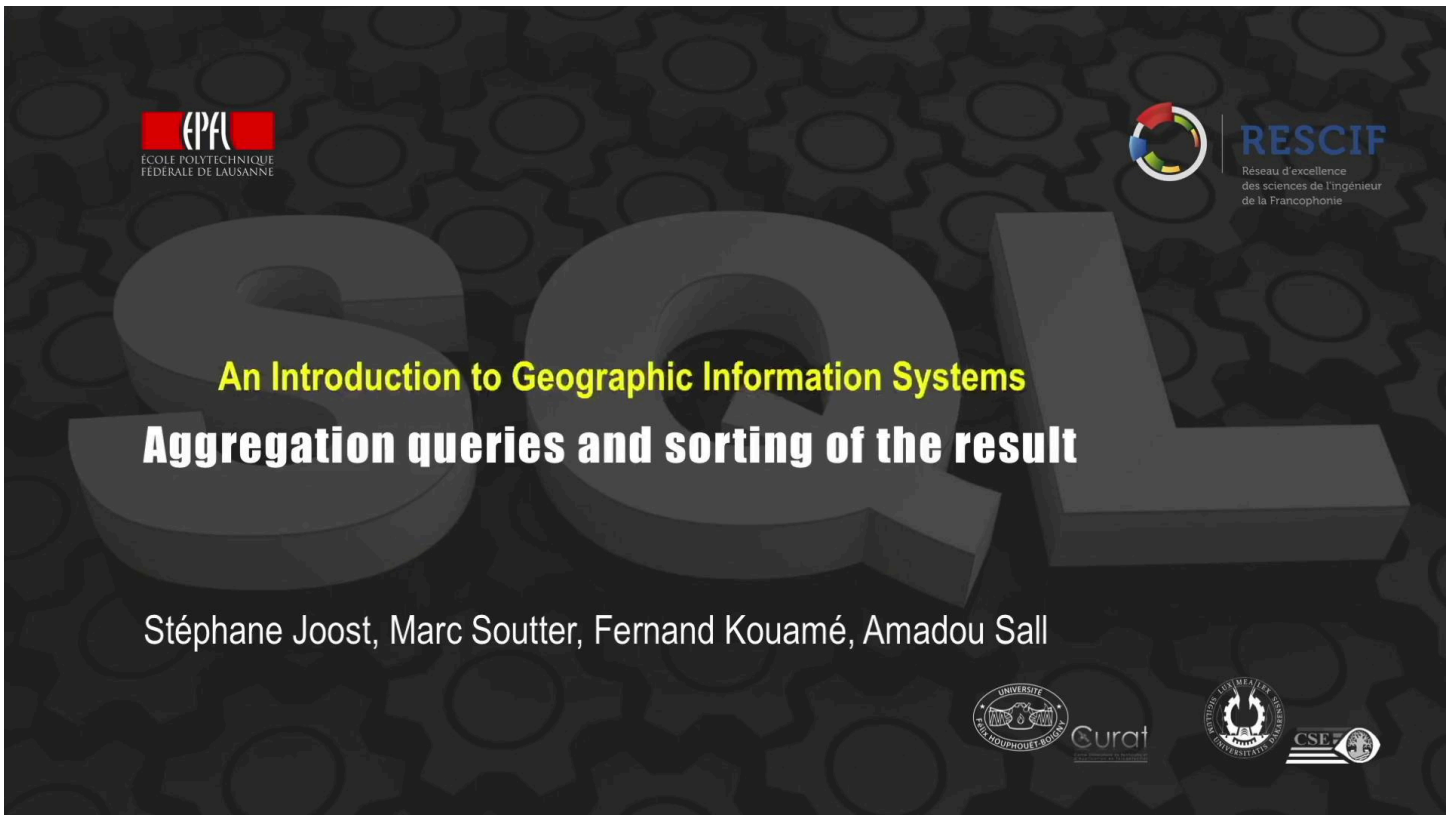
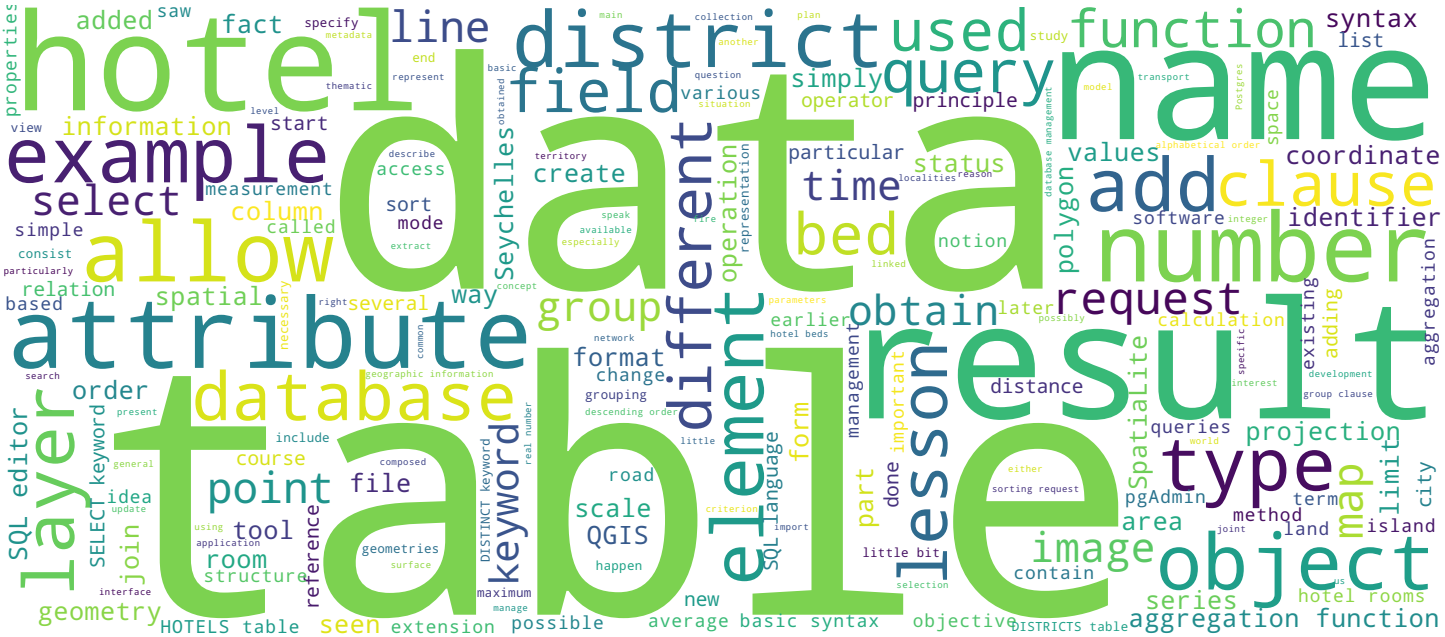
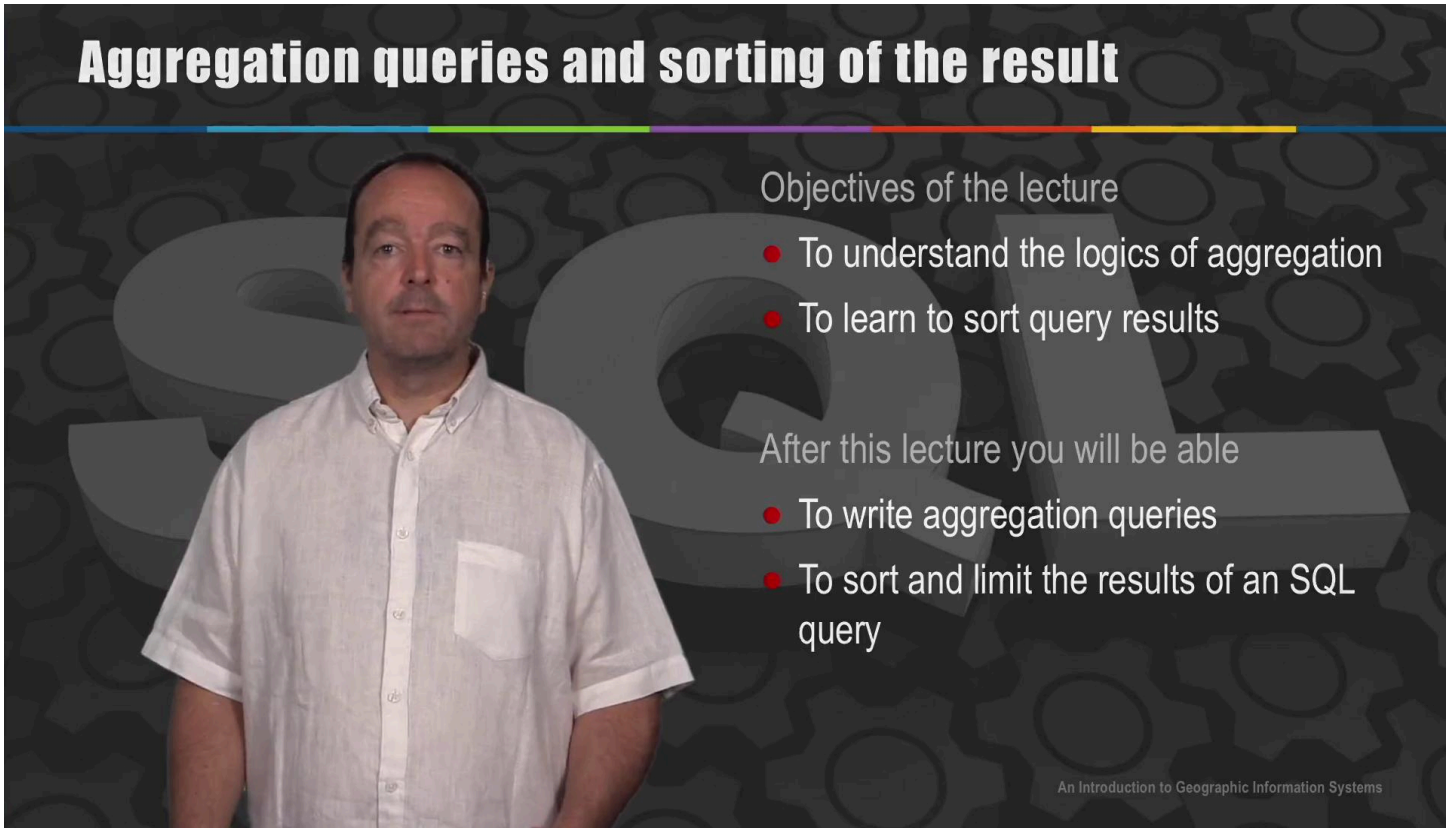




