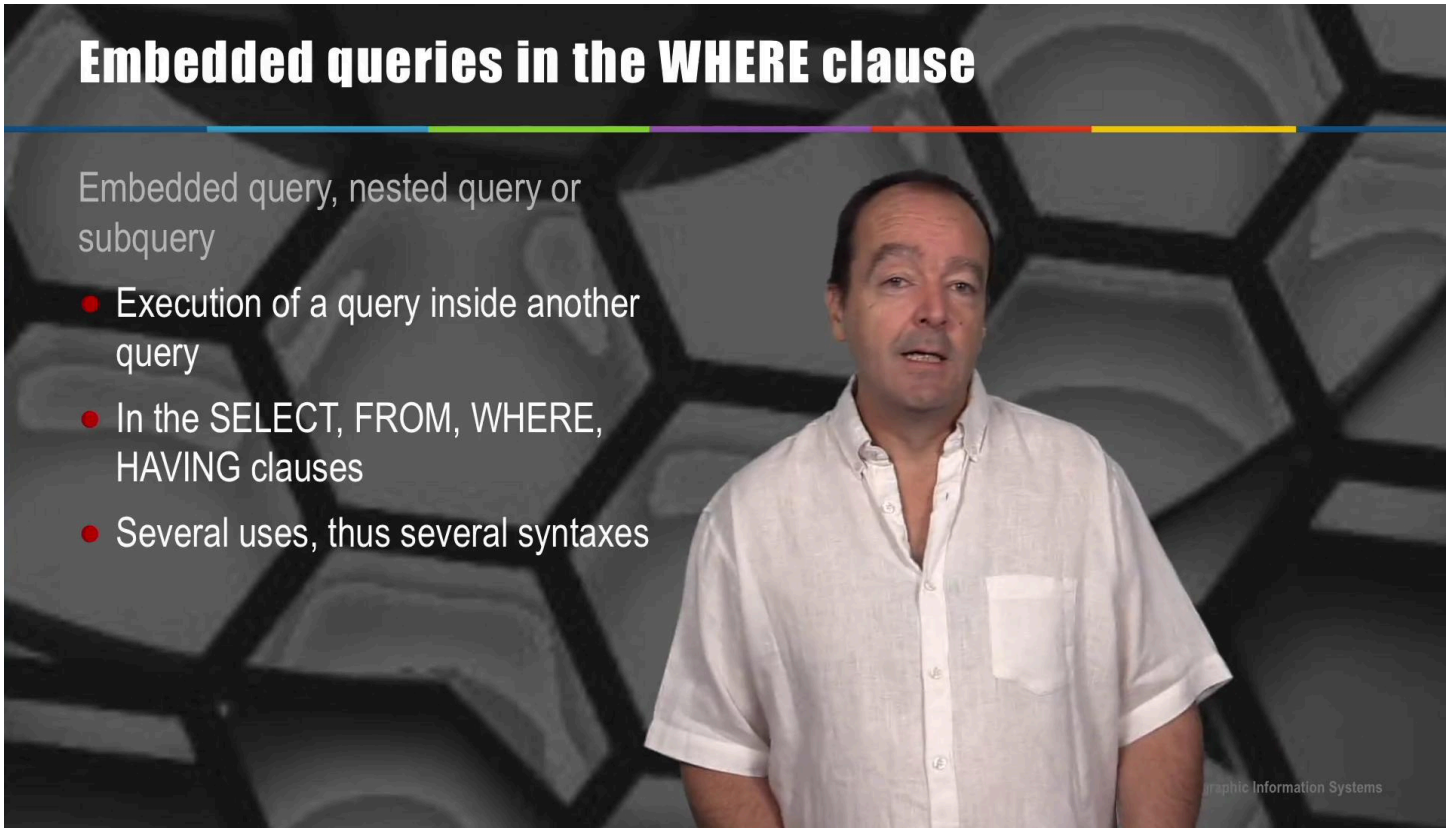


So we copy this clause and paste it at the end of the second request and we replace the attribute on which the filter focuses by the LOCALITY keyword in inverted commas since there is a dash that could cause a problem.

