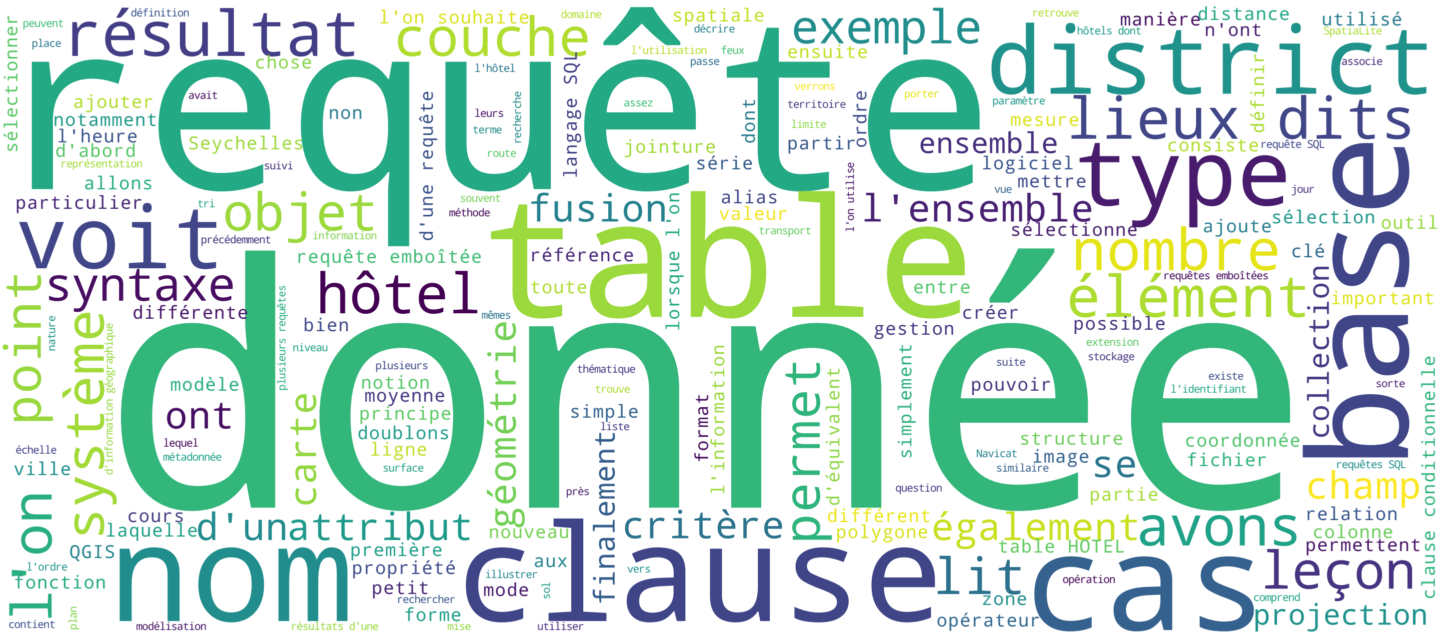
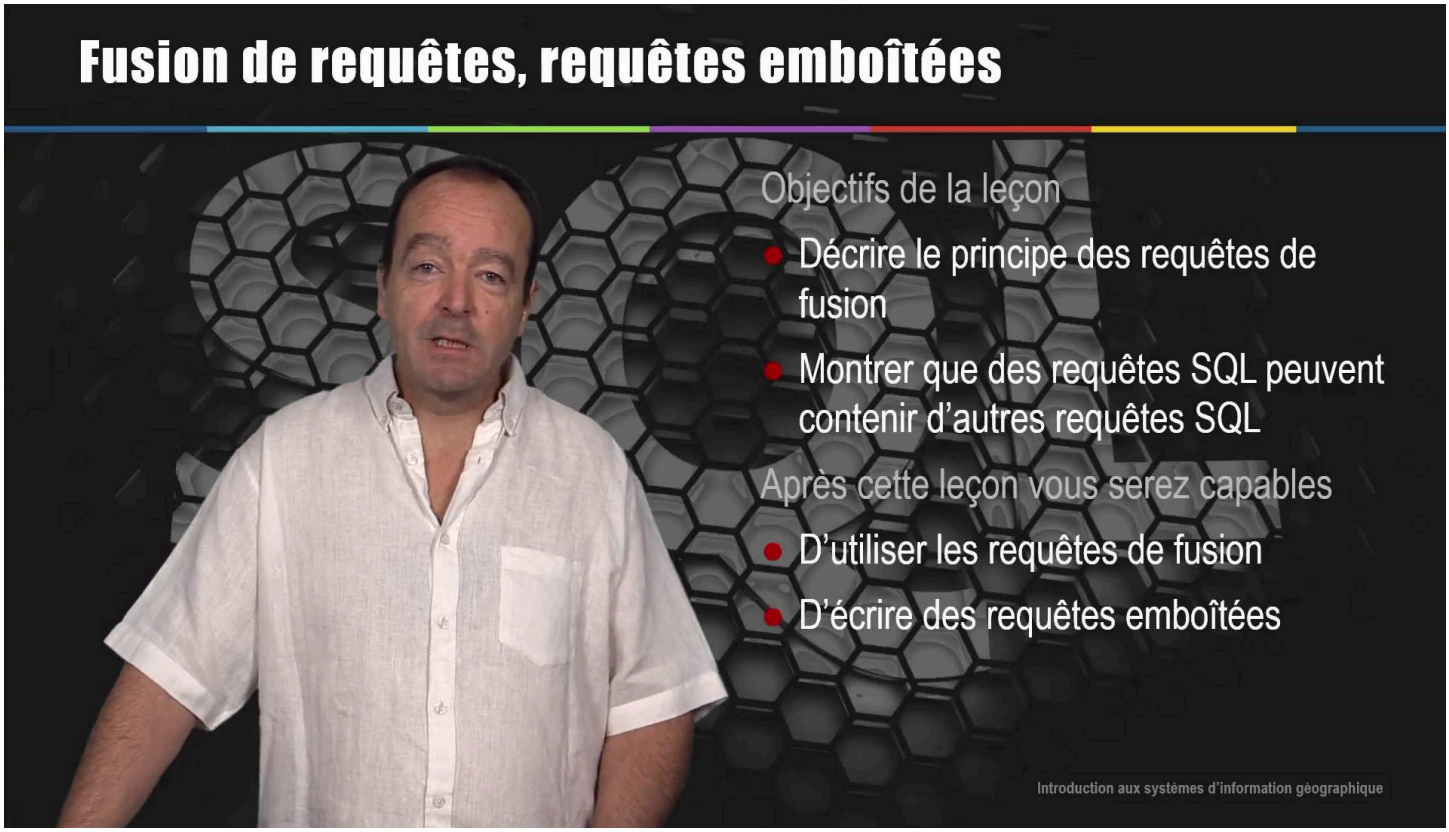


