SQL

Antoine OBLIGIS

Qu’est-ce que le SQL ?

Antoine OBLIGIS

Cours SQL : Requêtes fondamentales

Introduction

SQL (Structured Query Language) est un langage utilisé pour manipuler des bases de données relationnelles. Nous allons étudier plusieurs types de requêtes avec des exemples spécifiques.

Antoine OBLIGIS

1. Lecture de données simples avec SELECT

- **SELECT * FROM invoice;**
  Récupère toutes les colonnes et toutes les lignes de la table invoice.
- **SELECT * FROM article;**
  Récupère toutes les colonnes et toutes les lignes de la table article.
- **SELECT invoice_id, total FROM invoice;**
  Sélectionne uniquement les colonnes invoice_id et total de la table invoice.
- **SELECT article_name, unit_price FROM article;**
  Récupère les colonnes article_name et unit_price de la table article.

2. Élimination des doublons avec DISTINCT

- **SELECT DISTINCT invoice_id FROM invoice_item;**
  Récupère uniquement les valeurs uniques de invoice_id dans invoice_item.
- **SELECT DISTINCT article_id FROM invoice_item;**
  Récupère uniquement les valeurs uniques de article_id dans invoice_item.

3. Limiter les résultats avec LIMIT

- **SELECT * FROM invoice LIMIT 5;**
  Affiche uniquement les 5 premières lignes de la table invoice.

4. Pagination des résultats avec OFFSET

La clause `OFFSET` est utilisée pour ignorer un certain nombre de lignes au début du résultat d'une requête. Elle est souvent utilisée en combinaison avec `LIMIT` pour paginer les résultats.

- **SELECT * FROM invoice LIMIT 5 OFFSET 5;**
  Affiche uniquement les 5 lignes suivante des 5 premières lignes de la table invoice.

COUNT, SUM, AVG, MIN, MAX

Antoine OBLIGIS

4. Fonctions d’agrégation

- **SELECT COUNT(*) FROM invoice;**
  Compte le nombre total de lignes dans la table invoice.
- **SELECT SUM(total) FROM invoice;**
  Calcule la somme totale de la colonne total.
- **SELECT AVG(total) FROM invoice;**
  Calcule la moyenne des valeurs dans la colonne total.
- **SELECT MIN(total) FROM invoice;**
  Trouve la plus petite valeur dans la colonne total.
- **SELECT MAX(total) FROM invoice;**
  Trouve la plus grande valeur dans la colonne total.
-

=, >, <, >=, <=, <>,

BETWEEN ... AND ...

IN, LIKE, NOT IN, NOT LIKE,

AND, OR

IS NULL, IS NOT NULL

5. Conditions avec WHERE

- **SELECT * FROM invoice WHERE invoice_id = 160025;**
  Récupère les factures ayant l'identifiant 160025.
- **SELECT * FROM invoice WHERE total >= 10;**
  Récupère les factures dont le total est supérieur ou égal à 10.
- **SELECT * FROM invoice WHERE time BETWEEN '9:00' AND '9:30';**
  Récupère les factures générées entre 9h00 et 9h30.
- **SELECT * FROM article WHERE article_id IN (12, 13, 14, 15);**
  Récupère les articles ayant un article_id dans la liste donnée.
- **SELECT * FROM article WHERE article_name LIKE 'BAGUETTE%';**
  Récupère les articles dont le nom commence par "BAGUETTE".
- **SELECT * FROM article WHERE article_name NOT LIKE 'BAGUETTE%';**
  Récupère les articles dont le nom ne commence pas par "BAGUETTE".
- **SELECT * FROM invoice WHERE time BETWEEN '9:00' AND '9:30' AND total > 5;**
  Récupère les factures générées entre 9h00 et 9h30, et dont le total est supérieur à 5.
- Requête imbriquée :
  SELECT * FROM invoice_item WHERE article_id = (SELECT article_id FROM article WHERE article_name = 'CROISSANT');
  Récupère les lignes de invoice_item associées à l'article nommé "CROISSANT".
-

6. Trier les données avec ORDER BY

- **SELECT * FROM invoice ORDER BY total;**
  Trie les factures par total en ordre croissant.
- **SELECT * FROM invoice ORDER BY total DESC;**
  Trie les factures par total en ordre décroissant.

7. Groupement avec GROUP BY

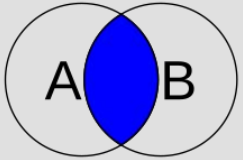
- **SELECT article_id, SUM(quantity) AS quantite_vendue_par_article FROM invoice_item GROUP BY article_id;**
  Calcule le total des quantités vendues pour chaque article_id.
- **SELECT article_id, SUM(quantity) AS quantite_vendue_par_article FROM invoice_item GROUP BY article_id HAVING quantite_vendue_par_article >= 5;**
  Même requête, mais affiche uniquement les articles avec une quantité vendue supérieure ou égale à 5.

