Les fiches récap de l'école O'clock

Bdd

Le langage SQL

Dernière modification: 21 mars 2023

Le langage SQL

Syntaxe – bases

Commentaires

- `#` ou `--` pour une ligne
- `/* ... */` pour plusieurs lignes

DDL : Data Definition Language

Définition des données. **Cela correspond à la création des bases, tables, et champs**.

Créer une table

```
CREATE TABLE `nomDeLaTable` (
    `nomDuChamp` typeDuChamp,
    `nomDuChamp2` typeDuChamp2,
    ...
);
```

Par exemple:

```
CREATE TABLE `user` (
    `login` VARCHAR(10),
    `email` VARCHAR(256),
    `birthday` DATE
);
```

Modifier une table

- `ALTER (database, table) ...` pour déclarer une modification.
- `... ADD (field, index, key)` pour ajouter un champ, un index ou une clef dans une table.
- `...MODIFY...` pour modifier un champ, index, ou clef existant.

Par exemple :

```
ALTER TABLE `user` ADD `password` VARCHAR(16);
```

Ajoute un champ « password » de type VARCHAR à la table « user ».

Supprimer ou vider une table

- `DROP TABLE `nomDeLaTable`` supprime purement et simplement la table.
- `TRUNCATE `nomDeLaTable`` vide la table sans la supprimer, comme un `DELETE` mais en plus rapide (utilisez `DELETE` pour supprimer une ou quelques lignes).

NOTE: Pour éviter les erreurs de syntaxe `table "maTable" does not exist`, on peut rajouter l'option `IF EXISTS` :

```
DROP TABLE IF EXISTS `user`;
```

DML : Data Manipulation Language

Cela correspond à la manipulation des données.

Opérations de base: CRUD

Create, **R**ead, **U**pdate, **D**elete (Ajout, Lecture, Modification, Suppression)

Lecture : `SELECT`

Depuis quelle table : `FROM`

- Lire tous les champs d'une table :

```sql
SELECT *
FROM nom_table
```

- Lire *des* champs spécifiques d'une table :

```sql
SELECT champ1, champ2
FROM nom_table
```

- Attribuer un *alias* à une table :

```sql
SELECT p.name, p.price
FROM product p
```

Condition(s) : `WHERE`

- Lire des lignes de la table qui respectent une condition

```sql
# Tous les étudiants qui s'appellent Pierre
SELECT *
FROM students
WHERE first_name = 'Pierre';
```

Conditions combinées : `AND`

- Lire des lignes de la table qui respectent plusieurs conditions

```sql
/* Tous les étudiants qui
- s'appellent Pierre
- sont nés après le 01/01/1989 */
SELECT *
FROM students
WHERE first_name = 'Pierre'
AND birthdate > '1989-01-01';
```

Classement des résultats : `ORDER BY (ASC) (DESC)`

- Lire toutes les lignes et les classer par ordre alphabétique :

```
SELECT * FROM students
ORDER BY name
```

  Par défaut, le sens du classement est `ASC` (ascendant, croissant), mais on peut également classer par ordre décroissant avec `DESC`.

- Lire les articles de blog, en les classant par date décroissante (les plus récents en premier) :

```
SELECT * FROM articles
ORDER BY publication_date DESC
```

  On peut également mixer les classements, par exemple si on souhaite classer d'abord par nom, puis par prénom si plusieurs noms sont identiques :

```
SELECT * FROM students
ORDER BY name ASC,  first_name ASC
```

Jointures

- « Lier » 2 tables grâce à leur relation :

```
SELECT p.name, c.name
FROM product p, category c
WHERE p.category_id = c.category_id
```

- Autre syntaxe :

```
SELECT product.name, category.name
FROM product
LEFT JOIN category ON product.category_id = category.id
```

  LEFT JOIN : prendre toutes les lignes de la première table, même s'il n'y a pas de valeur correspondante dans la deuxième (si *par exemple*, une catégorie associée à un produit a été supprimée).

Création : `INSERT INTO ... VALUES`

- Insérer 1 ligne avec les valeurs pour chaque champ, dans l'ordre.

```
INSERT INTO schools VALUES
(2, 'O''clock', 'Everywhere');
```

- Insérer 1 ligne en omettant certaines valeurs. Provoquera une erreur si le champ est NOT NULL et pas AUTO_INCREMENT (A_I).

```

# student.id: AUTO_INCREMENT
# student.first_name: NOT NULL
# student.last_name: NULL
# student.birthdate: NOT NULL
```

```
INSERT INTO students (first_name, birthdate) VALUES
('Joe', '1980-10-01')
```

- Insérer plusieurs lignes :

```
INSERT INTO students (first_name, birthdate)
VALUES ('Joe', '1980-10-01'), ('Jack', '1982-01-01'), ('John',
'1981-01-10');
```

Modification : `UPDATE ... SET`

```
UPDATE nom_table
SET champ1 = 'nouvelle valeur'
[WHERE condition]
```

Suppression : `DELETE`

```
# attention à la condition!
# s'il n'y en a pas, toutes les lignes sont supprimées sans condition.
DELETE FROM nom_table
WHERE condition
```

Opérateurs et fonctions de base

Comparaison de chaînes : `LIKE`

Utilisé dans la clause `WHERE`, permet de rechercher les chaînes qui respectent un modèle.

- Le caractère `%` est un joker, représente n'importe quelle chaîne de caractères.

```
# Etudiants dont le nom commence par un D
SELECT *
FROM students
WHERE last_name LIKE 'D%';
```

Date courante

NOW()

`NOW()` renvoi 'la date et l'heure courante' en format `DATETIME`

```
# Lignes dont la date limite est dépassée
# (limit_date est de type DATETIME)
SELECT *
FROM table
WHERE limit_date < NOW();
```

CURDATE()

`CURDATE()` renvoi 'la date courante' en format `DATE`

```sql
# Lignes dont la date limite est dépassée
# (limit_date est de type DATE)
SELECT *
FROM table
WHERE limit_date < CURDATE();
```

DATE()

`DATE()` prend un `DATETIME` en paramètre et renvoi un format `DATE`

```sql
# Lignes dont la date limite est dépassée
# (limit_date est de type DATE)
SELECT *
FROM table
WHERE limit_date < DATE( NOW() );
```

Fonctions d'agrégation

Les lignes peuvent être regroupées et éventuellement « agrégées » grâce à la clause `GROUP BY`. Cela permet d'exécuter certains calculs sur les lignes ainsi regroupées.

Nombre de lignes : COUNT()

```sql
# compte le nombre de lignes de la table products
SELECT COUNT(*)
FROM products
```

Addition : SUM()

```sql
SELECT person_id, person_name, SUM(tax_amount)
FROM taxes
GROUP BY person_id
```

Moyenne : AVG()

```sql
SELECT student_id, student_name, AVG(grade)
FROM grades
GROUP BY student_id
```