Search MOOC

Video

EPFL

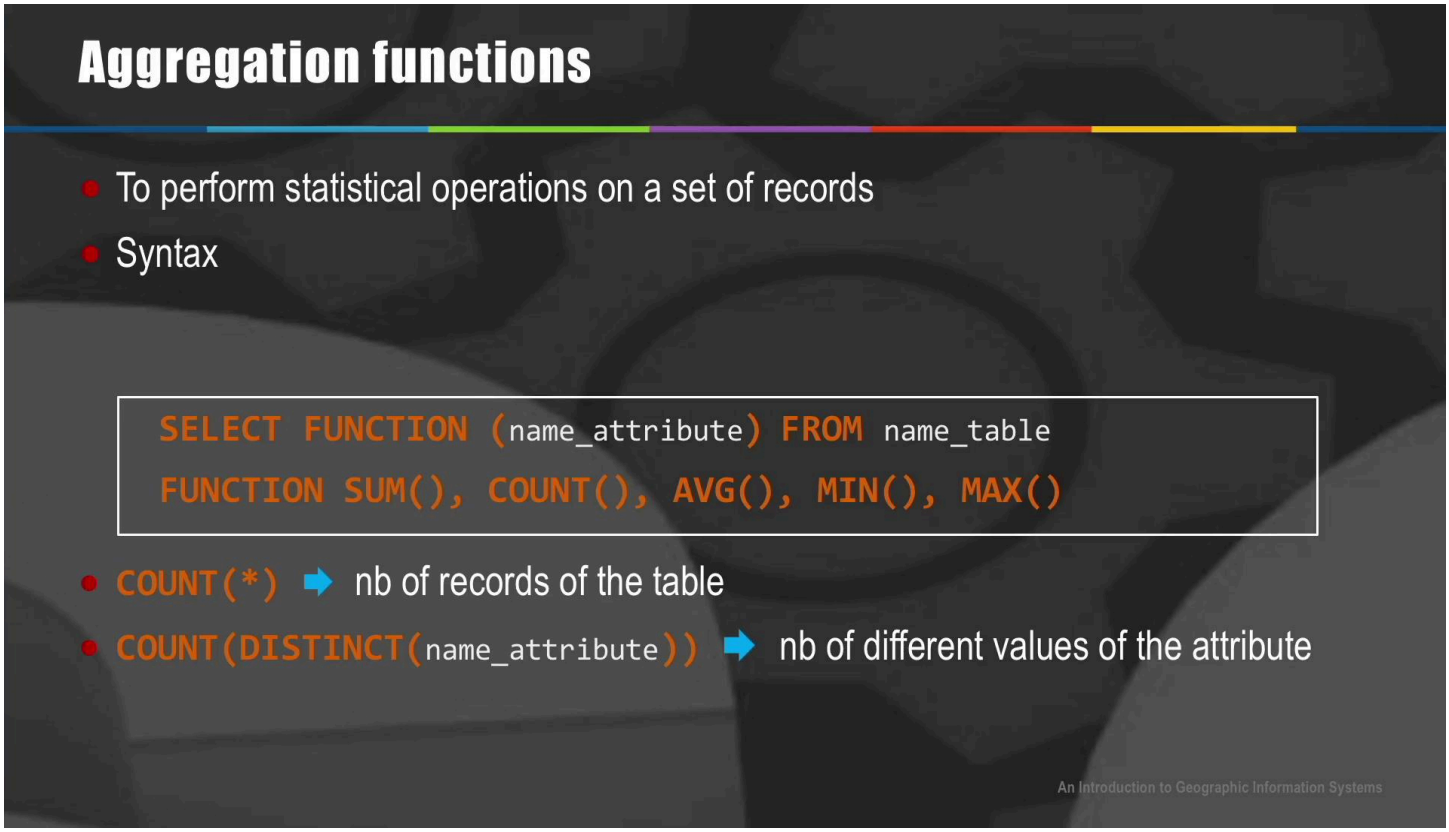


So we continue in this lesson our discovery of the richness and subtleties of the SQL language by addressing the issue of aggregation and sorting namely how to extract synthetic information from a set or a subset of data, to group them, filter them and sort them. The objective of the lesson is therefore to understand the aggregation logic and how to sort the results of a query so that you can thereafter write aggregation and sorting requests.

