La sélection de données dans MariaDB (suite)

Rappels

- La commande `SELECT` permet de sélectionner (lire) des données dans une table.
- Une requête `SELECT` comprend une liste de sélection indiquant quels champs de la table doivent être sélectionnés.
- Le caractère `*` permet de sélectionner tous les champs.
- La clause `WHERE` permet de filtrer les lignes qui seront retournées.
- La clause `ORDER BY` permet d'ordonner les lignes retournées selon le(s) champ(s) voulu(s), en ordre ascendant ou descendant.
- La clause `LIMIT` permet de limiter le nombre de résultats retournés.

Voici un exemple de requête utilisant toutes ces clauses:

```
SELECT titre, prix
    FROM livre
    WHERE id_editeur = 1
    ORDER BY date_parution DESC
    LIMIT 10;
```

Tables utilisées dans les exemples

Pour les exemples suivants, nous allons utiliser les tables "departement", "employe" et "client" créées par les requêtes suivantes:

```
CREATE TABLE departement (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(50) UNIQUE NOT NULL
);

CREATE TABLE employe (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL,
    nas CHAR(9),
    date_embauche DATE,
    taux_horaire DECIMAL(5,2),
    id_departement INT NOT NULL,
    no_telephone CHAR(10),
    courriel_personnel VARCHAR(50),
    courriel_entreprise VARCHAR(50),
    adresse_no_rue VARCHAR(50),
    adresse_code_postal VARCHAR(50),
    adresse_ville VARCHAR(50),
```

```
    adresse_province CHAR(2),
    FOREIGN KEY (id_departement) REFERENCES departement(id)
);

CREATE TABLE client (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL,
    no_telephone CHAR(10),
    courriel VARCHAR(50),
    programme_fidelite TINYINT
);
```

Utilisation d'expressions et mot-clé AS

Jusqu'à maintenant, nous avons seulement inclus des noms de champs dans nos listes de sélection. On peut aussi y inclure des expressions, par exemple des chaînes de caractères, des opérations mathématiques et des appels de fonctions. Voici un exemple avec la fonction `CONCAT`:

```
SELECT CONCAT(nom, ', ', prenom) FROM employe;
```

Cette requête aurait pour effet de retourner les nom et prénom de chaque employé dans une seule colonne, séparés par une virgule (par exemple: `Arcand, Marc`).

On peut aussi utiliser des opérations mathématiques. Par exemple, si on veut afficher le taux horaire actuel de chaque employé, suivi de son taux horaire majoré de 5%:

```
SELECT  nom,
        prenom,
        taux_horaire,
        taux_horaire * 1.05
    FROM employe;
```

On peut aussi utiliser le mot-clé `AS` pour donner un nom à la colonne dans les résultats:

```
SELECT  nom,
        prenom,
        taux_horaire,
        taux_horaire * 1.05 AS taux_horaire_majore
    FROM employe;
```

Le mot-clé `AS` peut en fait être utilisé pour renommer n'importe quelle colonne. On pourrait même s'en servir pour embellir l'affichage des résultats:

```
SELECT  CONCAT(nom, ', ', prenom) AS 'Nom et prénom',
        taux_horaire AS 'Taux horaire actuel',
        taux_horaire * 1.05 AS 'Taux horaire majoré'
    FROM employe
    ORDER BY nom, prenom;
```

Comme pour la liste de sélection, un filtre peut aussi inclure des expressions, par exemple:

```
SELECT * FROM employe WHERE taux_horaire = 10 + 15;
```

Expressions conditionnelles (CASE)

La clause `CASE` permet de sélectionner une valeur en fonction d'une autre valeur. Par exemple, si on veut sélectionner le nom du programme de fidélité auquel a souscrit chaque client, sachant que `1` signifie `Argent`, `2` signifie `Or` et 3 signifie `Platine`:

```
SELECT  id,
        nom,
        prenom,
        CASE programme_fidelite
            WHEN 1 THEN 'Argent'
            WHEN 2 THEN 'Or'
            WHEN 3 THEN 'Platine'
            ELSE 'Aucun programme de fidélité'
        END AS programme_fidelite
    FROM client
    ORDER BY nom, prenom;
```

Il existe une deuxième version de la clause `CASE`, plus flexible. En voici un exemple d'utilisation, permettant d'obtenir le grade d'un employé (`Junior`, `Intermédiaire` ou `Senior`) selon sa date d'embauche:

```
SELECT  *,
        CASE
            WHEN DATEDIFF(NOW(), date_embauche) / 365 >= 5
                THEN 'Senior'
            WHEN DATEDIFF(NOW(), date_embauche) / 365 >= 2
                THEN 'Intermédiaire'
            ELSE
                'Junior'
        END AS grade
    FROM employe;
```

Cet exemple utilise la fonction `NOW()`, qui permet d'obtenir la date l'heure courantes, ainsi que la fonction `DATEDIFF()`, qui permet d'obtenir le nombre de jours entre deux dates. Le grade sera `Senior` pour un

employé embauché il y a au moins 5 ans, `Intermédiaire` pour un employé embauché il y a au moins 2 ans, et `Junior` pour un employé embauché il y a moins de 2 ans.

