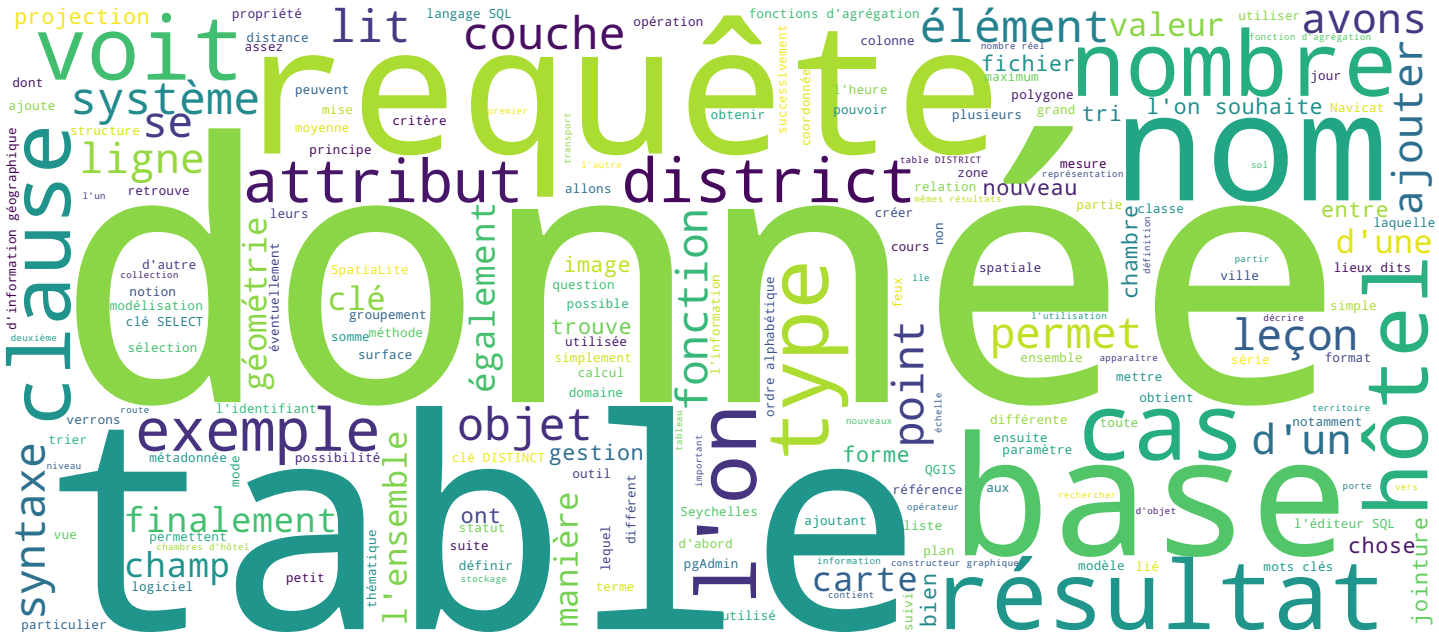
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

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

Requêtes d'agrégation et tri du résultat

Introduction aux systèmes d'information géographique

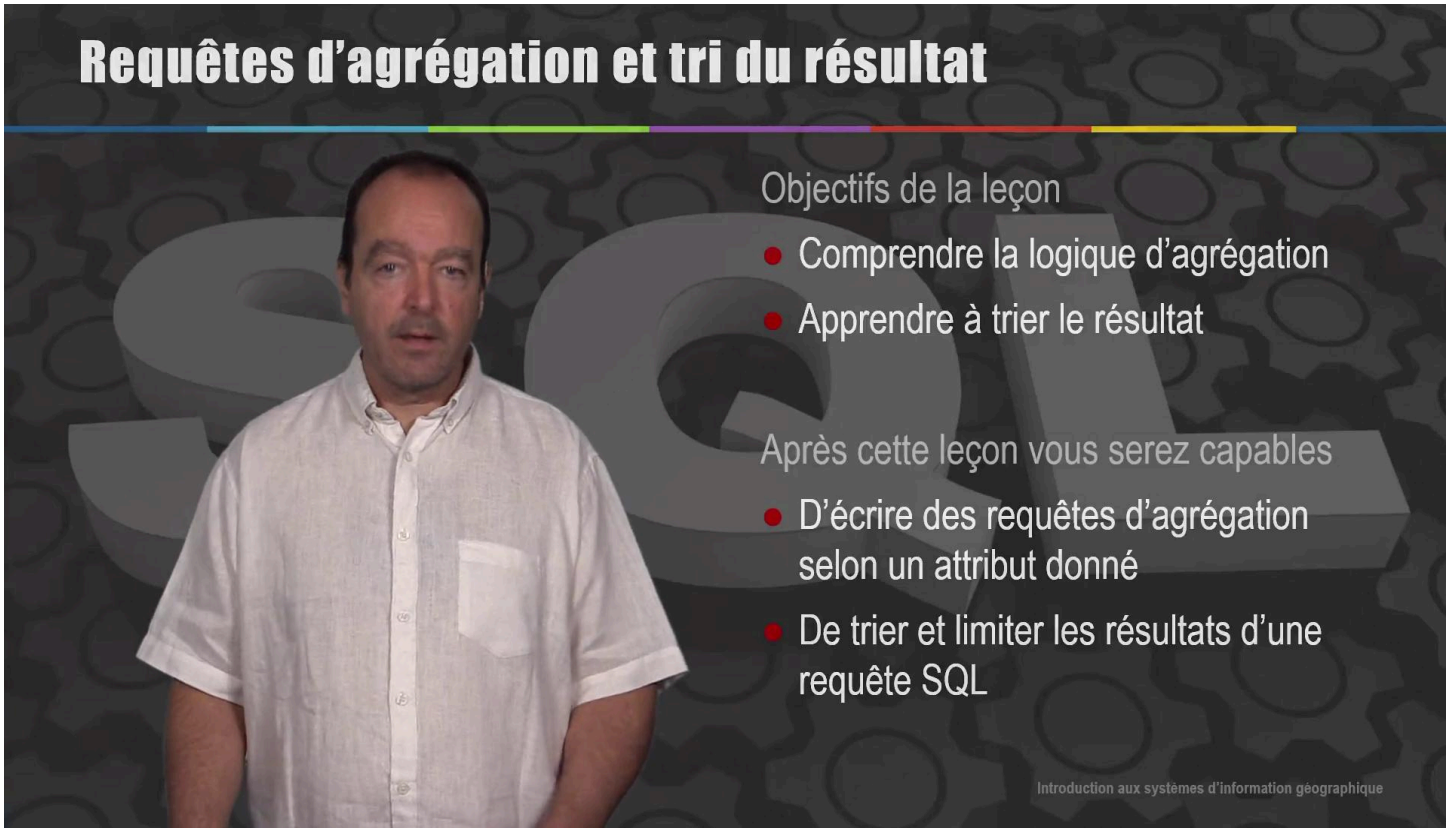
Stéphane Joost, Marc Soutter, Fernand Kouamé, Amadou Sall



