Introduction aux bases de données et à SQL

Partie 1 : Introduction aux bases de données

1. Qu'est-ce qu'une base de données ?

Une **base de données** est un ensemble organisé d'informations ou de données qui peuvent être facilement accédées, gérées et mises à jour. Les bases de données sont utilisées pour stocker et gérer des informations de manière efficace.

Exemple

- Un carnet d'adresses est une base de données simple : il contient des noms, des numéros de téléphone, des adresses e-mail, etc.
- Les plateformes comme Amazon utilisent des bases de données pour gérer les informations sur les produits, les commandes, les utilisateurs, etc.

2. Pourquoi utiliser une base de données ?

- **Stocker les données** : Permet de conserver les informations de manière sûre et organisée.
- **Accéder rapidement aux données** : Permet de retrouver facilement des informations précises.
- **Mise à jour simplifiée** : Les données peuvent être modifiées sans perturber le système.
- **Partage** : Les bases de données peuvent être partagées entre plusieurs utilisateurs.

3. Types de bases de données

Bases de données relationnelles :

Organisées en tables avec des relations entre elles. (Ex : MySQL, PostgreSQL)

Définition :

Les bases de données relationnelles organisent les données sous forme de **tables** (appelées aussi relations) composées de lignes (enregistrements) et de colonnes (champs). Les relations entre ces tables sont établies à l'aide de **clés primaires** et **clés étrangères**, ce qui permet de structurer et de relier les données efficacement.

Caractéristiques :

- Données bien structurées, respectant un schéma défini.
- Utilisation du langage SQL (Structured Query Language) pour les requêtes.
- Garantissent des transactions fiables avec les propriétés ACID (Atomicité, Cohérence, Isolation, Durabilité).

Exemples de bases relationnelles :

- **MySQL** : Utilisé couramment pour les applications web (WordPress, e-commerce, etc.).
- **PostgreSQL** : Réputé pour ses fonctionnalités avancées comme les types de données personnalisés.

Exemple concret :

Imaginons une base de données pour un site de réservation de vols. Nous avons deux tables principales :

Table Passagers :

ID_Passager	Nom	Email
1	Alice Dupont	alice@mail.com
2	Bob Martin	bob@mail.com

Table Réservations :

ID_Reservation	ID_Passager	Num_Vol	Date
101	1	AF123	2024-01-15
102	2	LH456	2024-01-20

Relation : La **clé étrangère** `ID_Passager` dans la table Réservations relie un passager à ses réservations.

Bases de données NoSQL :

Conçues pour les données non structurées (documents, graphes, etc.). (Ex : MongoDB)

Définition :

Les bases de données NoSQL sont conçues pour gérer des données **non structurées ou semi-structurées**. Contrairement aux bases relationnelles, elles n'utilisent pas de schéma strict et ne dépendent pas du SQL pour manipuler les données.

Types de bases NoSQL :

- **Bases orientées documents** : Stockent des données sous forme de documents (JSON, BSON). Exemple : MongoDB.
- **Bases orientées colonnes** : Optimisées pour l'analyse de données massives. Exemple : Cassandra.
- **Bases de graphes** : Utilisées pour les données liées (réseaux sociaux, chemins). Exemple : Neo4j.
- **Caches clé-valeur** : Pour les données rapides à récupérer. Exemple : Redis.

Exemple concret :

Avec MongoDB, une base de données pour une boutique en ligne pourrait être organisée comme suit (sous forme de documents JSON) :

Document Produits :

```
{
  "_id": "P123",
  "nom": "Smartphone",
  "marque": "Samsung",
  "prix": 699,
  "caractéristiques": {
```

```
    "écran": "6.4 pouces",
    "RAM": "8 Go",
    "batterie": "4500 mAh"
  }
}
```

Document Commandes :

```
{
  "_id": "C001",
  "client": "Alice Dupont",
  "produits": [
    {"id_produit": "P123", "quantité": 1},
    {"id_produit": "P124", "quantité": 2}
  ],
  "total": 1398
}
```

Bases de données hiérarchiques :

Organisées en structure arborescente.

Définition :

Les bases de données hiérarchiques organisent les données sous forme d'**arborescence** ou de structure en arbre, où chaque nœud est lié à un nœud parent. Ce modèle est adapté pour des données ayant une relation parent-enfant claire.

Caractéristiques :

- Les relations entre les données suivent une structure arborescente.
- Accès rapide aux données dans des scénarios où la hiérarchie est importante.
- Moins flexible comparé aux bases relationnelles ou NoSQL.

Exemple concret :

Un exemple typique serait un organigramme d'entreprise.

Table hiérarchique Employés :

ID_Employé	Nom	Poste	ID_Supérieur
1	Alice Dupont	PDG	NULL
2	Bob Martin	Manager	1
3	Claire Dubois	Développeur	2

Ici :

- Alice Dupont est au sommet (PDG).
- Bob Martin est un Manager sous Alice.
- Claire Dubois travaille sous Bob.

Les bases hiérarchiques sont souvent utilisées dans des systèmes anciens, comme IBM Information Management System (IMS).

Bases de données orientées objets :

Stockent des données sous forme d'objets.

Définition :

Les bases de données orientées objets stockent les données sous forme d'**objets**, similaires aux concepts utilisés en programmation orientée objet (POO). Chaque objet contient des données (attributs) et des méthodes (fonctions).

Caractéristiques :

- Intègrent des concepts POO comme l'héritage, les classes, et les encapsulations.
- Utile pour des systèmes où les données complexes (images, vidéos, géodonnées) doivent être modélisées directement sous forme d'objets.