Notes

Summary

0m 23s



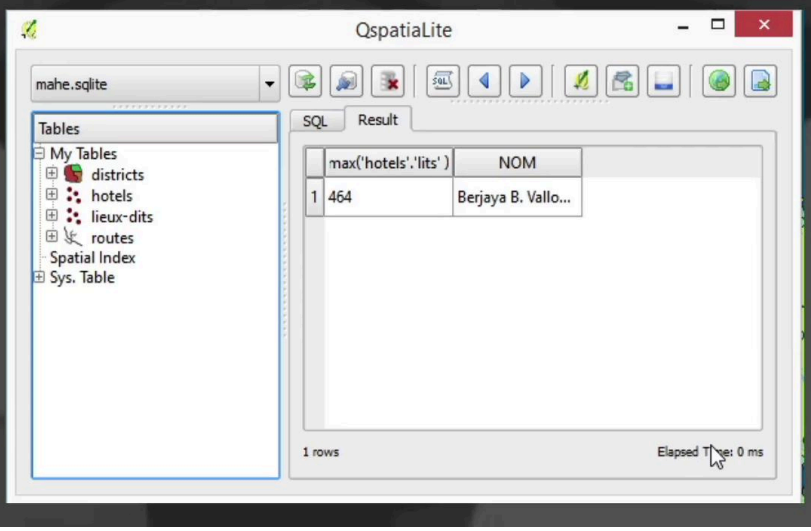
We will discuss successively the functions of aggregation, the concept of grouping, conditional aggregation and finally the elements of sorting and limitation of the results. Coming back to our summary table of the basic syntax of the SQL language, we find the clauses of aggregation that interest us here with the GROUP BY and HAVING keywords. But we will start with the aggregation functions which allow to carry out statistical operations on a set or subset of records with a type of syntax... the SELECT keyword, the keyword of the function with the attribute name on what the function is and then the FROM clause and the name of the table. And these functions are of various types, including the sum, the counting of the number of elements, the average, the minimum and the maximum. The COUNT function, with an attribute or asterisk as argument, gives the number of elements of the table whereas by adding the DISTINCT function, we obtain the number of values that the attribute used in argument can take.

Notes

Summary

0m 55s

Aggregation functions



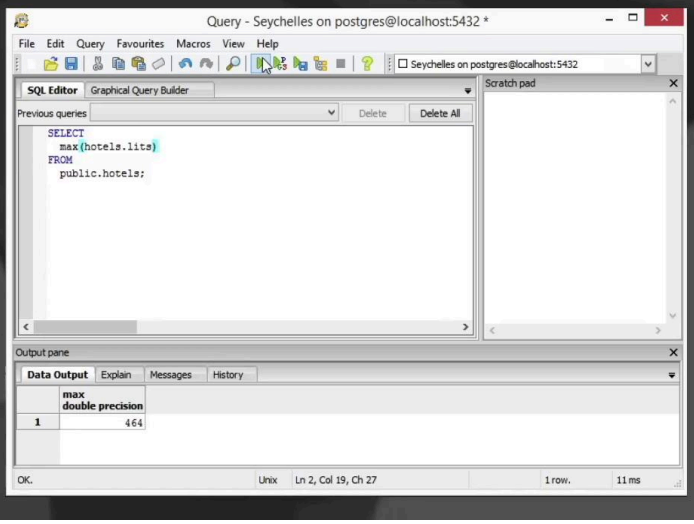
An example of the application of these aggregation functions in Spatialite with the number of rooms of the hotel table and we would like to have a table that gives us all the rooms present on the island so 2,518 rooms. We would also like to be able to count the STATUS column of this hotel, we see that there is 124 status corresponding to the 124 existing hotels and with the DISTINCT keyword, we will be able to regroup the different status which are 7 in total. Another example of using these aggregation functions, with the calculation of the average number of hotel beds obtained by dividing the total number of beds by the number of hotels. Note that we would obtain the same result using directly the average aggregation function or AVG. We thus obtain an average number of 38. If we want to have this value in real number in SpatiaLite, it is necessary to transform the sum which is an integer into a real one which is done with the CAST function on the "element" variable and followed by the "as float" suffix. The maximum, the maximum number of hotel beds is obtained with this max function, so 464. And then what is of interest to us is to see which of these hotels has the maximum of beds so we can add in SpatiaLite the NAME field as the result of the query and we see that it is the Berjaya B.

Notes

Summary

2m 17s

Aggregation functions



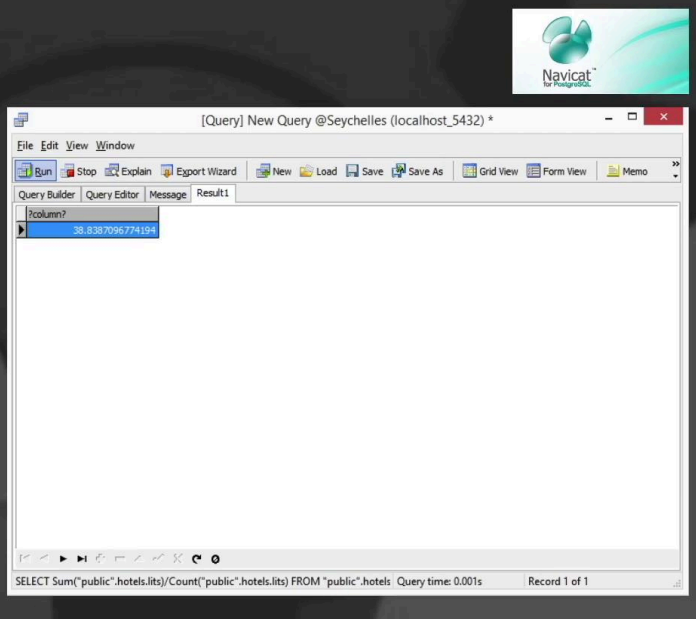
Vallon. It should be noted that this addition of the name is not an SQL standard and will cause an error with most DBMS. We will see later the correct SQL syntax to obtain this type of result. Same example, same situation for pgAdmin where we use the graphic constructor but we do not go very far because the aggregation functions must be written in the SQL editor in the same way as in SpatiaLite. So we saw the case of the sum of the rooms. Now, the counting of status against the 124 status corresponding to the 124 hotels again and with the DISTINCT keyword, we obtain the 7 types of status again characterizing the hotels of the islands. In the same way as previously, we can use these aggregation functions to compute a calculation so here the total of the hotel beds in relation to the number of hotels to obtain the average of the island hotels. And here we see that with Postgres we obtain directly a result in real number and not in integer number. Here we can also use the max function to obtain the number of beds of the biggest hotel of the Seychelles.

