Les jointures et les sous-requêtes

Rappels

- Au dernier cours, nous avons vu que le mot-clé `AS` permet de renommer des colonnes dans les résultats d'une requête.
- Nous avons aussi vu les clauses `UNION`, `INTERSECT` et `EXCEPT` qui permettent respectivement de combiner les résultats de plusieurs requêtes, de produire l'intersection des résultats de plusieurs requêtes, et de soustraire d'une requête les résultats d'une autre requête.
- Nous avons également introduit les sous-requêtes à l'aide de quelques exemples simples. Une sous-requête est une requête incluse dans une autre requête, et peut permettre par exemple de récupérer des valeurs d'une autre table, soit pour obtenir la valeur d'une clé étrangère (dans un `INSERT`, `UPDATE` ou `DELETE`) ou sélectionner les données d'une table liée (dans un `SELECT`).

Tables utilisées dans les exemples

Pour la plupart des exemples suivants, nous allons à nouveau utiliser les tables de la base de données "librairie", créées par les requêtes suivantes:

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

CREATE TABLE langue (
    code CHAR(2) PRIMARY KEY,
    nom VARCHAR(20) NOT NULL UNIQUE
);

CREATE TABLE livre (
    isbn CHAR(13) PRIMARY KEY,
    titre VARCHAR(50) NOT NULL,
    id_editeur INT,
    date_parution DATE NOT NULL,
    description TEXT,
    code_langue CHAR(2),
    prix DECIMAL(5,2) UNSIGNED,
        -- prix: Maximum de 5 chiffres dont 2 après la virgule
    FOREIGN KEY (id_editeur) REFERENCES editeur(id)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
    FOREIGN KEY (code_langue) REFERENCES langue(code)
        ON UPDATE CASCADE
```

```
        ON DELETE RESTRICT
);

CREATE TABLE auteur (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL
);

CREATE TABLE auteur_livre (
    id_auteur INT NOT NULL,
    isbn_livre CHAR(13) NOT NULL,
    PRIMARY KEY (id_auteur, isbn_livre),
    FOREIGN KEY (id_auteur) REFERENCES auteur(id)
        ON DELETE CASCADE
        ON UPDATE RESTRICT,
    FOREIGN KEY (isbn_livre) REFERENCES livre(isbn)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

Les jointures

- Les jointures (`JOIN`) servent à joindre des données de plusieurs tables (par exemple pour aller chercher les données d'une autre table liée par une clé étrangère).
- En ce sens, elles peuvent avoir une utilité semblable à celle des exemples de sous-requêtes dans un `SELECT` que nous avons vu la semaine dernière.
- Il existe plusieurs types de jointures:
    - Jointure interne (`INNER JOIN`)
    - Jointure externe (`OUTER JOIN`)
    - Jointure croisée (`CROSS JOIN`)

Les jointures internes (`INNER JOIN`)

Une jointure interne consiste à joindre des données de deux tables, de façon que seuls les enregistrements qui ont une référence commune dans les deux tables sont conservés. La jointure s'appuie généralement sur la liaison d'une clé étrangère à une clé primaire.

Par exemple, voici une jointure entre la table `livre` et la table `editeur`:

```
SELECT isbn, titre, date_parution, nom
    FROM livre
    INNER JOIN editeur
        ON id = id_editeur;
```

Note: L'inclusion du mot-clé `INNER` est facultative.

Cette requête aura pour effet de produire une liste de livres avec les champs `isbn`, `titre` et `date_parution` de la table `livre`, ainsi que le champ `nom` de la table `editeur`. Chaque ligne de la table `livre` sera associée à une ligne de la table `editeur` selon la correspondance entre le champ `id_editeur` de la table `livre` et le champ `id` de la table `editeur`.