Les jointures

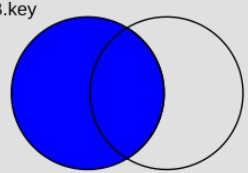
Antoine OBLIGIS

SQL JOINS

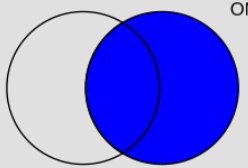
SELECT <fields>
FROM TableA A
INNER JOIN TableB B
ON A.key = B.key



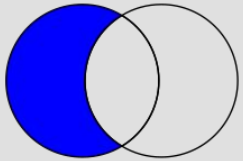
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key



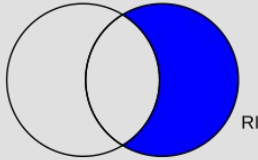
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key



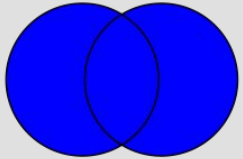
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key
WHERE B.key IS NULL



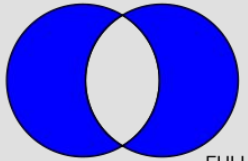
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL



SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key



SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL



8. Les jointures

- **SELECT * FROM invoice_item INNER JOIN article ON invoice_item.article_id = article.article_id;**
  Récupère les lignes communes aux tables invoice_item et article.
- **SELECT * FROM invoice_item LEFT JOIN article ON invoice_item.article_id = article.article_id;**
  Récupère toutes les lignes de invoice_item et complète avec article si possible.
- **SELECT * FROM invoice_item RIGHT JOIN article ON invoice_item.article_id = article.article_id;**
  Récupère toutes les lignes de article et complète avec invoice_item si possible.
- Requête jointe avec tri :
  SELECT item.invoice_id, a.article_name, item.quantity FROM invoice_item AS item INNER JOIN article AS a ON item.article_id = a.article_id ORDER BY item.invoice_id;
  Associe chaque facture avec le nom de l'article et la quantité, triées par invoice_id.
- Requête jointe complète :
  SELECT i.invoice_id, i.date, i.time, a.article_name, a.unit_price, item.quantity, i.total FROM invoice_item AS item INNER JOIN invoice AS i ON item.invoice_id = i.invoice_id INNER JOIN article AS a ON item.article_id = a.article_id ORDER BY i.invoice_id;
  Affiche les détails complets des factures, articles, et quantités vendues.

Explorer les données

- Lister tous les articles disponibles avec leurs prix.
- Afficher les factures ayant un total supérieur à 10 €.

Effectuer des calculs

- Trouver le montant total des ventes pour la journée.
- Identifier l'article le plus vendu.

Utiliser des jointures

- Associer chaque facture à ses articles et quantités correspondants.
- Lister tous les articles qui n'ont pas encore été vendus.

Créer des statistiques

- Calculer la quantité totale vendue pour chaque article.
- Trouver l’heure où les ventes sont les plus importantes.

Requêtes imbriquées

- Trouver toutes les factures contenant au moins un "CROISSANT".
- Lister les factures avec plus de 5 articles au total.

Antoine OBLIGIS

Explorer les données

- SELECT article_name, unit_price FROM article;
- SELECT * FROM invoice WHERE total > 10;

Effectuer des calculs

- SELECT SUM(total) AS montant_total FROM invoice;
- SELECT article_id, SUM(quantity) AS total_vendu
  FROM invoice_item
  GROUP BY article_id
  ORDER BY total_vendu DESC
  LIMIT 1;

Utiliser des jointures

- SELECT i.invoice_id, a.article_name, item.quantity
  FROM invoice_item AS item
  INNER JOIN article AS a ON item.article_id = a.article_id
  INNER JOIN invoice AS i ON item.invoice_id = i.invoice_id;
- SELECT article_name
  FROM article
  WHERE article_id NOT IN (
  SELECT DISTINCT article_id FROM invoice_item);

Antoine OBLIGIS

Créer des statistiques

- ```
  SELECT a.article_name, SUM(item.quantity) AS total_vendu
  FROM article AS a
  INNER JOIN invoice_item AS item ON a.article_id = item.article_id
  GROUP BY a.article_name;
  ```
- ```
  SELECT time, SUM(total) AS montant_total
  FROM invoice
  GROUP BY time
  ORDER BY montant_total DESC
  LIMIT 1;
  ```

