



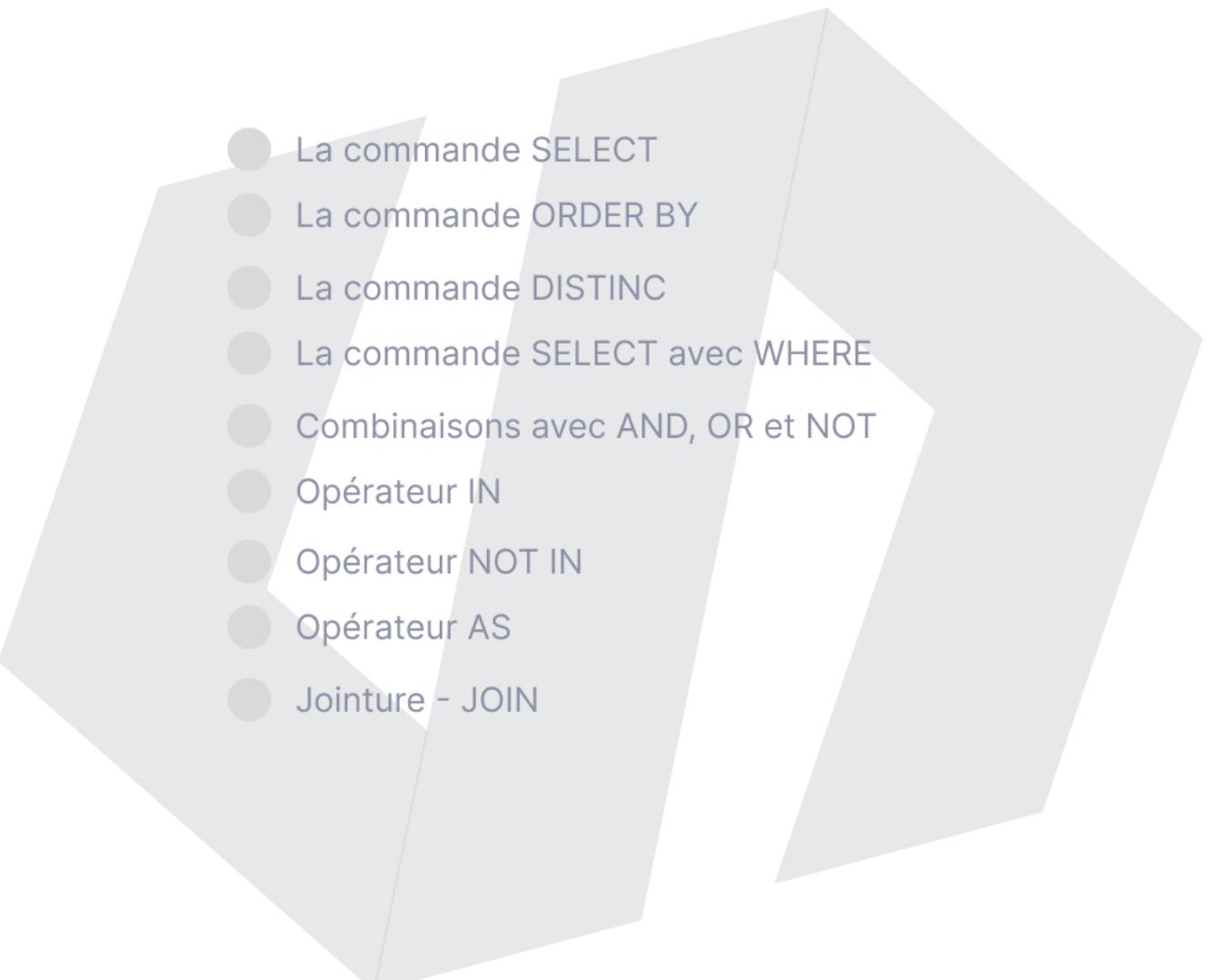
LA MINUTE DE CODE

MySQL™

SQL

PARTIE 2

AFFICHER | ORDONNER | COMBINER | OPÉRATEURS | REQUÊTES MULTIPLES

- 
- La commande SELECT
 - La commande ORDER BY
 - La commande DISTINCT
 - La commande SELECT avec WHERE
 - Combinaisons avec AND, OR et NOT
 - Opérateur IN
 - Opérateur NOT IN
 - Opérateur AS
 - Jointure - JOIN

La commande "SELECT" est une instruction SQL (Structured Query Language) qui permet de récupérer des données à partir d'une ou plusieurs tables dans une base de données relationnelle. Cette commande est l'une des plus importantes en SQL car elle permet de récupérer et afficher les informations stockées dans une base de données.

