Partie 1 : Introduction et installation

Chapitre 1 : Introduction et présentation du programme

1.1 Présentation du programme de la formation

Bienvenue dans cette formation de deux jours sur SQL et MySQL ! Ce programme a été conçu pour vous aider à maîtriser les concepts fondamentaux et les compétences pratiques nécessaires pour travailler avec les bases de données relationnelles en utilisant SQL et MySQL.

Au cours de cette formation, vous apprendrez à créer, gérer et interroger des bases de données à l'aide de différentes techniques et outils.

Dans tous les cas, je vous conseiller d'aller sur le site sql.sh qui est une mine d'or pour apprendre le SQL : https://sql.sh/.

Chapitre 2 : Qu'est-ce qu'une base de données ?

2.1 Introduction aux bases de données

Une base de données est un système organisé de stockage d'informations qui permet d'effectuer des opérations pour créer, modifier, récupérer et supprimer des données.

Les bases de données relationnelles, comme SQL, sont basées sur le modèle relationnel, qui organise les données en tables composées de lignes et de colonnes. Chaque table représente une entité (par exemple, un client ou un produit) et chaque ligne représente un enregistrement (une instance de cette entité).

Les termes qui reviennent souvent :

- SGBD (Système de Gestion de Base de Données) : Un SGBD est un logiciel qui permet de gérer et d'organiser les bases de données. Il offre des fonctionnalités pour la création, la modification, la suppression et la récupération des données, ainsi que pour la gestion des utilisateurs, la sécurité, les sauvegardes, etc. Les exemples courants de SGBD sont MySQL, Oracle, Microsoft SQL Server et PostgreSQL.
- Langage de base de données : Un langage de base de données est un langage spécifique utilisé pour communiquer avec un SGBD et manipuler les données. SQL (Structured Query Language) est le langage de base de données le plus couramment utilisé. Il permet d'effectuer des opérations telles que la création et la modification de schémas de base de

données, l'interrogation des données, l'insertion, la mise à jour et la suppression de données, etc.

- Schéma de base de données : Un schéma de base de données définit la structure logique d'une base de données. Il spécifie les tables, les colonnes, les contraintes, les relations et les autres objets de la base de données. Le schéma de base de données décrit la manière dont les données sont organisées et les règles qui s'appliquent à ces données.
- Modèle de base de données : Un modèle de base de données est une représentation conceptuelle de la structure d'une base de données. Les modèles de base de données couramment utilisés sont le modèle relationnel, le modèle hiérarchique, le modèle en réseau et le modèle orienté objet. Le modèle relationnel est le plus répandu et utilise des tables, des clés primaires et des relations pour organiser les données.
- Requête SQL : Une requête SQL est une instruction utilisée pour interroger une base de données et récupérer des données spécifiques. Les requêtes SQL peuvent être simples, comme une sélection de toutes les lignes d'une table, ou complexes, impliquant des opérations de jointure, des conditions et des agrégations.
- Index : Un index est une structure de données utilisée pour accélérer la recherche des données dans une base de données. Il améliore les performances en créant un moyen rapide d'accéder aux enregistrements en fonction de certaines colonnes. Les index sont souvent créés sur des colonnes fréquemment utilisées dans les requêtes de recherche pour réduire le temps de recherche.
- Une clause en SQL est une instruction ou une partie spécifique d'une requête qui définit une action ou une condition particulière. Les clauses sont utilisées pour spécifier les opérations à effectuer lors de l'exécution d'une requête SQL, telles que la sélection, la filtration, le tri ou la jointure des données. Les clauses sont généralement combinées pour former une requête SQL complète. Chaque clause a un objectif spécifique et contribue à déterminer les résultats et le comportement de la requête.

Chapitre 3 : Introduction à SQL et MySQL

3.1 Qu'est-ce que SQL ?

SQL (Structured Query Language) est un langage de programmation standardisé utilisé pour communiquer avec des bases de données relationnelles. SQL permet d'effectuer des opérations telles que la création, la modification, la suppression et la récupération de données à partir de bases de données.