Search MOOC

Video

EPFL

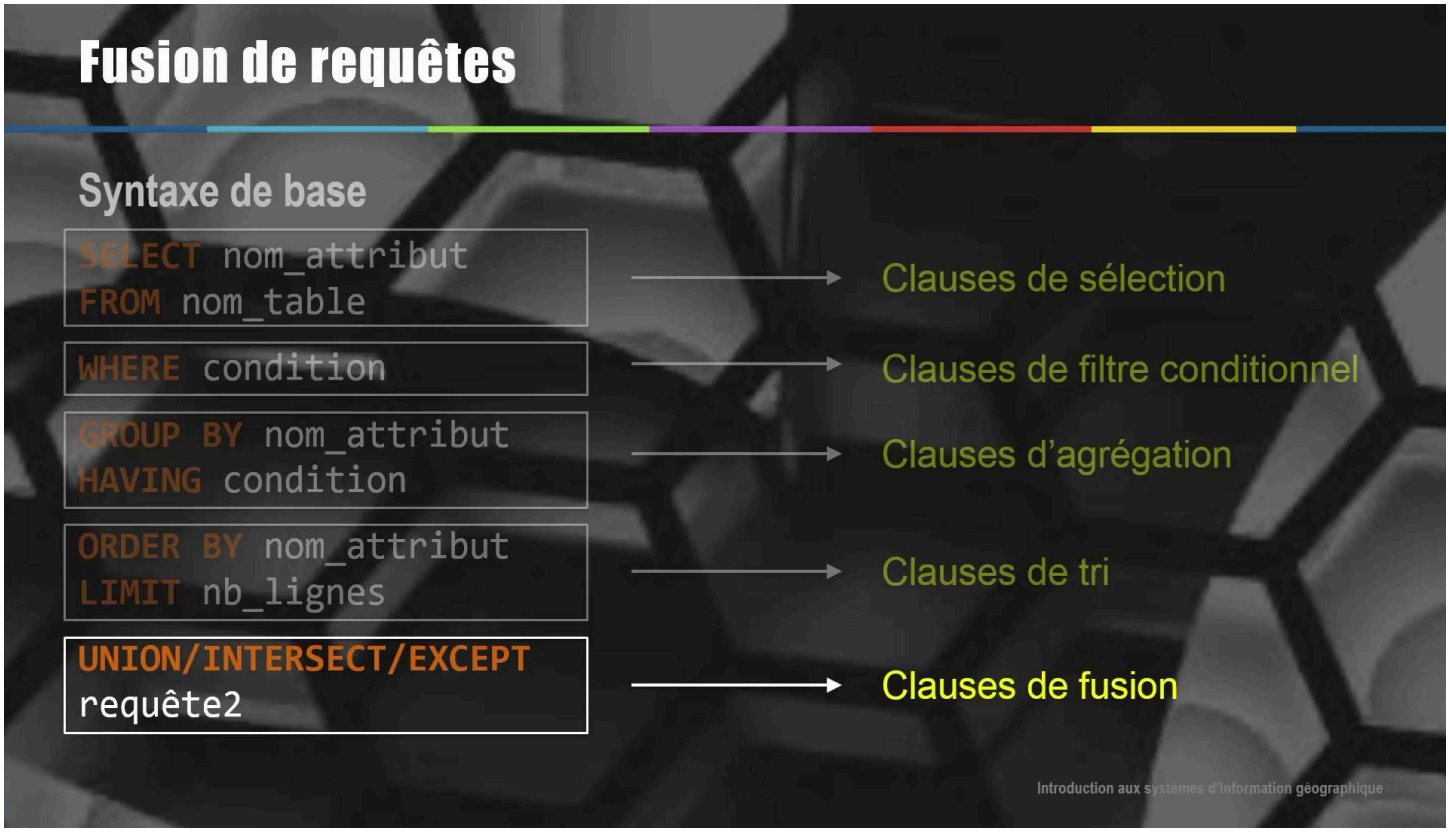


Bonjour. La présente leçon va porter sur la fusion de requêtes que l'on utilise lorsque l'on souhaite associer les résultats de deux ou plusieurs requêtes pour en extraire les éléments communs ou alors pour retirer de l'ensemble des résultats d'une requête le groupe des éléments qui seraient présents dans les résultats d'une autre requête. Nous aborderons également l'utilisation de résultats de requête comme élément d'une autre requête, donc le cas où on emboîte une requête dans une autre requête, ce que l'on fait par exemple lorsque l'on souhaite utiliser une fonction d'agrégation comme le nombre moyen d'employés d'une série d'entreprises comme critère pour sélectionner les PME qui auraient moins d'employés que la moyenne.

Notes

Summary

0m 22s



L'objectif de cette leçon est donc de décrire le principe des requêtes de fusion et de montrer que des requêtes SQL peuvent contenir d'autres requêtes SQL, de sorte qu'au terme de la leçon vous soyez capables d'utiliser des requêtes de fusion pour assembler des résultats, associer les résultats de plusieurs requêtes et d'écrire des requêtes emboîtées. Dans cette leçon nous allons donc aborder successivement le thème de la fusion de requêtes puis nous verrons les requêtes emboîtées dans la clause WHERE avant de passer aux requêtes emboîtées dans la clause FROM et nous terminerons avec l'utilisation des opérateurs IN et NOT IN dans les requêtes de fusion. En reprenant ce tableau qui résume les éléments de la syntaxe de base du langage SQL, nous retrouvons en fait l'ensemble des clauses de sélection, de filtre conditionnel, d'agrégation et de tri que nous avons vu lors des précédentes leçons. Nous voyons qu'il nous reste encore à traiter de ces clauses de fusions avec les mots-clés UNION, INTERSECT et EXCEPT.