La syntaxe de la commande SELECT est la suivante :

```
SELECT colonnes  
FROM tables  
WHERE conditions;
```

- "colonnes" spécifie les colonnes de la table que vous voulez sélectionner. Si vous voulez toutes les colonnes, vous pouvez utiliser l'astérisque (*).
- "tables" spécifie les tables dans lesquelles vous voulez effectuer la sélection.
- "conditions" spécifie les critères de sélection. Si vous ne voulez pas de conditions de sélection, vous pouvez omettre la clause WHERE.

La commande SELECT peut être utilisée de différentes manières, en fonction des besoins de l'utilisateur. Par exemple :

- Sélectionner toutes les colonnes d'une table :

```
SELECT *  
FROM table;
```

- Sélectionner des colonnes spécifiques d'une table :

```
SELECT colonne1, colonne2  
FROM table;
```

- Sélectionner des colonnes d'une table en fonction de certaines conditions :

```
SELECT colonne1, colonne2  
FROM table  
WHERE condition;
```

- Sélectionner des données dans plusieurs tables en utilisant des jointures :

```
SELECT colonne1, colonne2  
FROM table1  
INNER JOIN table2  
ON table1.colonne = table2.colonne;
```

La commande SELECT peut également être utilisée avec d'autres instructions SQL, telles que GROUP BY, ORDER BY, HAVING, etc.

La commande "ORDER BY" est une clause SQL qui est souvent utilisée avec la commande "SELECT". Elle permet de trier les résultats de la commande SELECT dans un ordre spécifié.

La syntaxe de la clause "ORDER BY" est la suivante :

```
SELECT colonnes  
FROM tables  
WHERE conditions  
ORDER BY colonne [ASC | DESC];
```

- "colonnes" spécifie les colonnes que vous voulez sélectionner.
- "tables" spécifie les tables dans lesquelles vous voulez effectuer la sélection.
- "conditions" spécifie les critères de sélection.
- "colonne" spécifie la colonne à utiliser pour trier les résultats.
- "ASC" spécifie un tri ascendant (par défaut) et "DESC" spécifie un tri descendant.

Par exemple, si vous voulez trier les résultats de la commande SELECT en fonction de la colonne "emp_nom" dans l'ordre alphabétique croissant, vous pouvez utiliser la commande suivante :

```
SELECT emp_nom  
FROM employes  
ORDER BY emp_nom ASC;
```

Si vous voulez trier les résultats en fonction de la colonne "emp_nom" dans l'ordre décroissant, vous pouvez utiliser la commande suivante :

```
SELECT emp_nom  
FROM employes  
ORDER BY emp_nom DESC;
```

La commande "ORDER BY" peut également être utilisée avec plusieurs colonnes. Dans ce cas, les résultats sont triés en fonction de la première colonne spécifiée, puis en fonction de la deuxième colonne spécifiée, et ainsi de suite. Par exemple :

```
SELECT shop_id, emp_nom  
FROM employes  
ORDER BY shop_id DESC,  
emp_nom ASC;
```

Dans cet exemple, les résultats sont d'abord triés par ordre décroissant en fonction de la colonne "shop_id", puis par ordre alphabétique croissant en fonction de la colonne "emp_nom".

Pour afficher de manière aléatoire:

```
SELECT * FROM employes ORDER BY RAND();
```

La commande "SELECT DISTINCT" en SQL est utilisée pour récupérer des valeurs uniques à partir d'une colonne d'une table ou d'une vue.

Elle permet de sélectionner toutes les occurrences distinctes d'une colonne spécifiée dans une table, en éliminant les doublons.

La syntaxe de la commande SELECT DISTINCT est la suivante :

```
SELECT DISTINCT colonne1, colonne2, ...
FROM table;
```

- "colonne1", "colonne2", etc. spécifient les colonnes de la table à partir desquelles vous voulez récupérer les valeurs uniques.
- "table" spécifie la table à partir de laquelle vous voulez récupérer les valeurs.