Notes

Summary

7m 41s

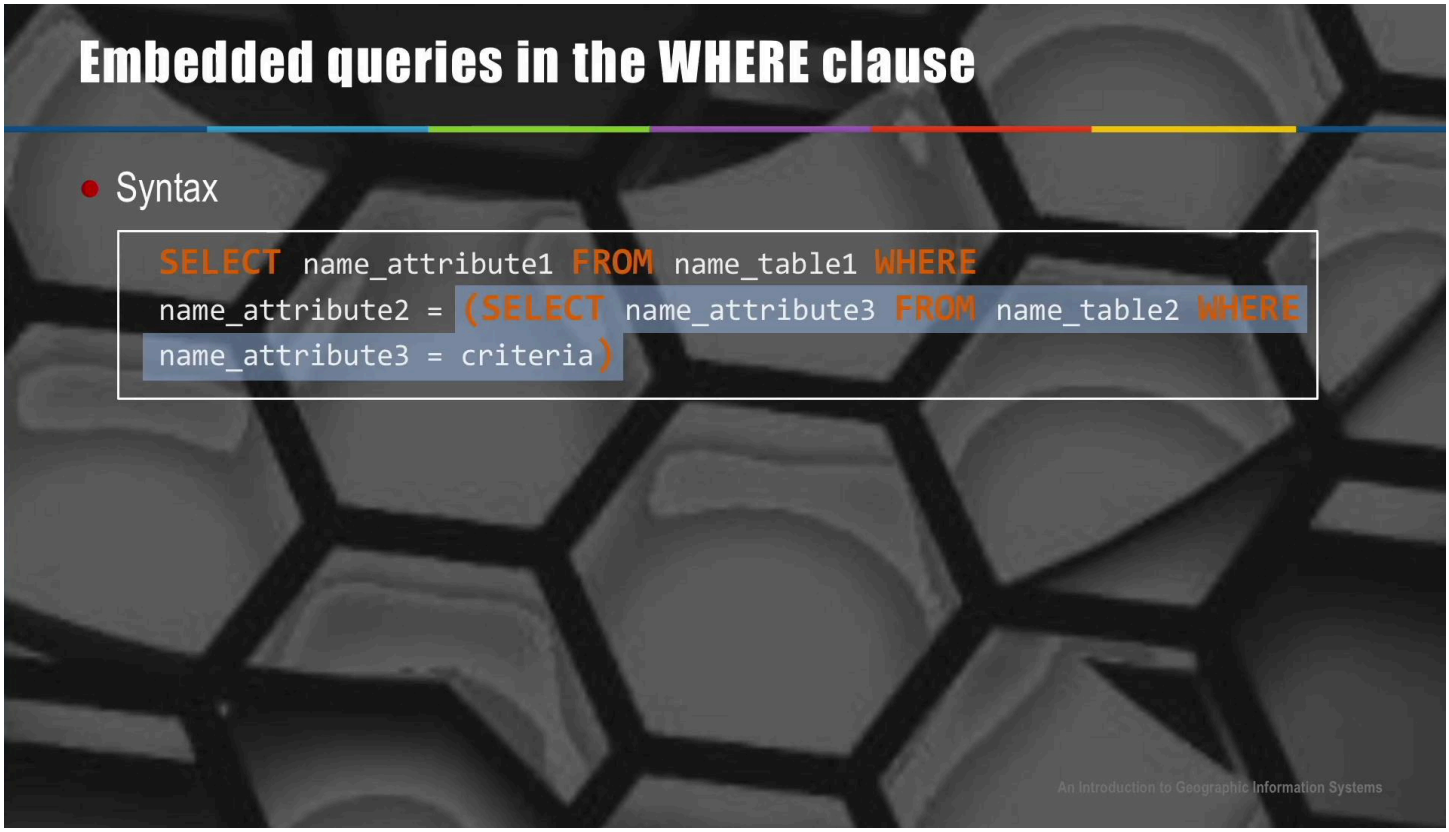


But we see that we have 4 results, so 4 districts and localities that start with T.