Requêtes imbriquées

- ```
  SELECT DISTINCT i.invoice_id
  FROM invoice AS i
  INNER JOIN invoice_item AS item ON i.invoice_id = item.invoice_id
  WHERE item.article_id = (
      SELECT article_id FROM article WHERE article_name = 'CROISSANT');
  ```
- ```
  SELECT i.invoice_id, SUM(item.quantity) AS total_articles
  FROM invoice AS i
  INNER JOIN invoice_item AS item ON i.invoice_id = item.invoice_id
  GROUP BY i.invoice_id
  HAVING total_articles > 5;
  ```

6. Requêtes avancées avec filtres

- Trouver toutes les factures contenant à la fois un "CROISSANT" ou une "BAGUETTE"
- Lister les articles dont le nom contient "PAIN"

7. Requêtes sur les ventes

- Identifier les articles ayant généré le plus de revenus (prix * quantité)
- Trouver les factures égale à 3 types d'articles différents ont été achetés

8. Analyse des horaires de vente

- Identifier l'heure où le montant total des ventes a été le plus élevé
- Trouver le nombre de factures émises par tranche horaire de 30 minutes (CONCAT)

9. Requêtes avec sous-requêtes et conditions

- Lister les articles vendus uniquement une fois (quantité totale = 1)
- Trouver les factures ayant au moins un article vendu plus de 5 fois

10. Requêtes avec jointures multiples

- Lister toutes les factures avec leurs articles et le montant total par article
- Comparer les factures ayant le même montant total

Antoine OBLIGIS

6. Requêtes avancées avec filtres

- ```
  SELECT item.invoice_id
  FROM invoice_item AS item
  WHERE item.article_id IN (
      SELECT article_id FROM article WHERE article_name IN ('CROISSANT', 'BAGUETTE')
  )
  GROUP BY item.invoice_id
  HAVING COUNT(DISTINCT item.article_id) = 2;
  ```
- ```
  SELECT article_name
  FROM article
  WHERE article_name LIKE '%PAIN%';
  ```

7. Requêtes sur les ventes

- SELECT a.article_name, SUM(item.quantity * a.unit_price) AS revenu_total FROM article AS a INNER JOIN invoice_item AS item ON a.article_id = item.article_id GROUP BY a.article_name ORDER BY revenu_total DESC;
- SELECT i.invoice_id, COUNT(DISTINCT item.article_id) AS types_articles FROM invoice AS i INNER JOIN invoice_item AS item ON i.invoice_id = item.invoice_id GROUP BY i.invoice_id HAVING types_articles > 3;

Antoine OBLIGIS

8. Analyse des horaires de vente

- SELECT time, SUM(total) AS montant_total FROM invoice GROUP BY time ORDER BY montant_total DESC LIMIT 1;
- SELECT CONCAT(HOUR(time), ':', IF(MINUTE(time) < 30, '00-29', '30-59')) AS tranche_horaire, COUNT(*) AS nombre_factures FROM invoice GROUP BY tranche_horaire;

9. Requêtes avec sous-requêtes et conditions

- SELECT a.article_name FROM article AS a INNER JOIN invoice_item AS item ON a.article_id = item.article_id GROUP BY a.article_name HAVING SUM(item.quantity) = 1;
- SELECT DISTINCT i.invoice_id FROM invoice_item AS item INNER JOIN invoice AS i ON item.invoice_id = i.invoice_id WHERE item.quantity > 5;

10. Requêtes avec jointures multiples

- SELECT i.invoice_id, a.article_name, item.quantity, (item.quantity * a.unit_price) AS montant_article FROM invoice AS i INNER JOIN invoice_item AS item ON i.invoice_id = item.invoice_id INNER JOIN article AS a ON item.article_id = a.article_id ORDER BY i.invoice_id;
- SELECT i1.invoice_id AS facture_1, i2.invoice_id AS facture_2, i1.total FROM invoice AS i1 INNER JOIN invoice AS i2 ON i1.total = i2.total AND i1.invoice_id < i2.invoice_id;

Antoine OBLIGIS

Les Fonctions sur les chaînes de caractères

Antoine OBLIGIS

Les fonctions de chaînes permettent de manipuler, analyser ou transformer les données textuelles dans SQL.

Fonction	Description	Exemple
LENGTH	Retourne la longueur d'une chaîne.	`SELECT LENGTH('SQL')` → 3
CHAR_LENGTH	Nombre de caractères dans une chaîne (différent de LENGTH pour Unicode).	`SELECT CHAR_LENGTH('é')` → 1
UPPER	Convertit une chaîne en majuscules.	`SELECT UPPER('sql')` → 'SQL'
LOWER	Convertit une chaîne en minuscules.	`SELECT LOWER('SQL')` → 'sql'
SUBSTRING / **SUBSTR**	Extrait une partie d'une chaîne.	`SELECT SUBSTRING('SQL Tutorial', 5, 8)` → 'Tutorial'
CONCAT	Concatène deux ou plusieurs chaînes.	`SELECT CONCAT('SQL', ' ', 'Tutorial')` → 'SQL Tutorial'
CONCAT_WS	Concatène des chaînes avec un séparateur.	`SELECT CONCAT_WS('-', '2025', '01', '13')` → '2025-01-13'
TRIM	Supprime les espaces ou caractères spécifiques en début/fin de chaîne.	`SELECT TRIM(' SQL ')` → 'SQL'

Les fonctions de chaînes permettent de manipuler, analyser ou transformer les données textuelles dans SQL.