Notes

Summary

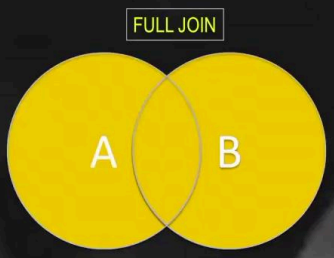
1m 14s

Fusion de requêtes

UNION, UNION ALL

- permet de mettre bout-à-bout les résultats de plusieurs requêtes
- mêmes nombre et type de colonnes, dans le même ordre
- Syntaxe

```
SELECT * FROM nom_table1 UNION
SELECT * FROM nom_table2
```



Structure A ≠ Structure B

La clause UNION du langage SQL permet de mettre bout à bout les résultats de plusieurs requêtes qui utilisent elles-mêmes la commande SELECT. C'est donc une commande qui permet de concaténer les résultats de deux requêtes ou davantage. Pour pouvoir utiliser cette clause de fusion il est important que les deux requêtes que l'on cherche à associer soient structurées de la même manière, donc elles ont le même nombre et le même type de colonnes et que ces colonnes apparaissent dans le même ordre dans les deux tables qui sont associées par la requête d'union. A partir de là la syntaxe est simple c'est deux clauses SELECT qui sont simplement reliées par le mot-clé UNION. Dans l'esprit il s'agit d'une opération qui est similaire à celle du FULL JOIN que nous avons vu dans les leçons précédentes, à ceci près que dans la jointure en fait la structure des deux ensembles que l'on associe peut être complètement différente. On a des champs qui sont totalement différents alors qu'ici, lorsqu'on fait l'union de deux tables, de deux requêtes, il faut vraiment que la structure soit la même dans les deux cas. La particularité de la clause UNION ALL consiste à ne pas éliminer les doublons donc si nous avons des éléments qui sont présents à la fois dans la première et dans la seconde table, ils apparaissent les deux alors que dans la requête d'union simple, les doublons sont éliminés.

Notes

Summary

2m 27s

Second type de requête de fusion, les requêtes d'intersection avec la clause INTERSECT qui permet d'obtenir l'intersection des résultats de deux requêtes. Comme dans le cas de l'union il est important, il est même essentiel que les deux tables que l'on utilise présentent le même nombre et le même type de colonnes et que ces colonnes soient listées dans le même ordre. A partir de là, la syntaxe est de même nature avec deux requêtes de sélection classique reliées par le mot-clé INTERSECT, par la clause INTERSECT. Comme précédemment on peut faire un lien avec l'idée de jointure où nous avions la jointure simple qui consistait en fait à relier les éléments de deux ensembles, de deux requêtes, par l'intermédiaire d'un champ commun. La différence ici c'est que l'intersection entre deux requêtes dans une clause de fusion implique que la structure des deux ensembles soit la même et non pas qu'on associe deux ensembles avec des attributs tout à fait variables. Troisième type de requête de fusion qui permet d'extraire des résultats d'une requête, ceux qui ne feraient pas partie d'une seconde requête. Avec la clause EXCEPT ou MINUS, dans certains SGBD, en particulier dans MySQL, c'est la clause MINUS qu'il faut utiliser.

Notes

Summary

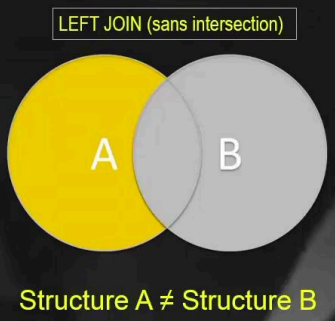
3m 53s

Fusion de requêtes

EXCEPT (MINUS)

- permet de récupérer les enregistrements d'une requête qui ne font pas partie d'une seconde requête
- mêmes nombre et type de colonnes, dans le même ordre
- Syntaxe

```
SELECT * FROM nom_table1 EXCEPT
SELECT * FROM nom_table2
```



Structure A ≠ Structure B

A nouveau, il faut avoir le même nombre et le même type de colonnes dans le même ordre dans les deux tables avec une syntaxe toujours de même nature, deux requêtes de sélection reliées par la clause de fusion. Ce type de fusion s'apparente à une jointure gauche sans intersection. En langage ensembliste avec cette représentation, à ceci près toujours que dans le cas d'une jointure les structures des tables ne sont pas les mêmes.

Notes

Summary

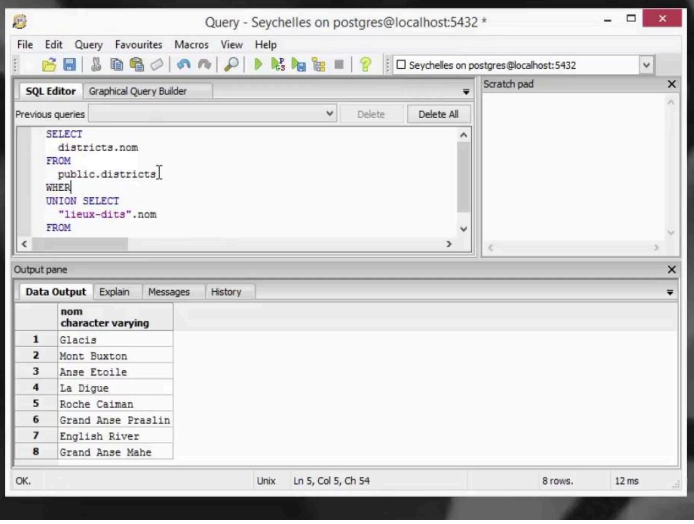
5m 23s

Fusion de requêtes

UNION, UNION ALL

INTERSECT

EXCEPT



Pour illustrer ces requêtes de fusion nous prenons donc la base de données des Seychelles avec la table des districts et celle des lieux-dits et les champs qui contiennent les noms de ces deux tables. La requête elle-même qui associerait ces deux tables s'écrit comme vous le voyez sous les yeux et on copie simplement cette requête pour la reproduire une seconde fois avant de mettre de l'ordre en supprimant dans le premier cas les éléments qui sont liés aux lieux-dits et dans le second cas les éléments qui sont liés aux districts de sorte à obtenir en fait la table qui associe les noms des districts et les noms des lieux-dits avec 330 résultats. On peut ensuite simplement rajouter le mot-clé ALL pour conserver les doublons et on voit qu'on passe à 371 retours donc il y a une quarantaine de doublons qui apparaissent. L'intersection des deux ensembles cette fois nous donne 17 résultats, donc 17 lieux-dits qui sont similaires dans leurs noms aux districts. Finalement, la soustraction des lieux-dits à l'ensemble des noms de districts donne 8 résultats, donc 8 districts qui n'ont pas d'équivalents dans les lieux-dits.