La requête que nous avons formulée peut cependant porter à confusion, puisque lorsqu'on la lit, ce n'est pas clair à quelle table appartient chaque champ. Il vaut donc mieux le préciser dans la requête:

```
SELECT  livre.isbn,
        livre.titre,
        livre.date_parution,
        editeur.nom
    FROM livre
    JOIN editeur
        ON editeur.id = livre.id_editeur;
```

Bien entendu, il peut être fastidieux de devoir répéter les mots "livre" et "editeur" partout dans notre requête. On peut donc utiliser des alias:

```
SELECT l.isbn, l.titre, l.date_parution, e.nom
    FROM livre AS l
    JOIN editeur AS e
        ON e.id = l.id_editeur;
```

Le mot-clé `AS` n'est par ailleurs pas obligatoire. On peut donc aussi écrire:

```
SELECT l.isbn, l.titre, l.date_parution, e.nom
    FROM livre l
    JOIN editeur e
        ON e.id = l.id_editeur;
```

Puisqu'il s'agit d'une jointure interne, les seules données qui apparaîtront dans les résultats sont celles pour lesquelles il existe une entrée dans les deux tables. Autrement dit, seuls les livres qui ont un éditeur seront inclus.

Il est bien sûr possible d'ajouter les autres clauses que nous avons vues suite à un `JOIN`, par exemple `WHERE`, `ORDER BY` et `LIMIT`:

```
SELECT l.isbn, l.titre, l.date_parution, e.nom
    FROM livre l
    JOIN editeur e
        ON e.id = l.id_editeur
    WHERE titre LIKE 'Harry Potter%'
    ORDER BY date_parution
    LIMIT 7;
```

Il est aussi possible d'inclure plusieurs `JOIN` dans une même requête:

```
SELECT l.isbn, l.titre, l.date_parution, e.nom AS editeur, lng.nom AS langue
    FROM livre l
    JOIN editeur e
        ON e.id = l.id_editeur
    JOIN langue lng
        ON lng.code = l.code_Langue
```

Si les tables d'une jointure ont une relation de type "un à plusieurs", on se retrouvera avec une ligne de résultats pour chaque entrée dans la table du côté "N" de la relation. Autrement dit, les résultats ressembleront à une table non normalisée! C'est par exemple ce qui se produirait avec la jointure suivante:

```
SELECT l.isbn, l.titre, l.date_parution, e.nom AS editeur, lng.nom AS langue
    FROM livre l
    JOIN editeur e
        ON e.id = l.id_editeur
    JOIN langue lng
        ON lng.code = l.code_Langue
```

Il existe une syntaxe alternative, qu'on appelle "jointure implicite", pour effectuer une jointure interne. Voici cette syntaxe, mais gardez en tête qu'elle n'est pas recommandée:

```
SELECT l.isbn, l.titre, l.date_parution, e.nom
    FROM livre l, editeur e
    WHERE l.id_editeur = e.id;
```

Les jointures externes (`OUTER JOIN`)

Rappelons qu'une jointure interne ne retourne que les données pour lesquelles il existe une entrée dans les deux tables. Une jointure externe (`OUTER JOIN`), pour sa part, permet d'inclure toutes les données d'une des deux

tables (ou des deux tables), même celles pour lesquelles il n'y a pas de correspondance dans l'autre table. Il en existe trois types principaux:

- `LEFT OUTER JOIN`
- `RIGHT OUTER JOIN`
- `FULL OUTER JOIN`

Le `LEFT OUTER JOIN` permet de joindre deux tables en incluant toutes les données qui se trouvent dans la table de gauche.

Par exemple, la requête suivante récupère chaque livre avec le nom de son éditeur, tout en incluant les livres qui n'ont pas d'éditeur:

```
SELECT l.isbn, l.titre, l.date_parution, e.nom
    FROM livre l
    LEFT OUTER JOIN editeur e
        ON e.id = l.id_editeur;
```

Note: L'inclusion du mot-clé `INNER` est facultative.

