Gérer le stockage des données

Table des matières

Module 1 : Introduction à la gestion des données

🎯 Objectifs pédagogiques :

À la fin de ce module, l'apprenant doit être capable de :

- Comprendre ce que sont les données et leur rôle dans les systèmes informatiques.
- Identifier les différents types de données.
- Comprendre les enjeux liés à la gestion des données dans une organisation.

---

1. 📌 Définition des données

🗣️ *Qu'est-ce qu'une donnée ?*

Une **donnée** est une représentation élémentaire d'une information brute, sans contexte ni interprétation. Elle peut être qualitative ou quantitative.

- **Exemples** :
  - `42` → un nombre sans contexte.
  - `"Dupont"` → un nom.
  - `"2024-12-01"` → une date.

🗣️ Différence entre donnée, information et connaissance :

Terme	Définition	Exemple
Donnée	Élément brut	`32`
Information	Donnée avec du contexte	Âge = 32 ans
Connaissance	Interprétation pour la prise de décision	Un client de 32 ans est dans la tranche de population à cibler

---

2. 📊 Types de données

🧱 Données structurées :

- Organisées en tableaux (lignes/colonnes), généralement stockées dans des bases relationnelles (SQL).
- **Exemple** : une base clients avec champs Nom, Email, Téléphone.

🧩 Données semi-structurées :

- Données ayant une structure flexible (tags, balises, paires clé/valeur).
- Stockées sous formats comme JSON, XML.
- **Exemple** :

```
{
  "nom": "Dupont",
  "email": "dupont@mail.com",
  "commandes": [123, 456]
}
```

Données non structurées :

- Sans format défini ni structure exploitable automatiquement.
- **Exemples** : images, vidéos, fichiers PDF, enregistrements audio.

---

3. Pourquoi gérer les données ?

1. Améliorer la prise de décision

- Exploiter les données permet de piloter l'activité avec des indicateurs fiables (KPI).
- Exemple : analyser les ventes pour ajuster les stocks.

2. Répondre aux exigences légales

- Conformité au **RGPD**, **HIPAA**, ou autres règlements selon les secteurs.

3. Assurer la sécurité et la confidentialité

- Prévention contre les fuites de données, les accès non autorisés, la perte de données.

4. Optimiser les performances système

- Une bonne gestion des index, des partitions, des formats de stockage réduit les temps de traitement.

---

4. Cycle de vie des données

Étape	Description	Exemple
Création	Collecte via formulaires, capteurs, imports	Client s'inscrit
Stockage	Sauvegarde dans une base, un fichier, un cloud	Enregistrement dans MySQL
Traitement	Calculs, tris, filtrages, IA	Filtrer les clients par région
Diffusion	Mise à disposition des utilisateurs	Dashboard, API, email
Archivage/Suppression	Conformité légale, optimisation	Suppression après 3 ans d'inactivité

---

5. Exemples d'utilisation concrète des données

- **E-commerce** : personnalisation de l'expérience utilisateur, recommandations de produits.
- **Santé** : dossiers patients électroniques, suivi d'évolution.
- **Banque** : détection de fraude par analyse en temps réel des transactions.

---

6. ⚠ Enjeux et défis de la gestion des données

Enjeu	Description	Risque en cas de négligence
Sécurité	Protéger contre vols et fuites	Amendes, atteinte à l'image
Qualité	Fiabilité et exactitude des données	Mauvaise décision
Volume	Gérer la montée en charge	Systèmes saturés
Accessibilité	Bon utilisateur, bon moment	Frein à la productivité
Conformité	Respect des lois	Sanctions légales

---

7. 🛠 Technologies et outils courants

- **Bases de données** : MySQL, PostgreSQL, SQL Server.
- **Stockage cloud** : AWS S3, Google Cloud Storage, Azure Blob.
- **Outils de traitement** : Python (Pandas), ETL (Talend, Airbyte).
- **Sécurité & gouvernance** : IAM, chiffrement, audit logs.

📝 Exercice pratique (optionnel)

Objectif : Identifier les types de données et proposer un mode de stockage adapté.

Scénario : une entreprise collecte les données suivantes :

- Nom du client
- Photo du produit
- Historique des commandes
- Avis clients (sous forme de texte libre)

Questions :

1. Classez chaque donnée selon son type (structurée, semi-structurée, non structurée).
2. Quel outil ou format utiliseriez-vous pour stocker chacune ?

✓ Résumé du module

- Les données sont à la base de tout système informatique.
- Bien les gérer permet d'en tirer de la valeur, d'éviter des risques et d'assurer la conformité.
- Il existe plusieurs types de données avec des méthodes de gestion différentes.
- Le cycle de vie des données aide à structurer leur traitement du début à la fin.

Module 2 : Introduction générale aux bases de données et notion de SGBDR

🎯 Objectifs pédagogiques

À l'issue de ce module, l'apprenant sera capable de :

- Expliquer ce qu'est une base de données.
- Identifier le rôle et les composants d'un Système de Gestion de Base de Données Relationnelle (SGBDR).
- Comprendre les concepts clés du modèle relationnel (tables, clés, relations).
- Distinguer les types de bases de données selon leur structure et usage.

1. 🎁 Qu'est-ce qu'une base de données ?

Une **base de données** est un ensemble organisé et structuré d'informations stockées électroniquement sur un système informatique. Elle permet de :

- **Conserver** des données de manière fiable,
- **Accéder** facilement à ces données,
- **Manipuler** et **gérer** les données (ajout, modification, suppression, recherche).

🔹 Exemple concret :

Une application de gestion d'école peut stocker :

- Les élèves,
- Les enseignants,
- Les notes,
- Les emplois du temps.

Ces éléments sont enregistrés dans des **tables** au sein d'une base de données.

2. ⚙ Qu'est-ce qu'un SGBD / SGBDR ?

🔹 Définition :

Un **Système de Gestion de Base de Données (SGBD)** est un logiciel qui permet d'interagir avec une base de données : stocker, interroger, modifier et sécuriser les données.

Un **SGBDR (Relationnel)** est un SGBD qui organise les données en **tables relationnelles**, avec un **modèle mathématique basé sur l'algèbre relationnelle**.

🔹 Exemples de SGBDR :

- **MySQL**
- **PostgreSQL**
- **Oracle Database**
- **SQL Server**
- **MariaDB**

3. 🔗 Le modèle relationnel : concepts fondamentaux

Concept	Description	Exemple
Table	Ensemble de lignes et de colonnes	`Clients`, `Commandes`
Enregistrement (ligne)	Une instance de donnée	`Client : ID = 1, Nom = Dupont`
Attribut (colonne)	Un champ de la table	`Nom`, `Email`, `Téléphone`
Clé primaire	Identifie de manière unique chaque ligne	`ID`
Clé étrangère	Référence une clé primaire d'une autre table	`ClientID` dans `Commandes`
Relation	Lien logique entre deux tables via les clés	Un client → plusieurs commandes

---

📚 Exemple de tables relationnelles :

Table Clients

ID	Nom	Email
1	Dupont	dupont@mail.com
2	Martin	martin@mail.com

Table Commandes

ID	ClientID	Date	Total
1	1	2024-11-01	125€
2	2	2024-11-03	90€

🔗 Relation :
`ClientID` dans la table **Commandes** est une **clé étrangère** pointant vers `ID` dans **Clients**.

---

4. 🧰 Avantages d'un SGBDR

Avantage	Description
Intégrité des données	Respect des contraintes (unicité, relations)
Sécurité	Contrôle des droits d'accès
Requêtage puissant	Utilisation du langage SQL
Cohérence	Transactions atomiques, cohérentes, isolées, durables (ACID)
Maintenance facilitée	Centralisation et outils d'administration

---

5. 🔐 Les propriétés ACID d'un SGBDR

Propriété	Description
Atomicité	Une transaction est un tout : elle réussit ou échoue totalement.
Cohérence	L'état de la base respecte les contraintes définies.
Isolation	Les transactions simultanées n'interfèrent pas entre elles.
Durabilité	Une fois validées, les modifications persistent, même après une panne.

---

6. ⚙️ Fonctionnalités d'un SGBDR

- Création de schémas et de tables
- Insertion, modification, suppression, recherche de données (SQL)
- Gestion des utilisateurs et des permissions
- Sauvegarde et restauration
- Indexation pour améliorer les performances
- Transactions

---

7. 🔍 Comparaison rapide : SGBDR vs NoSQL

Critère	SGBDR	NoSQL
Structure	Tables (relations)	Documents, graphes, colonnes, clé/valeur
Langage	SQL	Varies (MongoDB Query, CQL…)
Schéma	Fixe (schéma strict)	Flexible
Scalabilité	Verticale	Horizontale
Cas d'usage	Systèmes transactionnels	Big Data, temps réel, données non structurées

---

📝 Exercice de mise en pratique

Scénario : Créez un schéma de base de données simple pour une bibliothèque contenant :

- des **livres**,
- des **auteurs**,
- des **emprunts**.

Questions :

1. Quelles sont les tables nécessaires ?
2. Déterminez les clés primaires et étrangères.
3. Proposez une relation entre les entités.

✓ Résumé du module

- Une base de données est une structure utilisée pour stocker et organiser les données.
- Un SGBDR gère l'accès, la modification et la protection des données stockées dans des tables.
- Le modèle relationnel repose sur des concepts rigoureux : tables, clés, relations.
- Les SGBDR sont robustes, sûrs et adaptés aux systèmes transactionnels.