Notes

Summary

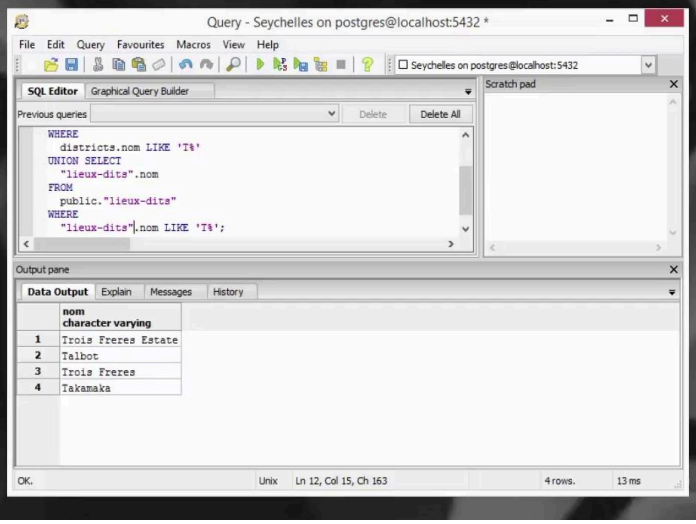
5m 56s

Fusion de requêtes

UNION, UNION ALL

INTERSECT

EXCEPT

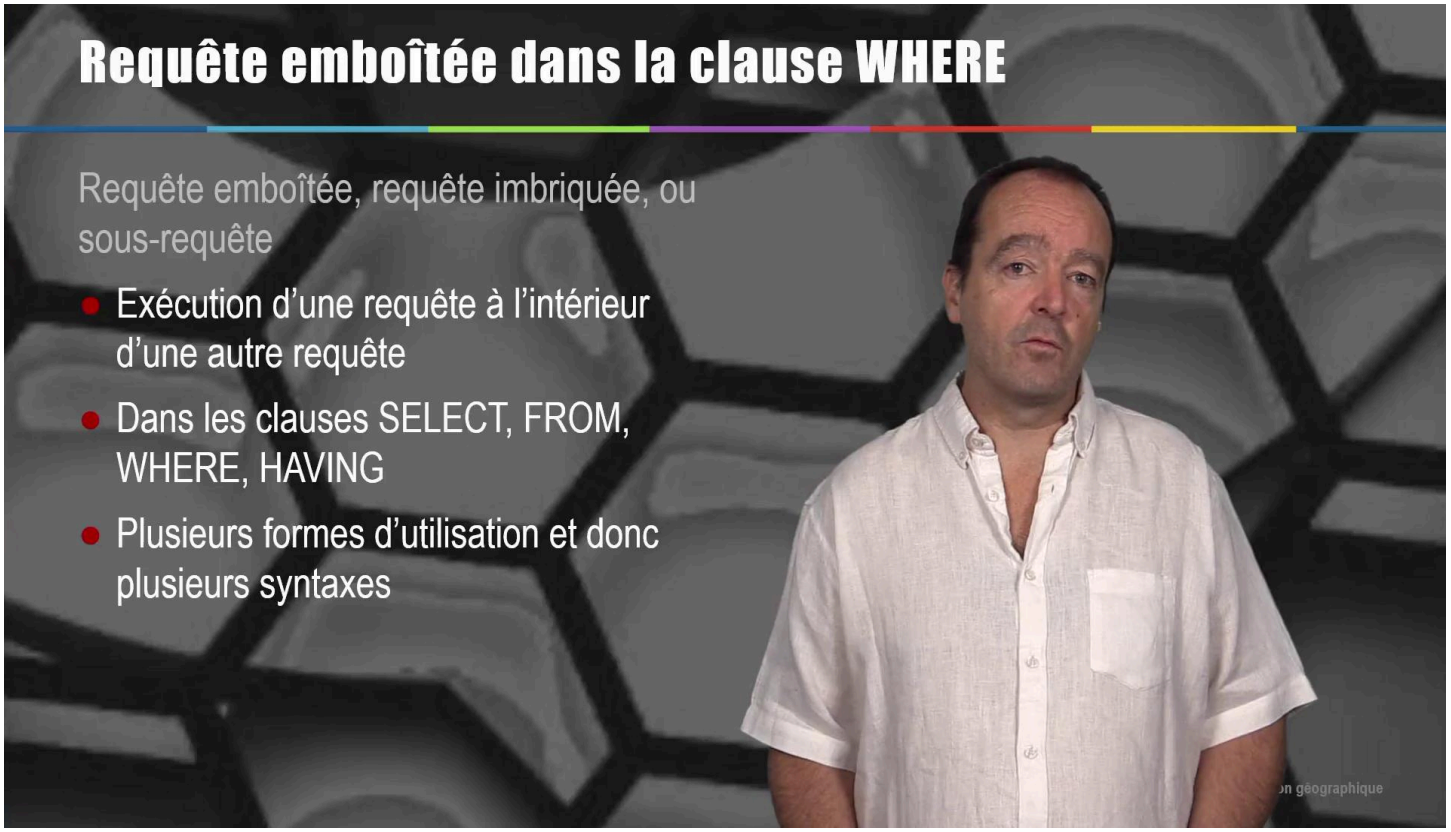


On peut affiner un petit peu la recherche en transformant les requêtes de sélection et en limitant en fait la recherche des districts à ceux qui commencent par la lettre T et en faisant la même chose ensuite pour les lieux-dits. Donc on copie cette clause, on la colle à la fin de la seconde requête et on remplace l'attribut sur lequel porte le filtre par le mot-clé LIEU-DIT entre guillemets puisqu'on a un tiret qui pourrait causer problème. Mais on voit qu'on a 4 résultats, donc 4 districts et lieux-dits qui commencent par T.