Search MOOC

Video

EPFL

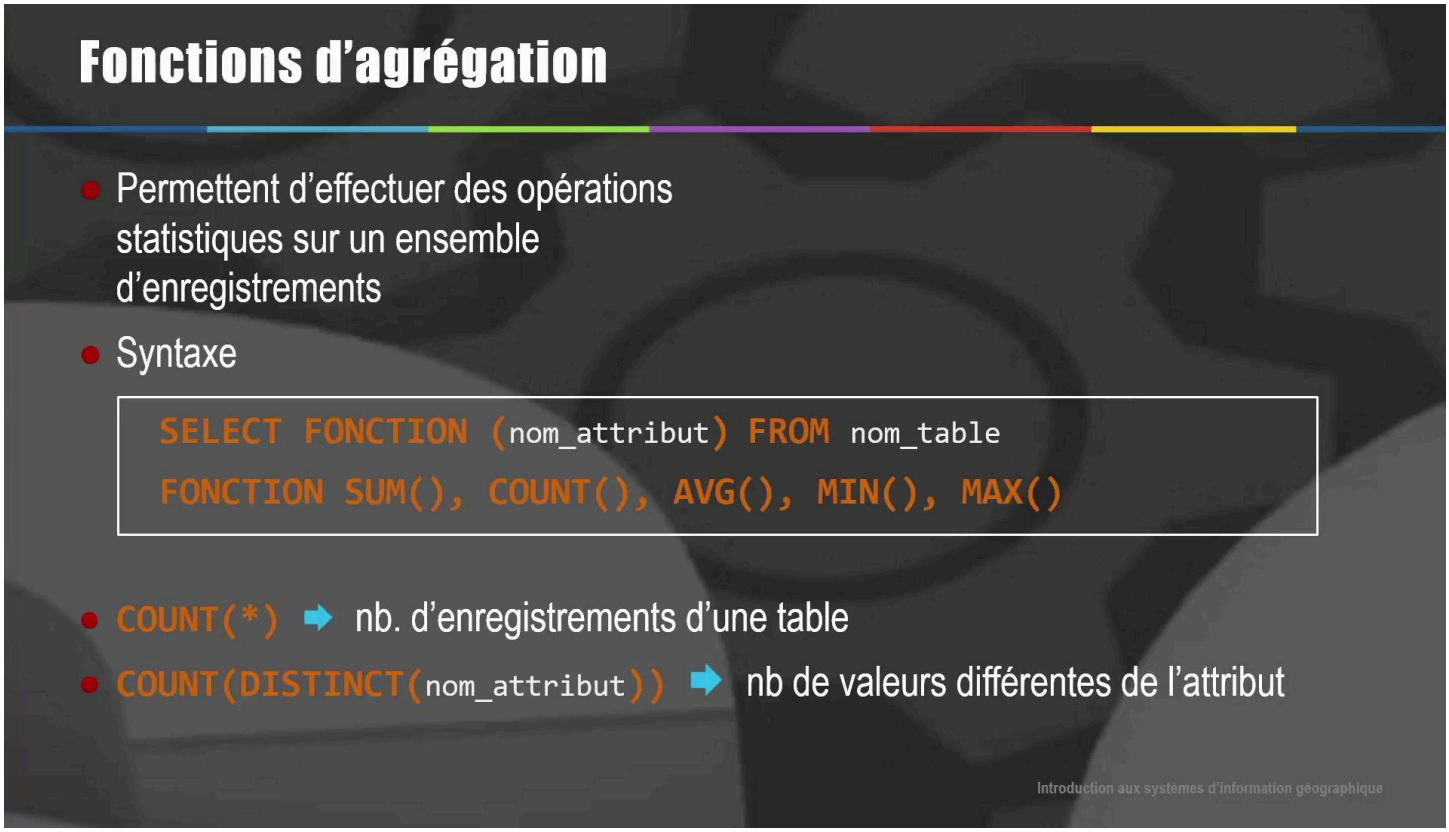


Nous poursuivons donc dans cette leçon notre découverte de la richesse et des subtilités du langage SQL en abordant la question de l'agrégation et du tri à savoir la manière d'extraire des informations synthétiques d'un ensemble ou d'un sous-ensemble de données, de les regrouper, de les filtrer et de les trier. L'objectif de la leçon est donc de comprendre la logique d'agrégation et la manière de trier les résultats d'une requête de sorte que vous puissiez par la suite écrire des requêtes d'agrégation et de tri.

Notes

Summary

0m 21s



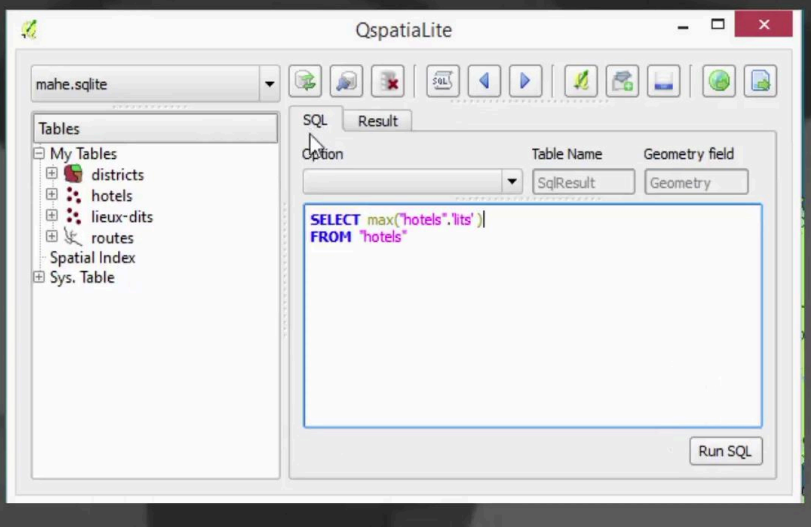
Nous aborderons donc successivement les fonctions d'agrégation, la notion de regroupement, l'agrégation conditionnelle et finalement, les éléments de tri et de limitation des résultats. En revenant à notre tableau de synthèse de la syntaxe de base du langage SQL, on retrouve les clauses d'agrégation qui nous intéressent ici avec les mots-clés GROUP BY et HAVING. Mais nous commencerons d'abord par les fonctions d'agrégation qui permettent d'effectuer des opérations statistiques sur un ensemble ou un sous-ensemble d'enregistrements avec une syntaxe de type... le mot-clé SELECT, le mot-clé de la fonction avec le nom d'attribut sur lequel porte la fonction et puis ensuite la clause FROM et le nom de la table. Et ces fonctions sont de divers types, notamment la somme, le comptage du nombre d'éléments, la moyenne, le minimum et le maximum. La fonction COUNT, avec un attribut ou l'astérisque comme argument, donne donc le nombre d'éléments de la table alors qu'en ajoutant la fonction DISTINCT, on obtient le nombre de valeurs que peut prendre l'attribut utilisé en argument.