Notes

Summary

8m 04s



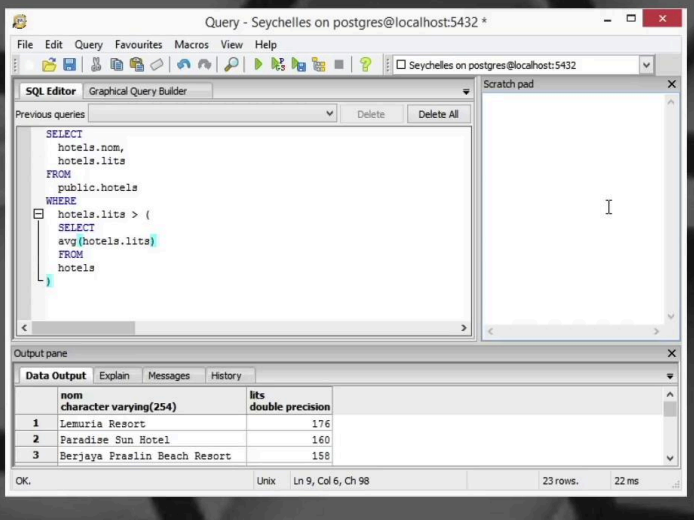
In the SQL world we talk about nested queries, imbricated queries or subqueries when a query is executed within another query. This kind of layout can be seen in the case of SELECT clauses, FROM clauses, WHERE clauses or HAVING clauses. As there are several ways of using the nested queries there are also several types of syntax that we will see a little more in detail starting with the WHERE clause. The syntax of a nested query in the WHERE clause consists in replacing in the condition that the WHERE clause must check, to replace the criterion with an SQL query placed brackets.

Notes

Summary

8m 16s

Embedded queries in the WHERE clause



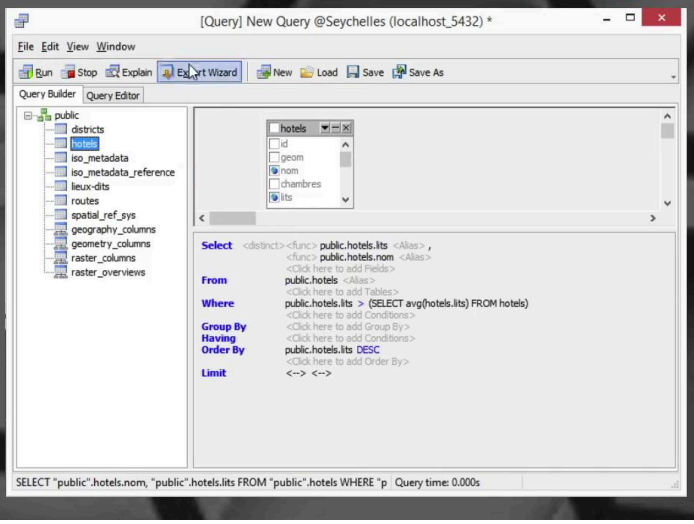
To illustrate the use of this type of nested query we take this example where we were looking for the maximum value of the number beds of the hotels in the Seychelles. We wanted to extract the hotel's name which had the largest number of beds, something that could be done quite easily if you remember it with SpatiaLite but that was not doable with Postgres. So we need a more elaborated request in this case so we select the name and the number of beds of the hotel, of the HOTELS table and as a condition in the WHERE clause we would like to select the hotels whose bed number corresponds to the maximum value of the number of beds. It is written here a little naively but in fact this maximum can be described by the request of selection that we had earlier so the maximum number of hotel beds from the table of hotels with as a condition... no condition. Here is the Berjaya. We can also transform the query to search for all the hotels whose number of beds is above average and there are 23 establishments. The average if we remember in a previous exercise we saw that it was 38 beds.