Notes

Summary

7m 19s



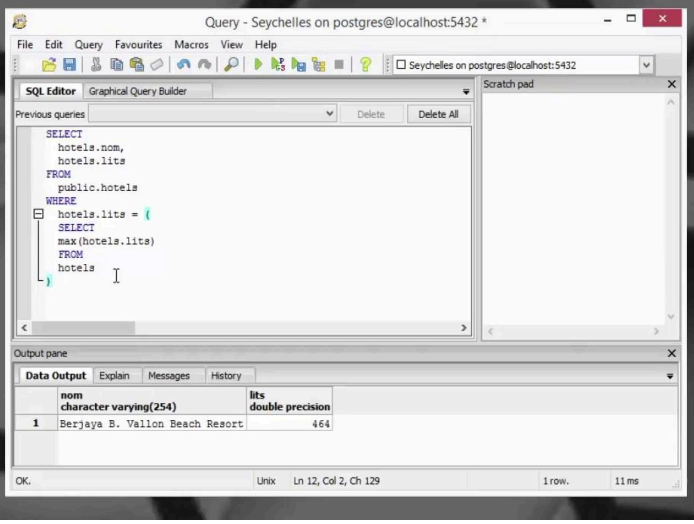
Dans le monde SQL on parle de requêtes emboîtées, des requêtes imbriquées ou encore de sous-requêtes lorsqu'une requête est exécutée à l'intérieur d'une autre requête. Ce genre de disposition peut se rencontrer dans le cas de clauses SELECT, de clauses FROM, de clauses WHERE ou encore de clauses HAVING. Comme il y a plusieurs formes d'utilisation de l'emboîtement des requêtes il existe aussi plusieurs types de syntaxe que nous allons voir un peu plus en détail en commençant par la clause WHERE.

Notes

Summary

8m 16s

Requête emboîtée dans la clause WHERE



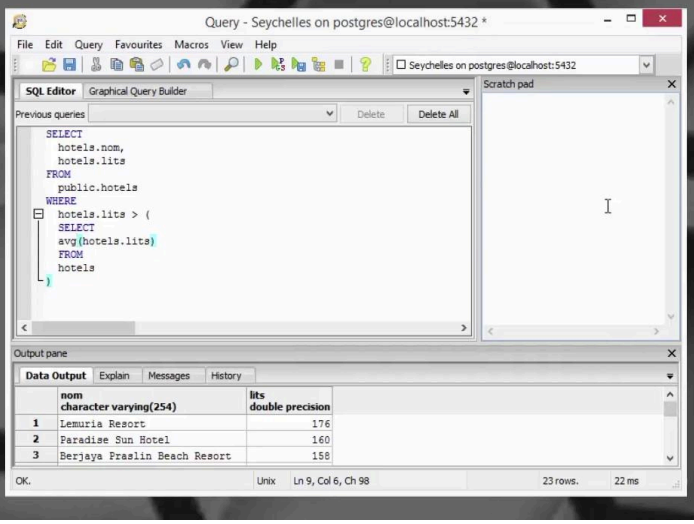
La syntaxe d'une requête emboîtée dans la clause WHERE consiste donc à remplacer dans la condition que doit vérifier la clause WHERE, à remplacer donc le critère par une requête SQL placée entre parenthèses. Pour illustrer l'utilisation de ce type de requête emboîtée on reprend cet exemple où l'on cherchait la valeur maximale du nombre de lits des hôtels des Seychelles. On souhaitait extraire en fait le nom de l'hôtel qui avait le plus grand nombre de lits, chose qui pouvait se faire assez facilement si vous vous en souvenez avec SpatiaLite mais qui n'était pas praticable avec Postgres. Donc il faut une requête un peu plus élaborée dans ce cas-là, donc on sélectionne le nom et le nombre de lits de l'hôtel, de la table HOTELS et comme condition dans la clause WHERE on aimerait sélectionner les hôtels dont le nombre de lits correspond à la valeur maximale du nombre de lits. On l'écrit ici un peu naïvement mais en fait ce maximum peut être décrit par la requête de sélection qu'on avait tout à l'heure, donc le maximum du nombre de lits des hôtels provenant de la table des hôtels avec comme condition... pas de condition. On retrouve Le Berjaya.

Notes

Summary

8m 50s

Requête emboîtée dans la clause WHERE



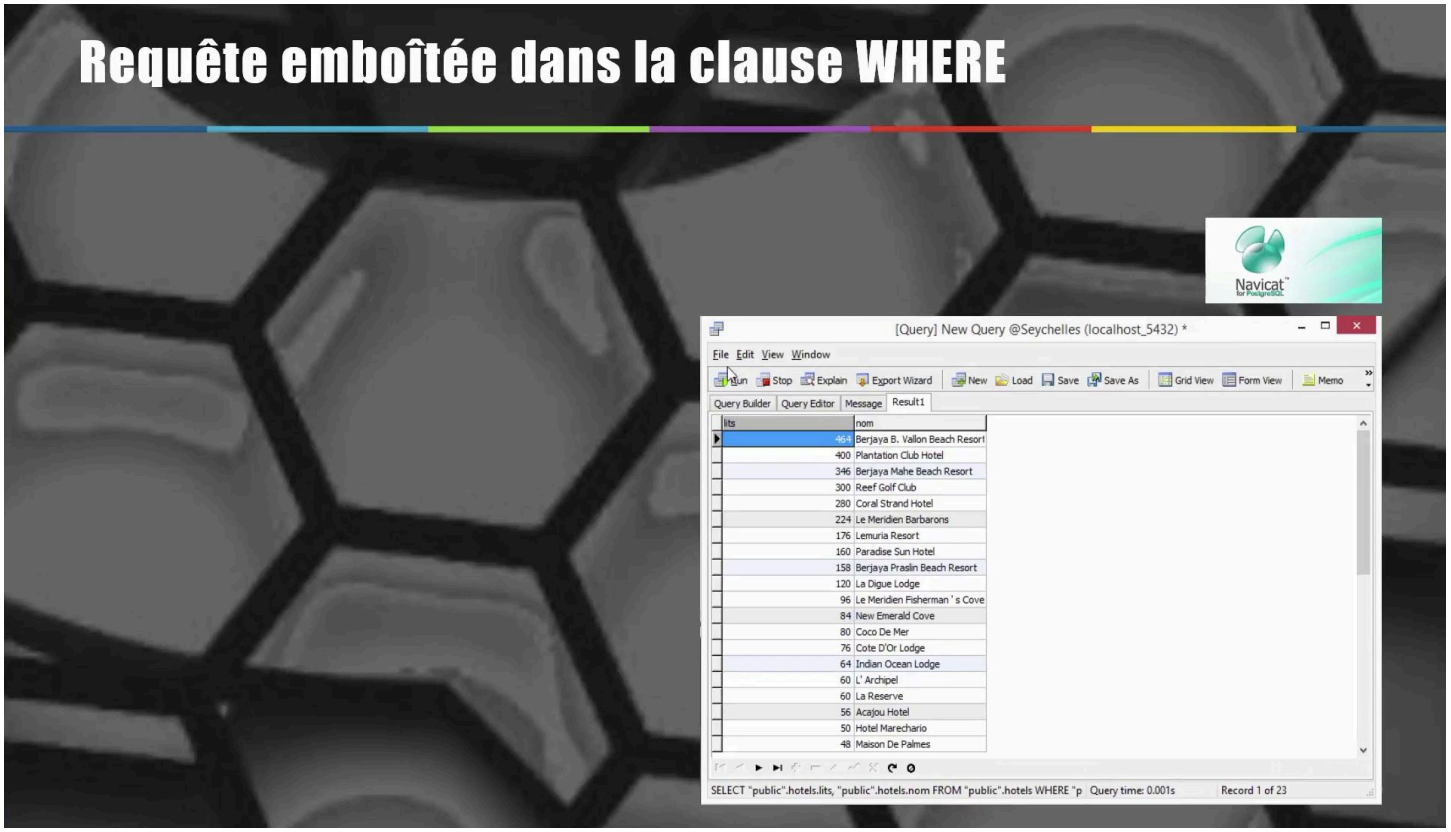
On peut aussi transformer la requête pour rechercher en fait l'ensemble des hôtels dont le nombre de lits est supérieur à la moyenne et on trouve 23 établissements. La moyenne si on s'en souvient dans un précédent exercice on avait vu qu'elle était de l'ordre de 38 lits.