Notes

Summary

4m 09s

Aggregation functions

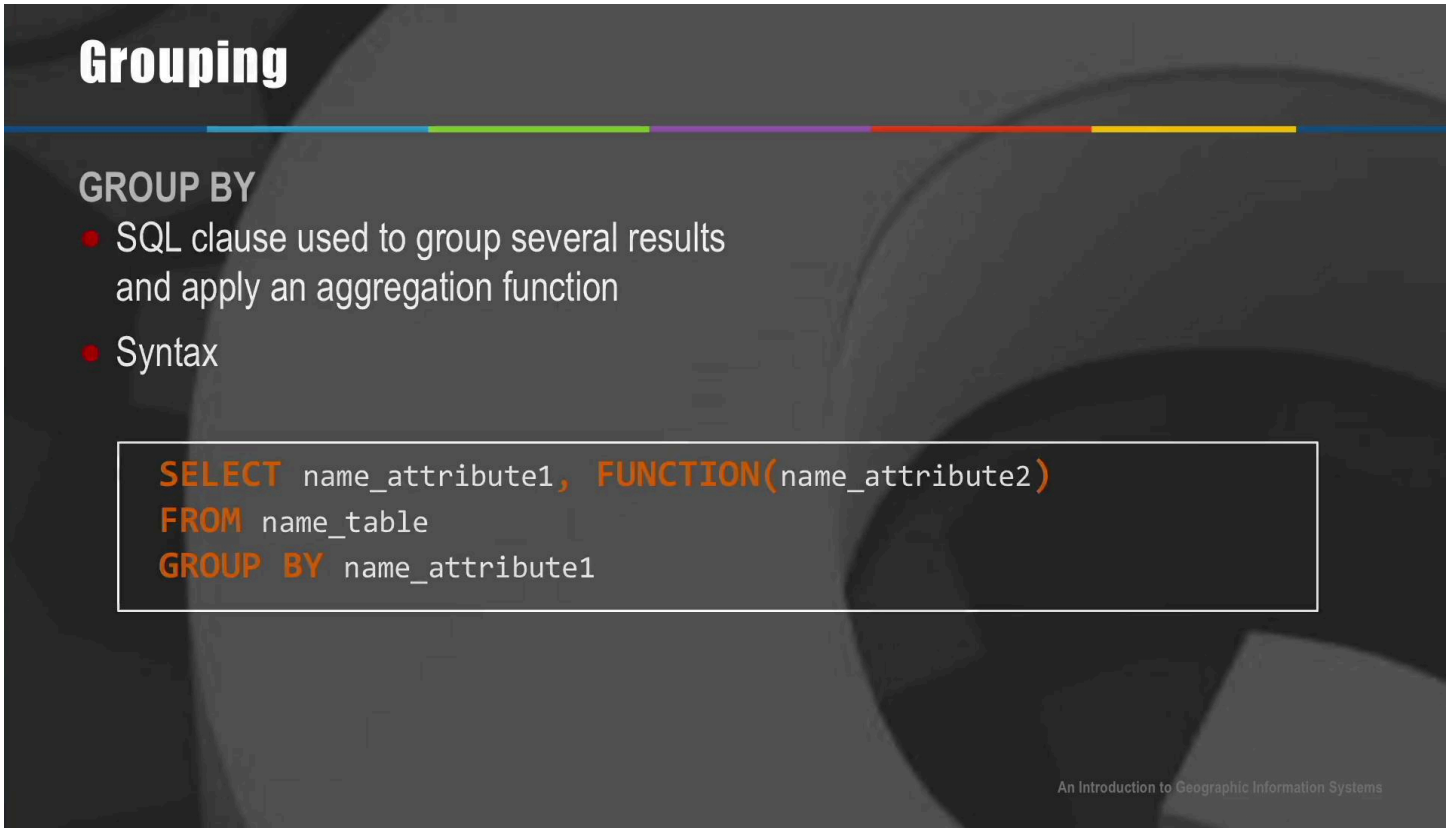


And we will see later how when we will speak of nested query, how to deal with this kind of questioning. The same example with the Navicat software, we add graphically again the elements and then the request can be composed in the graphic interface, so we can add the sum, add the counting section, change the attribute. We find the 124 status of the 124 hotels. On the other hand, to add the DISTINCT keyword, we must go also in this case to the SQL editor to write this keyword. Returning to the editor, we can this time replace the status with beds and then, and actually it is simpler to write this somewhat complicated query and which does the calculation of the average in the editor rather than in the graphic interface. And here again, we find the result.

Notes

Summary

5m 41s



So it is because it is Postgres that is behind directly in real number. The principle of the GROUP BY clause is to allow the grouping of several results and to apply an aggregation function to this group. The basic syntax includes the SELECT keyword with the attributes we want to see in the result with possibly a function which is exercised on one or the other of these attributes. The FROM keyword with the original table which is questioned and then the GROUP BY keyword followed by the attribute which will be used for grouping.

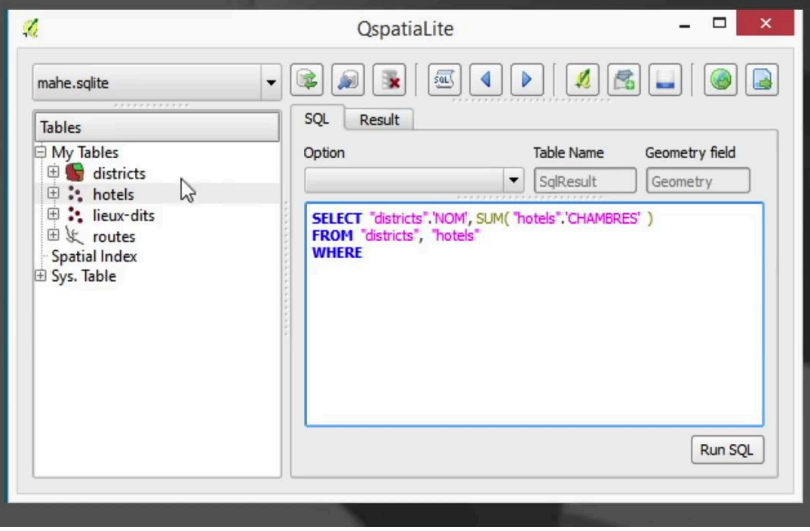
Notes

Summary

6m 54s

Grouping

GROUP BY



We take an example again with the SQL SpatiaLite query constructor integrated into QGIS. So, we write a query where we seek to obtain the set of hotel rooms available by district. So, this request includes as resulting field the names of districts and the sum made on the number of hotel rooms from the DISTRICTS table and HOTELS table with a joint as we have just seen between these 2 tables between the district identifier of the HOTELS table and the identifier, this same identifier in the DISTRICTS table itself.

