SQL RÉVISIONS

Jointure:

1. Jointure interne (INNER JOIN):

La jointure la plus courante.
Combine les enregistrements des deux tables où il y a une correspondance sur les colonnes spécifiées.

Syntaxe: SELECT * FROM table1 INNER JOIN table2 ON table1.colonne = table2.colonne;
Exemple:

Résultat:

Affiche les informations des clients qui ont passé des commandes, avec les détails de leurs commandes.

3. Jointure externe gauche (LEFT JOIN):

Inclut tous les enregistrements de la table de gauche, même s'il n'y a pas de correspondance dans la table de droite.
Syntaxe: SELECT * FROM table1 LEFT JOIN table2 ON table1.colonne = table2.colonne;
Exemple:

SQL
SELECT * FROM clients LEFT JOIN commandes ON clients.id = commandes.client_id;
Utilisez ce code avec précaution. En savoir plus
Résultat:

Affiche tous les clients, même ceux qui n'ont pas passé de commande.

4. Jointure externe droite (RIGHT JOIN):

Similaire à la jointure externe gauche, mais inclut tous les enregistrements de la table de droite.
Syntaxe: SELECT * FROM table1 RIGHT JOIN table2 ON table1.colonne = table2.colonne;

Exemple:

SQL
SELECT * FROM commandes RIGHT JOIN clients ON clients.id = commandes.client_id;
Utilisez ce code avec précaution. En savoir plus
Résultat:

Affiche toutes les commandes, même celles qui n'ont pas été passées par un client enregistré.

Top-N:

Définition:

Le Top-N en SQL est une technique permettant de sélectionner les N premiers enregistrements d'une table en fonction d'une valeur spécifique. Cette valeur peut être une colonne numérique, une date, une chaîne de caractères, etc.

Syntaxe de base:

SQL
SELECT * FROM table ORDER BY colonne DESC LIMIT N;

colonne: La colonne sur laquelle vous souhaitez trier les résultats.
DESC: Indique un tri décroissant (du plus grand au plus petit).
LIMIT N: Limite le nombre de résultats à N.
Exemple:

SQL
SELECT * FROM clients ORDER BY age DESC LIMIT 5;

Options avancées:

Tri croissant: Remplacez DESC par ASC pour trier par ordre croissant.
Plusieurs colonnes de tri: Ajoutez plusieurs colonnes après ORDER BY en précisant l'ordre de tri pour chacune.
Fonctions d'agrégation: Utilisez des fonctions comme SUM ou AVG pour obtenir le top N des valeurs agrégées.

Les fonctions:

Types de fonctions SQL:

Fonctions d'agrégation: Calculent des valeurs statistiques sur un ensemble de données, comme la somme (SUM), la moyenne (AVG), le nombre (COUNT), le minimum (MIN) et le maximum (MAX).
Fonctions de manipulation de chaînes: Modifient les chaînes de caractères, comme la conversion en majuscules (UPPER), en minuscules (LOWER), la suppression d'espaces (TRIM), la recherche de caractères (INSTR) et la concaténation (CONCAT).
Fonctions de conversion de données: Convertissent les données d'un type à un autre, comme la conversion d'une chaîne en nombre (CAST) ou d'une date en chaîne (DATE_FORMAT).
Fonctions de date et d'heure: Extraient des informations des dates et heures, comme l'année (YEAR), le mois (MONTH) et le jour (DAY), et permettent d'effectuer des calculs temporels.
Fonctions système: Accèdent aux informations du système, comme la date et l'heure actuelles (CURRENT_DATE), l'utilisateur connecté (USER) et la version du serveur de base de données (VERSION).

Les sous-totaux:

Comme group by

Calcul de progression:

Progression simple:

SQL
(valeur_actuelle / valeur_cible) * 100

valeur_actuelle: La valeur actuelle du processus.
valeur_cible: La valeur finale attendue.

on peut utiliser SUM

Correction dans une table:

On peut utiliser UPDATE
Ou selectionner des valeurs et leur SET un changement

Difference entre 2 tables ou lignes communes:

Différences entre deux tables avec ou sans ligne commune :

1. Jointures SQL:

Les jointures SQL permettent de combiner des données de deux tables en fonction de colonnes communes.