Notes

Summary

10m 29s

Requête emboîtée dans la clause WHERE



Donc on a la liste des hôtels qui ont plus de 38 lits. Dans le cas de Navicat, la chose se fait de manière graphique donc on sélectionne les champs dans la table et puis ensuite dans le constructeur de requêtes on peut ajouter les différents paramètres. On va chercher le nombre de lits supérieur à la moyenne. Alors la requête emboîtée doit quand même être écrite manuellement. L'interface graphique qui facilite la vie a ses limites. Donc comme tout à l'heure, on associe comme critères de recherche la valeur moyenne du nombre de lits du champ des hôtels et en plus ici on va trier en ordre descendant, en taille descendante ces hôtels.

Notes

Summary

10m 47s

Requête emboîtée dans la clause FROM

- Syntaxe

```
SELECT nom_attribut1 FROM (SELECT nom_attribut2 FROM
nom_table1 WHERE nom_attribut3 = critère)
WHERE nom_attribut2 = critère
```

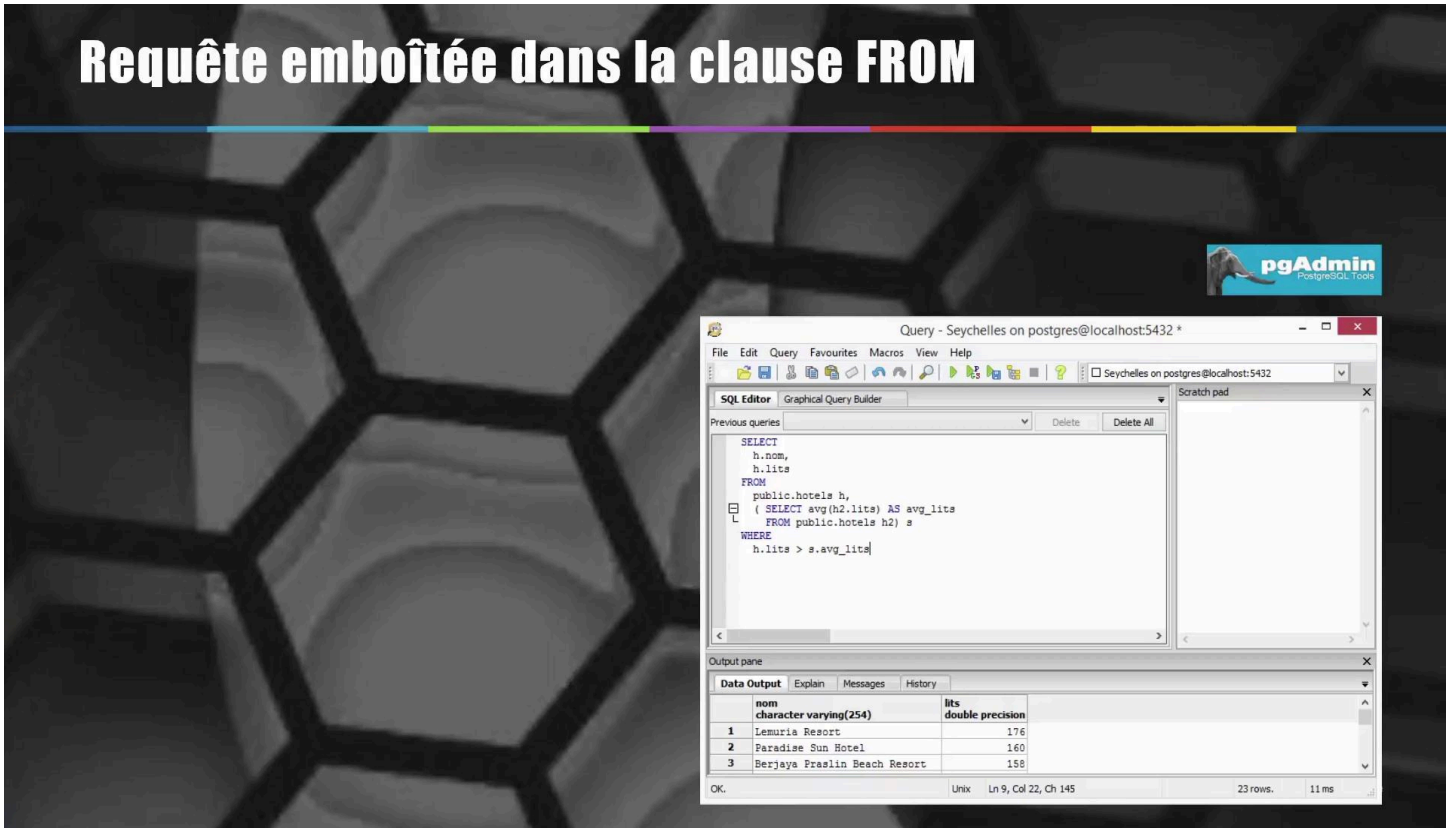

Dans le cas d'une requête emboîtée dans la clause FROM, le principe est exactement le même avec la requête de sélection qui remplace en fait le nom de la table sur laquelle porte la requête que l'on fait. Ce type de syntaxe est possible car une requête de sélection renvoie l'équivalent d'une table de la base de données.