Fonction	Description	Exemple
LTRIM	Supprime les espaces ou caractères en début de chaîne.	`SELECT LTRIM('  SQL')` → 'SQL'
RTRIM	Supprime les espaces ou caractères en fin de chaîne.	`SELECT RTRIM('SQL  ')` → 'SQL'
REPLACE	Remplace une sous-chaîne par une autre.	`SELECT REPLACE('SQL', 'S', 'T')` → 'TQL'
INSTR	Trouve la position d'une sous-chaîne dans une chaîne.	`SELECT INSTR('SQL Tutorial', 'Tutorial')` → 5
LOCATE	Identique à `INSTR`.	`SELECT LOCATE('T', 'SQL Tutorial')` → 5
LEFT	Retourne les premiers caractères d'une chaîne.	`SELECT LEFT('SQL', 2)` → 'SQ'
RIGHT	Retourne les derniers caractères d'une chaîne.	`SELECT RIGHT('SQL', 2)` → 'QL'
REVERSE	Inverse une chaîne de caractères.	`SELECT REVERSE('SQL')` → 'LQS'

Les Dates

Antoine OBLIGIS

Ces fonctions permettent de manipuler, analyser et formater les données de type `DATE`, `TIME`, `DATETIME`, etc.

Fonction	Description	Exemple
`CURDATE`	Retourne la date actuelle (AAAA-MM-JJ).	`SELECT CURDATE()` → '2025-01-13'
`CURRENT_DATE`	Synonyme de `CURDATE`.	`SELECT CURRENT_DATE()` → '2025-01-13'
`CURTIME`	Retourne l'heure actuelle (HH:MM:SS).	`SELECT CURTIME()` → '10:45:30'
`CURRENT_TIME`	Synonyme de `CURTIME`.	`SELECT CURRENT_TIME()` → '10:45:30'
`NOW`	Retourne la date et l'heure actuelles (AAAA-MM-JJ HH:MM:SS).	`SELECT NOW()` → '2025-01-13 10:45:30'
`DAY`	Retourne le jour d'une date (1-31).	`SELECT DAY('2025-01-13')` → 13
`MONTH`	Retourne le mois d'une date (1-12).	`SELECT MONTH('2025-01-13')` → 1
`YEAR`	Retourne l'année d'une date.	`SELECT YEAR('2025-01-13')` → 2025
`HOUR`	Retourne l'heure d'un datetime/heure.	`SELECT HOUR('10:45:30')` → 10

Fonction	Description	Exemple
MINUTE	Retourne les minutes d'un datetime/heure.	`SELECT MINUTE('10:45:30')` → 45
SECOND	Retourne les secondes d'un datetime/heure.	`SELECT SECOND('10:45:30')` → 30
DATE_FORMAT	Formate une date selon un modèle.	`SELECT DATE_FORMAT('2025-01-13', '%d/%m/%Y')` → '13/01/2025'
DATEDIFF	Retourne la différence en jours entre deux dates.	`SELECT DATEDIFF('2025-01-15', '2025-01-13')` → 2
TIMEDIFF	Retourne la différence entre deux heures.	`SELECT TIMEDIFF('10:45:30', '08:30:00')` → '02:15:30'
ADDDATE	Ajoute un intervalle à une date.	`SELECT ADDDATE('2025-01-13', INTERVAL 5 DAY)` → '2025-01-18'
SUBDATE	Soustrait un intervalle à une date.	`SELECT SUBDATE('2025-01-13', INTERVAL 5 DAY)` → '2025-01-08'
LAST_DAY	Retourne le dernier jour du mois d'une date.	`SELECT LAST_DAY('2025-01-13')` → '2025-01-31'
EXTRACT	Extrait une partie d'une date (année, mois, jour, heure, etc.).	`SELECT EXTRACT(YEAR FROM '2025-01-13')` → 2025

Les types

Antoine OBLIGIS

1. Types numériques

Entiers

- **TINYINT** : Entier très petit (1 octet).
- **SMALLINT** : Petit entier (2 octets).
- **MEDIUMINT** : Entier moyen (3 octets).
- **INT** / **INTEGER** : Entier standard (4 octets).
- **BIGINT** : Grand entier (8 octets).

