Gestion des Bases de Données

1. Bases de Données Relationnelles (SQL)

1.1 Conception de Schémas

La conception de schémas consiste à définir la structure d'une base de données. Cela inclut la définition des tables, des colonnes, des types de données, et des relations entre les tables (un à un, un à plusieurs, plusieurs à plusieurs).

Exemples de Tables

```sql
-- Table Utilisateurs
CREATE TABLE utilisateurs (
    id SERIAL PRIMARY KEY,
    nom VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    mot_de_passe VARCHAR(255) NOT NULL
);

-- Table Commandes
CREATE TABLE commandes (
    id SERIAL PRIMARY KEY,
    utilisateur_id INT REFERENCES utilisateurs(id),
    date_commande TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

1.2 Normalisation

La normalisation est le processus d'organisation des données pour réduire la redondance. Elle est souvent divisée en plusieurs formes normales (1NF, 2NF, 3NF).

Exemple de Normalisation

Au lieu de stocker les informations d'adresse dans la table `utilisateurs`, on peut créer une table `adresses` pour éviter la duplication.

```sql
CREATE TABLE adresses (
    id SERIAL PRIMARY KEY,
    utilisateur_id INT REFERENCES utilisateurs(id),
    rue VARCHAR(100),
    ville VARCHAR(50),
    code_postal VARCHAR(10)
);
```

1.3 Optimisation des Requêtes

L'optimisation des requêtes implique l'utilisation d'index, la réécriture de requêtes inefficaces, et l'analyse des plans d'exécution.

Exemple d'Indexation

```
-- Création d'un index sur la colonne email pour améliorer la recherche
CREATE INDEX idx_utilisateur_email ON utilisateurs(email);
```

---

2. Bases de Données NoSQL

2.1 Comprendre les Bases de Données NoSQL

Les bases de données NoSQL sont conçues pour traiter des données non structurées ou semi-structurées. Elles offrent une grande flexibilité et évolutivité.

2.2 Quand Utiliser NoSQL

- **MongoDB** : Idéal pour les applications avec des données flexibles et évolutives. Utilisé souvent pour les applications web et mobiles.
- **Cassandra** : Convient aux applications nécessitant une haute disponibilité et la scalabilité horizontale.
- **Redis** : Utilisé comme cache ou pour stocker des données temporaires, offrant des opérations rapides.

Exemple d'Insertion dans MongoDB

```
// Utilisation de MongoDB avec Mongoose
const mongoose = require('mongoose');

const utilisateurSchema = new mongoose.Schema({
    nom: String,
    email: String,
    mot_de_passe: String,
});

const Utilisateur = mongoose.model('Utilisateur', utilisateurSchema);

// Insertion d'un utilisateur
const nouvelUtilisateur = new Utilisateur({ nom: 'John Doe', email:
'john@example.com', mot_de_passe: 'securePassword' });
nouvelUtilisateur.save().then(() => console.log('Utilisateur enregistré'));
```

---

3. ORM (Object Relational Mapping)

L'ORM est un paradigme qui permet de mapper des objets de programmation à des tables de bases de données relationnelles. Cela facilite l'interaction avec la base de données en utilisant des objets au lieu d'écrire des requêtes SQL manuelles.

3.1 Hibernate (Java)

Hibernate est un ORM populaire pour Java qui simplifie l'interaction avec les bases de données relationnelles.

Exemple de Configuration avec Hibernate

Exemple de Mapping d'une Entité

```
import javax.persistence.*;

@Entity
@Table(name = "utilisateurs")
public class Utilisateur {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nom;

    private String email;

    private String motDePasse;

    // Getters et Setters
}
```

Exemple d'Utilisation de Hibernate

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class Main {
    public static void main(String[] args) {
        SessionFactory sessionFactory = new
Configuration().configure().buildSessionFactory();
        Session session = sessionFactory.openSession();

        session.beginTransaction();

        Utilisateur utilisateur = new Utilisateur();
        utilisateur.setNom("John Doe");
        utilisateur.setEmail("john@example.com");
        utilisateur.setMotDePasse("securePassword");
```

```
        session.save(utilisateur); // Sauvegarde de l'utilisateur

        session.getTransaction().commit();
        session.close();
    }
}
```

3.2 Prisma (JavaScript)

Prisma est un ORM moderne pour Node.js et TypeScript qui facilite la gestion des bases de données relationnelles.

Exemple de Configuration de Prisma

```
# Installation de Prisma
npm install @prisma/client prisma --save-dev

# Initialisation de Prisma
npx prisma init
```

Exemple de schéma Prisma

```
// fichier schema.prisma
model Utilisateur {
    id        Int     @id @default(autoincrement())
    nom       String
    email     String  @unique
    motDePasse String
}
```

Exemple d'Utilisation de Prisma

```
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();

async function main() {
    const nouvelUtilisateur = await prisma.utilisateur.create({
        data: {
            nom: 'John Doe',
            email: 'john@example.com',
            motDePasse: 'securePassword',
        },
    });
    console.log(nouvelUtilisateur);
}
```

```
main()
    .catch(e => console.error(e))
    .finally(async () => {
        await prisma.$disconnect();
    });
```

3.3 Sequelize (JavaScript)

Sequelize est un autre ORM pour Node.js qui offre une interface simple pour interagir avec des bases de données SQL.

Exemple de Configuration de Sequelize

```
# Installation de Sequelize
npm install sequelize mysql2
```

Exemple de Modèle Sequelize

```
const { Sequelize, DataTypes } = require('sequelize');
const sequelize = new Sequelize('ma_base', 'utilisateur', 'motdepasse', {
    host: 'localhost',
    dialect: 'mysql',
});

const Utilisateur = sequelize.define('Utilisateur', {
    nom: {
        type: DataTypes.STRING,
        allowNull: false,
    },
    email: {
        type: DataTypes.STRING,
        unique: true,
        allowNull: false,
    },
    motDePasse: {
        type: DataTypes.STRING,
        allowNull: false,
    },
});

// Synchroniser le modèle avec la base de données
sequelize.sync();
```

Exemple d'Utilisation de Sequelize

```
async function createUser() {
    const utilisateur = await Utilisateur.create({
        nom: 'John Doe',
        email: 'john@example.com',
        motDePasse: 'securePassword',
    });
    console.log(utilisateur.toJSON());
}

createUser().catch(console.error);
```

L'utilisation des ORM comme Hibernate pour Java et Prisma/Sequelize pour JavaScript simplifie la gestion des bases de données en permettant aux développeurs de travailler avec des objets plutôt qu'avec des requêtes SQL complexes. Cela améliore également la lisibilité du code et facilite la maintenance.