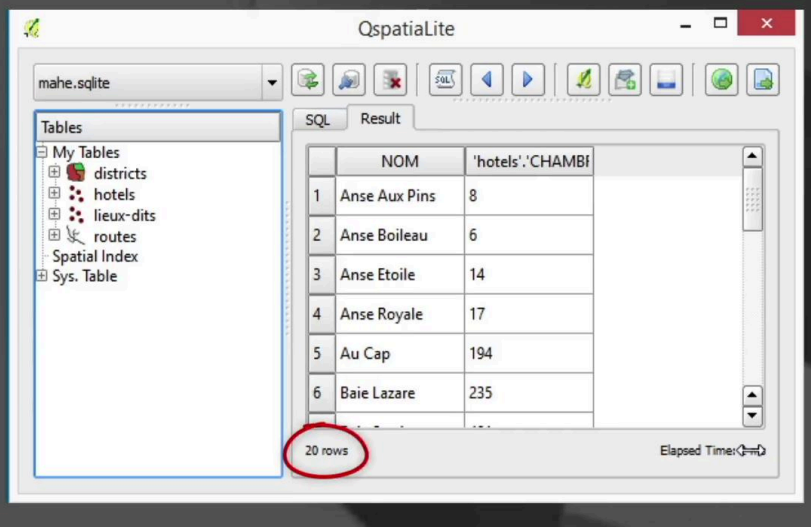
Notes

Summary

7m 42s

Grouping

GROUP BY



A regrouping clause must be added to this since the numbers... the number of hotel rooms is linked to each district name and that, the different hotels and districts, must be grouped together.

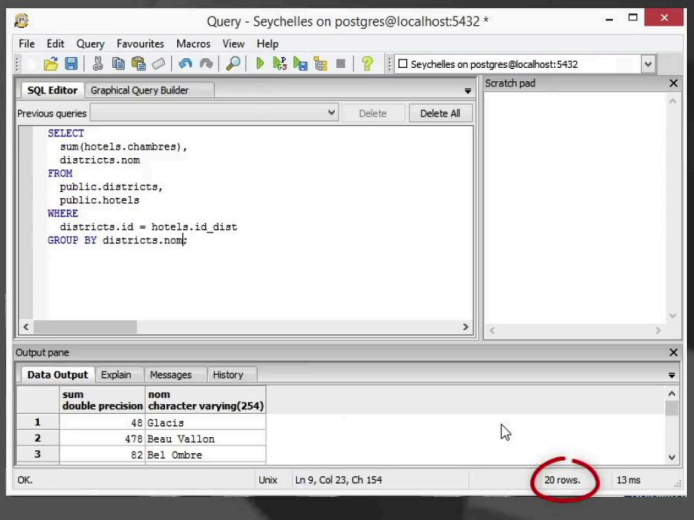
Notes

Summary

8m 26s

Grouping

GROUP BY



And we find about twenty results since there are 5 districts that do not have hotels. The same thing happens in pgAdmin, where this time we construct the query graphically with DISTRICTS tables and HOTELS tables. The joint, the 2 fields we would like to have in the result and we... we have to complete the request in the textual editor by adding the sum on the hotel rooms and adding the group clause, the GROUP BY, which is about the names of the districts since we will group the rooms by district. And we see that we also get again the same 20 results as before.