Décimaux et réels

- **DECIMAL(p, s)** / **NUMERIC(p, s)** : Nombre à virgule fixe avec précision (précision p et échelle s).
- **FLOAT(p)** : Nombre à virgule flottante avec précision simple (approximation).
- **DOUBLE** / **DOUBLE PRÉCISION** : Nombre à virgule flottante avec précision double.

2. Types de chaînes de caractères

Chaînes de longueur fixe

- **CHAR(n)** : Chaîne de longueur fixe (taille max : n caractères).

Chaînes de longueur variable

- **VARCHAR(n)** : Chaîne de longueur variable (taille max : n caractères).
- **TEXT** : Chaîne de grande taille (MySQL). Variantes :
  - **TINYTEXT** : Chaîne très petite (255 caractères max).
  - **TEXT** : Chaîne de taille moyenne (65 535 caractères max).
  - **MEDIUMTEXT** : Chaîne plus grande (16 Mo).
  - **LONGTEXT** : Chaîne très grande (4 Go).

Chaînes binaires

- **BINARY(n)** : Chaîne binaire de longueur fixe.
- **VARBINARY(n)** : Chaîne binaire de longueur variable.
- **BLOB** : Objet binaire de grande taille (MySQL). Variantes :
  - **TINYBLOB** : BLOB très petit.
  - **BLOB** : BLOB moyen.
  - **MEDIUMBLOB** : BLOB plus grand.
  - **LONGBLOB** : BLOB très grand.
-

3. Types de dates et d’heures

- **DATE** : Date (AAAA-MM-JJ).
- **TIME** : Heure (HH:MM:SS).
- **DATETIME** : Date et heure combinées (AAAA-MM-JJ HH:MM:SS).
- **TIMESTAMP** : Date et heure avec fuseau horaire (enregistrements relatifs au système).
- **YEAR** : Année (AAAA ou AA).

4. Types booléens et logiques

- **BOOLEAN** / **BOOL** : Vrai/Faux (souvent stocké comme `TINYINT` dans MySQL).
- **BIT(n)** : Champ binaire avec n bits.

5. Types géographiques

- **GEOMETRY** : Objet géographique générique.
- **POINT** : Point géographique.
- **LINESTRING** : Ligne.
- **POLYGON** : Polygone.
- **GEOMETRYCOLLECTION** : Collection de formes géométriques.

Antoine OBLIGIS

6. Types JSON et XML

- **JSON** : Stockage de données au format JSON.
- **XML** : Stockage de données au format XML.

7. Types spécialisés (selon les SGBDR)

PostgreSQL

- **SERIAL** / **BIGSERIAL** : Auto-incrémentation pour les entiers.
- **UUID** : Identifiant universel unique.
- **ARRAY** : Tableaux (exemple : `INT[]` pour un tableau d'entiers).
- **JSONB** : Format JSON binaire optimisé.

SQL Server

- **UNIQUEIDENTIFIER** : Identifiant global unique (GUID).
- **NTEXT** : Texte Unicode (obsolète, remplacé par **NVARCHAR(MAX)**).
- **MONEY** / **SMALLMONEY** : Valeurs monétaires.

Oracle

- **NUMBER(p, s)** : Nombre à virgule fixe (équivalent à **DECIMAL**).
- **CLOB** : Chaîne de grande taille (Character Large Object).
- **NCLOB** : CLOB en Unicode.
- **BFILE** : Référence à un fichier externe.

Antoine OBLIGIS

8. Types personnalisés

Certains SGBDR permettent de définir des types personnalisés :

- PostgreSQL : **CREATE TYPE** pour des énumérations ou structures complexes.
- MySQL : **ENUM** et **SET** pour des ensembles de valeurs prédéfinies.

Résumé

Catégorie	Exemples courants
Numériques	INT, BIGINT, DECIMAL, FLOAT
Chaînes	CHAR, VARCHAR, TEXT, BLOB
Dates/Heures	DATE, TIME, DATETIME
Booléens	BOOLEAN, BIT
Géographiques	POINT, POLYGON
JSON/XML	JSON, XML

CRUD

Antoine OBLIGIS

CREATE : Créer une table

La commande CREATE TABLE permet de créer une nouvelle table dans la base de données.

```
CREATE TABLE nom_table(
colonne1 TYPE1 [CONTRAINTES],
colonne2 TYPE2 [CONTRAINTES],
...
);
```

id : Identifiant unique (clé primaire).
nom : Nom de l'utilisateur (50 caractères maximum).
email : Adresse email (unique dans la table).
date_inscription : Date de l'inscription.