Les clauses `UNION`, `INTERSECT` et `EXCEPT`

La clause `UNION` permet de combiner les résultats de plusieurs requêtes. Par exemple, si on veut récupérer la liste des employés et des clients en une seule requête:

```
SELECT nom, prenom, no_telephone, courriel_personnel AS courriel FROM employe
UNION
SELECT nom, prenom, no_telephone, courriel FROM client;
```

Pour que l'union fonctionne, il faut que les deux requêtes `SELECT` retournent le même nombre de colonnes, et les colonnes aux mêmes positions doivent avoir les mêmes types. Les noms de colonnes de la première requêtes sont utilisés comme noms de colonnes dans les résultats.

Par défaut, les doublons sont retirés de l'union. On peut utiliser `UNION ALL` pour conserver les doublons.

```
SELECT nom, prenom, no_telephone, courriel_personnel AS courriel FROM employe
UNION DISTINCT
SELECT nom, prenom, no_telephone, courriel FROM client;
```

Si on ajoute une clause `ORDER BY` ou `LIMIT` à cette requête, elle s'appliquera à l'ensemble des résultats de l'union.

```
SELECT nom, prenom, no_telephone, courriel_personnel AS courriel FROM employe
UNION DISTINCT
SELECT nom, prenom, no_telephone, courriel FROM client
ORDER BY nom, prenom
LIMIT 50;
```

Si on veut appliquer un `ORDER BY` ou un `LIMIT` aux requêtes de l'union plutôt qu'à l'union complète, il faut utiliser des parenthèses.

```
(SELECT nom, prenom, no_telephone, courriel_personnel AS courriel
    FROM employe
    ORDER BY nom, prenom
    LIMIT 50)
UNION
(SELECT nom, prenom, no_telephone, courriel
    FROM client
    ORDER BY nom, prenom
    LIMIT 50)
```

La clause `INTERSECT` permet de sélectionner l'intersection de plusieurs requêtes, c'est-à-dire les lignes de résultats qui sont communes à ces requêtes. Par exemple, si on veut obtenir la liste des employés qui sont aussi clients:

```
SELECT nom, prenom, no_telephone, courriel_personnel AS courriel FROM employe
INTERSECT
SELECT nom, prenom, no_telephone, courriel FROM client;
```

La clause `EXCEPT` permet pour sa part de sélectionner les résultats de la première requête qui ne se trouvent pas aussi dans les résultats de la deuxième requête. Par exemple, si on veut obtenir la liste des employés qui ne sont PAS clients:

```
SELECT nom, prenom, no_telephone, courriel_personnel AS courriel FROM employe
EXCEPT
SELECT nom, prenom, no_telephone, courriel FROM client;
```

Les clauses `INTERSECT` et `EXCEPT` s'utilisent de la même façon que la clause `UNION`.

Introduction aux sous-requêtes

Il est possible d'inclure une requête dans une autre requête. On parle alors de **sous-requête**. En voici quelques exemples simples:

```
-- Utilisation d'une sous-requête dans un INSERT pour renseigner une clé étrangère
INSERT INTO employe(nom, prenom, id_departement)
    VALUES(
        'Paquette',
        'Micheline',
        (SELECT id FROM departement WHERE nom = 'Finances')
    );

-- Utilisation d'une sous-requête dans un UPDATE pour renseigner une clé étrangère
UPDATE employe
    SET id_departement = (SELECT id FROM departement WHERE nom = 'Ventes')
    WHERE nom = 'Paquette' AND prenom = 'Micheline';

/* Utilisation d'une sous-requête dans un DELETE pour supprimer les employés
 * des départements "Ventes" et "Finances"
 */
DELETE FROM employe
    WHERE id_departement IN (SELECT id FROM departement WHERE nom IN ('Finances',
'Ventes'));

/* Utilisation d'une sous-requête dans un SELECT pour récupérer une valeur
 * provenant d'une autre table
*/
SELECT  nom,
```

```
        prenom,
        (SELECT nom FROM departement WHERE id = id_departement) AS departement
    FROM employe
    ORDER BY nom, prenom;

-- Deuxième version du même exemple, cette fois-ci en précisant les noms des
tables
SELECT  nom,
        prenom,
        (SELECT nom
            FROM departement
            WHERE departement.id = employe.id_departement
        ) AS departement
    FROM employe
    ORDER BY nom, prenom;
```

Un autre exemple de sous-requête qui peut être particulièrement utile est l'utilisation de la fonction `LAST_INSERT_ID` pour utiliser l'ID auto-généré de la dernière insertion comme valeur d'une clé étrangère:

```
INSERT INTO departement(nom) VALUES('Marketing');

INSERT INTO employe(nom, prenom, id_departement)
    VALUES(
        'Croteau',
        'René',
        (SELECT LAST_INSERT_ID()) -- ID du département de Marketing
    );
```

Références

- Documentation officielle du DML de MariaDB
- Tutoriel sur MySQL de W3Schools