Requêtes Algèbre relationalle et SQL

et Comparaison entre MySQL et SQLite

Table des matières

1 Introduction

La manipulation et l'interrogation des données sont au cœur des systèmes d'information modernes. Le langage SQL (Structured Query Language) est la norme de facto pour interagir avec les bases de données relationnelles. L'algèbre relationnelle, quant à elle, fournit les fondements théoriques de ces opérations. Ce document se divise en deux parties principales :

1) La traduction de requêtes SQL spécifiques en leur équivalent en algèbre relationnelle, illustrant les opérations fondamentales telles que la projection, la sélection, la jointure et l'agrégation.
2) Une comparaison détaillée entre deux systèmes de gestion de bases de données (SGBD) largement utilisés : SQLite, une base de données embarquée, et MySQL, un SGBD client-serveur.

Cette analyse vise à renforcer la compréhension des mécanismes sous-jacents aux requêtes de bases de données et à guider le choix d'un SGBD adapté à des besoins spécifiques.

2 Traduction des Requêtes SQL en Algèbre Relationnelle

Dans cette section, chaque requête SQL fournie est accompagnée de sa traduction en algèbre relationnelle, décrite textuellement. Les opérations fondamentales de l'algèbre relationnelle, telles que la Projection (souvent notée Π), la Sélection (σ), la Jointure ($\bowtie$), la Jointure Externe Gauche, le Groupement et Agrégation (parfois noté $\mathcal{G}$), et la Différence ensembliste ($-$ ou anti-jointure), seront expliquées étape par étape pour chaque requête. Les noms de tables (relations) sont en italique, par exemple *Client*. Les noms d'attributs sont indiqués en utilisant une police à chasse fixe, par exemple `Id_Client`.

2.1 Requête A

Code SQL

```
    R.Id_Reservation,
    C.Nom_complet AS "Nom du client",
    H.Ville AS "Ville de l'hotel"
FROM
    Reservation R
    JOIN Client C
        ON R.Id_Client = C.Id_Client
    JOIN Concerner Con
        ON R.Id_Reservation = Con.Id_Reservation
    JOIN Type_Chambre TC
        ON Con.Id_Type = TC.Id_Type
    JOIN Chambre Ch
        ON TC.Id_Type = Ch.Id_Type
    JOIN Hotel H
        ON Ch.Id_Hotel = H.Id_Hotel
GROUP BY
    R.Id_Reservation,
    C.Nom_complet,
    H.Ville
```

```
ORDER BY
    R.Id_R servation;
```

Algèbre Relationnelle

```
Projection[Id_Réservation, Nom_complet, Ville]
(
  Jointure[Réservation.Id_Client = Client.Id_Client]
  (
    Jointure[Concerner.Id_Réservation = Réservation.Id_Réservation]
    (
      Jointure[Concerner.Id_Type = Type_Chambre.Id_Type]
      (
        Jointure[Type_Chambre.Id_Type = Chambre.Id_Type]
        (
          Jointure[Chambre.Id_Hotel = Hotel.Id_Hotel]
          (
            Réservation,
            Chambre
          ),
          Hotel
        ),
        Type_Chambre
      ),
      Concerner
    ),
    Client
  )
)
```

2.2 Requête B

Code SQL

```
SELECT * FROM Client WHERE Ville = 'Paris';
```

Algèbre Relationnelle

```
Sélection[Ville = 'Paris'](Client)
```

2.3 Requête C

Code SQL

```
SELECT
    C.Nom_complet AS "Nom du client",
    COUNT(R.Id_R servation) AS "Nombre de r servations"
FROM
    Client C
```

```sql
    LEFT JOIN Reservation R
        ON C.Id_Client = R.Id_Client
GROUP BY
    C.Id_Client,
    C.Nom_complet;
```