Notes

Summary

9m 06s

Embedded queries in the WHERE clause

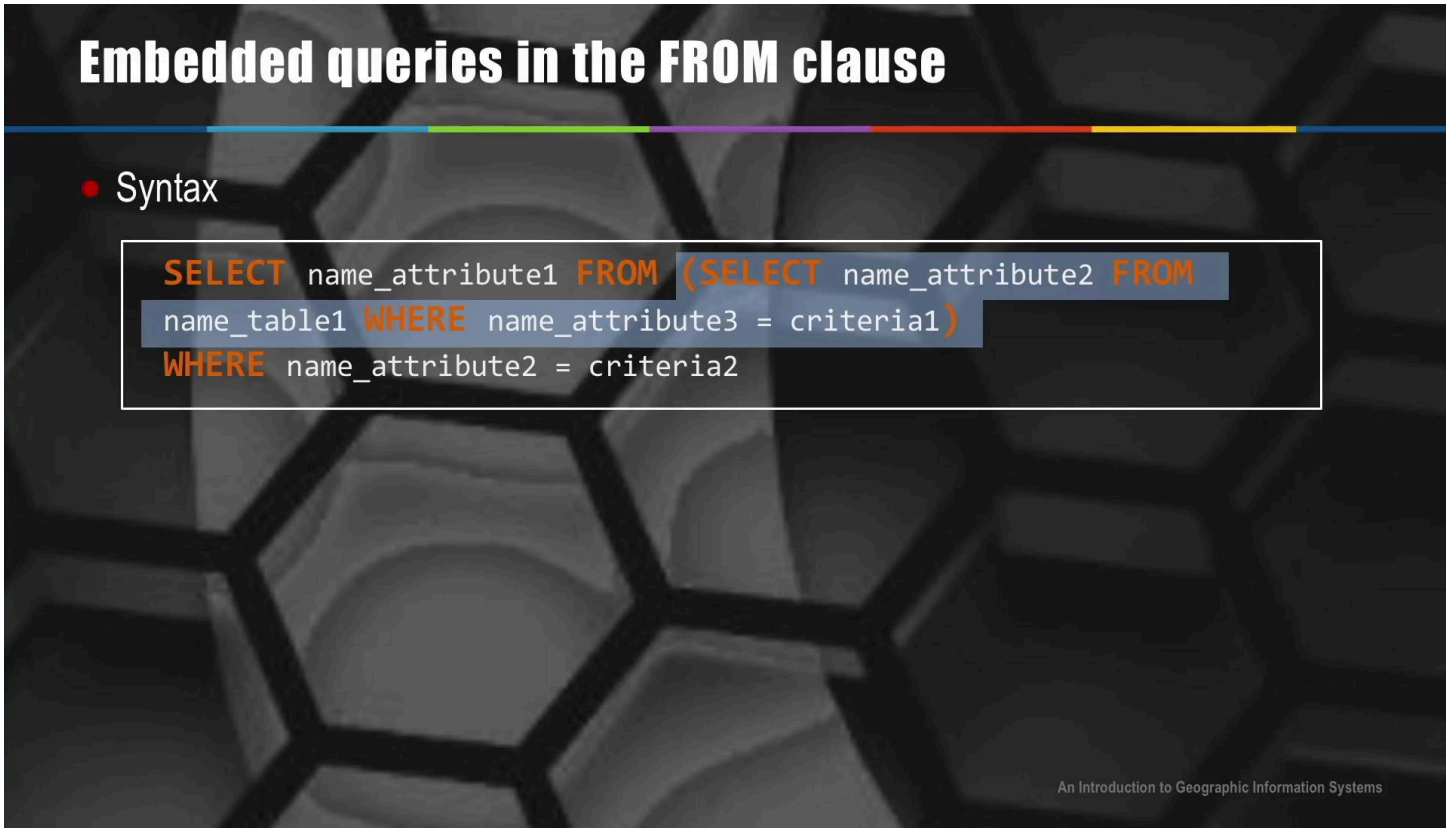


So we have the list of hotels that have more than 38 beds. In the case of Navicat the thing is done graphically so we select the fields in the table and then in the query constructor we can add the different parameters. We will search for the number of beds above average. The nested query must still be manually written. The graphic interface that makes life easier has its limits. So, as earlier, we