Quiz - Module 2 : SGBDR et bases de données relationnelles

Partie 1 : Questions à choix multiples (QCM)

1. Quel est le rôle principal d'un SGBDR ?
A. Afficher les pages web d'un site
B. Gérer et organiser les fichiers systèmes
C. Gérer, stocker et interroger des données organisées en tables
D. Convertir les données en langage machine

✓ Réponse attendue : **C**

---

2. Dans un modèle relationnel, une "clé primaire" sert à :
A. Créer des liens vers d'autres bases de données
B. Déterminer le mot de passe d'accès
C. Identifier de manière unique chaque enregistrement dans une table
D. Chiffrer les données de la table

✓ Réponse attendue : **C**

---

3. Quelle affirmation est vraie à propos des SGBDR ?
A. Ils stockent les données exclusivement au format XML
B. Ils n'autorisent pas plusieurs utilisateurs à accéder à la base simultanément
C. Ils permettent l'intégrité des données via des contraintes
D. Ils ne peuvent pas être utilisés avec des données textuelles

✓ Réponse attendue : **C**

---

4. Lequel de ces logiciels est un SGBDR ?
A. MongoDB
B. Oracle
C. Redis
D. Firebase

✓ Réponse attendue : **B**

Partie 2 : Vrai / Faux

5. Une clé étrangère permet de lier deux tables entre elles.
✅ Réponse : **Vrai**

6. Une base de données relationnelle ne peut contenir qu'une seule table.
✅ Réponse : **Faux**

7. Le langage SQL est utilisé uniquement pour créer des interfaces graphiques.
✅ Réponse : **Faux**

8. Le modèle relationnel repose sur des concepts mathématiques.
✅ Réponse : **Vrai**

Partie 3 : Questions ouvertes

9. Expliquez en quelques mots la différence entre une clé primaire et une clé étrangère.
✅ Réponse attendue :

- **Clé primaire** : identifie de manière unique chaque ligne d'une table.
- **Clé étrangère** : fait référence à la clé primaire d'une autre table pour établir une relation entre deux tables.

---

10. Donnez un exemple simple de deux tables reliées par une clé étrangère.

✅ Exemple attendu :

- Table **Clients** (ID, Nom, Email)
- Table **Commandes** (ID, Date, ClientID)

`ClientID` est une clé étrangère qui fait référence à `ID` dans **Clients**.

Exercice : Modélisation d'une base de données pour une bibliothèque

🎯 Objectif :

Concevoir une base de données relationnelle simple permettant de :

- Stocker des informations sur les **livres**, les **auteurs**, et les **emprunts**.
- Comprendre les relations entre ces entités.
- Identifier les clés primaires et étrangères.

Énoncé :

Une bibliothèque souhaite créer une base de données pour gérer :

1. Les **livres** (titre, date de publication, nombre d'exemplaires, auteur).
2. Les **auteurs** (nom, prénom, nationalité).
3. Les **lecteurs** (nom, prénom, email).
4. Les **emprunts** (date d'emprunt, date de retour, livre emprunté, lecteur concerné).

Correction / Solution

Étape 1 : Identification des tables

Nous avons besoin de **quatre tables principales** :

- `Auteurs`
- `Livres`
- `Lecteurs`
- `Emprunts`

Étape 2 : Définir les attributs et clés

Table `Auteurs`

Nom du champ	Type	Description
id_auteur	INT	Clé primaire
nom	VARCHAR	Nom de l'auteur
prenom	VARCHAR	Prénom de l'auteur
nationalite	VARCHAR	Nationalité de l'auteur

Table `Livres`

Nom du champ	Type	Description
id_livre	INT	Clé primaire
titre	VARCHAR	Titre du livre
date_pub	DATE	Date de publication
nb_exemplaires	INT	Nombre d'exemplaires disponibles
id_auteur	INT	Clé étrangère vers `Auteurs.id_auteur`

🔹 *Table `Lecteurs`*

Nom du champ	Type	Description
id_lecteur	INT	🔑 Clé primaire
nom	VARCHAR	Nom du lecteur
prenom	VARCHAR	Prénom du lecteur
email	VARCHAR	Adresse e-mail

🔹 *Table `Emprunts`*

Nom du champ	Type	Description
id_emprunt	INT	🔑 Clé primaire
date_emprunt	DATE	Date de l'emprunt
date_retour	DATE	Date de retour prévue
id_livre	INT	🔗 Clé étrangère vers `Livres.id_livre`
id_lecteur	INT	🔗 Clé étrangère vers `Lecteurs.id_lecteur`

🔹 Étape 3 : Relations entre les tables

- **Un auteur** peut avoir **plusieurs livres** → relation **1 → N**
- **Un livre** peut être emprunté **plusieurs fois** → relation **1 → N**
- **Un lecteur** peut faire **plusieurs emprunts** → relation **1 → N**

🗂️ Étape 4 : Schéma relationnel simplifié

```
scss
AUTEURS (id_auteur PK, nom, prenom, nationalite)

LIVRES (id_livre PK, titre, date_pub, nb_exemplaires, id_auteur FK)

LECTEURS (id_lecteur PK, nom, prenom, email)

EMPRUNTS (id_emprunt PK, date_emprunt, date_retour, id_livre FK, id_lecteur FK)
```

🛠️ Étape 5 : Exemple de requêtes SQL de création

🔹 *Création des tables (extrait)*

```
CREATE TABLE Auteurs (
    id_auteur INT PRIMARY KEY,
    nom VARCHAR(50),
    prenom VARCHAR(50),
    nationalite VARCHAR(50)
);

CREATE TABLE Livres (
    id_livre INT PRIMARY KEY,
    titre VARCHAR(100),
```

```
    date_pub DATE,
    nb_exemplaires INT,
    id_auteur INT,
    FOREIGN KEY (id_auteur) REFERENCES Auteurs(id_auteur)
);

CREATE TABLE Lecteurs (
    id_lecteur INT PRIMARY KEY,
    nom VARCHAR(50),
    prenom VARCHAR(50),
    email VARCHAR(100)
);

CREATE TABLE Emprunts (
    id_emprunt INT PRIMARY KEY,
    date_emprunt DATE,
    date_retour DATE,
    id_livre INT,
    id_lecteur INT,
    FOREIGN KEY (id_livre) REFERENCES Livres(id_livre),
    FOREIGN KEY (id_lecteur) REFERENCES Lecteurs(id_lecteur)
);
```

Module 3 : Création et alimentation de bases de données SQL

Objectifs pédagogiques

À la fin de ce module, l'apprenant sera capable de :

- Créer une base de données relationnelle.
- Créer des tables avec des types de données adaptés.
- Définir des clés primaires et étrangères.
- Alimenter la base avec des données via des requêtes `INSERT`.

1. Création d'une base de données SQL

Dans un **SGBDR** comme **MySQL**, **PostgreSQL**, ou **SQL Server**, une base de données est créée à l'aide de la commande SQL :

```
CREATE DATABASE Bibliotheque;
```

⚠ Certains SGBD nécessitent d'être connectés en tant qu'administrateur ou superutilisateur pour créer une base.

2. Sélection de la base de données

Avant de créer des tables, il faut se placer dans la base :

```
USE Bibliotheque;  -- pour MySQL
-- ou
\c Bibliotheque;   -- pour PostgreSQL
```

3. Création de tables

Les **tables** sont créées avec la commande `CREATE TABLE`, en définissant pour chaque colonne :

- un nom,
- un type de données (`INT`, `VARCHAR`, `DATE`, etc.),
- des **contraintes** (clé primaire, unique, non null, etc.).

Exemple : Création d'une table `Clients`

```
sql
CopierModifier
CREATE TABLE Clients (
    id INT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE,
    date_inscription DATE
);
```

Principaux types de données :

Type SQL	Utilisation	Exemple
`INT`	Entiers	`42`
`VARCHAR(n)`	Texte de longueur limitée	`'Dupont'`
`TEXT`	Texte long	Article de blog
`DATE`	Date (AAAA-MM-JJ)	`'2024-11-01'`
`BOOLEAN`	Vrai / Faux	`TRUE`, `FALSE`
`DECIMAL(10,2)`	Nombres décimaux (ex : prix)	`99.99`

4. Définir les clés

Clé primaire (`PRIMARY KEY`)

- Identifie de manière **unique** chaque enregistrement.
- Obligatoire pour chaque table.

```
id INT PRIMARY KEY
```

Clé étrangère (`FOREIGN KEY`)

- Crée une **relation entre deux tables**.
- Référence une clé primaire d'une autre table.

```
id_client INT,
FOREIGN KEY (id_client) REFERENCES Clients(id)
```

5. Insertion de données (`INSERT`)

La commande `INSERT INTO` permet d'ajouter une ou plusieurs lignes dans une table.

Exemple simple :

```
INSERT INTO Clients (id, nom, email, date_inscription)
VALUES (1, 'Martin', 'martin@mail.com', '2024-11-03');
```

Insertion multiple :

```
INSERT INTO Clients (id, nom, email, date_inscription) VALUES
(2, 'Dupont', 'dupont@mail.com', '2024-11-04'),
(3, 'Durand', 'durand@mail.com', '2024-11-05');
```

---

6. Ajout d'une contrainte après création (optionnel)

On peut ajouter une clé étrangère **après** la création d'une table :

```
ALTER TABLE Commandes
ADD CONSTRAINT fk_client
FOREIGN KEY (id_client) REFERENCES Clients(id);
```

---

7. Cas pratique

Créons un petit modèle pour une boutique en ligne :

Table `Produits`

```
CREATE TABLE Produits (
    id INT PRIMARY KEY,
    nom VARCHAR(100),
    prix DECIMAL(10,2)
);
```

Table `Commandes`

```
CREATE TABLE Commandes (
    id INT PRIMARY KEY,
    date_commande DATE,
    id_produit INT,
    FOREIGN KEY (id_produit) REFERENCES Produits(id)
);
```

Ajout de données :

```
INSERT INTO Produits (id, nom, prix) VALUES
(1, 'Clavier', 49.99),
(2, 'Souris', 29.99);

INSERT INTO Commandes (id, date_commande, id_produit) VALUES
(1, '2025-01-15', 1),
(2, '2025-01-16', 2);
```

Exercice d'application

Créez une base de données pour gérer un parc automobile contenant :

- des **voitures** (immatriculation, marque, modèle, année),
- des **conducteurs** (nom, prénom, permis),

- des **locations** (date début, date fin, voiture, conducteur).

Questions :

1. Écrivez les commandes SQL pour créer les trois tables.
2. Définissez les clés primaires et étrangères.
3. Insérez au moins 2 lignes par table.

✓ Résumé du module

- La création d'une base SQL se fait avec `CREATE DATABASE`.
- Les tables sont définies avec des types de données adaptés et des contraintes.
- Les relations sont établies avec des clés étrangères.
- L'insertion de données se fait avec `INSERT INTO`.
- Une bonne structuration dès la création garantit la cohérence des données.