Algèbre Relationnelle

```
Projection[Nom_complet, COUNT(Id_Réservation)]
(
  Regroupement[Id_Client]
  (
    Agréger[COUNT(Id_Réservation)]
    (
      JointureGauche[Client.Id_Client = Réservation.Id_Client](Client, Réservation)
    )
  )
)
```

2.4 Requête D

Code SQL

```sql
SELECT
    TC.Type AS "Type de chambre",
    COUNT(Ch.Id_Chambre) AS "Nombre de chambres"
FROM
    Type_Chambre TC
    LEFT JOIN Chambre Ch
        ON TC.Id_Type = Ch.Id_Type
GROUP BY
    TC.Id_Type,
    TC.Type;
```

Algèbre Relationnelle

```
Projection[Type, COUNT(Id_Chambre)]
(
  Regroupement[Id_Type]
  (
    Agréger[COUNT(Id_Chambre)]
    (
      JointureGauche[Type_Chambre.Id_Type = Chambre.Id_Type](Type_Chambre, Chambre)
    )
  )
)
```

2.5 Requête E

Code SQL

```sql
SELECT
    Ch.Id_Chambre,
    Ch.Numero,
    H.Ville,
    TC.Type AS "Type de chambre"
FROM
    Chambre Ch
    JOIN Hotel H
        ON Ch.Id_Hotel = H.Id_Hotel
    JOIN Type_Chambre TC
        ON Ch.Id_Type = TC.Id_Type
WHERE
    TC.Id_Type NOT IN (
        SELECT DISTINCT Con.Id_Type
        FROM Concerner Con
        JOIN Reservation R
            ON Con.Id_R servation = R.Id_R servation
        WHERE
            R.Date_arriv e <= '2025-12-31'
            AND R.Date_d part >= '2025-12-01'
    )
ORDER BY
    H.Ville,
    Ch.Numero;
```

Algèbre Relationnelle

```
Projection[Id_Chambre, Numero, Ville, Type]
(
  Jointure[Chambre.Id_Hotel = Hotel.Id_Hotel]
  (
    Jointure[Chambre.Id_Type = Type_Chambre.Id_Type]
    (
      Sélection[Id_Type NOT IN (
        Projection[Id_Type]
        (
          Jointure[Concerner.Id_Réservation = Réservation.Id_Réservation]
          (
            Sélection[Date_arrivée <= '2025-12-31' ET Date_départ >= '2025-12-01'](Rés
            Concerner
          )
        )
      )](Chambre),
      Type_Chambre
    ),
    Hotel
  )
)
```

3 Comparaison : SQLite vs. MySQL

3.1 Définitions et Caractéristiques Générales

3.1.1 SQLite

SQLite est un système de gestion de base de données relationnelle (SGBDR) **embarqué**. Il est implémenté sous forme de bibliothèque C et ne fonctionne pas sur un modèle client-serveur. La base de données entière (schéma, tables, index, données) est stockée dans un **unique fichier** sur le système de fichiers de l'hôte.

Caractéristiques Clés :

- **Serverless (Sans serveur) :** Aucun processus serveur distinct à gérer.
- **Zéro-configuration :** Pas d'installation ou d'administration complexe.
- **Transactionnel :** Support complet des propriétés ACID (Atomicité, Cohérence, Isolation, Durabilité).
- **Portable :** Le fichier de base de données est multiplateforme.
- **Léger :** Empreinte mémoire et disque très faible.
- **Domaine public :** Libre d'utilisation.

Cas d'usage typiques : Applications mobiles (Android, iOS), applications de bureau, stockage de données pour navigateurs web, tests unitaires, petits sites web à faible trafic, prototypage rapide.

3.1.2 MySQL

MySQL est l'un des SGBDR **open source client-serveur** les plus populaires. Un processus serveur MySQL s'exécute en continu, gérant les bases de données et répondant aux requêtes des applications clientes via le réseau.