Notes

Summary

0m 56s

Fonctions d'agrégation



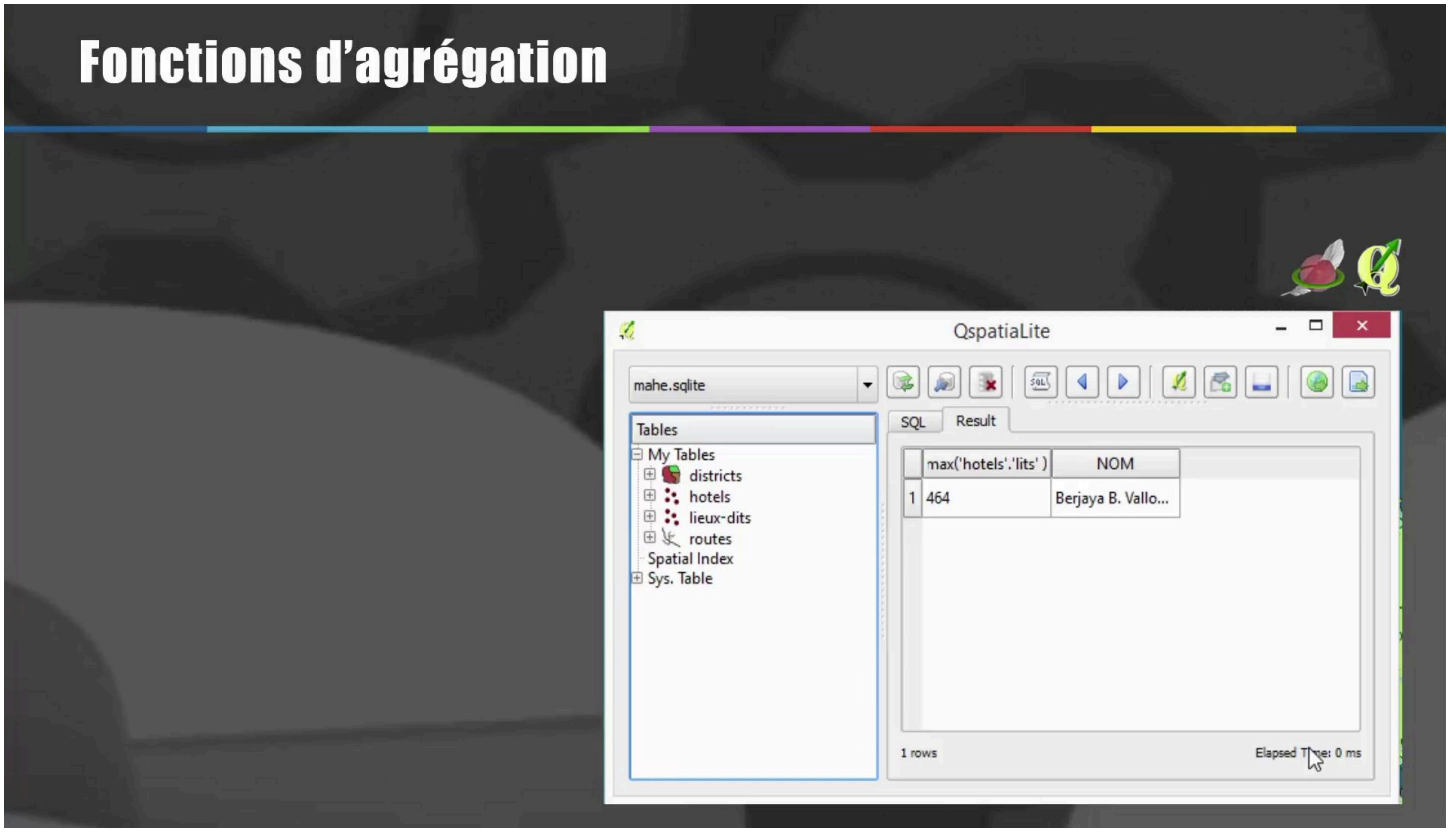
Un exemple de l'application de ces fonctions d'agrégation dans SpatiaLite avec le nombre de chambres de la table des hôtels et l'on souhaiterait avoir en fait un tableau qui nous donne l'ensemble des chambres présentes dans l'île donc 2'518 chambres. On aimerait aussi pouvoir compter en fait la colonne STATUT de cet hôtel, on voit qu'il y a 124 statuts qui correspondent aux 124 hôtels existants et avec le mot-clé DISTINCT, on va pouvoir regrouper en fait les différents statuts qui sont au nombre de 7 au total. Un autre exemple d'utilisation de ces fonctions d'agrégation, avec le calcul du nombre moyen de lits des hôtels obtenu en divisant le total des lits par le nombre d'hôtels. A noter que l'on obtiendrait le même résultat en utilisant directement la fonction d'agrégation moyenne ou AVG. On obtient donc un nombre moyen de 38. Si on veut avoir dans SpatiaLite cette valeur au nombre réel, il faut transformer la somme qui est en entier en un réel ce qui se fait avec la fonction CAST portant sur la variable "élément" et suivie du suffixe "as float". Le maximum, le nombre maximum de lits d'un hôtel est obtenu avec cette fonction max, donc 464.

Notes

Summary

2m 17s

Fonctions d'agrégation



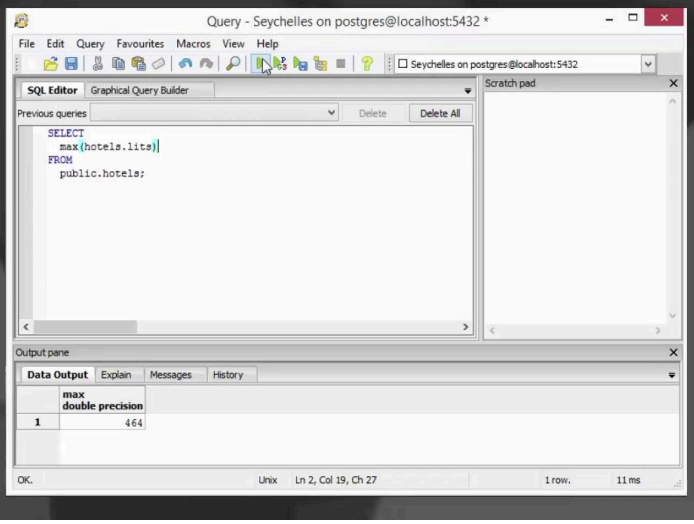
Et puis ce qui peut nous intéresser c'est de voir lequel de ces hôtels a ce maximum de lits donc on peut ajouter dans SpatiaLite le champ NOM comme résultat de la requête et on voit que c'est le Berjaya B. Vallon. Il faut noter que cet ajout du nom n'est pas un standard SQL et provoquera une erreur avec la plupart des SGBD.