Correction complète : Parc Automobile

Objectif rappelé :

Créer une base de données pour gérer :

- Les **voitures**
- Les **conducteurs**
- Les **locations**

Étape 1 : Création de la base de données

```
CREATE DATABASE ParcAutomobile;
USE ParcAutomobile;  -- pour MySQL
```

Étape 2 : Création des tables

Table `Voitures`

```
CREATE TABLE Voitures (
    immatriculation VARCHAR(10) PRIMARY KEY,
    marque VARCHAR(50),
    modele VARCHAR(50),
    annee INT
);
```

- `immatriculation` sert de **clé primaire**, car elle identifie de manière unique chaque véhicule.

Table `Conducteurs`

```
CREATE TABLE Conducteurs (
    id_conducteur INT PRIMARY KEY,
    nom VARCHAR(50),
    prenom VARCHAR(50),
    num_permis VARCHAR(20) UNIQUE
);
```

- `id_conducteur` est un identifiant unique.
- `num_permis` est déclaré `UNIQUE` pour éviter les doublons.

Table `Locations`

```
CREATE TABLE Locations (
    id_location INT PRIMARY KEY,
    date_debut DATE,
    date_fin DATE,
    immatriculation VARCHAR(10),
    id_conducteur INT,
    FOREIGN KEY (immatriculation) REFERENCES Voitures(immatriculation),
    FOREIGN KEY (id_conducteur) REFERENCES Conducteurs(id_conducteur)
);
```

- Cette table relie les voitures et les conducteurs avec **deux clés étrangères**.

Étape 3 : Insertion de données

Données dans `Voitures`

```
INSERT INTO Voitures (immatriculation, marque, modele, annee) VALUES
('AB-123-CD', 'Peugeot', '208', 2020),
('XY-987-ZT', 'Renault', 'Clio', 2019);
```

Données dans `Conducteurs`

```
INSERT INTO Conducteurs (id_conducteur, nom, prenom, num_permis) VALUES
(1, 'Durand', 'Alice', 'PERM12345'),
(2, 'Lemoine', 'Marc', 'PERM67890');
```

Données dans `Locations`

```
INSERT INTO Locations (id_location, date_debut, date_fin, immatriculation, id_conducteur)
VALUES
(1, '2025-05-01', '2025-05-05', 'AB-123-CD', 1),
(2, '2025-05-03', '2025-05-07', 'XY-987-ZT', 2);
```

Étape 4 : Requêtes de test

Afficher toutes les locations avec les noms des conducteurs et les modèles de voiture

```
SELECT L.id_location, C.nom, C.prenom, V.marque, V.modele, L.date_debut, L.date_fin
FROM Locations L
JOIN Conducteurs C ON L.id_conducteur = C.id_conducteur
JOIN Voitures V ON L.immatriculation = V.immatriculation;
```

Afficher toutes les voitures qui n'ont pas encore été louées

```
SELECT *
FROM Voitures
WHERE immatriculation NOT IN (
    SELECT immatriculation FROM Locations
);
```

✓ Résumé du schéma relationnel

- **Voitures** (`immatriculation`) ← relation 1 → N → **Locations**
- **Conducteurs** (`id_conducteur`) ← relation 1 → N → **Locations**

Module 4 : Interrogation d'une base de données SQL

Objectifs pédagogiques

À l'issue de ce module, l'apprenant saura :

- Utiliser le langage SQL pour interroger une base de données.
- Utiliser les commandes de base : `SELECT`, `WHERE`, `ORDER BY`, `GROUP BY`, `JOIN`.
- Extraire des données selon différents critères (filtres, conditions, agrégats).
- Comprendre et écrire des requêtes simples à intermédiaires.

1. La commande `SELECT` : lire les données

Structure de base :

```
SELECT colonne1, colonne2
FROM nom_table;
```

Exemple :

```
SELECT nom, prenom FROM Conducteurs;
```

- Cette requête retourne tous les noms et prénoms de la table `Conducteurs`.

2. Utiliser `WHERE` : filtrer les résultats

```
SELECT * FROM Voitures
WHERE marque = 'Renault';
```

Opérateurs logiques :

- `=` : égal
- `<>` ou `!=` : différent
- `<`, `>`, `<=`, `>=`
- `BETWEEN a AND b` : compris entre deux valeurs
- `IN (val1, val2)` : correspond à l'une des valeurs
- `LIKE` : recherche avec motif (`%` pour plusieurs caractères, `_` pour un caractère)
- `IS NULL` / `IS NOT NULL`

Exemple avec `LIKE` :

```
SELECT * FROM Clients
WHERE email LIKE '%@gmail.com';
```

🔍 3. Trier les résultats avec `ORDER BY`

```
SELECT * FROM Produits
ORDER BY prix DESC;
```

- `ASC` (par défaut) : ordre croissant
- `DESC` : ordre décroissant

🔍 4. Les fonctions d'agrégation

Fonction	Description	Exemple
`COUNT()`	Compter les lignes	`COUNT(*)`
`SUM()`	Somme d'une colonne	`SUM(prix)`
`AVG()`	Moyenne	`AVG(prix)`
`MIN()`	Valeur minimale	`MIN(date)`
`MAX()`	Valeur maximale	`MAX(prix)`

🔹 Exemple :

```
SELECT COUNT(*) FROM Conducteurs;
SELECT AVG(prix) FROM Produits;
```

🔍 5. Regrouper avec `GROUP BY`

🔹 Objectif :

Regrouper des lignes ayant des valeurs communes pour effectuer des calculs par groupe.

```
SELECT marque, COUNT(*) AS nombre_voitures
FROM Voitures
GROUP BY marque;
```

🔍 6. Regrouper avec condition : `HAVING`

Contrairement à `WHERE`, `HAVING` s'utilise **après un `GROUP BY`**.

```
SELECT marque, COUNT(*) AS nb
FROM Voitures
GROUP BY marque
HAVING COUNT(*) > 1;
```

🔍 7. Les jointures (`JOIN`)

Les jointures permettent de **lier plusieurs tables** ensemble.

🔹 Syntaxe générale :

```
SELECT ...
FROM TableA
JOIN TableB ON TableA.cle = TableB.cle
```

`INNER JOIN` : ne garde que les correspondances

```
SELECT L.id_location, C.nom, V.marque
FROM Locations L
INNER JOIN Conducteurs C ON L.id_conducteur = C.id_conducteur
INNER JOIN Voitures V ON L.immatriculation = V.immatriculation;
```

`LEFT JOIN` : conserve tous les enregistrements de la table de gauche

```
SELECT V.immatriculation, L.date_debut
FROM Voitures V
LEFT JOIN Locations L ON V.immatriculation = L.immatriculation;
```

Cas pratique

Sur la base de données du **parc automobile**, écrivez les requêtes suivantes :

1. Afficher tous les conducteurs dont le nom commence par "D".
2. Lister les voitures dont l'année est postérieure à 2020.
3. Afficher le nombre de locations par conducteur.
4. Afficher les voitures qui **n'ont jamais été louées**.
5. Afficher la liste complète des locations, avec le nom du conducteur et le modèle de voiture.

Corrigé

```
-- 1.
SELECT * FROM Conducteurs
WHERE nom LIKE 'D%';

-- 2.
SELECT * FROM Voitures
WHERE annee > 2020;

-- 3.
SELECT C.nom, COUNT(*) AS nb_locations
FROM Locations L
JOIN Conducteurs C ON L.id_conducteur = C.id_conducteur
GROUP BY C.nom;

-- 4.
SELECT * FROM Voitures
WHERE immatriculation NOT IN (
    SELECT immatriculation FROM Locations
);

-- 5.
SELECT L.id_location, C.nom, C.prenom, V.modele, L.date_debut
FROM Locations L
JOIN Conducteurs C ON L.id_conducteur = C.id_conducteur
JOIN Voitures V ON L.immatriculation = V.immatriculation;
```

✓ Résumé du module

- `SELECT` permet d'extraire des données d'une ou plusieurs tables.
- Les filtres `WHERE`, les tris `ORDER BY`, les regroupements `GROUP BY` et les jointures `JOIN` sont les bases de l'interrogation SQL.
- Bien maîtriser ces commandes permet de créer des tableaux de bord, des rapports, ou d'alimenter des applications.

Interrogation d'une base de données SQL

avec **corrections** complètes.

Contexte (base ParcAutomobile)

Nous utilisons les 3 tables suivantes déjà créées dans les modules précédents :

- `Voitures(immatriculation, marque, modele, annee)`
- `Conducteurs(id_conducteur, nom, prenom, num_permis)`
- `Locations(id_location, date_debut, date_fin, immatriculation, id_conducteur)`

Exercice 1 : Sélection simple

Question :
Afficher les noms et prénoms de tous les conducteurs ayant une adresse e-mail se terminant par "@gmail.com".
(Supposons qu'un champ `email` a été ajouté à la table `Conducteurs`)

Correction :

```
SELECT nom, prenom
FROM Conducteurs
WHERE email LIKE '%@gmail.com';
```

Exercice 2 : Filtrage avec conditions

Question :
Lister tous les véhicules de marque "Peugeot" immatriculés après 2019.

Correction :

```
SELECT *
FROM Voitures
WHERE marque = 'Peugeot' AND annee > 2019;
```

➽ Exercice 3 : Trie des résultats

Question :
Afficher les voitures classées par année décroissante.

Correction :

```
SELECT *
FROM Voitures
ORDER BY annee DESC;
```

➽ Exercice 4 : Agrégation + regroupement

Question :
Combien de locations ont été effectuées par conducteur ?

Correction :

```
SELECT id_conducteur, COUNT(*) AS nb_locations
FROM Locations
GROUP BY id_conducteur;
```

➽ Exercice 5 : Filtrage après regroupement (HAVING)

Question :
Afficher uniquement les conducteurs ayant effectué **plus de 1 location**.

Correction :

```
SELECT id_conducteur, COUNT(*) AS nb_locations
FROM Locations
GROUP BY id_conducteur
HAVING COUNT(*) > 1;
```

➽ Exercice 6 : Jointure simple

Question :
Afficher les noms des conducteurs et les dates de leurs locations.

Correction :

```
SELECT C.nom, C.prenom, L.date_debut, L.date_fin
FROM Locations L
JOIN Conducteurs C ON L.id_conducteur = C.id_conducteur;
```

➽ Exercice 7 : Jointure multiple

Question :
Afficher la liste complète des locations avec :

- nom/prénom du conducteur,
- modèle et marque de la voiture,
- date de début et de fin.