3.2 Qu'est-ce que MySQL ?

MySQL est un système de gestion de bases de données relationnelles (SGBD / RDBMS en anglais) open source et gratuit. Un SGBD est un logiciel qui permet a des utilisateurs d'interagir avec une base de données sans avoir à gérer de nombreux aspects complexes (par exemple, le stockage des données, la sécurité, les sauvegardes, etc.).

MySQL est l'un des SGBD les plus populaires au monde. Il est utilisé par de nombreuses entreprises et organisations, dont Facebook, Twitter, YouTube, Netflix, Airbnb, Uber, etc.

3.3 Qu'est-ce que phpMyAdmin ?

phpMyAdmin est un outil de gestion de bases de données MySQL basé sur le web. Il permet d'effectuer des opérations telles que la création, la modification, la suppression et la récupération de données à partir de bases de données MySQL.

PHPMyAdmin n'est pas un système de gestion de base de données (SGBD) à proprement parler. PHPMyAdmin est une application web basée sur PHP qui fournit une interface conviviale pour gérer et administrer des bases de données MySQL. Il agit en tant qu'outil de gestion pour interagir avec un SGBD spécifique, qui est MySQL dans ce cas.

MySQL est le SGBD réel qui gère et stocke les données dans une base de données relationnelle. PHPMyAdmin facilite l'interaction avec MySQL en fournissant une interface visuelle pour effectuer des opérations courantes telles que la création de tables, l'exécution de requêtes SQL, l'importation et l'exportation de données, la gestion des utilisateurs et des privilèges, etc.

Chapitre 4 : Installation de MySQL et d'un outil de gestion

4.1 Installation de MySQL

Pour installer MySQL sur votre ordinateur, suivez les instructions fournies par la documentation officielle : https://dev.mysql.com/doc/refman/8.0/en/installing.html

4.2 Installation d'un outil de gestion

Après avoir installé MySQL, il est recommandé d'installer un outil de gestion de bases de données pour faciliter le travail avec MySQL. Voici deux options populaires :

- MySQL Workbench : un outil graphique de gestion de bases de données MySQL. Pour l'installer, suivez les instructions sur le site officiel : https://www.mysql.com/products/workbench/

- phpMyAdmin : un outil web de gestion de bases de données MySQL. Pour l'installer, suivez les instructions sur le site officiel : https://www.phpmyadmin.net/

Pour nous faciliter la vie, nous allons utiliser un package de logiciels, comprenant MySQL et phpMyAdmin, qui s'appelle Xampp. Pour l'installer, suivez les instructions sur le site officiel : https://www.apachefriends.org/fr/index.html

Il permet d'installer d'un coup : Apache, MySQL, PHP et Perl.

Puis nous telechargerons Workbench de son coté.

Partie 2 : Les premières requêtes SQL de selection.

Chapitre 1 : Bases de SQL et syntaxe

1.1 Structure d'une requête SQL

Une requête SQL est une instruction qui permet d'interagir avec une base de données pour récupérer, insérer, mettre à jour ou supprimer des données. Les requêtes SQL sont composées de clauses, qui déterminent les actions à effectuer sur les données.

```sql
SELECT colonne1, colonne2
FROM nom_de_la_table
WHERE condition;
```

Ici, la requête est composé de trois clauses.

Par convention, les mots clés SQL sont écrits en majuscules. Cependant, il est possible d'écrire les mots clés en minuscules, mais cela peut rendre la requête plus difficile à lire.

De plus, il est conseillé de revenir à la ligne après chaque clause pour améliorer la lisibilité de la requête.

1.2 Les types de données SQL

Les types de données SQL sont utilisés pour définir le type de données que peut contenir une colonne. Voici quelques types de données courants :

- INT : un entier (nombre entier)
- FLOAT : un nombre à virgule flottante (nombre réel)
- VARCHAR(n) : une chaîne de caractères de longueur variable, où n est la longueur maximale
- DATE : une date (AAAA-MM-JJ)
- DATETIME : une date et heure (AAAA-MM-JJ HH:MM:SS)

Dans ce cours, on ne va pas construire nous même les bases de données, nous allons utiliser des bases de données déjà existantes. Nous n'avons donc pas d'intérêt a développer davantage les types de données.