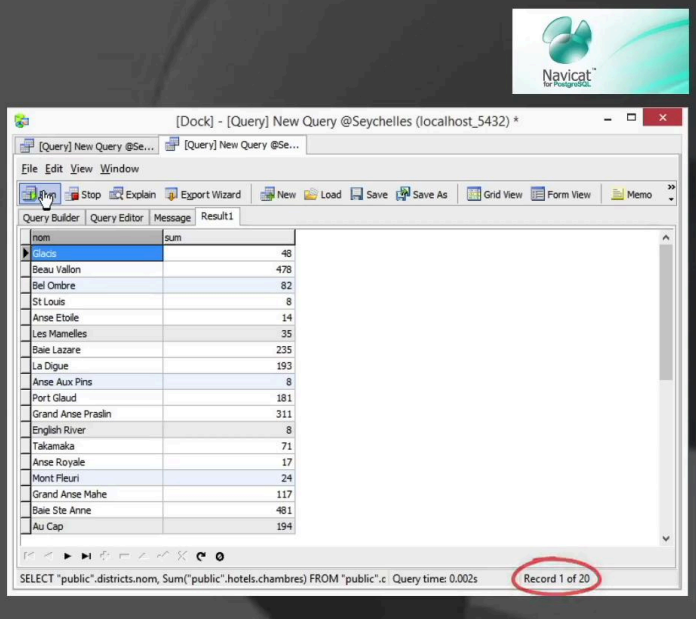
Notes

Summary

8m 42s

Grouping

GROUP BY

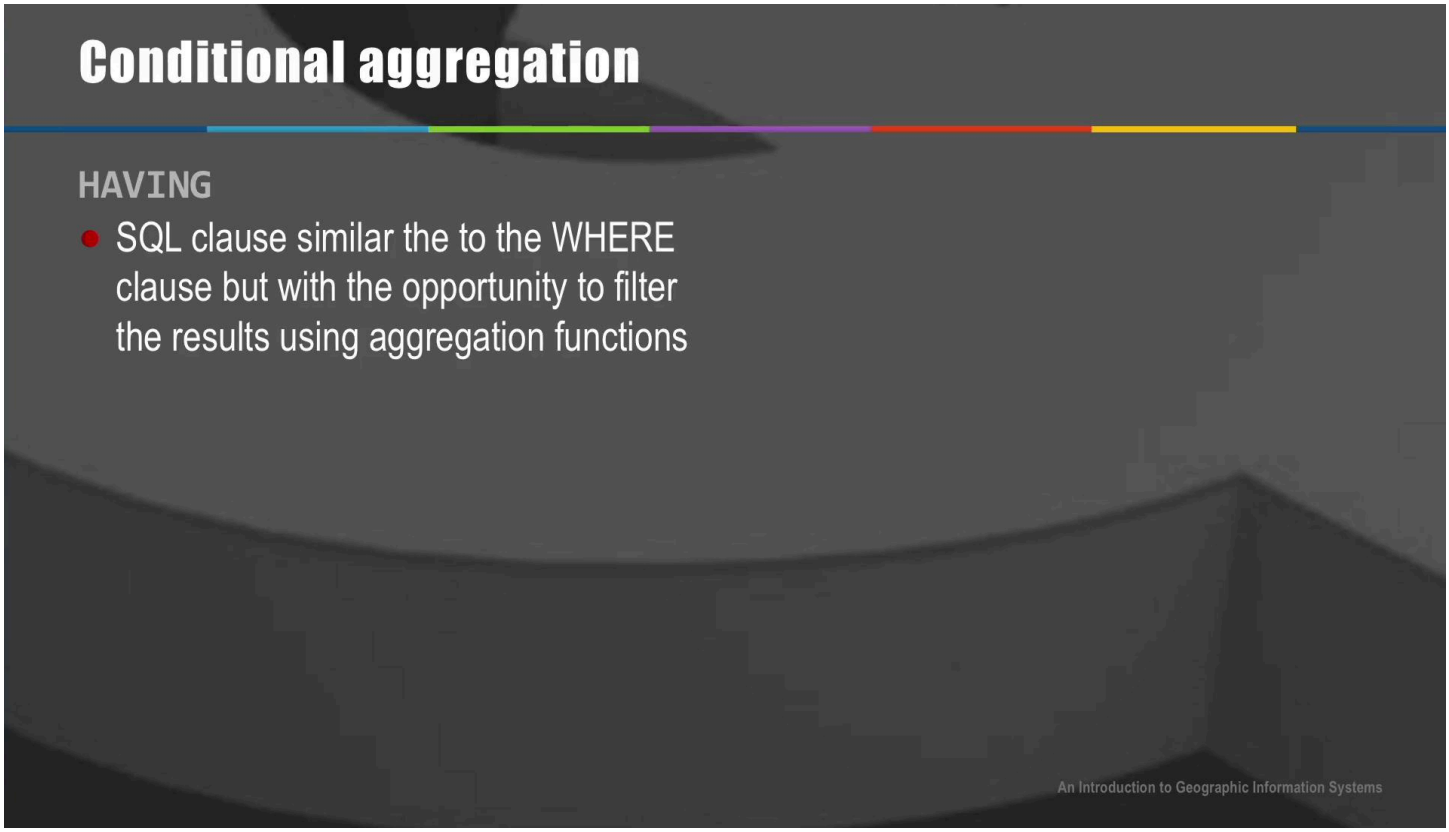


In Navicat, things happen a little bit in the same way. We add the tables in the graphical constructor we make the join, we select the fields we want to see appearing in the result and then we can this time in the integrated SQL editor, directly select the aggregation function and then we see that automatically, the grouping has been added, the same 20 results are obtained.

Notes

Summary

9m 32s



The HAVING clause is similar to the WHERE clause but allows to filter the results with the help of aggregation functions, it is the reason why this time we refer to conditional aggregation.

Notes

Summary

10m 08s

Conditional aggregation

HAVING

- SQL clause similar the to the WHERE clause but with the opportunity to filter the results using aggregation functions
- Syntax

```
SELECT name_attribute1, FUNCTION(name_attribute2)
FROM name_table
GROUP BY name_attribute1
HAVING FUNCTION(name_attribute2) operator criteria
```

An Introduction to Geographic Information Systems

In terms of syntax, the basic syntax always includes the SELECT keyword, the list of attributes that we want to group, that we want to have in the resulting table with possibly a function on one of these attributes. The FROM with the original table, the group clause with the attribute or attributes which intervene in this group clause and finally a filter function based on the HAVING keyword which combines an attribute function, an operator and a criterion.

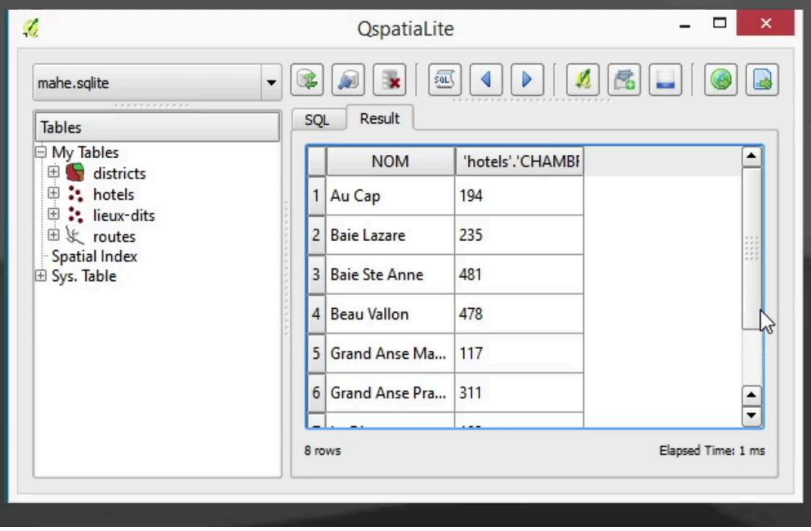
Notes

Summary

10m 20s

Conditional aggregation

HAVING



So, we will see right away how it happens with the same example as before where we have this request which allows to collect the number of rooms per district but we will simply filter this result to get all the districts in which this number of rooms is greater than 100.

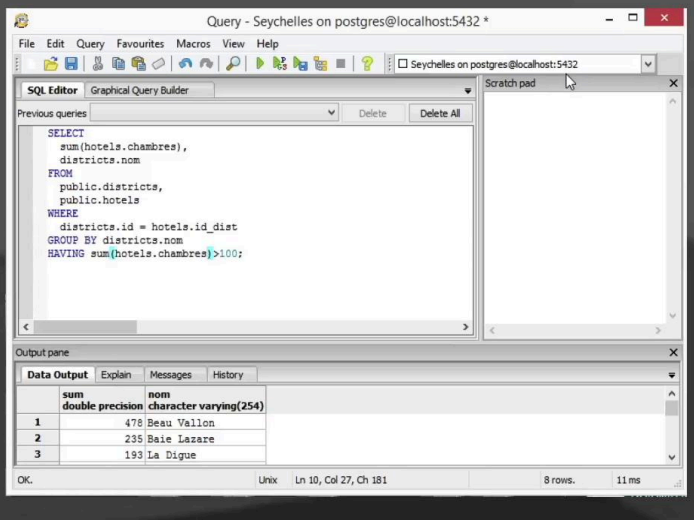
Notes

Summary

10m 55s

Conditional aggregation

HAVING



And we see that there are 8 districts that have more rooms, that have more than 100 rooms. The same applies to the request we had earlier in the case of PostgreSQL with the same way, we are obliged to use the SQL editor to add this filter clause. The graphic constructor being in this case of no use. And again, we find the same 8 results.