Correction :

```
SELECT C.nom, C.prenom, V.marque, V.modele, L.date_debut, L.date_fin
FROM Locations L
JOIN Conducteurs C ON L.id_conducteur = C.id_conducteur
JOIN Voitures V ON L.immatriculation = V.immatriculation;
```

➖ Exercice 8 : Sous-requête

Question :
Afficher les informations des voitures **qui n'ont jamais été louées**.

Correction :

```
SELECT *
FROM Voitures
WHERE immatriculation NOT IN (
    SELECT immatriculation FROM Locations
);
```

➖ Exercice 9 : TOP N résultats

Question :
Afficher les **3 conducteurs** ayant effectué **le plus de locations**.

Correction (MySQL/PostgreSQL) :

```
SELECT C.nom, C.prenom, COUNT(*) AS nb_locations
FROM Locations L
JOIN Conducteurs C ON L.id_conducteur = C.id_conducteur
GROUP BY C.nom, C.prenom
ORDER BY nb_locations DESC
LIMIT 3;
```

➖ Exercice 10 : Recherche conditionnelle avancée

Question :
Afficher tous les conducteurs ayant **loué une voiture de marque "Renault"**.

Correction :

```
SELECT DISTINCT C.nom, C.prenom
FROM Locations L
JOIN Conducteurs C ON L.id_conducteur = C.id_conducteur
JOIN Voitures V ON L.immatriculation = V.immatriculation
WHERE V.marque = 'Renault';
```

Module 5 : La configuration du stockage

Objectifs pédagogiques

À la fin de ce module, l'apprenant sera capable de :

- Comprendre les principes fondamentaux du stockage dans une base de données.
- Identifier les options de stockage possibles (local, réseau, cloud).
- Configurer correctement le stockage d'un SGBD selon les contraintes de performance, sécurité, et fiabilité.
- Choisir un moteur de stockage adapté (InnoDB, MyISAM, etc.).
- Optimiser l'utilisation de l'espace disque.

1. Introduction au stockage dans un SGBD

Un **SGBD** utilise des fichiers systèmes pour stocker :

- les données (tables),
- les index,
- les logs (journalisation),
- les métadonnées (structure des bases),
- les fichiers temporaires.

Ces fichiers sont organisés différemment selon :

- le **moteur de base de données** utilisé (MySQL, PostgreSQL, SQL Server, etc.),
- le **moteur de stockage** (InnoDB, MyISAM...).

2. Types de stockage selon l'environnement

Type de stockage	Description	Usage courant
Stockage local	Sur disque dur interne	Bases de test, petits projets
Stockage NAS/SAN	Réseau (partagé, haute dispo)	Systèmes en entreprise
Stockage SSD	Plus rapide, accès aléatoire optimisé	Bases critiques
Stockage cloud (S3, EBS)	À la demande, élastique	Bases cloud (AWS RDS, Azure SQL...)
RAM Disk / Mémoire	Temporaire, en mémoire volatile	Bases in-memory, caches

3. Configuration du stockage dans MySQL (exemple)

Choix du moteur de stockage :

Chaque table peut utiliser un moteur spécifique.

```
CREATE TABLE Produits (
    id INT,
    nom VARCHAR(100)
) ENGINE=InnoDB;
```

Moteur	Avantages	Limites
InnoDB	Transactions, intégrité, ACID	Légèrement plus lent que MyISAM
MyISAM	Rapide en lecture	Pas de transactions, pas de FK
MEMORY	Très rapide (stockée en RAM)	Volatil, non persistant

☞ InnoDB est généralement recommandé pour les bases relationnelles modernes.

🔹 Emplacement des fichiers de données (ex : MySQL)

- Par défaut, les fichiers sont stockés dans :

```
swift
/var/lib/mysql/         (Linux)
C:\ProgramData\MySQL\   (Windows)
```

- Fichiers principaux :
  - `.frm` : définition de table (obsolète sur MySQL 8)
  - `.ibd` : fichier de données InnoDB
  - `ib_logfile0` / `ib_logfile1` : logs de transactions

📂 4. Partitionnement et tablespaces

🔸 Partitionnement :

Découper logiquement une table en plusieurs segments stockés séparément.

```
CREATE TABLE Logs (
    id INT,
    date_log DATE
)
PARTITION BY RANGE (YEAR(date_log)) (
    PARTITION p2022 VALUES LESS THAN (2023),
    PARTITION p2023 VALUES LESS THAN (2024)
);
```

✅ Avantages :

- Meilleure gestion de très grandes tables
- Meilleure performance sur certaines requêtes

🔸 Tablespaces :

Permettent de séparer physiquement le stockage de différentes bases ou tables sur plusieurs disques.

```
CREATE TABLESPACE ts_clients
  ADD DATAFILE '/data/clients.ibd'
  ENGINE = InnoDB;
```

🔐 5. Bonnes pratiques de configuration

✅ Emplacement physique :

- Stocker les données sur un disque dédié (pas sur le disque système).
- Utiliser **RAID 10** ou **RAID 5** pour la tolérance de panne.

✅ Taille des fichiers :

- Surveiller la taille des fichiers `ibdata` ou `log` : purger régulièrement les logs.

✅ Logs et binlogs :

- Activer les logs (`binlog`, `slow query log`) mais les stocker sur un disque séparé si possible.

✅ TempDB (SQL Server) ou tables temporaires :

- Utiliser un disque rapide (SSD) pour les fichiers temporaires.

🔧 6. Exemple : PostgreSQL

🔹 Configuration des chemins dans `postgresql.conf` :

```
data_directory = '/mnt/postgres_data'
log_directory = '/mnt/logs_pgsql'
```

- **Tablespaces** :

```
CREATE TABLESPACE archive LOCATION '/mnt/tablespace_archive';
CREATE TABLE archives (
    id SERIAL,
    titre TEXT
) TABLESPACE archive;
```

❓ Exercice d'application

Objectif : Optimiser une base PostgreSQL contenant une très grande table `logs` (plus de 500 millions de lignes).

Questions :

1. Comment configurer PostgreSQL pour que cette table utilise un autre disque dur ?
2. Quelle stratégie proposer pour éviter que la base principale ne soit saturée ?
3. Implémentez un partitionnement mensuel de la table `logs`.

✅ Corrigé suggéré

1. Créer un **tablespace dédié** :

```
CREATE TABLESPACE ts_logs LOCATION '/mnt/disque_rapide/logs';
```

2. Créer la table `logs` dans ce tablespace :

```
CREATE TABLE logs (
    id SERIAL,
    message TEXT,
    date_log DATE
) TABLESPACE ts_logs;
```

3. Partitionner la table :

```
CREATE TABLE logs_2025_05 PARTITION OF logs
FOR VALUES FROM ('2025-05-01') TO ('2025-06-01');
```

📌 Résumé du module

- Le **stockage physique** a un impact direct sur les performances et la fiabilité.
- Le bon **choix du moteur**, des **tablespaces**, et de la **structure physique** est essentiel.
- Chaque SGBD a ses propres outils et fichiers à bien configurer.

telier guidé pas à pas : Configurer une base de données sur disque externe

Dans cet atelier, nous allons configurer une base de données MySQL (ou MariaDB) pour utiliser un disque externe ou un autre emplacement de stockage afin d'optimiser les performances, surtout en cas de grosse charge ou de grande quantité de données. Nous allons notamment :

- Modifier les chemins de stockage des bases de données.
- Déplacer les fichiers de données de la base vers un disque externe.
- Configurer MySQL pour utiliser ce nouveau chemin de stockage.

Pré-requis

- Un système Linux (ex : Ubuntu ou CentOS), ou Windows (avec un disque externe connecté).
- MySQL ou MariaDB installé.
- Accès `root` ou `sudo` pour effectuer des changements de configuration système.
- Un **disque externe** formaté et monté (ex : `/mnt/disque_externe`).

Étape 1 : Vérification de l'emplacement actuel des fichiers de MySQL

1. Connectez-vous à votre serveur où MySQL est installé.
    - **Sur Linux** : Ouvrez le terminal et connectez-vous à MySQL avec la commande suivante :

    ```
    bash
    mysql -u root -p
    ```

2. Vérifiez l'emplacement actuel des fichiers de données (fichiers `.ibd`, `.frm`, etc.) avec la commande :

    ```
    sql
    SHOW VARIABLES LIKE 'datadir';
    ```

    Cela vous donnera un chemin comme `/var/lib/mysql/` où MySQL stocke actuellement ses données.

Étape 2 : Arrêter le service MySQL

Avant de déplacer les fichiers, il faut **arrêter** le service MySQL pour éviter toute corruption de données.

- **Sur Linux** (ex. Ubuntu) :

  ```
  sudo systemctl stop mysql
  ```

- **Sur Windows** :
  - Ouvrez **Services** (services.msc), trouvez **MySQL** et cliquez sur "Arrêter".

Étape 3 : Monter le disque externe

Assurez-vous que le disque externe est monté correctement sur votre système. Si ce n'est pas déjà fait, voici comment procéder :

Sur Linux :

1. Vérifiez que le disque est détecté : `lsblk`

   Vous devriez voir un disque comme `/dev/sdb1`.

2. Créez un répertoire pour le point de montage :

   ```
   sudo mkdir /mnt/disque_externe
   ```

3. Montez le disque à cet emplacement :

   ```
   sudo mount /dev/sdb1 /mnt/disque_externe
   ```

Sur Windows :

- Ouvrez l'explorateur de fichiers et vérifiez que votre disque externe est visible sous un lecteur, par exemple `E:\`.

Étape 4 : Copier les fichiers de données MySQL vers le disque externe

Sur Linux :

1. **Copiez** le répertoire des données MySQL vers le disque externe.
   - Le répertoire des données par défaut sur MySQL est généralement `/var/lib/mysql/`.

   ```
   sudo cp -R /var/lib/mysql /mnt/disque_externe/
   ```

2. **Vérifiez que les fichiers ont bien été copiés** et assurez-vous que les permissions sont correctes pour MySQL :

   ```
   sudo chown -R mysql:mysql /mnt/disque_externe/mysql
   sudo chmod -R 755 /mnt/disque_externe/mysql
   ```

Sur Windows :

- Utilisez l'Explorateur de fichiers pour copier le répertoire `C:\ProgramData\MySQL\MySQL Server 8.0\data` vers votre disque externe, par exemple `E:\mysql_data`.