1.3 Les opérateurs SQL

Les opérateurs SQL permettent de comparer, combiner ou effectuer des opérations sur les données. Voici quelques opérateurs courants :

- = : égal
- <> ou != : différent
- < : inférieur
- > : supérieur
- <= : inférieur ou égal
- >= : supérieur ou égal
- AND : combine deux conditions, toutes deux doivent être vraies
- OR : combine deux conditions, l'une d'elles doit être vraie
- NOT : inverse la condition

1.4 Les clauses SELECT, FROM et WHERE

- SELECT : spécifie les colonnes à récupérer dans la requête
- FROM : spécifie la table à partir de laquelle récupérer les données
- WHERE : spécifie une condition que les enregistrements doivent respecter pour être récupérés

```sql
SELECT nom, prenom
FROM clients
WHERE email = 'jean.dupont@email.com';
```

Chapitre 2 : Filtrage et tri des données

2.1 Utilisation de AND, OR et NOT

Les opérateurs AND, OR et NOT permettent de combiner ou de modifier des conditions dans une clause WHERE.

```
-- Exemple avec AND
SELECT nom, prenom
FROM clients
WHERE email = 'jean.dupont@email.com' AND prenom = 'Jean';

-- Exemple avec OR
SELECT nom, prenom
FROM clients
WHERE email = 'jean.dupont@email.com' OR email = 'marie.durand@email.com';

-- Exemple avec NOT
SELECT nom, prenom
FROM clients
WHERE NOT email = 'jean.dupont@email.com';
```

2.2 Opérateurs de comparaison

Les opérateurs de comparaison permettent de comparer les valeurs des colonnes aux valeurs spécifiées.

```
-- Exemple avec < (inférieur)
SELECT nom, prenom
FROM clients
WHERE age < 30;

-- Exemple avec >= (supérieur ou égal)
SELECT nom, prenom
FROM clients
WHERE age >= 18;
```

2.3 Fonctions d'agrégation (COUNT, SUM, AVG, MIN, MAX)

Les fonctions d'agrégation permettent de réaliser des calculs sur un ensemble de lignes. Voici quelques fonctions d'agrégation courantes :

- COUNT(): compte le nombre de lignes
- SUM(): calcule la somme des valeurs d'une colonne
- AVG(): calcule la moyenne des valeurs d'une colonne

- MIN(): trouve la valeur minimale d'une colonne
- MAX(): trouve la valeur maximale d'une colonne

```sql
-- Exemple avec COUNT()
SELECT COUNT(*) AS nombre_clients
FROM clients;

-- Exemple avec SUM()
SELECT SUM(salaire) AS total_salaires
FROM employes;

-- Exemple avec AVG()
SELECT AVG(salaire) AS salaire_moyen
FROM employes;

-- Exemple avec MIN()
SELECT MIN(salaire) AS salaire_min
FROM employes;

-- Exemple avec MAX()
SELECT MAX(salaire) AS salaire_max
FROM employes;
```

Attention ! Il faut pratiquer :)

> Dans le cadre du cours, il est conseillé de réaliser les premiers exercices pour mettre en pratique tout cela ! Voir : bdd-employees

---

2.4 Tri des données avec ORDER BY

La clause ORDER BY permet de trier les résultats d'une requête selon une ou plusieurs colonnes.

```sql
-- Trier les clients par nom (ordre croissant)
SELECT nom, prenom, email
FROM clients
ORDER BY nom ASC;

-- Trier les clients par nom (ordre décroissant)
SELECT nom, prenom, email
FROM clients
ORDER BY nom DESC;

-- Trier les clients par nom, puis par prénom
SELECT nom, prenom, email
FROM clients
ORDER BY nom ASC, prenom ASC;

-- Trier les clients par nom croissant, puis par prénom (ordre décroissant)
-- C'est étrange mais possible !
SELECT nom, prenom, email
FROM clients
ORDER BY nom ASC, prenom DESC;
```

2.5 Limitation des résultats avec LIMIT

La clause LIMIT permet de limiter le nombre de lignes renvoyées par une requête.

```sql
-- Récupérer les 10 premiers clients
SELECT nom, prenom, email
FROM clients
ORDER BY nom ASC
LIMIT 10;

-- Récupérer les clients 11 à 20
SELECT nom, prenom, email
FROM clients
ORDER BY nom ASC
LIMIT 10 OFFSET 10;
```

Chapitre 3 : Les requêtes de tri et de regroupement personnalisées.