Par exemple, si vous avez une table "employes" avec les colonnes "emp_nom" vous pouvez utiliser la commande suivante pour récupérer une liste de tous les nom distincts:

```
SELECT DISTINCT emp_nom
FROM employes;
```

Cela renverra une liste de tous les nom différents.

Il est important de noter que la commande SELECT DISTINCT peut être plus lente que la commande SELECT normale, car elle nécessite que la base de données élimine les doublons dans la table. Par conséquent, elle doit parcourir l'ensemble de la table, ce qui peut prendre plus de temps si la table est très grande.

Pour afficher un enregistrement particulier dans SQL, vous pouvez utiliser la commande SELECT avec une clause WHERE pour spécifier les critères de recherche.

La syntaxe générale de la commande SELECT avec la clause WHERE est la suivante :

```
SELECT colonne1, colonne2, ...
FROM table
WHERE condition;
```

- "colonne1", "colonne2", etc. spécifient les colonnes de la table que vous voulez afficher.
- "table" spécifie la table à partir de laquelle vous voulez afficher les données.
- "condition" spécifie les critères de recherche pour identifier l'enregistrement que vous souhaitez afficher.

Par exemple, si vous avez une table "clients" avec les colonnes "nom", "prénom" et "email", vous pouvez utiliser la commande suivante pour afficher l'enregistrement d'un client en particulier en utilisant son adresse e-mail :

```
SELECT emp_nom
FROM employes
WHERE shop_id = 2;
```

Cela renverra l'enregistrement de la table "employes" qui correspond à l'id du magasin 2.

Il est important de noter que la condition de recherche doit être assez précise pour identifier un enregistrement unique, sinon la commande SELECT renverra tous les enregistrements correspondant à la condition de recherche.

Pour combiner plusieurs conditions d'affichage dans SQL, vous pouvez utiliser les opérateurs logiques tels que "AND", "OR" et "NOT". Vous pouvez les combiner pour créer des conditions plus complexes et afficher les enregistrements qui répondent à ces conditions.

La syntaxe générale pour combiner plusieurs conditions dans une commande SELECT est la suivante :

```
SELECT colonne1, colonne2, ...
FROM table
WHERE condition;
```

- "colonne1", "colonne2", etc. spécifient les colonnes de la table que vous voulez afficher.
- "table" spécifie la table à partir de laquelle vous voulez afficher les données.
- "condition1", "condition2", etc. spécifient les conditions que les enregistrements doivent remplir pour être affichés.
- "[opérateur_logique]" est un opérateur logique tel que "AND", "OR" ou "NOT" qui combine les conditions.

Par exemple, si vous avez une table "employes" avec les colonnes "emp_nom", "emp_prénom", "shop_id" vous pouvez utiliser la commande suivante pour afficher tous les employés qui travail dans un magasin spécifique et par rapport à un deuxième cristère.

```
SELECT emp_prenom
FROM employes
WHERE shop_id = 2 AND emp_nom = 'Marie';
```

Cela renverra tous les enregistrements de la table "clients" qui correspondent aux critères de recherche.

Vous pouvez également utiliser l'opérateur "OR" pour afficher les enregistrements qui répondent à l'une des deux conditions. Par exemple :

```
SELECT emp_prenom  
FROM employes  
WHERE shop_id = 2 OR emp_nom = 'Marie';
```

Cela renverra tous les enregistrements de la table "employes" qui travail dans le magasin qui a l'id 2 et qui a le nom 'Marie'.

Il est important de noter que les opérateurs logiques doivent être utilisés avec parcimonie pour éviter de créer des conditions trop complexes, qui peuvent être difficiles à gérer et à comprendre.

L'opérateur IN en SQL est utilisé pour spécifier une condition dans laquelle une colonne doit correspondre à l'une des valeurs spécifiées dans une liste.

La syntaxe de l'opérateur IN est la suivante :

```
SELECT colonne1, colonne2, ... FROM nom_table WHERE  
colonne_x IN (valeur_1, valeur_2, ..., valeur_n);
```

Dans cet exemple, la requête sélectionne les valeurs des colonnes colonne1, colonne2, ... de la table nom_table où la colonne_x correspond à l'une des valeurs spécifiées dans la liste de valeurs (valeur_1, valeur_2, ..., valeur_n).