Créer une table `utilisateurs` pour stocker les informations des utilisateurs :

```
CREATE TABLE utilisateurs (
id INT AUTO_INCREMENT PRIMARY KEY,
nom VARCHAR(50) NOT NULL,
email VARCHAR(100) NOT NULL UNIQUE,
date_inscription DATE
);
```

Antoine OBLIGIS

INSERT INTO : Ajouter des données

La commande `INSERT INTO` permet d'insérer de nouvelles lignes dans une table.

```
INSERT INTO nom_table
(colonne1, colonne2, ...) VALUES
(valeur1, valeur2, ...);
```

Ajouter un utilisateur dans la table `utilisateurs` :

```
INSERT INTO utilisateurs (nom, email, date_inscription)
VALUES ('Jean Dupont', 'jean.dupont@example.com', '2025-01-01');
```

Insère un utilisateur nommé "Jean Dupont" avec son email et sa date d'inscription.

```
INSERT INTO utilisateurs (nom, email, date_inscription)
VALUES
('Marie Curie', 'marie.curie@example.com', '2025-01-02'),
('Albert Einstein', 'albert.einstein@example.com', '2025-01-03');
```

UPDATE : Modifier des données

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

```
UPDATE nom_table SET
colonne1 = nouvelle_valeur1,
colonne2 = nouvelle_valeur2, ...
WHERE condition;
```

Modifier plusieurs colonnes

```
UPDATE utilisateurs SET
nom = 'Jean Dupont-Maj',
email = 'jean.dupont.maj@example.com'
WHERE id = 1;
```

Modifier l'email d'un utilisateur avec l'`id = 1` :

```
UPDATE utilisateurs SET
email = 'nouvel.email@example.com'
WHERE id = 1;
```

Attention : Toujours utiliser une clause `WHERE` pour éviter de modifier toutes les lignes de la table.

DELETE : Supprimer des données

La commande DELETE permet de supprimer des lignes dans une table.

```
DELETE FROM nom_table
WHERE condition;
```

Supprimer l'utilisateur avec l'`id = 2` :

```
DELETE FROM utilisateurs
WHERE id = 2;
```

Supprimer toutes les données d'une table :

```
DELETE FROM utilisateurs;
```

Antoine OBLIGIS

Résumé

Commande	Action
CREATE TABLE	Créer une nouvelle table.
INSERT INTO	Ajouter de nouvelles lignes dans une table.
UPDATE	Modifier des données existantes dans une table.
DELETE	Supprimer des lignes d'une table.

Exercice pratique

1. Créer une table produits avec les colonnes suivantes :
    - id : Identifiant unique (entier, clé primaire, auto-increment).
    - nom : Nom du produit (chaîne de caractères).
    - prix : Prix du produit (décimal).
2. Insère les produits suivants :
    - Nom : "Produit A", Prix : 10.99
    - Nom : "Produit B", Prix : 15.49
    - Nom : "Produit C", Prix : 7.99
3. Mets à jour le prix du "Produit A" à 12.99.
4. Supprime le produit "Produit C".

Correction

1. Créer une table produits

```
CREATE TABLE produits (
 id INT AUTO_INCREMENT PRIMARY KEY,
nom VARCHAR(100) NOT NULL,
prix DECIMAL(10, 2) NOT NULL );
```

2. Insérer les produits

```
INSERT INTO produits (nom, prix)
VALUES
('Produit A', 10.99),
('Produit B', 15.49),
('Produit C', 7.99);
```

3. Mettre à jour le prix du "Produit A"

```
UPDATE produits SET prix = 12.99
WHERE nom = 'Produit A';
```

4. Supprimer le produit "Produit C"

```
DELETE FROM produits WHERE nom = 'Produit C';
```

Vérification des données

```
SELECT * FROM produits;
```

Antoine OBLIGIS

DROP : Supprimer une table ou une base de données

1.1. Supprimer une table

```
DROP TABLE nom_table;
DROP TABLE produits;
```

1.2. Supprimer une base de données

```
DROP DATABASE nom_base;
DROP DATABASE magasin;
```

1.3. Supprimer une colonne dans une table

```
ALTER TABLE nom_table DROP
COLUMN nom_colonne;

ALTER TABLE produits
DROP COLUMN description;
```

Foreign Key : Clé étrangère

La clé étrangère est définie lors de la création d'une table ou via une commande `ALTER TABLE`.

```
CREATE TABLE table_enfant (
colonne1 TYPE1,
colonne2 TYPE2,
colonne_foreign TYPE3,
FOREIGN KEY (colonne_foreign) REFERENCES table_parent(colonne_clé_prim) );
```

Créer une relation entre les tables `commandes` et `clients` :

```
CREATE TABLE clients (
client_id INT AUTO_INCREMENT PRIMARY KEY,
nom VARCHAR(100) NOT NULL );
CREATE TABLE commandes ( commande_id INT AUTO_INCREMENT PRIMARY KEY,
client_id INT,
montant DECIMAL(10, 2),
FOREIGN KEY (client_id) REFERENCES clients(client_id) );
```