3.1 Utilisation de LIKE et NOT LIKE

L'opérateur LIKE permet de rechercher des enregistrements qui correspondent à un modèle spécifique. Il est souvent utilisé avec le caractère générique %, qui correspond à n'importe quelle

chaîne de caractères.

```sql
-- Récupérer les clients dont le nom commence par 'D'
SELECT nom, prenom, email
FROM clients
WHERE nom LIKE 'D%';

-- Récupérer les clients dont le nom se termine par 'on'
SELECT nom, prenom, email
FROM clients
WHERE nom LIKE '%on';

-- Récupérer les clients dont le nom contient 'on'
SELECT nom, prenom, email
FROM clients
WHERE nom LIKE '%on%';

-- Récupérer les clients dont le nom ne contient pas 'on'
SELECT nom, prenom, email
FROM clients
WHERE nom NOT LIKE '%on%';
```

3.2 Utilisation de IN et NOT IN

L'opérateur IN permet de rechercher des enregistrements dont la valeur d'une colonne correspond à l'une des valeurs spécifiées.

```sql
-- Récupérer les clients dont le nom est 'Dupont' ou 'Durand'
SELECT nom, prenom, email
FROM clients
WHERE nom IN ('Dupont', 'Durand');

-- Récupérer les clients dont le nom n'est pas 'Dupont' ou 'Durand'
SELECT nom, prenom, email
FROM clients
WHERE nom NOT IN ('Dupont', 'Durand');
```

3.3 Utilisation de BETWEEN et NOT BETWEEN

L'opérateur BETWEEN permet de rechercher des enregistrements dont la valeur d'une colonne est comprise entre deux valeurs spécifiées.

```sql
-- Récupérer les clients dont l'âge est compris entre 20 et 30 ans
SELECT nom, prenom, email
FROM clients
WHERE age BETWEEN 20 AND 30;

-- Récupérer les clients dont l'âge n'est pas compris entre 20 et 30 ans
SELECT nom, prenom, email
FROM clients
WHERE age NOT BETWEEN 20 AND 30;
```

3.4 Utilisation de IS NULL et IS NOT NULL

L'opérateur IS NULL permet de rechercher des enregistrements dont la valeur d'une colonne est NULL.

```sql
-- Récupérer les clients dont l'adresse est NULL
SELECT nom, prenom, email
FROM clients
WHERE adresse IS NULL;

-- Récupérer les clients dont l'adresse n'est pas NULL
SELECT nom, prenom, email
FROM clients
WHERE adresse IS NOT NULL;
```

3.5 Utilisation de GROUP BY et HAVING

La clause GROUP BY permet de regrouper les enregistrements ayant la même valeur dans une colonne. La clause HAVING permet de filtrer les résultats d'une requête GROUP BY.

```sql
-- Récupérer le nombre de clients par ville
SELECT ville, COUNT(*) AS nombre_clients
FROM clients
GROUP BY ville;

-- Récupérer les villes ayant plus de 10 clients
SELECT ville, COUNT(*) AS nombre_clients
FROM clients
GROUP BY ville
HAVING COUNT(*) > 10;
```

3.6 Utilisation de DISTINCT

L'opérateur DISTINCT permet de supprimer les doublons des résultats d'une requête.

```
-- Récupérer les villes des clients
SELECT ville
FROM clients;

-- Récupérer les villes des clients (sans doublons)
SELECT DISTINCT ville
FROM clients;
```

3.7 Utilisation de AS

L'opérateur AS permet de renommer une colonne ou une table dans une requête.

```
-- Récupérer les villes des clients (avec un alias)
SELECT DISTINCT ville AS ville_client
FROM clients;
```

Dans cette session, nous avons couvert les bases de la syntaxe SQL, les opérateurs, les types de données, et les clauses pour filtrer, trier et limiter les résultats des requêtes. Dans la session suivante, nous aborderons des sujets plus avancés, tels que les manipulations de données et les jointures.