Étape 5 : Modifier la configuration de MySQL pour utiliser le disque externe

1. **Éditez le fichier de configuration MySQL**. Ce fichier est généralement situé dans `/etc/mysql/my.cnf` ou `/etc/my.cnf`.

   ```
   sudo nano /etc/mysql/my.cnf
   ```

2. **Modifiez la ligne `datadir`** pour pointer vers le nouveau répertoire sur le disque externe :

   ```
   ini
   [mysqld]
   datadir = /mnt/disque_externe/mysql
   ```

   Si vous avez déplacé vos fichiers de journalisation également, vous devrez modifier également `log-error` et `log-bin` :

   ```
   ini
   log-error = /mnt/disque_externe/mysql/mysql_error.log
   log-bin = /mnt/disque_externe/mysql/mysql-bin
   ```

3. **Sauvegardez** et fermez le fichier de configuration.

Étape 6 : Réinitialiser le service MySQL

Après avoir modifié la configuration, vous devez redémarrer MySQL pour qu'il prenne en compte les nouveaux chemins de stockage.

- **Sur Linux** :

  ```
  sudo systemctl start mysql
  ```

- **Sur Windows** :
  - Retournez dans **Services**, trouvez **MySQL**, et cliquez sur "Démarrer".

Étape 7 : Vérification du bon fonctionnement

1. **Connectez-vous à MySQL** pour vérifier que tout fonctionne correctement :

   ```
   mysql -u root -p
   ```

2. **Vérifiez les variables** pour vous assurer que le chemin a bien été modifié :

   ```
   SHOW VARIABLES LIKE 'datadir';
   ```

   Cela devrait maintenant afficher le chemin du disque externe, par exemple `/mnt/disque_externe/mysql/`.

3. Vous pouvez aussi vérifier l'état des fichiers de journalisation avec la commande :

   ```
   SHOW VARIABLES LIKE 'log_error';
   ```

   Cela doit pointer vers le fichier de log sur le disque externe.

Étape 8 : Tester l'accès aux bases de données

Enfin, vérifiez que les bases de données et les tables sont accessibles et fonctionnent normalement :

1. **Vérifiez la liste des bases de données** :

   ```
   SHOW DATABASES;
   ```

2. **Effectuez une requête simple** sur l'une de vos bases de données pour tester l'intégrité des données :

   ```
   SELECT * FROM <nom_table> LIMIT 10;
   ```

Étape 9 : Sauvegarde et sécurité

- **Vérification des sauvegardes** : Assurez-vous que vous avez une stratégie de sauvegarde solide, en particulier avec des données stockées sur un disque externe.
- **Permissions** : Vérifiez que les permissions du répertoire de données sur le disque externe sont correctement définies afin que l'utilisateur `mysql` puisse y accéder.

Résumé de l'atelier

- Nous avons configuré MySQL pour utiliser un disque externe pour stocker ses fichiers de données.
- Cette configuration permet d'améliorer les performances, de réduire l'utilisation du disque principal du serveur et de faciliter la gestion des données volumineuses.
- La clé de cette opération réside dans une bonne gestion des permissions, des sauvegardes et de la configuration.

Module 6 : La sauvegarde des données

Objectifs pédagogiques

À la fin de ce module, l'apprenant sera capable de :

- Comprendre l'importance de la sauvegarde des données pour assurer la sécurité et la disponibilité des informations.
- Mettre en place une stratégie de sauvegarde adaptée à son système de gestion de base de données (SGBD).
- Utiliser différentes méthodes de sauvegarde : complètes, différentielles, incrémentielles.
- Automatiser et sécuriser le processus de sauvegarde.
- Restaurer les données à partir des sauvegardes.

---

1. Introduction à la sauvegarde des données

La **sauvegarde des données** est un processus essentiel pour toute organisation ou système utilisant une base de données. Elle permet de prévenir la perte de données due à :

- La **corruption des fichiers**.
- Les **pannes matérielles** (disques durs défectueux, serveurs en panne).
- Les **erreurs humaines** (suppression accidentelle, erreurs de manipulation).
- Les **attaques malveillantes** (ransomware, piratage).

Le but d'une sauvegarde est de garantir que vous pouvez restaurer les données dans leur état le plus récent possible, avec un minimum de perte.

---

2. Types de sauvegardes

Sauvegarde complète (Full Backup)

Une **sauvegarde complète** consiste à copier l'intégralité de la base de données, y compris tous les fichiers de données, les journaux de transactions et autres fichiers nécessaires à sa restauration.

- **Avantages** :
  - Simple à mettre en place et à restaurer.
  - Pas de dépendance à d'autres sauvegardes.
- **Inconvénients** :
  - Temps et espace de stockage plus importants, surtout pour les grandes bases de données.

Sauvegarde incrémentielle

Une **sauvegarde incrémentielle** ne copie que les **modifications** apportées depuis la dernière sauvegarde, qu'elle soit complète ou incrémentielle.

- **Avantages** :
  - Moins d'espace de stockage et de temps nécessaires.
- **Inconvénients** :
  - La restauration peut être plus longue car elle nécessite de restaurer la sauvegarde complète suivie de toutes les sauvegardes incrémentielles.

Sauvegarde différentielle

Une **sauvegarde différentielle** copie toutes les modifications depuis la **dernière sauvegarde complète**.

- **Avantages** :
  - Plus rapide que la sauvegarde complète.
  - Restauration plus rapide que l'incrémentielle (un seul fichier de sauvegarde à appliquer après la complète).
- **Inconvénients** :
  - Nécessite un peu plus d'espace que l'incrémentielle, mais moins que la complète.

⚙ 3. Outils de sauvegarde pour SGBD

🔹 Sauvegarde MySQL / MariaDB

Méthode 1 : Utilisation de `mysqldump`

`mysqldump` est l'outil standard pour effectuer des sauvegardes de bases de données MySQL et MariaDB. Il permet de créer une sauvegarde au format SQL, contenant les instructions nécessaires pour recréer la base de données et y insérer les données.

- **Commande de sauvegarde complète** :

```
mysqldump -u root -p --all-databases > sauvegarde_complète.sql
```

- **Commande de sauvegarde d'une seule base** :

```
mysqldump -u root -p nom_base_de_donnees > sauvegarde_base.sql
```

- **Commande de sauvegarde avec option de compression** (plus rapide et moins d'espace disque) :

```
mysqldump -u root -p nom_base_de_donnees | gzip > sauvegarde_base.sql.gz
```

- **Commande pour effectuer une sauvegarde incrémentielle** :

MySQL ne dispose pas de fonctionnalité directe pour les sauvegardes incrémentielles via `mysqldump`. Cependant, vous pouvez activer les binlogs (journaux de transactions) pour capturer les modifications.

```
mysqlbinlog --start-position=xxxx --stop-position=yyyy > sauvegarde_incrémentielle.sql
```

Méthode 2 : Utilisation de `Xtrabackup`

Pour des bases MySQL plus volumineuses, `Xtrabackup` (un outil de Percona) permet de faire des sauvegardes **à chaud**, c'est-à-dire sans arrêter la base de données, ce qui est essentiel pour des environnements en production.

```
xtrabackup --backup --target-dir=/path/to/backup
```

🔹 Sauvegarde PostgreSQL

Méthode 1 : Utilisation de `pg_dump`

`pg_dump` est l'outil standard pour effectuer des sauvegardes de bases PostgreSQL. Il peut être utilisé pour sauvegarder une base de données entière ou une partie spécifique de celle-ci.

- **Commande de sauvegarde complète** :

```
pg_dump -U utilisateur -W -F c -b -v -f "sauvegarde_complète.backup" nom_base_de_donnees
```

- `-F c` : format compressé.
- `-b` : inclure les blobs (grands objets binaires).
- `-v` : mode verbeux pour afficher le détail de l'opération.

Méthode 2 : Sauvegarde avec `pg_basebackup` (sauvegarde à chaud)

- **Commande de sauvegarde** :

```
pg_basebackup -D /path/to/backup_directory -Ft -z -P
```

- `-Ft` : format tar.
- `-z` : compression des données.
- `-P` : affiche la progression de la sauvegarde.

🔒 4. Automatisation des sauvegardes

L'automatisation des sauvegardes est essentielle pour garantir qu'elles soient effectuées régulièrement, sans erreur humaine.

🔹 Automatiser les sauvegardes avec cron (Linux)

Exemple : Automatisation de `mysqldump` avec cron

1. Ouvrez la crontab de l'utilisateur root :

   ```
   sudo crontab -e
   ```

2. Ajoutez une ligne pour effectuer une sauvegarde quotidienne à 3h du matin :

   ```
   0 3 * * * mysqldump -u root -p'MotDePasse' --all-databases >
   /chemin/sauvegarde/backup_$(date +\%F).sql
   ```

Cela exécutera la commande tous les jours à 3h du matin, en créant une sauvegarde avec un nom basé sur la date (`backup_YYYY-MM-DD.sql`).

🔹 Automatiser les sauvegardes avec des outils de gestion (Windows/Linux)

Utiliser des outils comme **Bacula**, **Amanda**, **Duplicity**, ou des outils de cloud comme **AWS Backup** peut faciliter la gestion, la planification, et la récupération des sauvegardes de manière plus automatisée.

🛠️ 5. Restauration des données

La restauration des données dépend du type de sauvegarde effectué.

🔹 Restauration MySQL / MariaDB

- **Restauration d'une sauvegarde complète** avec `mysqldump` :

```
mysql -u root -p < sauvegarde_complète.sql
```

- **Restauration d'une sauvegarde comprimée** :

```
gunzip < sauvegarde_complète.sql.gz | mysql -u root -p
```

- **Restauration des binlogs (journaux de transactions)** :

```
mysqlbinlog /path/to/binlog | mysql -u root -p
```

🔻 Restauration PostgreSQL

- **Restauration d'une sauvegarde avec `pg_restore` :**

```
pg_restore -U utilisateur -d nom_base_de_donnees /path/to/sauvegarde_complète.backup
```

- **Restauration d'une sauvegarde avec `pg_dump` :**

```
psql -U utilisateur -d nom_base_de_donnees < sauvegarde_complète.sql
```

⚠ 6. Bonnes pratiques en matière de sauvegarde

- **Fréquence de sauvegarde** : Adaptez la fréquence selon l'importance des données. Une sauvegarde quotidienne est souvent un bon compromis pour des bases de données très utilisées.
- **Rotation des sauvegardes** : Conservez plusieurs versions de sauvegarde (par exemple, les 7 dernières).
- **Test de restauration** : Testez régulièrement la procédure de restauration pour garantir son efficacité en cas de besoin.
- **Stockage hors site** : Conservez des copies de vos sauvegardes dans un autre emplacement physique ou dans le cloud pour éviter les pertes dues à des catastrophes locales (incendies, inondations, etc.).