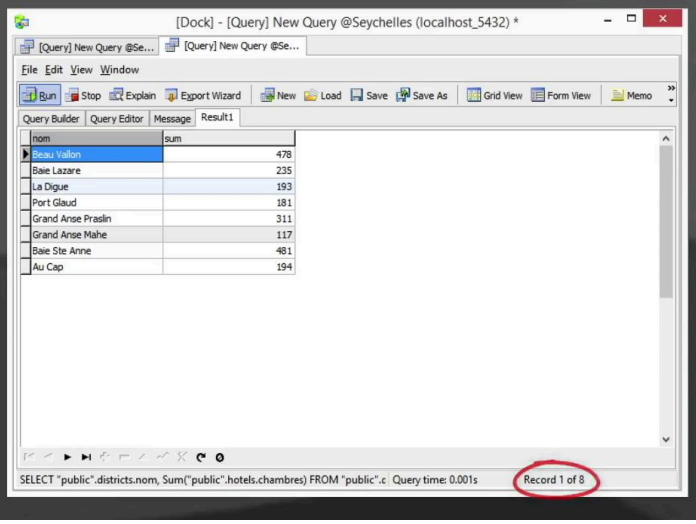
Notes

Summary

11m 23s

Conditional aggregation

HAVING

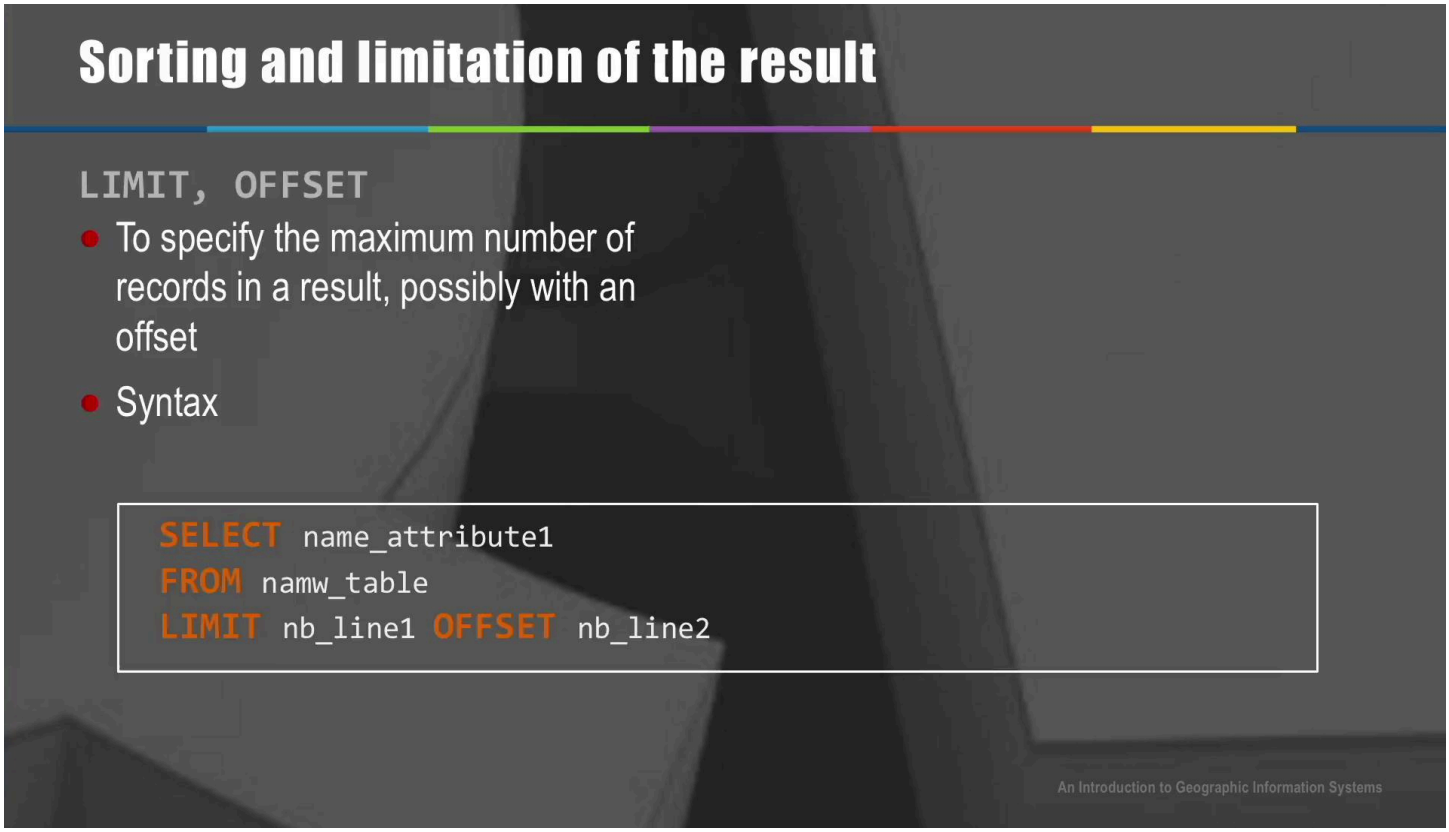


Finally, in the case of Navicat, we also start from the previous request and we see that we have in the integrated editor the possibility of graphically constructing this query, which makes life a little bit easier compared to a standard SQL editor.

Notes

Summary

11m 52s



And in the result, we find again these 8 hotels, these 8 districts, sorry. The sorting and limitation clauses are also part of the series of basic syntactic elements of the SQL language with the ORDER BY and LIMIT keywords. ORDER BY is an SQL clause which allows to sort the lines in a result on one or more columns in ascending or descending order. The basic syntax is the following: the SELECT keyword, the attributes we want to see appearing in the result, the FROM clause with the name of the table which hosts these attributes and then the ORDER BY sort clause with successively the different attributes that will be used as a sorting key with the attribute number 1 in descending order, the attribute number 2 in ascending order and so on. By default, if nothing is specified, the sortig order will be ascending but it is often preferable to specify the sorting order to eliminate any ambiguity and especially to facilitate the reading of the request. The LIMIT keyword allows to specify the number of results we wish to obtain and the OFFSET keyword is used to define an offset in the RESULTS table to search for that maximum number of results. The syntax is then SELECT name of the attribute, FROM name of the table LIMIT the number of lines and optionally OFFSET the number of lines.

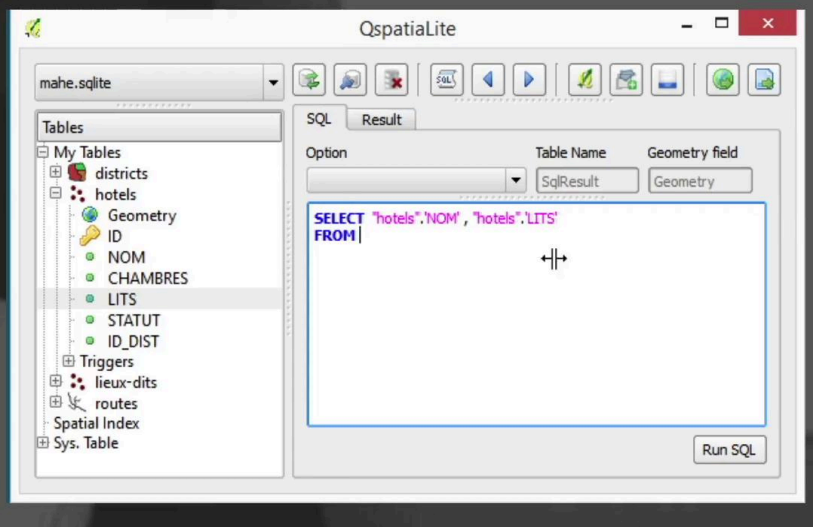
Notes

Summary

12m 16s

Sorting and limitation of the result

ORDER BY
LIMIT, OFFSET



By applying these ideas in the case of the SpatiaLite database, we write a SELECT query with several attributes, hotel names, number of beds. And that is it actually.

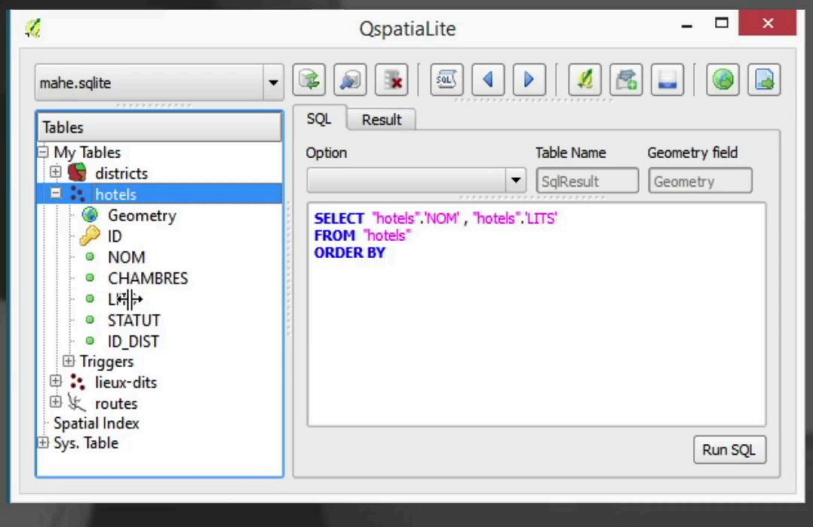
Notes

Summary

14m 04s

Sorting and limitation of the result

ORDER BY
LIMIT, OFFSET



The FROM clause with the name of the table and then an ORDER BY sorting request and we will sort by decreasing number of beds.