Notes

Summary

3m 55s

Fonctions d'agrégation

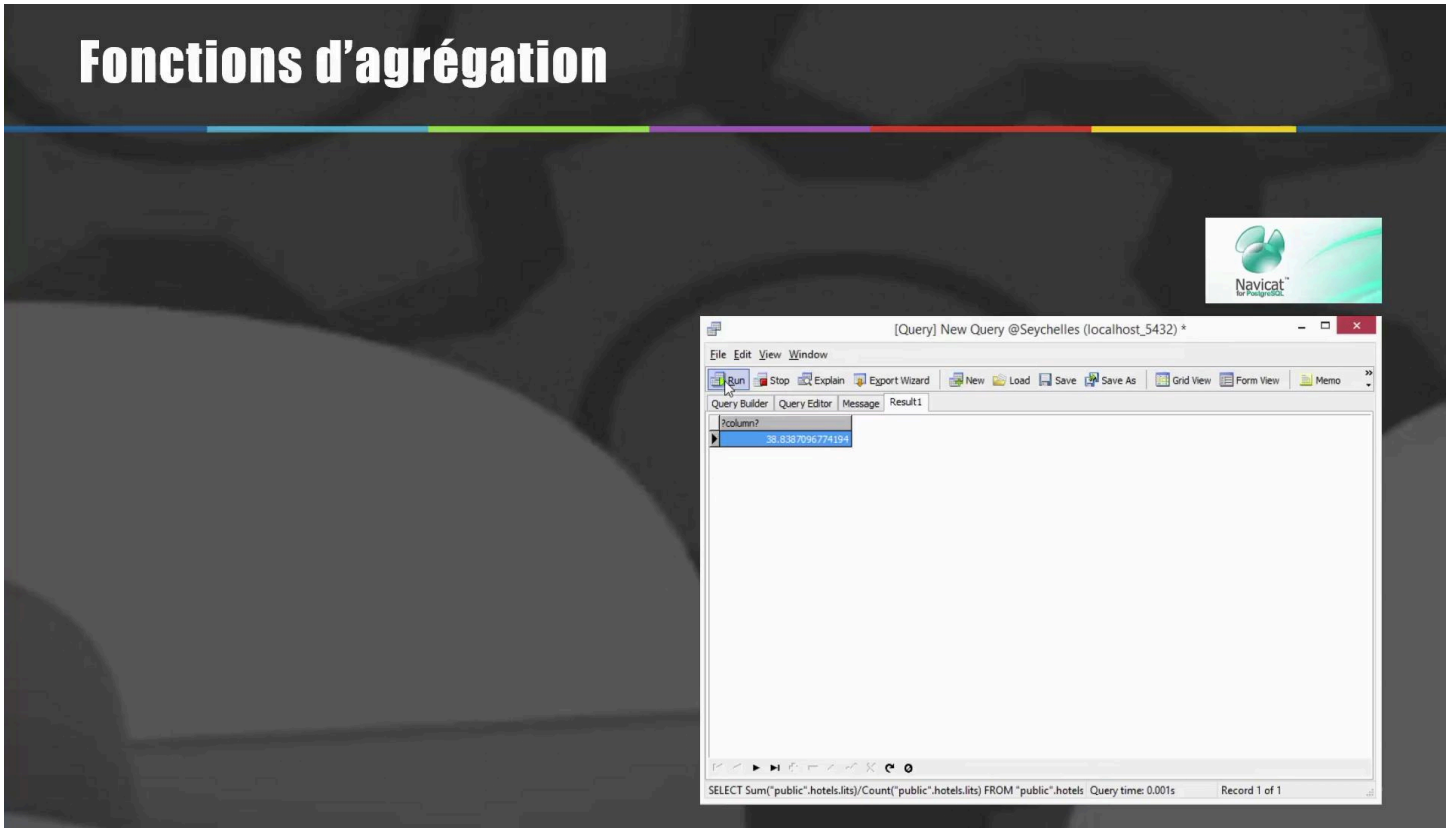


Nous verrons plus tard la syntaxe SQL correcte pour obtenir ce type de résultat. Même exemple, même situation pour pgAdmin où on utilise le constructeur graphique mais on ne va pas très loin parce que les fonctions d'agrégation doivent être écrites dans l'éditeur SQL un peu de la même manière que dans SpatiaLite. Donc, on a vu le cas de la somme des chambres. Maintenant, le comptage des statuts contre de nouveaux les 124 statuts correspondant aux 124 hôtels et avec le mot clé DISTINCT, on obtient de nouveaux les 7 types de statuts caractérisant les hôtels des îles. De la même manière que précédemment, on peut utiliser ces fonctions d'agrégation pour composer un calcul donc ici la la somme des lits de l'hôtel rapportée au nombre d'hôtels pour obtenir en fait la moyenne des hôtels des îles. Et là on voit qu'avec Postgres on obtient directement un résultat en nombre réel et non pas en nombre entier. On peut ici également utiliser la fonction max pour obtenir le nombre de lits du plus grand hôtel des Seychelles.

