Reasliser par Salima Ouassas et Nourelhouda Elkasra

Projet Bases de Données - Algèbre Relationnelle

Licence MIP - IAP S4 - 2025

Encadrée par Pr. J.ZAHIR

Requêtes SQL traduites en Algèbre Relationnelle

a. Afficher la liste des réservations avec le nom du client et la ville de l'hôtel réservé

Requête en algèbre relationnelle :

```
PROJECT[id_Reservation, nom_client, ville](
    JOIN[id_Hotel](
        JOIN[id_Chambre](
            JOIN[id_Reservation](
                JOIN[id_Client](Reservation, Client),
                Reservation_Chambre
            ),
            Chambre
        ),
        Hotel
    )
)
```

b. Afficher les clients qui habitent à Paris

Requête en algèbre relationnelle :

```
SELECT[ville = 'Paris'](Client)
```

c. Calculer le nombre de réservations faites par chaque client

Reasliser par Salima Ouassas et Nourelhouda Elkasra

Requête en algèbre relationnelle :

```
GROUP_BY[id_Client, nom_client ; COUNT(id_Reservation) AS nb_Reservations](
    LEFT_JOIN[id_Client](Client, Reservation)
)
```

d. Donner le nombre de chambres pour chaque type de chambre

Requête en algèbre relationnelle :

```
GROUP_BY[id_Type, type_chambre ; COUNT(id_Chambre) AS nb_chambres](
    LEFT_JOIN[id_Type](Type_Chambre, Chambre)
)
```

e. Afficher les chambres non réservées pour une période donnée

Requête en algèbre relationnelle :

Reasliser par Salima Ouassas et Nourelhouda Elkasra

```
PROJECT[id_Chambre, num_Chambre, etage, type_chambre, ville](
    MINUS(
        JOIN[id_Hotel](
            JOIN[id_Type](Chambre, Type_Chambre),
            Hotel
        ),
        PROJECT[id_Chambre, num_Chambre, etage, type_chambre, ville](
            JOIN[id_Reservation](
                JOIN[id_Chambre](
                    JOIN[id_Hotel](
                        JOIN[id_Type](Chambre, Type_Chambre),
                        Hotel
                    ),
                    Reservation_Chambre
                ),
                SELECT[NOT(date_depart < date_debut OR date_arrivee > date_fin)]
(Reservation)
            )
        )
    )
)
```

Opérations utilisées

- **SELECT** : Sélection (condition)
- **PROJECT** : Projection (colonnes)
- **JOIN** : Jointure naturelle
- **LEFT_JOIN** : Jointure externe gauche
- **GROUP_BY** : Regroupement avec agrégation
- **MINUS** : Différence ensembliste
- **OR** : OU logique
- **NOT** : NON logique

Notes sur les correspondances SQL vs Algèbre Relationnelle

Reasliser par Salima Ouassas et Nourelhouda Elkasra

1. **LEFT JOIN** en SQL correspond à **LEFT_JOIN** en algèbre relationnelle
2. **GROUP BY** avec **COUNT()** correspond à l'opérateur **GROUP_BY**
3. **NOT IN** avec sous-requête correspond à la différence ensembliste (**MINUS**)
4. Les conditions **WHERE** correspondent à l'opérateur **SELECT**
5. **SELECT** (liste de colonnes) correspond à l'opérateur **PROJECT**

4) Qu'est ce que SQLite, quelle différence avec MySQL?

SQLite

SQLite est un système de gestion de base de données relationnelle (SGBDR) léger et embarqué. Contrairement aux bases de données traditionnelles, SQLite est une bibliothèque logicielle qui s'intègre directement dans les applications.

Caractéristiques principales de SQLite :

- **Sans serveur** : Pas besoin d'un processus serveur séparé
- **Auto-contenu** : Toute la base de données est stockée dans un seul fichier
- **Configuration zéro** : Aucune installation ou configuration complexe
- **Multiplateforme** : Fonctionne sur tous les systèmes d'exploitation
- **Très léger** : Bibliothèque de moins de 1 MB
- **ACID compliant** : Respecte les propriétés de transaction

Principales différences avec MySQL

Architecture

- **SQLite** : Base de données embarquée, fonctionne comme une bibliothèque
- **MySQL** : Serveur de base de données traditionnel avec architecture client-serveur

Installation et configuration

- **SQLite** : Aucune installation, juste inclure la bibliothèque
- **MySQL** : Installation complète du serveur, configuration des utilisateurs, ports, etc.

Accès concurrent

- **SQLite** : Limité en écriture simultanée (un seul écrivain à la fois)
- **MySQL** : Optimisé pour de nombreuses connexions simultanées

Taille et performance

- **SQLite** : Idéal pour des applications légères, mobiles, prototypage
- **MySQL** : Conçu pour des applications web à fort trafic et grandes bases de données

Cas d'usage typiques

SQLite convient pour :

- Applications mobiles (Android, iOS)
- Applications desktop

- Prototypage rapide
- Sites web à trafic modéré
- Applications embarquées
- Tests et développement

MySQL convient pour :

- Applications web à fort trafic
- Sites e-commerce
- Systèmes d'entreprise
- Applications nécessitant de nombreuses connexions simultanées
- Bases de données volumineuses avec plusieurs utilisateurs

Syntaxe SQL

Les deux supportent le SQL standard, mais avec quelques différences mineures dans les types de données et certaines fonctions spécifiques.

En résumé, SQLite est parfait pour des projets simples ou embarqués, tandis que MySQL excelle dans les environnements multi-utilisateurs et les applications web complexes.