Partie 3 : Les requêtes de selection avancées

Chapitre 1 : Jointures entre tables

Les jointures permettent de combiner des données provenant de plusieurs tables. Il existe différents types de jointures, dont les plus courants sont INNER JOIN, LEFT JOIN, RIGHT JOIN et FULL OUTER JOIN.

1.1 INNER JOIN

L'INNER JOIN renvoie les enregistrements qui ont des correspondances dans les deux tables.

```sql
-- Récupérer les clients et leurs commandes
SELECT clients.nom, clients.prenom, commandes.date
FROM clients
INNER JOIN commandes ON clients.id = commandes.client_id;
```

1.2 LEFT JOIN

Le LEFT JOIN renvoie tous les enregistrements de la table de gauche, et les enregistrements correspondants de la table de droite. Si aucune correspondance n'est trouvée, les résultats de la table de droite sont NULL.

```sql
-- Récupérer tous les clients et leurs commandes éventuelles
SELECT clients.nom, clients.prenom, commandes.date
FROM clients
LEFT JOIN commandes ON clients.id = commandes.client_id;
```

1.3 RIGHT JOIN

Le RIGHT JOIN renvoie tous les enregistrements de la table de droite, et les enregistrements correspondants de la table de gauche. Si aucune correspondance n'est trouvée, les résultats de la table de gauche sont NULL.

```sql
-- Récupérer toutes les commandes et leurs clients éventuels
SELECT clients.nom, clients.prenom, commandes.date
FROM clients
RIGHT JOIN commandes ON clients.id = commandes.client_id;
```

1.4 FULL OUTER JOIN

Le FULL OUTER JOIN renvoie les enregistrements lorsqu'il y a une correspondance dans l'une des tables. Les résultats de la table de gauche ou de droite sont NULL s'il n'y a pas de correspondance.

```sql
-- Récupérer toutes les commandes et tous les clients, avec leurs correspondances
éventuelles
SELECT clients.nom, clients.prenom, commandes.date
FROM clients
FULL OUTER JOIN commandes ON clients.id = commandes.client_id;
```

Chapitre 2 : Sous-requêtes

2.1 Utilisation de sous-requêtes dans une clause WHERE

Une sous-requête est une requête imbriquée dans une autre requête. Elle peut être utilisée dans une clause WHERE pour filtrer les résultats d'une requête.

Cette solution existe mais il est préférable d'utiliser les jointures, qui sont plus performantes et surtout plus lisibles.

```sql
-- Récupérer les clients ayant passé au moins une commande
SELECT nom, prenom
FROM clients
WHERE id IN (
    SELECT client_id
    FROM commandes
);
```

2.2 Utilisation de sous-requêtes dans une clause FROM

Une sous-requête peut être utilisée dans une clause FROM pour créer une table temporaire à partir de laquelle récupérer les données.

```sql
-- Récupérer les clients ayant passé au moins une commande
SELECT clients.nom, clients.prenom, commandes.nombre
FROM (
    SELECT id, nom, prenom
    FROM clients
) AS clients
INNER JOIN (
    SELECT client_id, COUNT(*) AS nombre
    FROM commandes
    GROUP BY client_id
) AS commandes ON clients.id = commandes.client_id;
```

Chapitre 3 : Vues

3.1 Création d'une vue

Une vue est une table virtuelle qui est créée à partir d'une requête SELECT. Elle peut être utilisée comme une table normale dans les requêtes SELECT, INSERT, UPDATE et DELETE.

```sql
-- Créer une vue pour récupérer les clients ayant passé au moins une commande
CREATE VIEW clients_ayant_commande AS
SELECT clients.nom, clients.prenom, commandes.nombre
FROM (
    SELECT id, nom, prenom
    FROM clients
) AS clients
INNER JOIN (
    SELECT client_id, COUNT(*) AS nombre
    FROM commandes
    GROUP BY client_id
) AS commandes ON clients.id = commandes.client_id;
```