associate, as search criteria, the average value number of beds in the hotel field and here we will sort the hotels in descending order, in descending size.

Notes

Summary

10m 48s



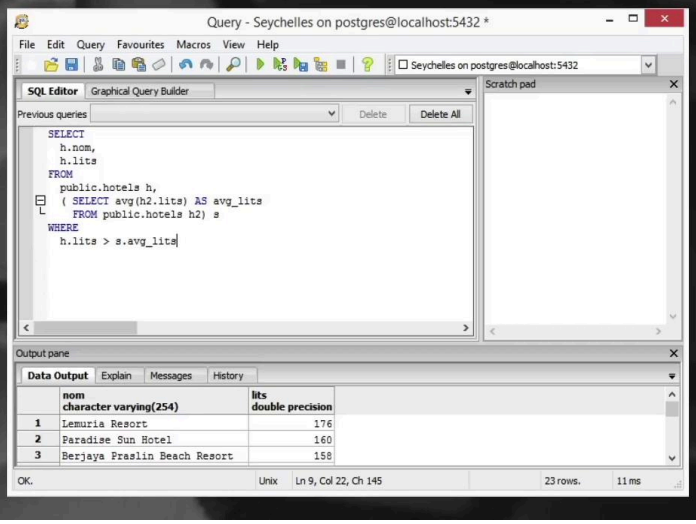
In the case of a request nested in the FROM clause, the principle is exactly the same with the selection request which replaces the table name on which the query is made. This type of syntax is possible because a selection query returns the equivalent of a table in the database.