Notes

Summary

4m 16s

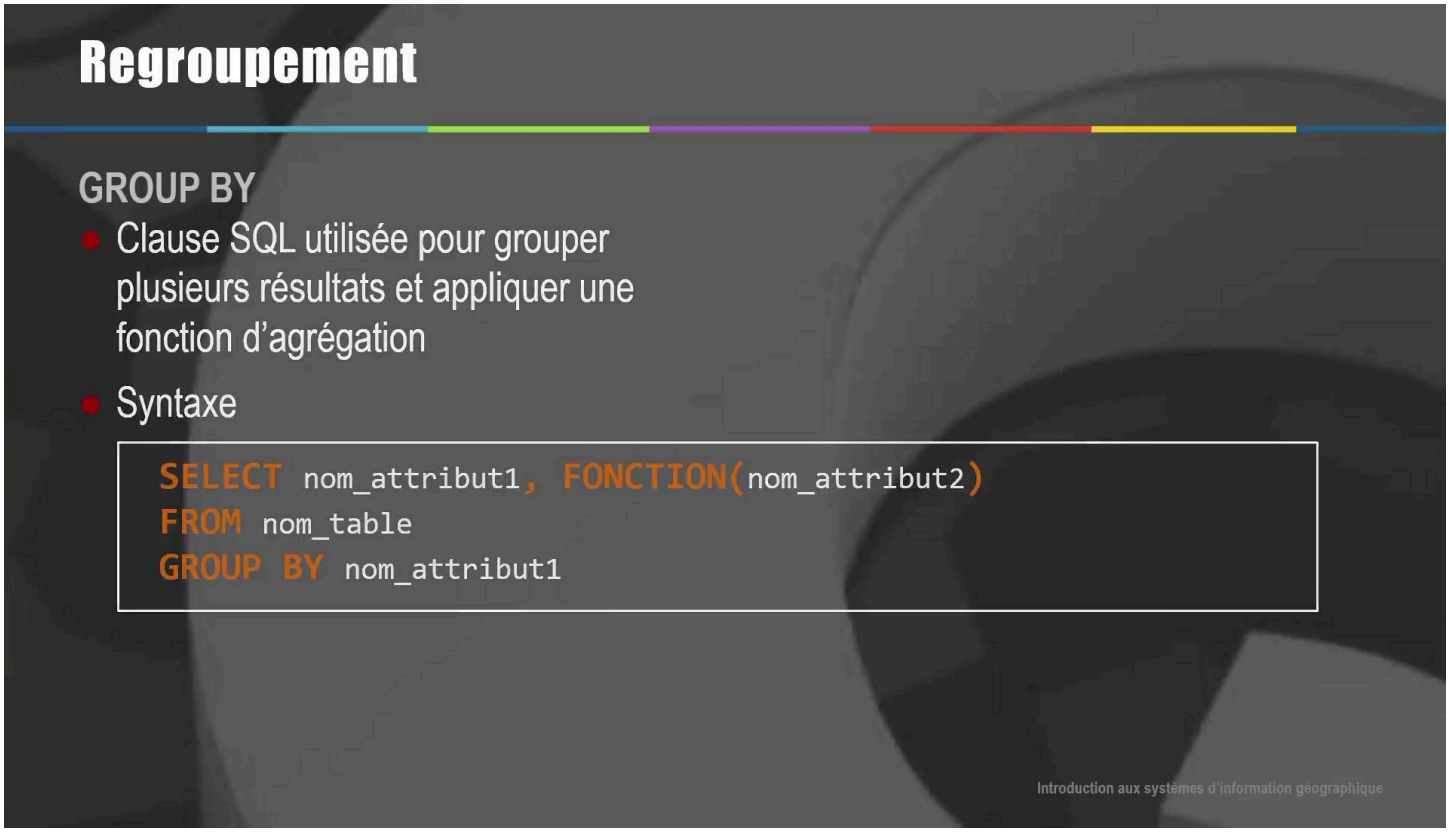


Et nous verrons par la suite, lorsqu'on parlera de requête emboîtée, comment il faut traiter ce type d'interrogation. Le même exemple encore avec le logiciel Navicat, on ajoute de nouveau graphiquement les éléments et puis ici la requête peut être composée dans l'interface graphique, donc on peut ajouter la somme, ajouter la rubrique de comptage, changer l'attribut. On trouve les 124 statuts des 124 hôtels. Par contre, pour ajouter le mot-clé DISTINCT, on doit passer également dans ce cas-là dans l'éditeur SQL pour écrire à pied ce mot-clé. En revenant dans l'éditeur, on peut cette fois remplacer le statut par les lits et puis et finalement c'est plus simple d'écrire cette requête un peu compliquée et qui fait le calcul de la moyenne dans l'éditeur plutôt que dans l'interface graphique. Et ici à nouveau, on trouve le résultat. Alors c'est parce que c'est Postgres qui est derrière directement en nombre réel.

Notes

Summary

5m 41s



Le principe de la clause GROUP BY est de permettre le regroupement de plusieurs résultats et d'appliquer une fonction d'agrégation sur ce groupe. La syntaxe de base comprend donc le mot-clé SELECT avec les attributs que l'on souhaite voir dans le résultat avec éventuellement une fonction qui est exercée sur l'un ou l'autre de ces attributs. Le mot-clé FROM avec la table d'origine qui est interrogée et puis le mot-clé GROUP BY suivi de l'attribut qui va être utilisé pour le regroupement.

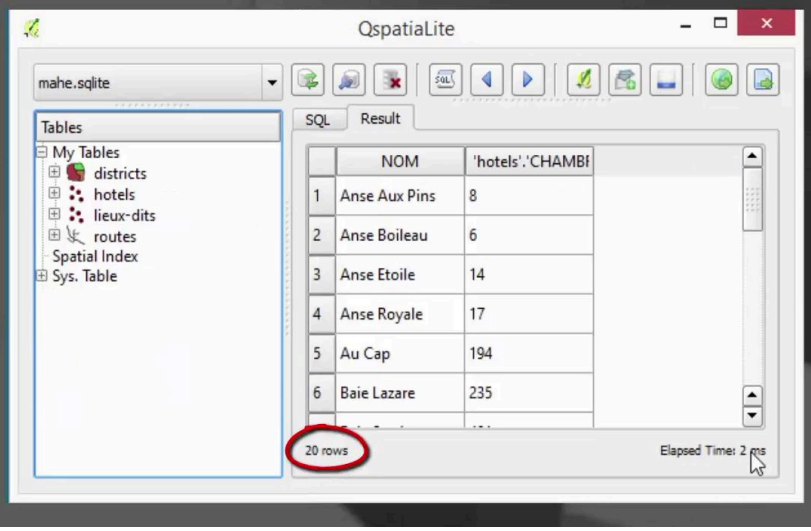
Notes

Summary

7m 06s

Regroupement

GROUP BY



Nous prenons à nouveau un exemple avec le constructeur de requête SQL SpatiaLite intégré à QGIS. Donc, on écrit une requête où l'on cherche à obtenir en fait l'ensemble de chambres d'hôtel disponibles par district. Donc, cette requête comprend donc comme champ résultant le nom de districts et la somme effectuée sur le nombre de chambres des hôtels provenant de la table DISTRICTS et de la table HOTELS avec une jointure comme on l'a vu tout à l'heure entre ces 2 tables, entre l'identifiant de districts de la table HOTELS et l'identifiant, ce même identifiant dans la table DISTRICTS même. A cela, il faut ajouter maintenant une clause de regroupement puisque les nombres... le nombre de chambres d'hôtel est lié à chaque nom de district et qu'il faut, pour les différents hôtels et les différents districts, regrouper ces éléments.