Caractéristiques Clés :

- **Client-Serveur :** Architecture nécessitant un serveur dédié.
- **Haute performance et scalabilité :** Capable de gérer de grandes bases de données et un trafic élevé.
- **Fonctionnalités avancées :** Support des procédures stockées, triggers, vues, réplication, clustering.
- **Sécurité :** Mécanismes robustes de gestion des utilisateurs, rôles et permissions.
- **Moteurs de stockage multiples :** InnoDB (transactionnel, verrouillage par ligne), MyISAM (rapide pour lectures), etc.
- **Large communauté et support.**

Cas d'usage typiques : Applications web (de petite à très grande échelle), systèmes de gestion de contenu (WordPress, Drupal), plateformes e-commerce, applications d'entreprise nécessitant une base de données centralisée et partagée.

Table 1 – Différences clés entre SQLite et MySQL

Caractéristique	**SQLite**	**MySQL**
Architecture	Embarqué (bibliothèque C)	Client-Serveur
Stockage des données	Fichier unique	Multiples fichiers gérés par le serveur
Configuration	Nulle (zéro-configuration)	Installation et configuration du serveur
Administration	Minimale (permissions du fichier)	Requiert un administrateur de base de données (DBA) pour les déploiements complexes
Accès concurrent	Limité (verrouillage au niveau du fichier, amélioré récemment mais pas au niveau de MySQL)	Élevé (verrouillage fin au niveau des lignes avec InnoDB)
Scalabilité	Verticale (ressources de la machine hôte), limitée pour la concurrence	Horizontale et verticale, support du clustering et de la réplication
Typage des données	Dynamique (manifest typing)	Statique et strict (type défini par colonne)
Gestion des utilisateurs	Aucune gestion intégrée (via permissions du système de fichiers)	Système complet de gestion des utilisateurs et des droits
Accès réseau	Non natif (l'application hôte doit l'exposer si besoin)	Natif, conçu pour l'accès réseau
Fonctionnalités SQL	Standard SQL, certaines fonctionnalités avancées peuvent manquer	Support étendu du standard SQL et de nombreuses extensions propriétaires
Complexité	Très simple	Modérée à complexe

3.2 Tableau Comparatif des Différences Majeures

3.3 Conclusion de la Comparaison

Le choix entre SQLite et MySQL dépend fondamentalement des exigences du projet :

— **Choisissez SQLite** si :
  – Votre application est autonome (desktop, mobile) et n'a pas besoin de partager des données de manière centralisée et concurrente.
  – Vous avez besoin d'une solution de stockage simple, sans configuration ni administration.
  – La portabilité (un seul fichier) est un avantage majeur.
  – Vous faites du prototypage ou des tests qui nécessitent une base de données légère.

— **Choisissez MySQL** si :
  – Vous développez une application web ou un service qui nécessite un accès concurrent par plusieurs utilisateurs/processus.
  – La base de données doit être accessible via le réseau.
  – Vous avez besoin de fonctionnalités avancées, d'une forte scalabilité et de mécanismes de sécurité robustes.
  – La gestion de volumes de données importants et la performance sous forte charge sont critiques.

En somme, SQLite excelle par sa simplicité et son intégration, tandis que MySQL brille par sa puissance et sa capacité à gérer des applications distribuées et à grande échelle.

4 les liens

Le liens vers le Repo GitHub : GitHub Repo
Le lien vers la capture vidéo de l'interface : Védio

5 Conclusion Générale

Ce document a exploré deux facettes importantes de la gestion des données : la logique formelle des requêtes via l'algèbre relationnelle et les aspects pratiques du choix d'un SGBD à travers la comparaison de SQLite et MySQL. La traduction des requêtes SQL en algèbre relationnelle permet de mieux saisir les opérations effectuées par le SGBD. Comprendre les différences fondamentales entre des SGBD comme SQLite et MySQL est crucial pour concevoir des architectures logicielles efficaces et adaptées aux contraintes spécifiques de chaque projet. Le choix judicieux d'outils et une bonne compréhension des concepts sous-jacents sont des atouts majeurs pour tout développeur ou architecte de systèmes d'information.