Notes

Summary

11m 35s

Embedded queries in the FROM clause



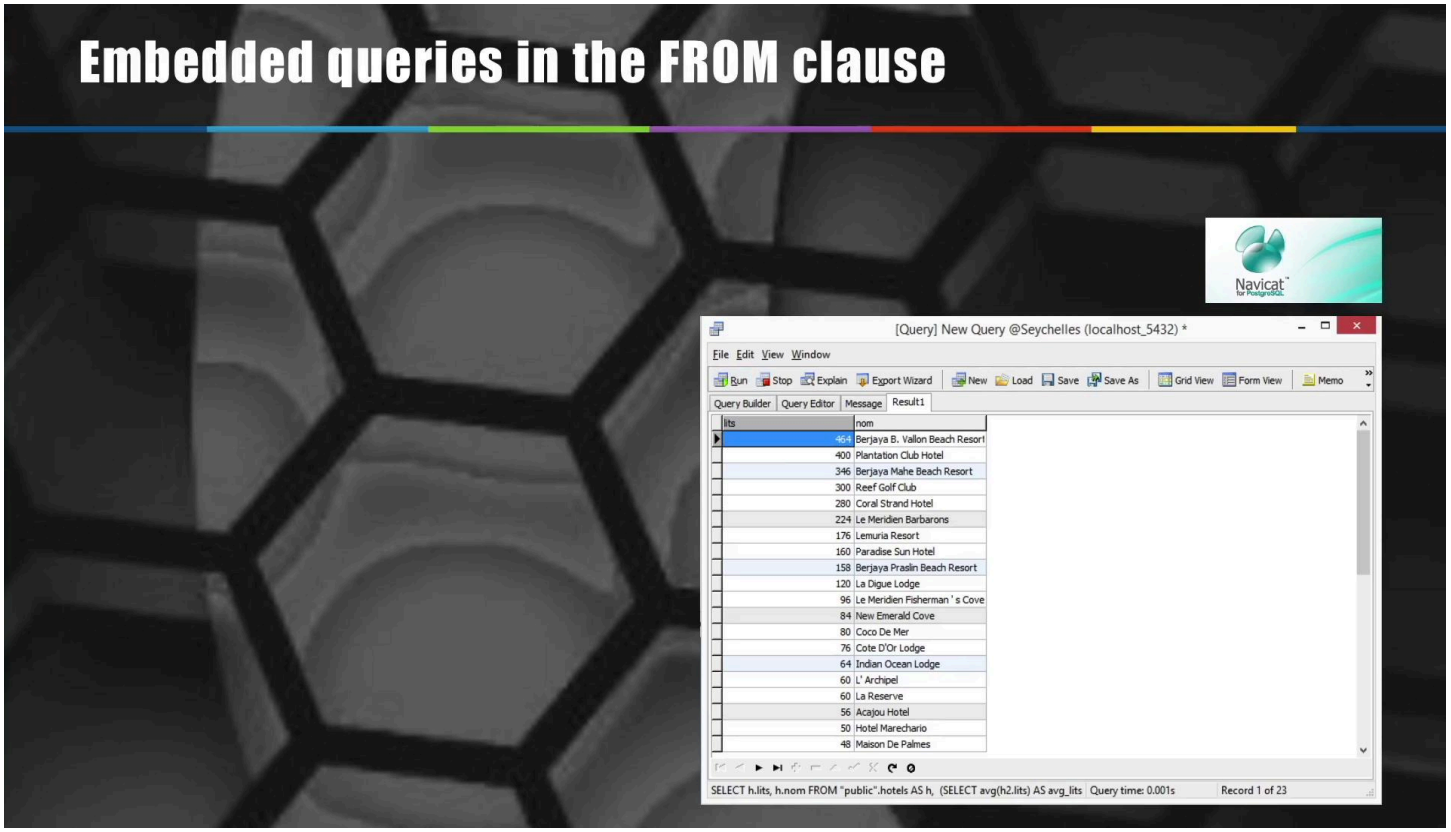
As we will use the same HOTELS table several times in the FROM clause, we will have to resort to aliases, which makes it easier to write the SQL query. So we begin by adding a second FROM clause which is in the form of a request and in this query we will look for the average number of beds of the hotels, of a HOTELS table to which will give the H2 alias to distinguish it from the first that the H alias. This average number of beds extracted from this second table will be used in the conditional clause with the necessity to use an alias for that table of the FROM clause, S for the synthesis table and an alias also for the column which contains the average values so that we can reference this alias in the conditional clause. We get the same 23 results as if we used the WHERE clause.