Par exemple, si nous avons une table nommée "produits" avec une colonne "categorie" qui peut prendre les valeurs "fruits", "légumes", "viandes", "produits laitiers", etc., nous pouvons utiliser l'opérateur IN pour trouver tous les produits qui sont des fruits ou des légumes, comme suit :

```
SELECT nom_produit, categorie FROM produits WHERE  
categorie IN ('fruits', 'legumes');
```

Cette requête renverra tous les produits qui sont des fruits ou des légumes, et affichera leur nom et leur catégorie.

L'opérateur NOT IN en SQL est l'inverse de l'opérateur IN.

Il est utilisé pour spécifier une condition dans laquelle une colonne ne doit pas correspondre à l'une des valeurs spécifiées dans une liste.

La syntaxe de l'opérateur NOT IN est la suivante :

```
SELECT colonne1, colonne2, ... FROM nom_table WHERE  
colonne_x NOT IN (valeur_1, valeur_2, ..., valeur_n);
```

Dans cet exemple, la requête sélectionne les valeurs des colonnes colonne1, colonne2, ... de la table nom_table où la colonne_x ne correspond pas à l'une des valeurs spécifiées dans la liste de valeurs (valeur_1, valeur_2, ..., valeur_n).

Par exemple, si nous avons une table nommée "employes" avec une colonne "emp_nom" . nous pouvons utiliser l'opérateur NOT IN pour trouver tous les employés et exclure un employé par rapport à son nom.

```
SELECT emp_nom FROM employes WHERE emp_nom NOT IN  
( 'Peter' );
```

L'opérateur AS en SQL est utilisé pour donner un alias à une colonne ou à une table dans une requête SELECT.

Pour les colonnes, l'opérateur AS est utilisé pour renommer la colonne et rendre le résultat de la requête plus lisible. La syntaxe est la suivante :

```
SELECT nom_colonne AS nouvel_alias FROM nom_table;
```

Par exemple, si nous avons une table nommée "employes" avec les colonnes "emp_nom" et "emp_prenom", nous pouvons utiliser l'opérateur AS pour renommer la colonne "emp_nom" en "salaries" :

```
SELECT emp_nom AS 'salaries', emp_prenom FROM employes;
```

Cette requête renverra tous les noms des employés sous l'alias "Salaries", et les prénoms sous leur prenom d'origine.

Pour les tables, l'opérateur AS est utilisé pour donner un alias à une table dans une requête JOIN. La syntaxe est la suivante :

```
SELECT colonne1, colonne2, ... FROM nom_table AS  
nouvel_alias JOIN nom_table2 ON  
nom_table.colonne = nom_table2.colonne;
```

Par exemple, si nous avons deux tables nommées "shops" et "employés", et que nous voulons joindre ces tables pour afficher les noms des employés qui travail dans le magasin qui à l'id 1:

```
SELECT employes.emp_nom, employes.emp_prenom  
FROM employes  
JOIN shops ON employes.shop_id = shops.shop_id  
WHERE shops.shop_id = 1;
```

Dans cette requête, la clause "JOIN" relie les tables "employés" et "shops" en utilisant la colonne commune "shop_id". La condition de jointure est spécifiée dans la clause "ON".

Ensuite, la requête utilise la clause "WHERE" pour filtrer les données en ne renvoyant que les employés travaillant dans le magasin avec l'identifiant "1".

1. Créer une base de données "sport"
2. Créer une table "Equipes" avec un champ clé primaire equ_id et un champ varchar de 30 caractères "equ_ville"
3. Ajouter à "equ_ville" les valeurs "Paris", "Monaco", "Lyon".
4. Créer une autre table "joueurs" avec les champs suivant: un champ clé primaire "joueur_id", deux champs varchar "joueur_nom" , "joueur_prenom" et un champ init "equ_id".
5. Créer 3 joueurs à Lyon, 3 joueurs à Paris, 3 joueurs à Monaco en remplaçant les valeurs correspondantes. Chaque équipe doit avoir un joueur qui a un nom commun 'Pierre'
6. Réaliser la commande pour afficher seulement les joueurs de Lyon.
7. Réaliser la commande pour afficher tous les joueurs qui ont le nom "Pierre" de toutes les équipes confondues



laminutedecode.io

@LAMINUTEDECODE