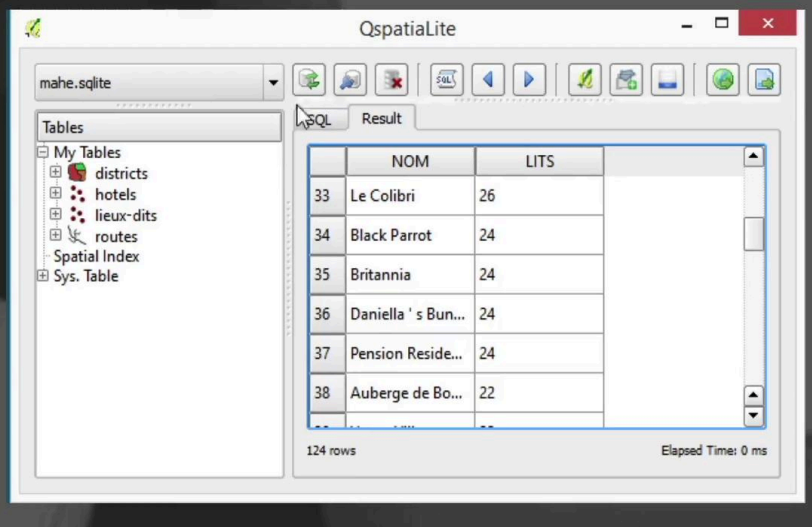
Notes

Summary

14m 25s

Sorting and limitation of the result

ORDER BY
LIMIT, OFFSET



So the BEDS fields is added with the DECREASING keyword then a second sorting criterion which will simply be the name of the hotels in ascending alphabetical order. So, we find the biggest hotel in the lead and then… within each class of beds a alphabetical sorting of hotels.

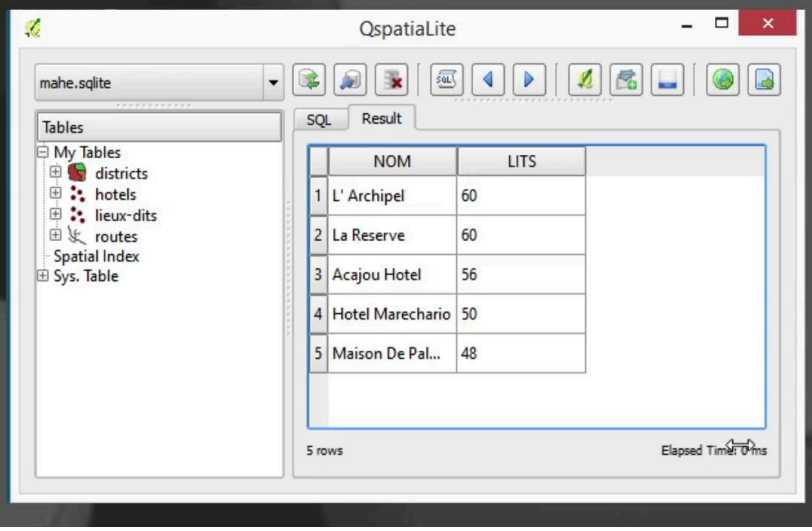
Notes

Summary

14m 38s

Sorting and limitation of the result

ORDER BY
LIMIT, OFFSET



We can now limit this result and say that we are only interested in the first 5 values. So we have the biggest five and in each category in alphabetical order and possibly shifting this selection 5, 15 lines and we land at the beginning with a series of hotels in the 60 lines.

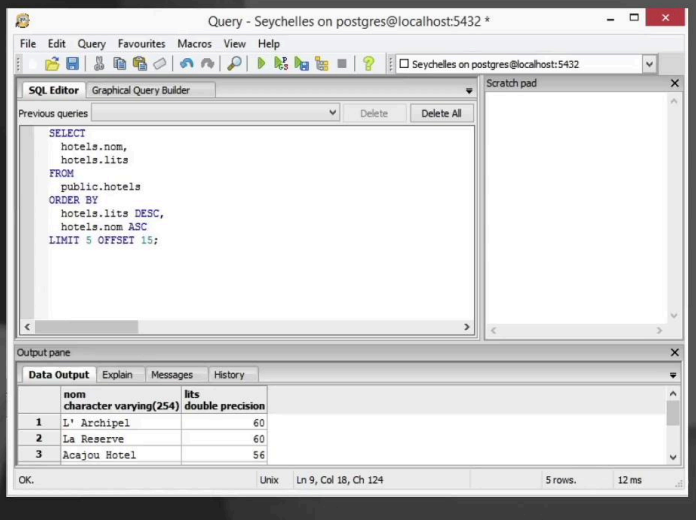
Notes

Summary

15m 07s

Sorting and limitation of the result

ORDER BY
LIMIT, OFFSET



Same operation in the pgAdmin environment for the Postgres database where we select graphically the fields and we can also write in the ad hoc tab, also graphically, the sorting request. We get the same result with the hotels sorted by number of beds and then by names and we can then add this time in the SQL editor the limit of 5 so we see very well the same 5 hotels as earlier and the possible OFFSET here of 15 lines as before.

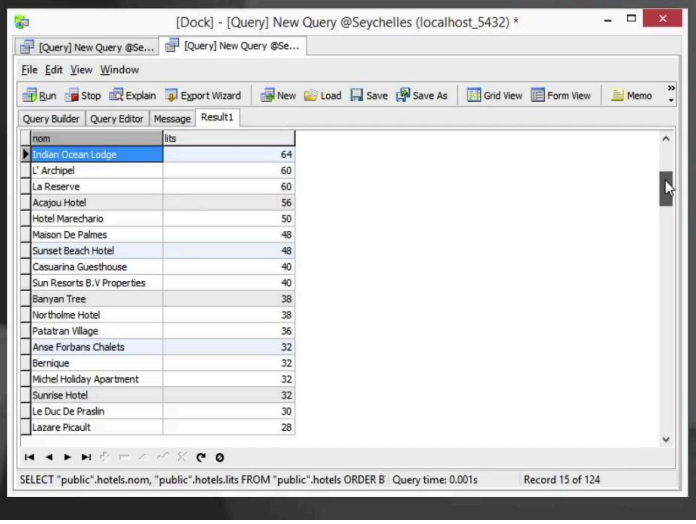
Notes

Summary

15m 32s

Sorting and limitation of the result

ORDER BY
LIMIT, OFFSET



Last example, with Navicat, where we compose again graphically the query with the name, of the hotel and the beds, and then in the SQL graphic editor we can add the sorting fields, define the sorting type here the beds in descending order, so beds in descending number and the hotels in alphabetical order. The result corresponds well to what was expected and as for the LIMIT and OFFSET, it happens in the same way as in pgAdmin III.

Notes

Summary

16m 22s