Notes

Summary

11m 47s

Requête emboîtée dans la clause FROM



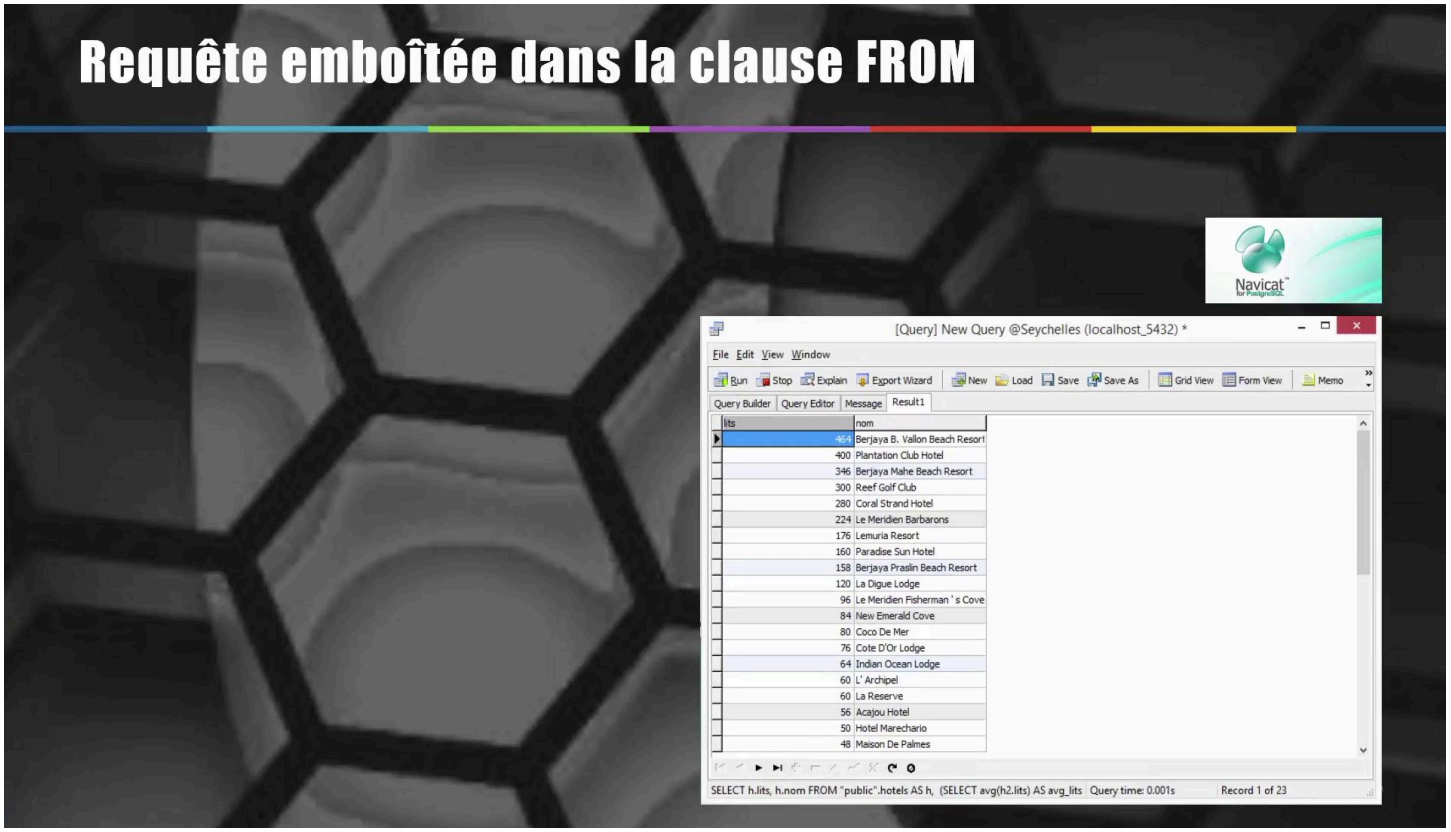
Comme nous allons utiliser plusieurs fois la même table HOTELS dans la clause FROM, nous allons devoir recourir à des alias, ce qui facilite un petit peu l'écriture de la requête SQL. Donc on commence par ajouter une deuxième clause FROM qui se présente sous la forme d'une requête et dans cette requête on va rechercher la moyenne du nombre de lits des hôtels, d'une table HOTELS à laquelle on va donner l'alias H2 pour la distinguer de la première qui a l'alias H. Ce nombre moyen de lits extrait de cette deuxième table va être utilisé dans la clause conditionnelle avec la nécessité d'utiliser un alias pour cette table de la clause FROM, donc S pour la table de synthèse et puis un alias également pour la colonne qui contient les valeurs moyennes de sorte que l'on puisse référencer cet alias dans la clause conditionnelle. On obtient bien les mêmes 23 résultats que dans le cas où on utilisait la clause WHERE.