Antoine OBLIGIS

Si la table est déjà créée, on peut ajouter une clé étrangère avec `ALTER TABLE`.

```
ALTER TABLE table_enfant
ADD CONSTRAINT nom_contrainte FOREIGN KEY (colonne_foreign)
REFERENCES table_parent(colonne_clé_prim);
```

Ajouter une clé étrangère à une table `commandes` existante :

```
ALTER TABLE commandes
ADD CONSTRAINT fk_client_commande FOREIGN KEY (client_id)
REFERENCES clients(client_id);
```

Pour supprimer une clé étrangère, on doit d'abord connaître le nom de la contrainte.

```
ALTER TABLE table_enfant
DROP FOREIGN KEY nom_contrainte;
```

Supprimer une clé étrangère nommée `fk_client_commande` :

```
ALTER TABLE commandes
 DROP FOREIGN KEY fk_client_commande;
```

Si une catégorie est supprimée, les produits associés doivent aussi être supprimés :

```
CREATE TABLE produits (
produit_id INT AUTO_INCREMENT PRIMARY KEY,
nom VARCHAR(100) NOT NULL,
categorie_id INT, FOREIGN KEY (categorie_id) REFERENCES
categories(categorie_id) ON DELETE CASCADE );
```

Si une catégorie est mise à jour, les produits associés doivent suivre la modification :

```
FOREIGN KEY (categorie_id) REFERENCES categories(categorie_id) ON UPDATE CASCADE
```

Commande	Action
DROP TABLE	Supprime une table avec ses données.
DROP DATABASE	Supprime une base de données entière.
ALTER TABLE DROP COLUMN	Supprime une colonne d'une table.
FOREIGN KEY	Définit une relation entre deux tables.
ON DELETE CASCADE	Supprime les lignes liées lorsqu'une ligne parent est supprimée.
ON UPDATE CASCADE	Met à jour les lignes liées lors de la modification de la ligne parent.

INDEX

Antoine OBLIGIS

Les index en SQL améliorent la performance des requêtes en facilitant la recherche des données dans les tables. Cependant, ils augmentent les besoins en espace de stockage et peuvent ralentir les opérations d'insertion, de mise à jour et de suppression.

1.1. Index simple

Un index simple est créé sur une seule colonne.

Syntaxe :

```
CREATE INDEX nom_index ON nom_table(colonne);
```

Exemple :

Créer un index sur la colonne `nom` de la table `utilisateurs` :

```
CREATE INDEX idx_nom ON utilisateurs(nom);
```

1.2. Index unique

Un index unique garantit que les valeurs dans la colonne indexée sont uniques.

Syntaxe :

```
CREATE UNIQUE INDEX nom_index ON nom_table(colonne);
```

Exemple :

Créer un index unique sur la colonne `email` :

```
CREATE UNIQUE INDEX idx_email_unique ON utilisateurs(email);
```

1.3. Index composite

Un index composite est créé sur plusieurs colonnes. Il est utile pour les requêtes qui filtrent ou trient par plusieurs colonnes.

Syntaxe :

```
CREATE INDEX nom_index ON nom_table(colonne1, colonne2);
```

Exemple :

Créer un index sur les colonnes nom et prénom :

```
CREATE INDEX idx_nom_prenom ON utilisateurs(nom, prenom);
```

1.4. Index plein texte (Full-Text Index)

Un index plein texte est utilisé pour les recherches textuelles avancées dans des colonnes de type texte.

Syntaxe :

```
CREATE FULLTEXT INDEX nom_index ON nom_table(colonne);
```

Exemple :

Créer un index plein texte sur la colonne `description` :

```
CREATE FULLTEXT INDEX idx_description ON articles(description);
```

1.5. Index spatial

Un index spatial est utilisé pour les colonnes contenant des données géographiques.

Syntaxe :

```
CREATE SPATIAL INDEX nom_index ON nom_table(colonne);
```

Exemple :

Créer un index spatial sur une colonne `location` :

```
CREATE SPATIAL INDEX idx_location ON points(location);
```

Exercices pratiques

Types d'index

1. Créer un index simple sur la colonne `nom` de la table `utilisateurs`.
2. Créer un index composite sur les colonnes `nom` et `prénom`.
3. Créer un index unique sur la colonne `email`.

Gestion des Privilèges

Antoine OBLIGIS

Les privilèges en SQL permettent de contrôler l'accès aux bases de données, tables et autres objets. Ils sont attribués à des utilisateurs ou des rôles via les commandes `GRANT` et `REVOKE`.

1. Commande `GRANT`

Attribue des privilèges à un utilisateur.

Syntaxe :

```
GRANT privilege ON objet TO utilisateur [WITH GRANT OPTION];
```

- **`privilege`** : Type de privilège (voir tableau ci-dessous).
- **`objet`** : Nom de la table, base de données ou colonne.
- **`utilisateur`** : Nom de l'utilisateur ou rôle.
- **`WITH GRANT OPTION`** : Permet à l'utilisateur de transmettre les privilèges.