Types de jointures:

Jointure interne (INNER JOIN): Ne conserve que les enregistrements présents dans les deux tables.
Jointure externe gauche (LEFT JOIN): Conserve tous les enregistrements de la table de gauche, même s'ils n'ont pas de correspondance dans la table de droite.
Jointure externe droite (RIGHT JOIN): Conserve tous les enregistrements de la table de droite, même s'ils n'ont pas de correspondance dans la table de gauche.
Jointure complète (FULL JOIN): Conserve tous les enregistrements des deux tables, même s'ils n'ont pas de correspondance.
2. Table avec ligne commune:

Lorsque deux tables partagent une ou plusieurs colonnes communes, il est possible de les joindre pour obtenir des informations plus complètes.

Exemple:

Table "Clients" avec les noms et adresses.
Table "Commandes" avec les produits achetés et les identifiants clients.
Une jointure sur la colonne "id_client" permet d'afficher les noms des clients et les produits qu'ils ont achetés.

3. Tables sans ligne commune:

Lorsque deux tables n'ont aucune colonne commune, il n'est pas possible de les joindre directement.

Solutions:

Ajouter une colonne commune artificielle: Créer une nouvelle colonne dans une ou les deux tables pour établir une correspondance.
Utiliser des jointures subqueries: Utiliser des requêtes imbriquées pour comparer les données des deux tables.
Exemple:

Table "Clients" avec les noms et adresses.
Table "Produits" avec les noms et descriptions.
Ajouter une colonne "id_categorie" commune aux deux tables permet de les joindre et d'afficher les noms des clients et les produits qu'ils ont achetés.

Points clés à retenir:

Les jointures SQL permettent de combiner des données de plusieurs tables.

Le choix de la jointure dépend de la relation entre les tables et des informations que vous souhaitez obtenir.
Il est possible de joindre des tables avec ou sans ligne commune.
N'hésitez pas à me poser des questions pour plus d'informations.

Sous-requête:

Les sous-requêtes en SQL : Fiches de révision
Définition:

Une sous-requête est une requête imbriquée dans une autre requête. Elle permet d'exécuter une requête secondaire et d'utiliser son résultat comme une valeur dans la requête principale.

Utilisations:

Filtrer les données: Afficher uniquement les enregistrements qui correspondent à une condition définie par la sous-requête.
Effectuer des calculs complexes: Combiner les résultats de plusieurs requêtes pour obtenir des informations plus précises.
Récupérer des données de plusieurs tables: Joindre des tables de manière indirecte en utilisant les résultats de la sous-requête.
Types de sous-requêtes:

1. Sous-requêtes scalaires: Retournent une seule valeur.

Syntaxe: SELECT (SELECT colonne FROM table WHERE condition);
Exemple:

SQL
SELECT nom FROM clients WHERE id_client IN (SELECT id_client FROM commandes);

Ce script affiche les noms des clients qui ont passé une commande.

Optimisation des requetes:

Pour optimiser une requête on peut utiliser with qui regroup une table par exemple

Définition:

La fonction WITH en SQL permet de créer des clauses nommées et réutilisables dans une requête. Elle est utile pour simplifier la syntaxe des requêtes complexes et améliorer la lisbilité du code.

Avantages:

Réduire la duplication de code: Définir des expressions complexes une seule fois et les utiliser ensuite dans différentes parties de la requête.
Améliorer la lisbilité: Rendre le code plus clair et plus facile à comprendre.
Augmenter la modularité: Décomposer les requêtes complexes en modules plus petits et réutilisables.
Syntaxe de base:

SQL
```
WITH nom_clause AS (
  SELECT ...
),
nom_clause2 AS (
  SELECT ...
),
...
SELECT ...
FROM ...
WHERE ...
ORDER BY ...;
```
Utilisez ce code avec précaution. En savoir plus
nom_clause: Le nom de la clause définie.
SELECT ...: La requête qui définit le contenu de la clause.
...: D'autres clauses nommées peuvent être définies.
SELECT ...: La requête principale qui utilise les clauses nommées.
Exemple:

SQL
```
WITH clients_actifs AS (
  SELECT * FROM clients WHERE derniere_connexion > '2023-12-01'
)
SELECT nom, email
FROM clients_actifs
ORDER BY nom;
```
Utilisez ce code avec précaution. En savoir plus
Ce script définit une clause nommée clients_actifs qui sélectionne les clients qui ont été actifs au cours du dernier mois. La requête principale utilise ensuite cette clause pour afficher les noms et les adresses e-mail des clients actifs.

Cas d'utilisation:

Calculs complexes: Définir des expressions intermédiaires pour les utiliser dans des calculs ultérieurs.
Filtrage de données: Définir des clauses de filtrage pour les utiliser dans plusieurs parties de la requête.

Jointures complexes: Définir des clauses de jointure pour simplifier la syntaxe de la requête principale.