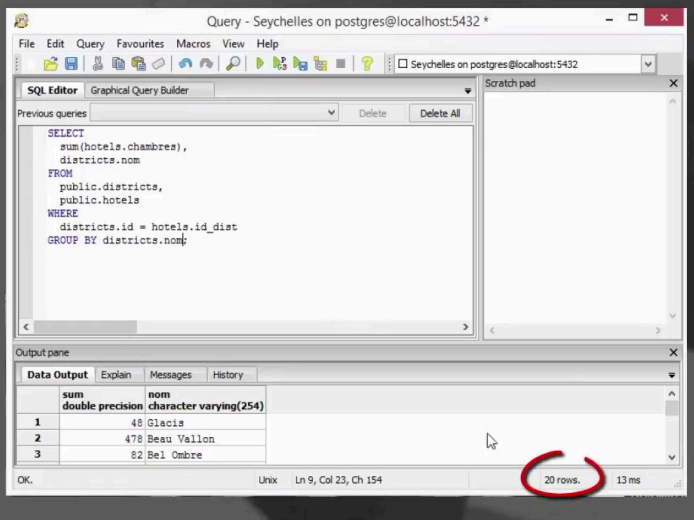
Notes

Summary

7m 42s

Regroupement

GROUP BY



Et on trouve une vingtaine de résultats puisqu'il y a 5 districts qui n'ont pas d'hôtels. Même chose dans le cas de pgAdmin, où on construit cette fois la requête graphiquement avec les tables DISTRICTS et les tables HOTELS. La jointure, les 2 champs que l'on aimerait avoir dans le résultat et on... on doit compléter la requête dans l'éditeur textuel en ajoutant la somme sur les chambres d'hôtel et en ajoutant la clause de groupement, le GROUP BY qui porte donc sur les noms des districts puisqu'on va regrouper les chambres par district.

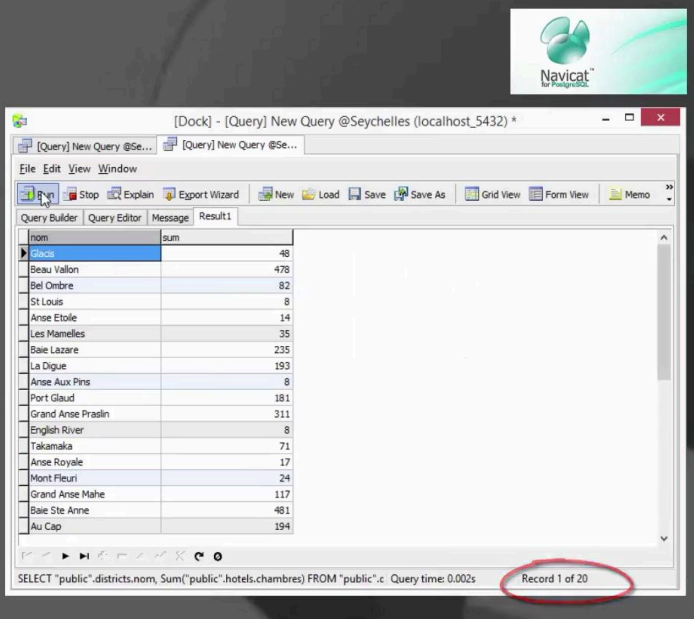
Notes

Summary

8m 42s

Regroupement

GROUP BY



Et l'on voit que l'on obtient de nouveau également les 20 mêmes résultats que tout à l'heure. Dans Navicat, les choses se passent un petit peu de la même manière. On ajoute dans le constructeur graphique les tables, on fait la jointure, on sélectionne les champs que l'on souhaite voir apparaître dans le résultat et puis, on peut cette fois dans l'éditeur SQL intégré, sélectionner directement la fonction d'agrégation et puis on voit qu'automatiquement, le groupement a été ajouté, on obtient les mêmes 20 résultats.

Notes

Summary

9m 28s

Agrégation conditionnelle

HAVING

- Clause SQL similaire à WHERE mais permettant de filtrer des résultats à l'aide de fonctions d'agrégation
- Syntaxe

```
SELECT nom_attribut1, FONCTION(nom_attribut2)
FROM nom_table
GROUP BY nom_attribut1
HAVING FONCTION(nom_attribut2) operateur critere
```


La clause HAVING est similaire à la clause WHERE mais permet de filtrer des résultats à l'aide de fonctions d'agrégation, raison pour laquelle on parle cette fois d'agrégation conditionnelle. Sur le plan de la syntaxe, la syntaxe de base comprend toujours le mot-clé SELECT, la liste des attributs que l'on souhaite grouper, que l'on souhaite avoir dans la table résultante avec éventuellement une fonction sur l'un de ces attributs. Le FROM avec la table d'origine, la clause de groupement avec l'attribut ou les attributs qui interviennent dans cette clause de groupement et finalement une fonction de filtre basée sur le mot-clé HAVING qui associe en fait une fonction attributaire, un opérateur et un critère.

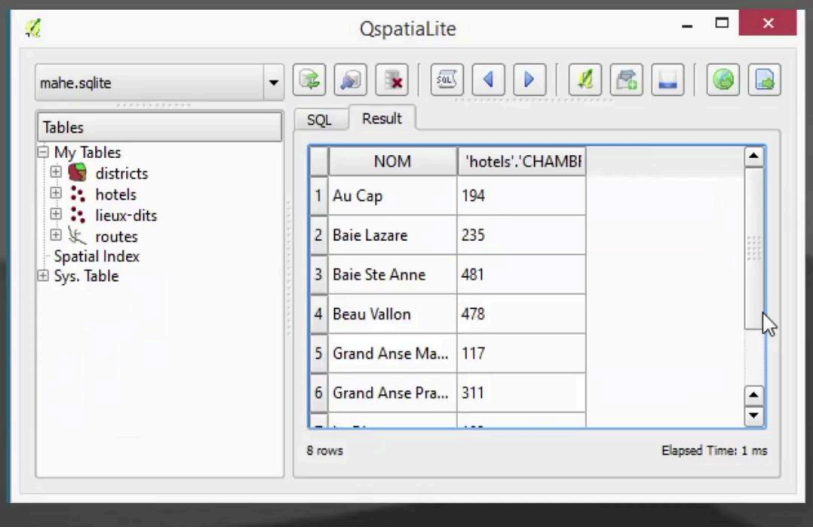
Notes

Summary

10m 08s

Agrégation conditionnelle

HAVING



Alors, on va voir tout de suite comment ça se passe avec le même exemple que tout à l'heure où on a cette requête qui permet de réunir les nombres de chambres par district mais on va simplement filtrer ce résultat pour avoir l'ensemble des districts dans lesquels ce nombre de chambres est supérieur à 100.

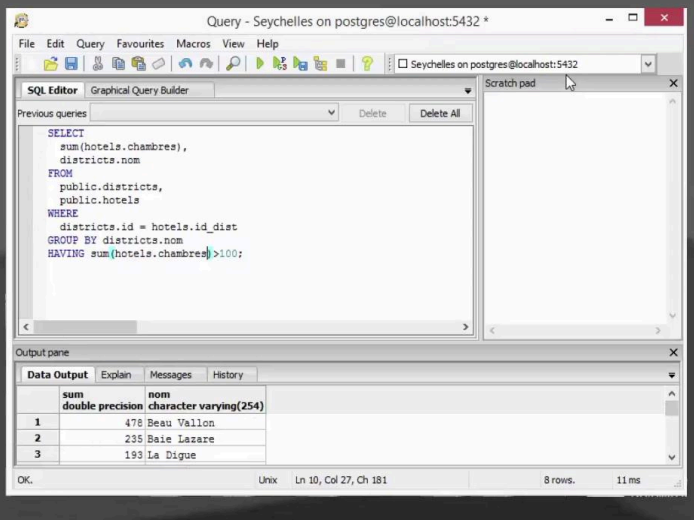
Notes

Summary

10m 56s

Agrégation conditionnelle

HAVING



Et on voit qu'on a 8 districts qui ont davantage de chambres, qui ont plus de 100 chambres. La même chose en reprenant la requête qu'on avait tout à l'heure dans le cas de PostgreSQL avec la même manière, on est obligé d'utiliser l'éditeur SQL pour ajouter cette clause de filtre. Le constructeur graphique n'étant dans ce cas d'aucune utilité. Et à nouveau, on trouve les mêmes 8 résultats.