📝 Résumé du module

Dans ce module, vous avez appris les bases de la sauvegarde des données, les différents types de sauvegarde, et comment automatiser et sécuriser le processus de sauvegarde. Nous avons également exploré les méthodes de restauration pour garantir que vous pouvez récupérer rapidement vos données en cas de besoin.

Module 7 : La Configuration de la Réplication

La réplication de bases de données est une technique permettant de synchroniser plusieurs bases de données entre différents serveurs. Elle est couramment utilisée pour améliorer la disponibilité des données, la performance, la tolérance aux pannes, et pour le scaling horizontal. Ce module se concentre sur la configuration de la réplication MySQL, un des systèmes de gestion de bases de données relationnelles (SGBDR) les plus populaires.

Objectifs du Module :

1. **Comprendre les principes de la réplication de bases de données.**
2. **Configurer la réplication maître-esclave (master-slave) dans MySQL.**
3. **Configurer la réplication multi-maître pour assurer une meilleure tolérance aux pannes.**
4. **Analyser la gestion des erreurs et la surveillance de la réplication.**
5. **Tester et valider la réplication.**

---

1. Introduction à la Réplication

1.1 Qu'est-ce que la réplication de bases de données ?

La réplication de bases de données consiste à créer une copie exacte de la base de données principale (le **master**) sur un ou plusieurs serveurs secondaires (les **slaves**). Ces copies sont mises à jour en temps réel ou avec un certain retard, afin de garantir la disponibilité des données en cas de panne.

Les principaux types de réplication sont :

- **Réplication Maître-Esclave (Master-Slave)** : Un serveur principal (master) envoie ses modifications de données à un ou plusieurs serveurs esclaves (slaves).
- **Réplication Maître-Maître (Master-Master)** : Chaque serveur agit à la fois comme un maître et un esclave, permettant une écriture et une lecture sur plusieurs serveurs.

1.2 Avantages de la réplication

- **Haute disponibilité** : En cas de panne du maître, un esclave peut être promu pour prendre sa place.
- **Scalabilité** : Permet d'équilibrer la charge en lisant les données sur les serveurs esclaves.
- **Sécurité** : En cas de corruption de la base de données sur le maître, un esclave peut être utilisé pour la récupération.
- **Performances améliorées** : Répartition des charges de lecture entre le maître et les esclaves.

2. Configuration de la Réplication MySQL

2.1 Préparations initiales

1. **Installation de MySQL** : Assurez-vous que MySQL est installé sur tous les serveurs impliqués dans la réplication (master et slaves).

   Vous pouvez vérifier cela avec la commande :

   ```
   mysql --version
   ```

2. **Modification des fichiers de configuration MySQL** :

   Pour activer la réplication, vous devez modifier le fichier de configuration MySQL (`my.cnf` ou `my.ini`) sur les serveurs maître et esclave.

   Ouvrez ce fichier avec un éditeur de texte :

   ```
   sudo nano /etc/mysql/my.cnf
   ```

3. **Configurer le serveur maître** :

   Ajoutez ou modifiez les lignes suivantes dans la section `[mysqld]` du fichier de configuration du maître (sur le serveur maître) :

   ```
   ini
   [mysqld]
   server-id = 1
   log-bin = /var/log/mysql/mysql-bin.log
   binlog-do-db = nom_de_votre_base_de_donnees
   ```

- `server-id = 1` : Définit un identifiant unique pour le serveur maître.
- `log-bin` : Active l'enregistrement du journal binaire (requis pour la réplication).
- `binlog-do-db` : Indique quelle base de données sera répliquée (vous pouvez aussi utiliser `binlog-ignore-db` pour ignorer certaines bases).

Redémarrez le service MySQL pour appliquer les modifications :

```
sudo service mysql restart
```

4. **Configurer le serveur esclave** :

   Sur le serveur esclave, ouvrez le fichier `my.cnf` et ajoutez les configurations suivantes :

```
ini
[mysqld]
server-id = 2
relay-log = /var/log/mysql/mysql-relay-bin.log
```

   - `server-id = 2` : Un identifiant unique pour le serveur esclave.
   - `relay-log` : Indique où les événements de réplication seront enregistrés localement sur l'esclave.

   Redémarrez également MySQL sur l'esclave :

```
sudo service mysql restart
```

2.2 Création d'un utilisateur de réplication

1. **Sur le serveur maître**, connectez-vous à MySQL :

```
mysql -u root -p
```

2. **Créez un utilisateur de réplication** qui aura l'autorisation de se connecter depuis l'esclave :

```
CREATE USER 'replication_user'@'%' IDENTIFIED BY 'password';
GRANT REPLICATION SLAVE ON *.* TO 'replication_user'@'%';
FLUSH PRIVILEGES;
```

   - `replication_user` : Le nom de l'utilisateur de réplication.
   - `password` : Le mot de passe de l'utilisateur.

3. **Vérifiez la position de la réplication** en obtenant le **log position** et le **nom du log binaire** :

```
SHOW MASTER STATUS;
```

   Cela retournera des informations comme :

   - `File: mysql-bin.000001`
   - `Position: 1234`

   Vous aurez besoin de ces informations pour configurer l'esclave.

2.3 Configurer l'esclave pour commencer la réplication

Sur le serveur esclave, connectez-vous à MySQL et exécutez les commandes suivantes pour initialiser la réplication :

1. **Sur le serveur esclave**, connectez-vous à MySQL :

```
mysql -u root -p
```

2. **Indiquez à l'esclave le maître et le log binaire à partir duquel il doit commencer la réplication** :

```
CHANGE MASTER TO
MASTER_HOST = 'adresse_ip_du_maître',
MASTER_USER = 'replication_user',
MASTER_PASSWORD = 'password',
MASTER_LOG_FILE = 'mysql-bin.000001',
MASTER_LOG_POS = 1234;
```

   - `MASTER_HOST` : L'adresse IP ou le nom de domaine du serveur maître.
   - `MASTER_USER` et `MASTER_PASSWORD` : Les informations d'identification de l'utilisateur de réplication que vous avez créé.
   - `MASTER_LOG_FILE` et `MASTER_LOG_POS` : Les informations du log binaire que vous avez obtenues à l'étape précédente.

3. **Démarrez la réplication** :

```
START SLAVE;
```

4. **Vérifiez l'état de la réplication** :

```
SHOW SLAVE STATUS\G
```

   Vous devriez voir des informations indiquant que l'esclave suit le maître, avec un état comme `Slave_IO_Running: Yes` et `Slave_SQL_Running: Yes`.

---

3. Vérification de la Réplication

1. **Testez la réplication en insérant des données sur le maître** :

   Sur le serveur maître, insérez une nouvelle donnée dans la base répliquée :

```
USE test_sauvegarde;
INSERT INTO clients (nom, email) VALUES ('Alice', 'alice@example.com');
```

2. **Vérifiez sur l'esclave** que les données ont été répliquées :

   Sur l'esclave, exécutez la même requête SELECT :

```
USE test_sauvegarde;
SELECT * FROM clients;
```

   Vous devriez voir que la ligne insérée sur le maître est désormais présente sur l'esclave.

---

4. Surveillance et Gestion des Erreurs

1. **Vérification de l'état de la réplication** :

   Sur l'esclave, utilisez la commande suivante pour vérifier régulièrement l'état de la réplication :

```
SHOW SLAVE STATUS\G
```

2. **Gérer les erreurs de réplication** :

   Si des erreurs surviennent (par exemple, une erreur de `IO thread` ou `SQL thread`), vous pouvez redémarrer la réplication avec :

```
STOP SLAVE;
START SLAVE;
```

   Pour résoudre des erreurs spécifiques, il peut être nécessaire de **mettre à jour le fichier de log ou la position**.

---

5. Configuration de la Réplication Multi-Maître (Optionnel)

1. **Configurer deux serveurs MySQL en tant que maîtres et esclaves** :
   - Modifiez les fichiers de configuration des deux serveurs pour activer la réplication dans les deux sens.
   - Créez des utilisateurs de réplication pour chaque serveur.
   - Assurez-vous que chaque serveur connaît l'autre comme un maître.

   La réplication **multi-maître** permet à chaque serveur d'agir comme un maître et un esclave en même temps, ce qui offre une meilleure tolérance aux pannes, mais nécessite une gestion plus complexe des conflits de données.

---

6. Conclusion

Dans ce module, nous avons appris à configurer la réplication dans MySQL, à configurer une réplication maître-esclave, et à tester son bon fonctionnement. La réplication est un outil puissant pour assurer la haute disponibilité et la scalabilité de vos bases de données. Cependant, elle nécessite une gestion attentive, notamment en matière de surveillance et de gestion des erreurs.

Module 8 : Connaissance des différents types de stockage

Dans ce module, nous allons explorer les différents types de **stockage** utilisés pour la gestion des données, en particulier en ce qui concerne les bases de données. Le stockage des données joue un rôle crucial dans la performance, la sécurité, et la scalabilité des systèmes d'information. Ce module vise à donner une vue d'ensemble des types de stockage disponibles, ainsi que de leurs avantages, inconvénients, et cas d'utilisation.

Objectifs du Module :

1. Comprendre les différents types de **stockage** utilisés pour les bases de données.
2. Identifier les avantages et inconvénients de chaque type de stockage.
3. Apprendre à choisir le bon type de stockage en fonction des besoins de l'application.
4. Explorer les technologies de stockage utilisées pour des systèmes de gestion de bases de données (SGBD).