Notes

Summary

12m 13s

Requête emboîtée dans la clause FROM



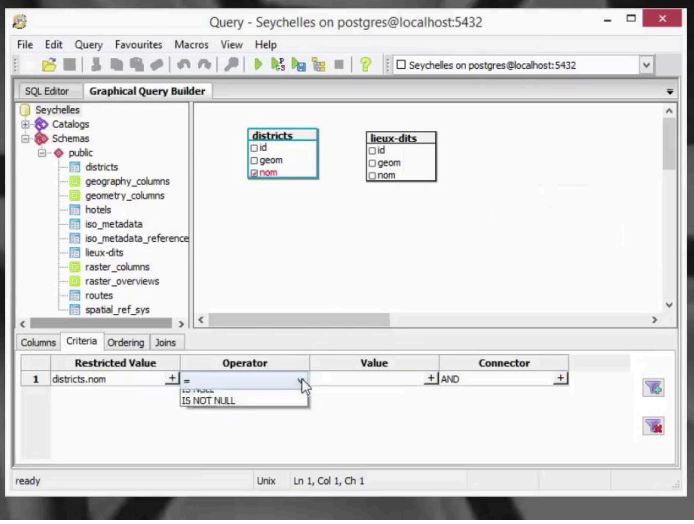
Même chose ici dans le cas de Navicat où l'on remplace la table HOTELS par un alias ce qui met à jour automatiquement les champs et après la suite peut de nouveau se faire de la même manière dans l'éditeur. On copie ici la requête de calcul de la moyenne, on met des alias sur la deuxième table d'hôtels, un alias sur cette requête emboîtée, un alias sur la moyenne, ce qui permet de compléter la requête de sélection, ce qui permet de compléter la clause conditionnelle et puis on conserve l'idée du tri en ordre décroissant.

Notes

Summary

13m 29s

Fusion et opérateurs IN et NOT IN



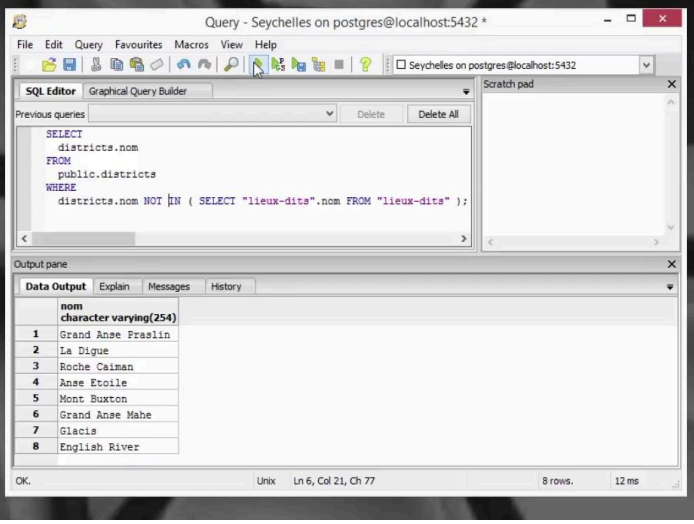
Nous avons vu dans le cas de la requête emboîtée dans la clause WHERE que l'on utilise en fait une requête SQL pour définir le critère qui est utilisé dans la condition que doit vérifier la clause conditionnelle. C'est un cas particulier en fait où la requête qu'on utilise renvoie une seule valeur. Mais il peut arriver, et même souvent, que la requête utilisée renvoie plusieurs valeurs et que la condition que l'on souhaite vérifier c'est que le critère de sélection corresponde à l'une des valeurs de cette collection. C'est dans ce cas-là qu'on va utiliser les opérateurs IN et NOT IN pour tester en fait les résultats de la requête emboîtée. La syntaxe se présente comme précédemment avec simplement l'opérateur qui change devenant IN à la place du "égal" ou du "plus grand que". Pour illustrer ce type d'utilisation de requête emboîtée, nous allons reprendre l'exemple des requêtes d'intersection où l'on recherchait en fait les districts qui ont le même nom que des lieux-dits ou au contraire des districts qui n'ont pas d'équivalents dans la collection des lieux-dits.

Notes

Summary

14m 36s

Fusion et opérateurs IN et NOT IN



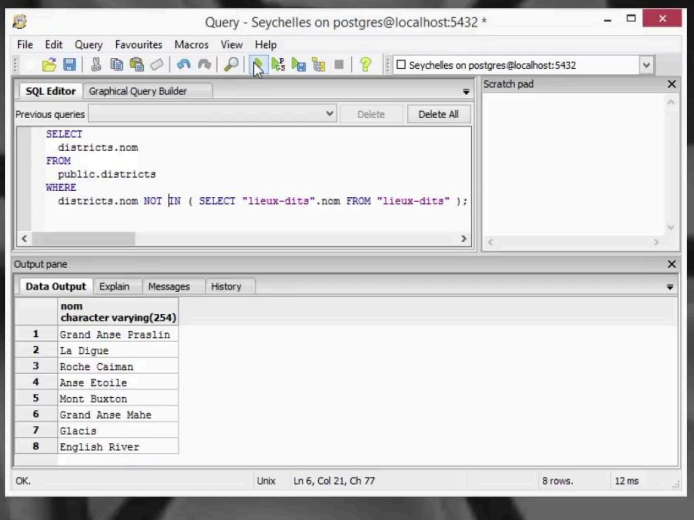
Nous ajoutons donc la table des districts et celle des lieux-dits, sélectionnons le nom de la table des districts et on ajoute un critère pour définir la clause WHERE, critère qui va porter sur le nom de districts et un opérateur IN pour dire qu'on aimerait que le nom figure dans la collection des noms de lieux-dits. Dans cette collection des noms de lieux-dits on doit l'écrire à pied avec le nom de la table toujours entre guillemets à cause des tirets, la clause FROM et le critère étant défini, on peut encore mettre un peu d'ordre en supprimant la table des lieux-dits de la clause principale. Et lorsqu'on exécute cette requête on voit que l'on trouve bien les 17 districts qui ont un équivalent dans la collection des lieux-dits. Pour retrouver les districts qui n'ont pas d'équivalents on inverse simplement le sens de l'opérateur NOT IN et on trouve les 8 districts qui n'ont pas de correspondant dans la collection des lieux-dits.

Notes

Summary

15m 54s

Fusion et opérateurs IN et NOT IN



Dans le cas de Navicat on ajoute donc le district, on sélectionne le nom et puis dans la clause conditionnelle on va ajouter simplement le champ "nom de districts", remplacer l'opérateur, alors ici il s'appelle IS IN LIST mais qui est traduit en SQL par simplement le IN et puis comme précédemment, il faut écrire la requête emboîtée de manière explicite sans oublier les guillemets pour la table LIEU-DIT et en désactivant l'ajout automatique d'apostrophes dans le critère de sélection. Lorsqu'on exécute cette requête, on trouve bien les 17 districts que l'on recherchait et pour le complément on inverse l'opérateur et on trouve les 8 districts qui n'ont pas d'équivalent dans la collection des lieux-dits.

Notes

Summary

17m 00s