Notes

Summary

12m 12s

Embedded queries in the FROM clause

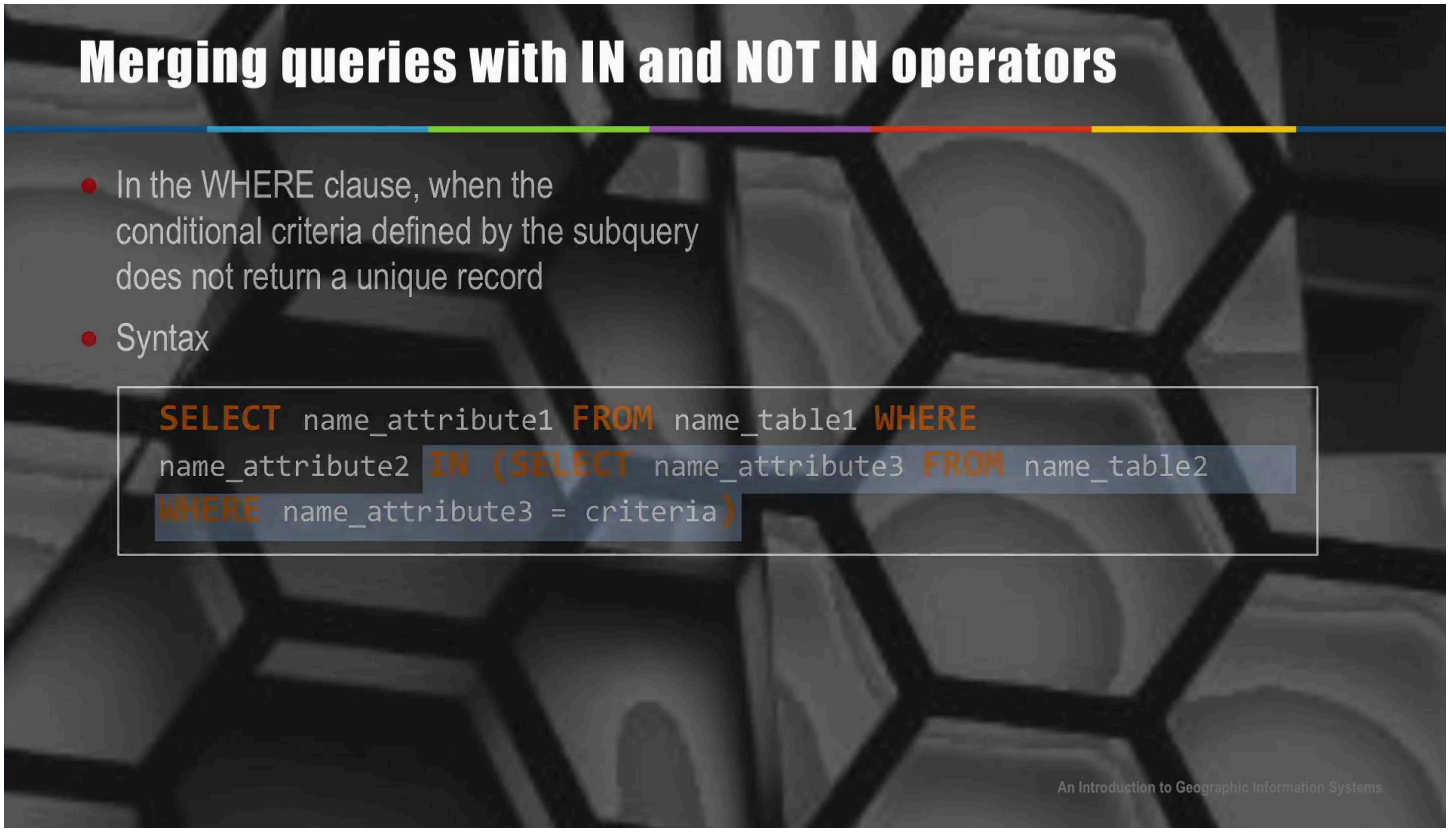


Same thing here in the case of Navicat where we replace the HOTELS table by an alias which automatically updates the fields and afterwards can be done again the same way in the editor. We copy here the average calculation query, we put aliases on the second table of hotels, one alias on this nested query, an alias on the average, which allows to complete the selection request, which allows to complete the conditional clause and we keep the idea of sorting in decreasing order.