---

1. Types de stockage des données

Il existe plusieurs types de stockage qui peuvent être utilisés pour gérer les données, et chacun a des caractéristiques qui le rendent plus ou moins adapté à certains scénarios. Les principaux types sont :

1. **Stockage sur disque local (Direct Attached Storage - DAS)**
2. **Stockage en réseau (Network Attached Storage - NAS)**
3. **Stockage en réseau partagé (Storage Area Network - SAN)**
4. **Stockage en cloud**
5. **Stockage de données en mémoire (In-Memory Storage)**

---

2. Stockage sur disque local (DAS)

Le **DAS (Direct Attached Storage)** fait référence au stockage directement connecté à un serveur ou un ordinateur sans passer par un réseau. Il s'agit généralement de disques durs internes ou de disques SSD.

Caractéristiques :

- **Connexion directe** : Le stockage est directement connecté à un serveur ou une machine, sans infrastructure réseau.
- **Usage localisé** : Idéal pour des environnements où le volume de données n'est pas excessif, ou pour des applications nécessitant un accès rapide et local aux données.
- **Faible coût** : Comparé aux autres solutions, le DAS est moins coûteux à déployer.

Avantages :

- **Haute performance** en termes de latence, car le disque est localement attaché à la machine.
- **Installation facile** et faible coût de mise en œuvre.

Inconvénients :

- **Scalabilité limitée** : Difficulté à étendre le stockage à plusieurs serveurs ou machines.
- **Pas de redondance** : Si le disque local échoue, la récupération des données peut être difficile sans sauvegarde.

Cas d'utilisation :

- Applications ou bases de données locales qui n'ont pas besoin de se connecter avec plusieurs machines ou utilisateurs.

---

3. Stockage en réseau (NAS)

Le **NAS (Network Attached Storage)** est un type de stockage partagé accessible via un réseau, souvent utilisé pour les sauvegardes et le stockage de données non structurées.

Caractéristiques :

- **Connecté au réseau** : Accessible par tous les dispositifs connectés au réseau, ce qui permet un stockage centralisé.

- **Fichiers partagés** : Utilisé pour des environnements où de multiples utilisateurs ou serveurs ont besoin d'accéder aux mêmes fichiers.
- **Interface réseau (Ethernet)** : Le NAS se connecte via des connexions réseau traditionnelles comme Ethernet.

Avantages :

- **Centralisation des données** : Permet de stocker et d'accéder aux fichiers depuis plusieurs machines.
- **Évolutivité** : Facile à étendre en ajoutant plus de disques à l'unité NAS.
- **Facilité d'utilisation** : Souvent, les NAS sont faciles à configurer et à gérer.

Inconvénients :

- **Vitesse d'accès réduite** par rapport à un stockage local (latence réseau).
- **Moins adapté pour les applications à haute performance** nécessitant un accès rapide aux données.

Cas d'utilisation :

- Partage de fichiers entre plusieurs utilisateurs ou serveurs dans un réseau.
- Sauvegarde des données non structurées et de petites bases de données.

---

4. Stockage en réseau partagé (SAN)

Le **SAN (Storage Area Network)** est une architecture de stockage réseau qui permet de connecter des disques de stockage directement à des serveurs via un réseau haute vitesse (Fibre Channel, iSCSI).

Caractéristiques :

- **Réseau dédié** : Contrairement au NAS, le SAN est un réseau à part entière conçu spécifiquement pour gérer des besoins de stockage.
- **Accès au niveau du bloc** : Le SAN permet un accès au stockage au niveau des blocs, plutôt qu'au niveau des fichiers comme avec un NAS.
- **Haute disponibilité et performance** : Utilise des réseaux de haute performance pour garantir la disponibilité et réduire la latence.

Avantages :

- **Haute performance** : Accès rapide et faible latence grâce à des réseaux dédiés.
- **Évolutivité** : Facile à étendre avec de nombreux disques de stockage.
- **Résilience et disponibilité** : Très redondant, avec des options pour la haute disponibilité et la tolérance aux pannes.

Inconvénients :

- **Coût élevé** : Le SAN est plus cher à installer et à maintenir, notamment à cause de l'infrastructure réseau spécialisée.
- **Complexité** : Sa configuration et gestion peuvent être complexes.

Cas d'utilisation :

- Environnements de production à grande échelle, comme les data centers, où de hautes performances et une haute disponibilité sont nécessaires.

- Applications nécessitant des accès rapides aux données à grande échelle, telles que les bases de données transactionnelles à fort volume.

---

5. Stockage en cloud

Le **stockage en cloud** désigne le stockage des données sur des serveurs distants accessibles via Internet. Les fournisseurs de cloud comme AWS, Google Cloud, et Microsoft Azure offrent des services de stockage évolutifs.

Caractéristiques :

- **Accès à distance** : Les données sont stockées dans des data centers distants et accessibles via Internet.
- **Évolutivité** : Le stockage cloud est hautement scalable, vous payez uniquement pour l'espace que vous utilisez.
- **Service managé** : Le fournisseur de cloud s'occupe de la gestion de l'infrastructure, de la maintenance, des sauvegardes, etc.

Avantages :

- **Scalabilité quasi infinie** : Vous pouvez augmenter ou réduire la capacité de stockage à la demande.
- **Faible coût initial** : Pas besoin d'investir dans des équipements physiques.
- **Accessibilité** : Les données sont accessibles de partout avec une connexion Internet.

Inconvénients :

- **Dépendance à la connexion Internet** : Un accès rapide et fiable à Internet est essentiel.
- **Sécurité** : Les données sont stockées hors site, ce qui nécessite une bonne stratégie de sécurité et de gestion des droits d'accès.

Cas d'utilisation :

- Stockage de données volumineuses pour des applications distribuées ou à grande échelle.
- Sauvegarde de données et services de récupération après sinistre.
- Applications qui bénéficient de l'évolutivité et de la gestion simplifiée des ressources.

---

6. Stockage de données en mémoire (In-Memory Storage)

Le **stockage en mémoire** utilise la RAM pour stocker les données au lieu du disque. Il est principalement utilisé dans les bases de données en mémoire et pour les caches.

Caractéristiques :

- **Accès ultra-rapide** : Les données sont stockées directement dans la mémoire vive, ce qui permet des accès très rapides.
- **Pas de persistance par défaut** : Les données en mémoire sont volatiles, ce qui signifie qu'elles sont perdues en cas de redémarrage du système, sauf si elles sont périodiquement écrites sur le disque.

Avantages :

- **Latence extrêmement faible** : Permet des performances très élevées, idéal pour les applications nécessitant un accès ultra-rapide aux données.

- **Utilisé pour des caches et des bases de données en temps réel** : Parfait pour les systèmes de gestion de sessions ou les bases de données NoSQL en mémoire.

Inconvénients :

- **Coût élevé** : La RAM est plus coûteuse par rapport au stockage sur disque.
- **Volatilité** : Les données sont perdues si le système s'arrête de manière inattendue, sauf mise en place de mécanismes de sauvegarde.

Cas d'utilisation :

- Systèmes de cache (comme Redis, Memcached).
- Bases de données en mémoire pour des besoins de traitement en temps réel (ex. SAP HANA, Redis).
- Applications nécessitant une latence minimale, comme le trading haute fréquence.

---

7. Conclusion et choix du stockage adapté

Le choix du type de stockage dépend de plusieurs facteurs tels que la taille des données, la performance requise, les coûts, la redondance, et la tolérance aux pannes. Voici quelques recommandations générales :

- **DAS (Direct Attached Storage)** : Idéal pour des environnements simples et des petites entreprises, ou pour des bases de données locales à faible volume.
- **NAS (Network Attached Storage)** : Parfait pour un stockage centralisé avec un accès par plusieurs machines et pour des fichiers partagés.
- **SAN (Storage Area Network)** : Convient aux grandes entreprises nécessitant une haute performance, une redondance et une scalabilité.
- **Cloud** : Idéal pour les applications distribuées et évolutives avec une gestion facile et un faible coût initial.
- **In-Memory Storage** : Utilisé pour des applications nécessitant des performances ultra-rapides ou un traitement en temps réel des données.

Exercices pratiques :

1. Comparez les performances de MySQL en utilisant un stockage DAS vs un stockage NAS.
2. Implémentez un service de sauvegarde avec un stockage cloud (AWS, Google Cloud).
3. Configurez un cache en mémoire avec Redis pour améliorer les performances d'une application web.
4. Mesurez la latence de lecture et d'écriture dans un environnement SAN par rapport à un environnement NAS.

Exercice 1 : Comparaison des performances de MySQL en utilisant un stockage DAS vs un stockage NAS

Objectif :

Comparer les performances de MySQL lorsque les données sont stockées sur un **disque local (DAS)** et sur un **NAS**.

Méthode :

1. **Configurer MySQL avec DAS :**
   - Installez MySQL sur une machine avec un disque local (par exemple, un SSD ou un disque dur local).
   - Configurez une base de données simple et insérez plusieurs milliers d'enregistrements.
   - Exécutez des requêtes `SELECT`, `INSERT`, `UPDATE`, et `DELETE` pour mesurer la performance de ces opérations.
2. **Configurer MySQL avec NAS :**

- Configurez un **NAS** accessible via un réseau local.
- Déplacez les fichiers de données de MySQL sur le NAS et assurez-vous que MySQL peut y accéder correctement.
- Répétez les mêmes tests de performance (requêtes `SELECT`, `INSERT`, `UPDATE`, et `DELETE`).

Analyse des résultats :

- **DAS :** Vous devriez observer des performances plus élevées pour les opérations de lecture et d'écriture en raison de l'accès direct au disque local.
- **NAS :** Le NAS pourrait entraîner une légère augmentation de la latence en raison des appels réseau pour accéder aux données.

Conclusion :

Le **DAS** est plus rapide pour les applications locales nécessitant un accès direct aux données. En revanche, un **NAS** peut offrir des avantages en termes de partage et de centralisation des données, mais il peut introduire de la latence dans les opérations de bases de données.

Exercice 2 : Implémentation d'un service de sauvegarde avec un stockage cloud (AWS, Google Cloud)

Objectif :

Mettre en place un service de sauvegarde des données dans un stockage cloud (par exemple AWS S3 ou Google Cloud Storage).