3.2 Utilisation d'une vue

Une vue peut être utilisée comme une table normale dans les requêtes SELECT, INSERT, UPDATE et DELETE.

```sql
-- Récupérer les clients ayant passé au moins une commande
SELECT nom, prenom, nombre
FROM clients_ayant_commande;
```

3.3 Suppression d'une vue

Une vue peut être supprimée avec la commande DROP VIEW.

```sql
-- Supprimer la vue
DROP VIEW clients_ayant_commande;
```

3.4 Modification d'une vue

Une vue peut être modifiée avec la commande ALTER VIEW.

```sql
-- Modifier la vue pour inclure le montant total des commandes
CREATE OR REPLACE VIEW clients_avec_commandes AS
SELECT clients.*, COUNT(commandes.id) as nombre_commandes, SUM(commandes.montant) as
montant_total
FROM clients
LEFT JOIN commandes ON clients.id = commandes.client_id
GROUP BY clients.id;
```

Chapitre 4 : Procédures stockées

4.1 Création d'une procédure stockée

Les procédures stockées sont des routines qui sont enregistrées dans la base de données et peuvent être exécutées avec des paramètres.

Pour créer une procédure stockée, utilisez la commande CREATE PROCEDURE.

```
-- Créer une procédure stockée pour récupérer les clients ayant passé au moins une commande
CREATE PROCEDURE clients_ayant_commande()
BEGIN
    SELECT clients.nom, clients.prenom, commandes.nombre
    FROM (
        SELECT id, nom, prenom
        FROM clients
    ) AS clients
    INNER JOIN (
        SELECT client_id, COUNT(*) AS nombre
        FROM commandes
        GROUP BY client_id
    ) AS commandes ON clients.id = commandes.client_id;
END;
```

4.2 Utilisation d'une procédure stockée

Une procédure stockée peut être exécutée avec la commande CALL.

```
-- Exécuter la procédure stockée
CALL clients_ayant_commande();
```

4.3 Suppression d'une procédure stockée

Une procédure stockée peut être supprimée avec la commande DROP PROCEDURE.

```
-- Supprimer la procédure stockée
DROP PROCEDURE clients_ayant_commande;
```

4.4 Modification d'une procédure stockée

Une procédure stockée peut être modifiée avec la commande ALTER PROCEDURE.

```
-- Modifier la procédure stockée pour inclure le montant total des commandes
CREATE OR REPLACE PROCEDURE clients_avec_commandes()
BEGIN
    SELECT clients.*, COUNT(commandes.id) as nombre_commandes, SUM(commandes.montant)
as montant_total
    FROM clients
    LEFT JOIN commandes ON clients.id = commandes.client_id
    GROUP BY clients.id;
END;
```

Dans cette partie, nous avons comment utiliser les jointures, les sous-requêtes, les vues et les procédures stockées pour interroger des données. Dans la partie suivante, nous aborderons les requêtes de modification de données, telles que l'insertion, la mise à jour et la suppression de données.

Partie 4 : Les requêtes de modification de données

Chapitre 1 : Manipulation des données

1.1 Insertion de données avec INSERT INTO

La commande INSERT INTO permet d'ajouter de nouveaux enregistrements à une table.

```
-- Insérer un client dans la table clients
INSERT INTO clients (nom, prenom, email)
VALUES ('Martin', 'Pierre', 'pierre.martin@email.com');
```

1.2 Mise à jour des données avec UPDATE

La commande UPDATE permet de modifier des données existantes dans une table.

```
-- Mettre à jour l'adresse e-mail d'un client
UPDATE clients
SET email = 'pierre.martin@nouveau-email.com'
WHERE id = 1;
```

1.3 Suppression des données avec DELETE

La commande DELETE permet de supprimer des enregistrements d'une table.

```sql
-- Supprimer un client de la table clients
DELETE FROM clients
WHERE id = 1;
```