Exemple concret :

Dans une base orientée objets pour une application de géolocalisation, vous pourriez avoir une classe `Lieu` avec des sous-classes `Restaurant` et `Musée`.

Classe Lieu :

- Attributs : `Nom`, `Coordonnées_GPS`.
- Méthodes : `Calculer_Distance(autre_lieu)`.

Sous-classe Restaurant :

- Attributs spécifiques : `Cuisine`, `Prix_Moyen`.

Instance dans la base de données :

```
{
  "type": "Restaurant",
  "nom": "Chez Alice",
  "coordonnées_GPS": "48.8566, 2.3522",
  "cuisine": "Française",
  "prix_moyen": 30
}
```

différences entre base orientée objet et mongoDB :

Critère	Base orientée objets	MongoDB (NoSQL)
Structure des données	Objets avec classes, héritage et méthodes	Documents JSON flexibles
Relations	Héritage, agrégation	Champs imbriqués ou références
Langage de manipulation	API orientées objets (POO)	MongoDB Query Language (MQL)
Performance	Efficace pour des données complexes et logiques métier	Performant pour des données massives, hétérogènes
Flexibilité	Faible, dépend des classes	Très élevée, pas de schéma rigide
Cas d'utilisation	Simulation, multimédia, systèmes POO	Big Data, catalogues, logs, applications web

Tableau résumé des types de bases de données

Type de base de données	Avantages	Désavantages	Cas d'utilisation	Exemples

Relationnelle	- Structure stricte et organisée (schémas définis).	- Schéma rigide, peu adapté aux données non structurées.	- Systèmes bancaires et financiers (transactions fiables).	MySQL, PostgreSQL
	- Fiabilité des relations entre données via clés primaires/ étrangères.	- Performances limitées avec des volumes massifs ou des écritures fréquentes.	- CRM, ERP, applications e-commerce (utilisateurs, produits, commandes).	Oracle Database
	- Support du SQL pour manipuler et interroger facilement les données.			
NoSQL	- Très flexible (pas de schéma rigide).	- Faibles garanties ACID pour les transactions complexes.	- Applications web/mobiles nécessitant une scalabilité massive.	MongoDB, Cassandra
	- Scalabilité horizontale pour gérer de grandes quantités de données.	- Risque de duplication de données (absence de normalisation).	- Gestion de contenu, big data, catalogues de produits.	Redis, Neo4j
	- Modèles variés : document, graphe, clé-valeur, colonnes.	- Courbe d'apprentissage pour des langages spécifiques (ex. : MQL pour MongoDB).		
Hiérarchique	- Structure arborescente simple pour des relations parent-enfant.	- Rigide, difficile à adapter si les relations deviennent complexes.	- Gestion de fichiers, systèmes LDAP, catalogues	IBM IMS, Windows Registry

			produits simples.	
	- Performant pour des relations fixes et des accès rapides.	- Faible capacité pour des relations complexes ou croisées.	- Applications nécessitant un traitement rapide de données hiérarchiques.	
Orientée objets	- Modèle naturel pour représenter des données sous forme d'objets (OOP).	- Moins mature et moins adopté que les bases relationnelles.	- Applications nécessitant une forte intégration avec des langages OOP.	db4o, ObjectDB
	- Pas besoin de traduction entre objets et tables (mapping).	- Moins adapté aux transactions complexes et au reporting.	- Jeux vidéo, CAO/FAO, simulations scientifiques.	

4. Processus de Conception de Base de Données

En général, la création d'un **diagramme relationnel** (ou d'un **modèle conceptuel des données**) se fait **avant** la création des tables réelles dans une base de données. Le diagramme relationnel est une **représentation logique** de la base de données, tandis que les tables physiques sont une **implémentation** de ce modèle dans un système de gestion de base de données (SGBD).

Voici comment cela se passe généralement dans la conception :

1. **Analyse des besoins (modèle conceptuel)** :
    - Avant de créer un diagramme relationnel, il faut analyser les besoins du système et comprendre les données à gérer. Cela permet de comprendre les entités et leurs relations. Par exemple, si vous créez une application de bibliothèque, vous devrez identifier des entités comme **Emprunteur**, **Livre**, et les **Emprunts**.
2. **Modélisation conceptuelle (diagramme entité-association)** :
    - Une fois que vous avez identifié les entités, un diagramme entité-association (E/A) est souvent créé pour visualiser les entités et leurs

relations sans trop se soucier des détails techniques. C'est une **vue abstraite** des données.

3. **Modélisation logique (diagramme relationnel)** :
    - À partir du modèle conceptuel, vous créez un **diagramme relationnel** qui est une traduction de ce modèle conceptuel en termes de tables et de relations, en prenant en compte les **clés primaires** et **clés étrangères**. Ce diagramme est encore indépendant du SGBD que vous allez utiliser.
4. **Modélisation physique (création des tables)** :
    - Enfin, une fois que le modèle relationnel est validé, vous pouvez créer les tables dans le SGBD. Vous transformez les entités et relations du diagramme en tables physiques, avec des types de données spécifiques pour chaque colonne, et appliquez des contraintes (par exemple, contraintes d'intégrité).

Installation

Partie 2 : Les bases de données relationnelles

Les types de données

Partie 3 : Modélisation et conception

Normalisation

Exercices : Normalisation

Création d'une Table en SQL avec Intégrité des Données

Exercices Relations

EXERCICES

Partie 4 : Introduction au langage SQL

ressources : https://sqlzoo.net/wiki/SELECT_basics