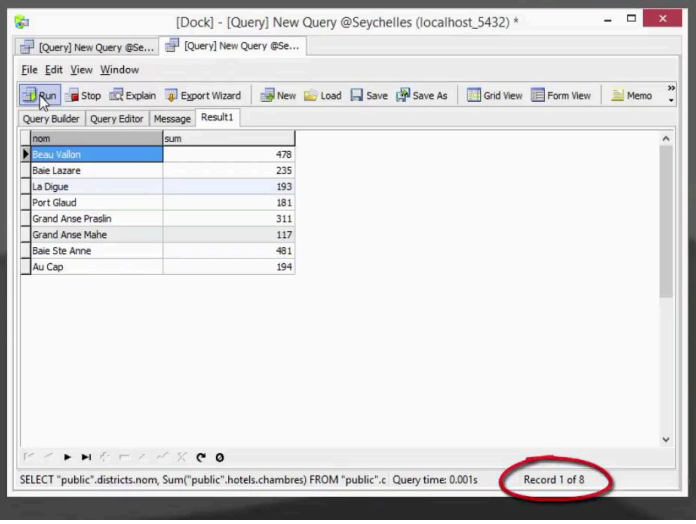
Notes

Summary

11m 23s

Agrégation conditionnelle

HAVING

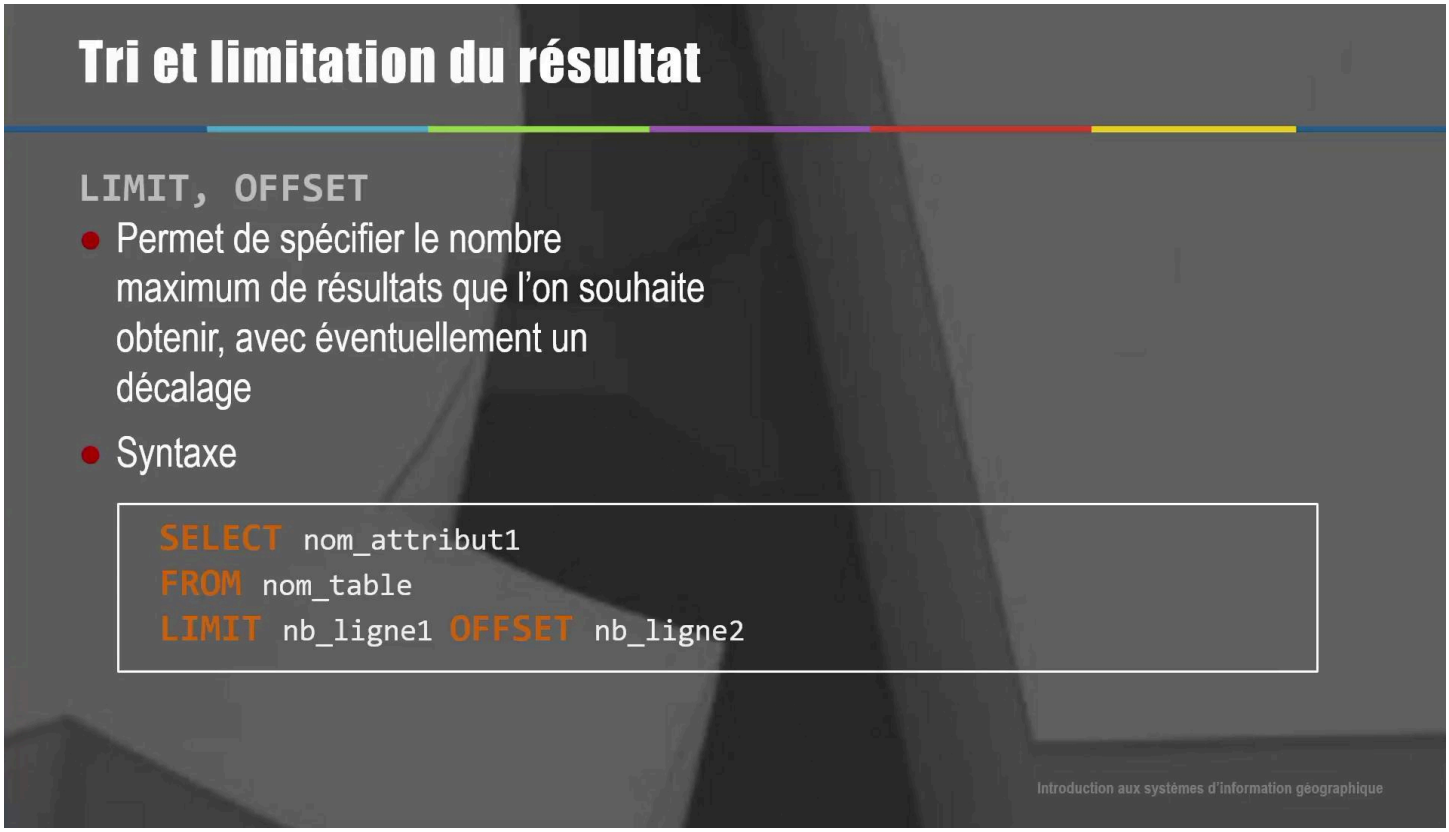


Finalement, dans le cas de Navicat, on repart également de la requête précédente et on voit qu'on a dans l'éditeur intégré la possibilité de construire graphiquement cette requête, ce qui facilite un tout petit peu la vie par rapport à un éditeur SQL standard.