Notes

Summary

13m 29s



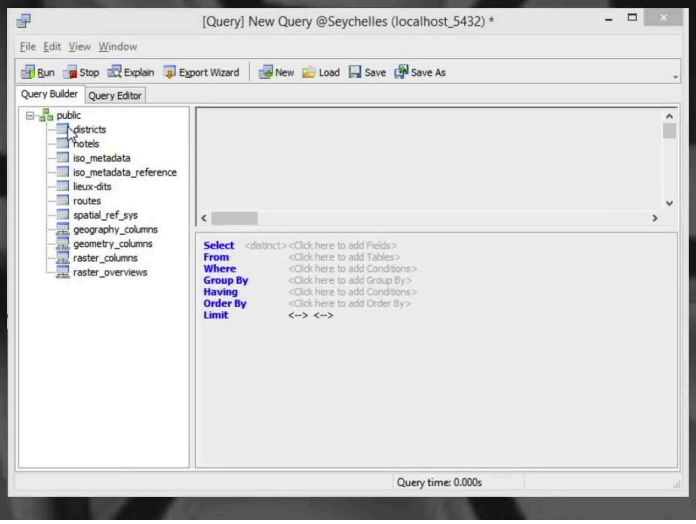
We saw in the case of the nested query in the WHERE clause that an SQL query is used to define the criterion which is used in the condition that the conditional clause must verify. This is a special case where the request that we use returns only one value. But it can happen, and often, that the query used returns multiple values and that the condition we want to check is that the selection criterion corresponds to one of the values of this collection. It is in this case that we will use the IN and NOT IN operators to test the results of the nested query. The syntax is presented as previously with simply the operator that changes becoming IN instead of "equal" or "greater than".

Notes

Summary

14m 36s

Merging queries with IN and NOT IN operators



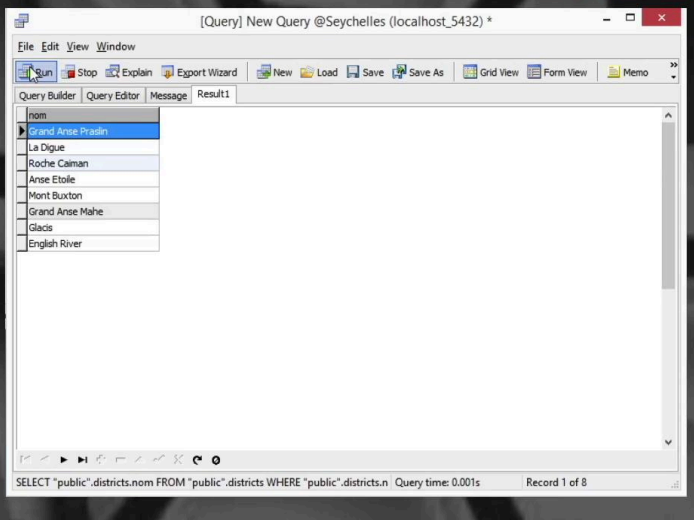
To illustrate this type of nested query use we will take the example of the intersection queries where we sought the districts that have the same name as localities or on the contrary the districts that have no equivalents in the collection of localities. So we add the table of districts and localities, we select the name of the district table and add a criterion to define the WHERE clause, a criterion that will focus on the district name and an IN operator to say that we would like the name to appear in the collection of locality names. In this collection of locality names we have to write it on foot with the table name always in inverted commas because of the dash, the FROM clause and the criterion being defined, we can still sort out by removing the table of localities from the main clause. And when we execute this query we see that we find the 17 districts which have an equivalent in the collection of localities. To find the districts that have no equivalents we simply reverse the sense of the NOT IN operator and we find the 8 districts that do not have a correspondent in the collection of localities.

Notes

Summary

15m 35s

Merging queries with IN and NOT IN operators



In the case of Navicat we add the district we select the name and in the conditional clause we will simply add the "name of districts" field replace the operator so here it is called IS IN LIST but which is translated in SQL by simply the IN and as before, it is necessary to write the nested query in an explicit manner without forgetting the the quotation marks for the LOCALITY table and by disabling the automatic addition of apostrophes in the selection criterion. When this query is executed we find the 17 districts we were looking for and for the complement the operator is reversed and we find the 8 districts which have no equivalents in the collection of localities.

Notes

Summary

17m 00s