Exemple :

Attribuer le privilège `SELECT` à un utilisateur sur une table :

```
GRANT SELECT ON base_donnees.table TO 'utilisateur'@'localhost';
```

Antoine OBLIGIS

2. Commande REVOKE

Révoque les privilèges précédemment accordés.

Syntaxe :

```
REVOKE privilege ON objet FROM utilisateur;
```

Exemple :

Révoquer le privilège SELECT :

```
REVOKE SELECT ON base_donnees.table FROM 'utilisateur'@'localhost';
```

Privilège	Description
`ALL`	Tous les privilèges disponibles.
`SELECT`	Lire les données.
`INSERT`	Ajouter des données.
`UPDATE`	Modifier des données.
`DELETE`	Supprimer des données.
`CREATE`	Créer des bases ou des objets (tables, vues, etc.).
`DROP`	Supprimer des objets (tables, vues, etc.).
`INDEX`	Créer ou supprimer des index.
`ALTER`	Modifier la structure d'une table ou d'un objet.
`EXECUTE`	Exécuter des procédures stockées et fonctions.
`GRANT OPTION`	Autoriser à attribuer les privilèges à d'autres utilisateurs.

4. Gestion des utilisateurs

Créer un utilisateur :

```
CREATE USER 'utilisateur'@'localhost' IDENTIFIED BY 'mot_de_passe';
```

Supprimer un utilisateur :

```
DROP USER 'utilisateur'@'localhost';
```

Changer le mot de passe d'un utilisateur :

```
ALTER USER 'utilisateur'@'localhost' IDENTIFIED BY 'nouveau_mot_de_passe';
```

Afficher les privilèges :

Afficher les privilèges d'un utilisateur :

```
SHOW GRANTS FOR 'utilisateur'@'localhost';
```

Exercices pratiques

Privilèges

1. Ajouter un utilisateur `admin` et `user1` avec tous les droits
2. Attribuer à un utilisateur appelé `admin` les privilèges complets sur une base de données `article`.
3. Révoque le privilège `DELETE` de l'utilisateur `user1` sur la table `invoices`.
4. Affiche les privilèges de l'utilisateur `admin`.

Vues

Antoine OBLIGIS

Vues

VUES : Simplifier les requêtes complexes

Une vue (`VIEW`) est une table virtuelle basée sur le résultat d'une requête SQL. Elle ne contient pas de données proprement dites, mais fournit une manière simplifiée ou personnalisée d'accéder aux données.

1. Création d’une vue

Syntaxe :

```
CREATE VIEW nom_vue AS
SELECT colonnes
FROM table
WHERE condition;
```

Antoine OBLIGIS

1.1. Créer une vue pour les produits les plus chers :

```
CREATE VIEW produits_chers AS
SELECT nom, prix
FROM produits
WHERE prix > 20.00;
```

Utilisation :

```
SELECT * FROM produits_chers;
```

Antoine OBLIGIS

1.2. Vue pour afficher les factures détaillées :

```
CREATE VIEW factures_detaillees AS
SELECT i.invoice_id, i.date, i.total, a.article_name, item.quantity
FROM invoice AS i
INNER JOIN invoice_item AS item ON i.invoice_id = item.invoice_id
INNER JOIN article AS a ON item.article_id = a.article_id;
```

Utilisation :

```
SELECT * FROM factures_detaillees WHERE total > 50;
```

2. Modification d’une vue

Pour mettre à jour une vue existante, utilise `CREATE OR REPLACE VIEW` :

```
CREATE OR REPLACE VIEW produits_chers AS
SELECT nom, prix, categorie_id
FROM produits
WHERE prix > 30.00;
```

3. Supprimer une vue

Syntaxe :

```
DROP VIEW nom_vue;
```

Exemple : Supprimer la vue `produits_chers` :

```
DROP VIEW produits_chers;
```

3. Avantages des vues

- **Lisibilité :** Simplifie les requêtes complexes.
- **Réutilisation :** Une fois définie, une vue peut être utilisée plusieurs fois.
- **Sécurité :** Masquer certaines colonnes ou données sensibles.
- **Modularité :** Permet de structurer des requêtes en couches.

4. Exercices pratiques

Exercice 1 : Créer une vue simple

Créer une vue `produits_abordables` qui contient les produits avec un prix inférieur à 10 €.

Exercice 2 : Vue complexe

Créer une vue `ventes_detaillees` qui associe chaque facture avec les articles vendus, les quantités, et le montant total pour chaque article.

Exercice 3 : Mise à jour d'une vue

Mettre à jour la vue `produits_abordables` pour inclure uniquement les produits avec un prix inférieur à 5 €.

MERCI

Antoine OBLIGIS