Notes

Summary

11m 52s



Et dans le résultat, on retrouve à nouveau ces 8 hôtels, ces 8 districts pardon. Les clauses de tri et de limitation font donc également partie de la série des éléments syntaxiques de base du langage SQL avec les mots-clés ORDER BY et LIMIT. ORDER BY est donc une clause SQL qui permet de trier les lignes dans un résultat sur une ou plusieurs colonnes par ordre ascendant ou descendant. La syntaxe de base est la suivante : donc le mot-clé SELECT, les attributs que l'on souhaite voir apparaître dans le résultat, la clause FROM avec le nom de la table qui héberge ces attributs et puis la clause de tri ORDER BY avec successivement les différents attributs qui vont être utilisés comme clé de tri avec l'attribut numéro 1 en ordre descendant, l'attribut numéro 2 en ordre ascendant etc. Par défaut, si rien n'est précisé, l'ordre de tri sera ascendant mais il est souvent préférable de préciser l'ordre de tri pour éliminer toute ambiguïté et surtout faciliter la lecture de la requête. Le mot-clé LIMIT permet de spécifier le nombre de résultats que l'on souhaite obtenir et le mot-clé OFFSET permet de définir un décalage dans la table RESULTATS pour rechercher ce nombre limite maximum de résultats. La syntaxe est donc du type SELECT nom d'attribut, FROM nom de table, LIMIT le nombre de lignes et éventuellement OFFSET le nombre de lignes.

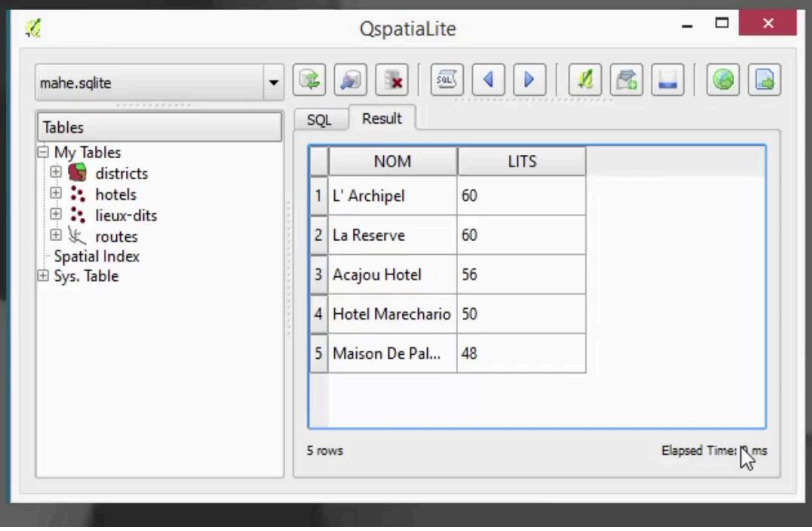
Notes

Summary

12m 16s

Tri et limitation du résultat

ORDER BY
LIMIT, OFFSET



En appliquant ces idées dans le cas de la base de données SpatiaLite, on écrit donc une requête SELECT avec plusieurs attributs, le nom des hôtels, le nombre de lits. Et c'est tout en fait. La clause FROM avec le nom de la table et puis une requête de tri ORDER BY et on va trier par nombre de lits décroissant. Donc, on ajoute le champ LITS avec le mot-clé DECROISSANT et puis un deuxième critère de tri qui va être simplement le nom des hôtels par ordre alphabétique croissant. Donc, on retrouve donc le plus grand des hôtels en tête et puis des... au sein de chaque classe de nombre de lits un tri alphabétique des hôtels. On peut limiter maintenant ce résultat et se dire qu'on s'intéresse qu'aux 5 premières valeurs. Donc, on a les 5 plus grands et dans chaque catégorie par ordre alphabétique et éventuellement décaler cette sélection de 5 de 15 lignes et on atterrit un peu au début avec une série d'hôtels dans les 60 lignes.

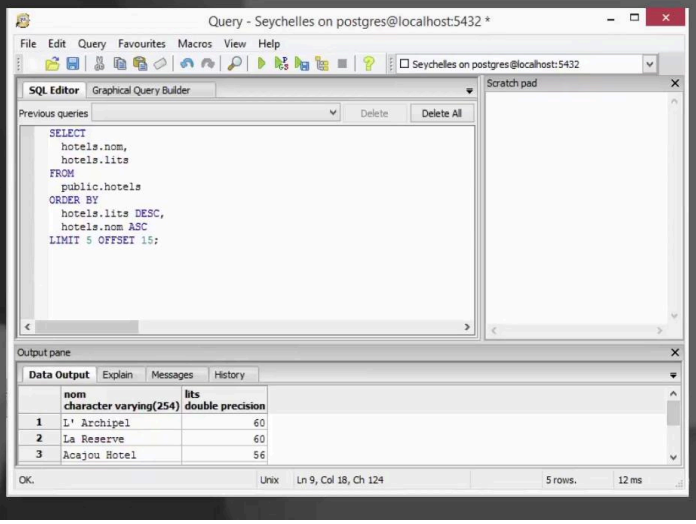
Notes

Summary

14m 04s

Tri et limitation du résultat

ORDER BY
LIMIT, OFFSET



Même opération dans l'environnement pgAdmin pour la base des données Postgres où l'on sélectionne graphiquement les champs et l'on peut composer dans l'onglet ad hoc, graphiquement aussi, la requête de tri. On obtient bien le même résultat avec les hôtels triés par nombre de lits puis par noms et l'on peut ajouter alors cette fois dans l'éditeur SQL la limite de 5 donc on voit très bien les 5 mêmes hôtels que tout à l'heure et l'OFFSET éventuel ici de 15 lignes comme tout à l'heure.

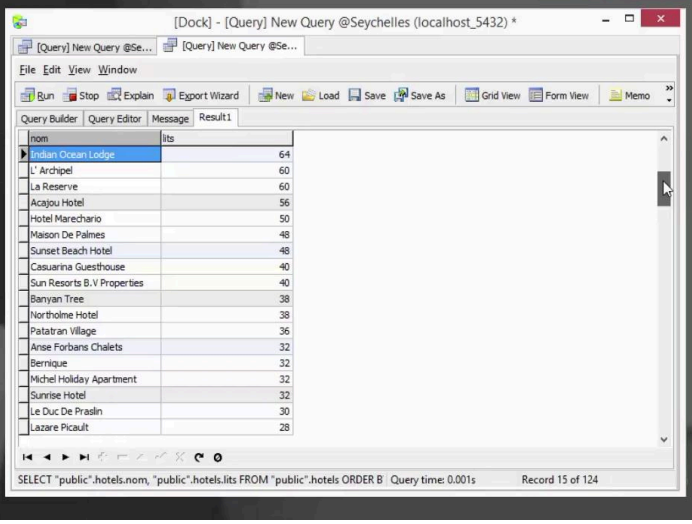
Notes

Summary

15m 32s

Tri et limitation du résultat

ORDER BY
LIMIT, OFFSET



Dernier exemple, avec Navicat où l'on compose de nouveau graphiquement la requête avec le nom, de l'hôtel et les lits, et puis dans l'éditeur graphique du SQL on peut ajouter les champs de tri, définir le type de tri, ici les lits en ordre descendant, donc des lits en nombre descendant et les hôtels en ordre alphabétique. Le résultat correspond bien à ce que l'on attendait et pour ce qui est de LIMIT et OFFSET, ça se passe de la même manière que dans pgAdmin III.

Notes

Summary

16m 21s