Si on voulait plutôt prioriser la table `editeur` plutôt que la table `livre`, on pourrait faire un `RIGHT OUTER JOIN`:

```
SELECT l.isbn, l.titre, l.date_parution, e.nom
    FROM livre l
    RIGHT OUTER JOIN editeur e
        ON e.id = l.id_editeur;
```

Et si on voulait inclure toutes les données de part et d'autre, on pourrait faire un `FULL OUTER JOIN`. Malheureusement, ce type de jointure n'est pas supporté par MariaDB! La solution est donc de faire l'union d'un `LEFT OUTER JOIN` et d'un `RIGHT OUTER JOIN`:

```
SELECT l.isbn, l.titre, l.date_parution, e.nom
    FROM livre l
    LEFT JOIN editeur e
        ON e.id = l.id_editeur
UNION
SELECT l.isbn, l.titre, l.date_parution, e.nom
    FROM livre l
    RIGHT JOIN editeur e
        ON e.id = l.id_editeur;
```

Si on voulait obtenir seulement les enregistrements qui n'ont **PAS** de correspondance à gauche ou à droite, on pourrait ajouter des clauses `WHERE`:

```
SELECT l.isbn, l.titre, l.date_parution, e.nom
    FROM livre l
    LEFT JOIN editeur e
        ON e.id = l.id_editeur
    WHERE e.id IS NULL
UNION
SELECT l.isbn, l.titre, l.date_parution, e.nom
    FROM livre l
    RIGHT JOIN editeur e
        ON e.id = l.id_editeur
    WHERE l.isbn IS NULL;
```

Les jointures croisées (`CROSS JOIN`)

Une jointure croisée retourne le produit cartésien des deux tables, c'est-à-dire que les résultats contiendront toutes les combinaisons possibles des lignes de la table de gauche avec les lignes de la table de droite.

```
SELECT l.isbn, l.titre, l.date_parution, e.nom
    FROM livre l
    CROSS JOIN editeur e;
```

En MariaDB, on peut omettre le mot-clé `CROSS`, car la jointure croisée est le comportement par défaut d'une jointure sans option `ON`.

Les sous-requêtes

Nous avons vu des exemples de sous-requêtes simples au dernier cours, avec les tables "employe" et "departement". Voici un rappel de ces exemples:

```
-- 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;
```

Voici deux exemples supplémentaires, cette fois-ci avec les tables `livre` et `editeur`:

```
-- Sous-requête dans la clause WHERE d'un SELECT
SELECT *
    FROM livre
    WHERE id_editeur = (SELECT id FROM editeur WHERE nom = 'Gallimard Jeunesse');

-- Sous-requête dans la clause FROM
SELECT titre, nom_editeur
    FROM (
        SELECT l.*, e.nom AS nom_editeur
            FROM livre l
            JOIN editeur e
                ON e.id = l.id_editeur
    ) AS livre_editeur;

/*
 * Attention: la requête ci-dessus est présente à des fins d'exemples uniquement,
 * et aurait très bien pu être écrite sans sous-requête:
 */
```

```sql
SELECT l.titre, e.nom AS nom_editeur
    FROM livre l
    JOIN editeur e
        ON e.id = l.id_editeur;
```

Opérateur `EXISTS`

Au dernier cours, nous avons vu un exemple où une clause `INTERSECT` entre une requête sur table "employe" et une autre sur une table "client" était utilisée pour obtenir la liste des employés qui sont aussi des clients. Voici un rappel de cet exemple:

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

Or, nous aurions pu obtenir le même résultat avec une sous-requête, à l'aide de l'opérateur `EXISTS`:

```sql
SELECT nom, prenom, no_telephone, courriel_personnel AS courriel
    FROM employe e
    WHERE EXISTS (
        SELECT * FROM client c
            WHERE   c.nom = e.nom
                AND c.prenom = e.prenom
                AND c.no_telephone = e.no_telephone
                AND c.courriel = e.courriel_personnel
    );
```

Références

- Documentation officielle du DML de MariaDB
- Documentation de MariaDB sur les jointures et les sous-requêtes
- Tutoriel sur MySQL de W3Schools