Méthode :

1. **Création d'un bucket dans le cloud :**
   - Sur **AWS** :
     1. Connectez-vous à votre compte AWS et allez dans le service **S3**.
     2. Créez un **bucket** pour stocker vos fichiers de sauvegarde.
   - Sur **Google Cloud** :
     1. Connectez-vous à votre compte Google Cloud et accédez au service **Cloud Storage**.
     2. Créez un **bucket** similaire pour y stocker les sauvegardes.
2. **Configurer MySQL pour effectuer une sauvegarde automatique :**
   - Utilisez la commande `mysqldump` pour créer une sauvegarde de votre base de données.

     ```
     mysqldump -u root -p my_database > backup.sql
     ```

   - Transférez cette sauvegarde vers le stockage cloud à l'aide de la CLI :
     - Sur **AWS**, utilisez `aws s3 cp` :

       ```
       aws s3 cp backup.sql s3://your-bucket-name/
       ```

     - Sur **Google Cloud**, utilisez `gsutil cp` :

       ```
       gsutil cp backup.sql gs://your-bucket-name/
       ```

3. **Automatiser la sauvegarde :**
   - Configurez une tâche cron (sur Linux) pour effectuer la sauvegarde à intervalles réguliers :

     ```
     crontab -e
     ```

Ajoutez une ligne pour exécuter le script de sauvegarde tous les jours à minuit :

```
0 0 * * * /path/to/backup_script.sh
```

Analyse des résultats :

- Vous devriez maintenant avoir des sauvegardes automatiques stockées dans le cloud, ce qui garantit la sécurité des données et leur accessibilité à partir de n'importe où.

Conclusion :

Le stockage dans le **cloud** offre une solution fiable et évolutive pour les sauvegardes. Les **services comme AWS S3** et **Google Cloud Storage** sont populaires pour la gestion des sauvegardes, car ils offrent une haute disponibilité, une redondance et une facilité de gestion.

Exercice 3 : Configuration d'un cache en mémoire avec Redis pour améliorer les performances d'une application web

Objectif :

Mettre en place un **cache en mémoire** avec **Redis** pour améliorer les performances d'une application web en réduisant la latence des requêtes répétées.

Méthode :

1. **Installer Redis :**
   - Sur un serveur Ubuntu, vous pouvez installer Redis avec les commandes suivantes :

     ```
     sudo apt update
     sudo apt install redis-server
     ```

2. **Configurer Redis pour l'application :**
   - Dans votre application web (par exemple en PHP, Python, ou Node.js), connectez-vous à Redis en utilisant un client Redis.
     - Pour **Python**, vous pouvez utiliser la bibliothèque `redis-py` :

       ```
       pip install redis
       ```

     - Ensuite, dans votre code Python :

       ```
       import redis

       r = redis.Redis(host='localhost', port=6379, db=0)
       r.set('user:1000', 'Alice')  # Stocker une donnée dans Redis
       user_name = r.get('user:1000')  # Récupérer la donnée
       print(user_name)  # Affiche "Alice"
       ```

3. **Utiliser Redis pour mettre en cache les résultats des requêtes fréquentes :**
   - Par exemple, au lieu d'exécuter une requête de base de données à chaque fois, stockez les résultats dans Redis pour un accès rapide.
     - Si la requête est déjà présente dans Redis, renvoyez la réponse directement.
     - Si la requête n'est pas dans Redis, effectuez la requête dans la base de données et stockez le résultat dans Redis pour les prochaines demandes.

Analyse des résultats :

- Vous devriez constater une réduction significative du temps de réponse pour les requêtes fréquentes, car les données sont stockées en mémoire dans Redis.
- Cela peut considérablement améliorer les performances de votre application web, en particulier pour des données qui ne changent pas fréquemment.

Conclusion :

L'utilisation de Redis comme cache en mémoire permet de réduire la charge sur la base de données et d'améliorer les performances de l'application en diminuant la latence. Redis est une solution idéale pour les scénarios où la rapidité d'accès aux données est cruciale.

Exercice 4 : Mesurer la latence de lecture et d'écriture dans un environnement SAN par rapport à un environnement NAS

Objectif :

Comparer la **latence de lecture et d'écriture** entre un stockage **SAN** et un stockage **NAS**.

Méthode :

1. **Configurer un environnement SAN** :
   - Assurez-vous que votre serveur est connecté à un réseau SAN et que le stockage SAN est bien configuré pour MySQL.
   - Effectuez des tests de performance en exécutant des requêtes `SELECT`, `INSERT`, et `UPDATE` sur la base de données.
2. **Configurer un environnement NAS** :
   - Configurez un NAS sur le même réseau et montez le volume de stockage sur le serveur.
   - Répétez les mêmes tests de performance que précédemment.
3. **Mesurer la latence** :
   - Utilisez des outils comme `sysbench` ou des requêtes personnalisées pour mesurer la latence des opérations de lecture et d'écriture.
     Exemple avec `sysbench` pour tester les performances de MySQL :

```
sysbench --db-driver=mysql --mysql-user=root --mysql-password=password --
mysql-db=test --test=oltp --oltp-table-size=100000 --num-threads=4 run
```

Analyse des résultats :

- Le **SAN** devrait offrir une **latence plus faible** et des performances globalement supérieures en raison de son réseau dédié et de son accès aux blocs de données.
- Le **NAS** pourrait afficher une latence légèrement plus élevée en raison de la dépendance à un réseau partagé et de l'accès aux données au niveau des fichiers.

Conclusion :

Le **SAN** est une meilleure solution pour les environnements à forte demande de performance et de faible latence, tandis que le **NAS** est plus adapté aux environnements de stockage partagé avec un volume de données moins critique en termes de performance.

Résumé des corrections :

- **DAS vs NAS** : Le **DAS** offre des performances supérieures pour des applications locales, mais le **NAS** est utile pour un stockage centralisé et partagé.
- **Cloud** : Le stockage cloud est une solution évolutive et fiable pour la sauvegarde, avec une gestion simplifiée.
- **Redis** : Le cache en mémoire avec Redis permet d'accélérer les applications en réduisant la latence des données fréquemment utilisées.
- **SAN vs NAS** : Le **SAN** est plus performant en termes de latence et de débit pour les bases de données de grande envergure, tandis que le **NAS** est plus adapté à des besoins de stockage partagé.

Module 9 : Les familles de bases de données NoSQL

Objectifs du Module :

1. Comprendre les concepts fondamentaux des bases de données NoSQL.
2. Explorer les principales familles de bases de données NoSQL : clé-valeur, colonne, document et graphes.
3. Identifier les cas d'utilisation spécifiques à chaque famille de bases de données NoSQL.
4. Comparer les bases de données NoSQL avec les bases de données relationnelles (SQL).
5. Comprendre quand choisir une base de données NoSQL par rapport à une base de données SQL.

---

1. Introduction aux bases de données NoSQL

Les bases de données NoSQL (Not Only SQL) désignent un ensemble de technologies de gestion de bases de données qui ne reposent pas sur un modèle relationnel classique. Ces systèmes sont conçus pour gérer des volumes massifs de données non structurées ou semi-structurées, souvent dans des environnements distribués et scalables.

Caractéristiques principales des bases de données NoSQL :

- **Scalabilité horizontale** : Les bases de données NoSQL sont souvent conçues pour être facilement scalables, c'est-à-dire qu'elles peuvent s'étendre facilement en ajoutant des serveurs supplémentaires.
- **Modèle flexible** : Elles ne nécessitent pas un schéma strict comme les bases de données relationnelles. Les données peuvent être stockées sous diverses formes, adaptées aux besoins de l'application.
- **Haute disponibilité** : De nombreuses bases de données NoSQL offrent une tolérance aux pannes et une disponibilité continue grâce à des architectures distribuées.
- **Optimisation des performances pour certains cas d'usage** : Certaines bases NoSQL sont optimisées pour des cas spécifiques, comme les recherches en texte plein ou la gestion de graphes complexes.

---

2. Les différentes familles de bases de données NoSQL

Les bases de données NoSQL se regroupent généralement en plusieurs catégories, selon leur modèle de données et leur mécanisme de gestion. Les quatre principales familles sont :

1. **Les bases de données clé-valeur (Key-Value Store)**
2. **Les bases de données orientées documents (Document Store)**
3. **Les bases de données en colonnes (Column Store)**
4. **Les bases de données orientées graphes (Graph Database)**

2.1. Bases de données clé-valeur (Key-Value Store)

Dans les bases de données **clé-valeur**, les données sont stockées sous forme de paires clé-valeur, où chaque clé est unique et permet d'accéder à la valeur associée. Ce modèle est très simple et efficace pour les cas d'utilisation nécessitant des lectures et écritures rapides.

Exemples populaires :

- **Redis**
- **Riak**
- **Memcached**

Caractéristiques :

- **Simplicité** : Les données sont stockées sous forme de paires clé-valeur.
- **Performance** : Ce modèle est très performant pour des accès rapides en lecture et écriture, idéal pour des sessions utilisateurs, des caches ou des comptages.
- **Flexibilité** : Bien qu'elles soient simples, ces bases peuvent stocker des valeurs très variées (strings, objets, tableaux, etc.).

Cas d'utilisation :

- **Sessions utilisateurs** : Stockage des données de session utilisateur dans une application web.
- **Caches** : Accélération des performances des applications en gardant les données fréquemment demandées en mémoire.
- **Comptage et suivi d'événements** : Comptage d'occurrences d'événements ou de visites.

Exemple :

Dans une application de commerce en ligne, vous pourriez utiliser Redis pour stocker les articles dans le panier d'achat d'un utilisateur avec une clé unique (par exemple, l'ID de l'utilisateur).

2.2. Bases de données orientées documents (Document Store)

Les bases de données **orientées documents** stockent des données sous forme de documents, généralement au format JSON, BSON ou XML. Ces bases sont particulièrement adaptées pour le stockage de données semi-structurées.

Exemples populaires :

- **MongoDB**
- **CouchDB**
- **Couchbase**

Caractéristiques :

- **Structure flexible** : Les documents peuvent contenir des champs avec des types de données variés, y compris des tableaux ou des objets imbriqués.
- **Indexation puissante** : Ces bases permettent de créer des index sur des champs spécifiques pour des recherches efficaces.

- **Support des requêtes complexes** : Contrairement aux clés-valeurs, les bases orientées documents permettent des requêtes plus complexes sur des structures imbriquées.

Cas d'utilisation :

- **Applications web** : Pour gérer des documents ou des données semi-structurées dans des applications web (par exemple, profils utilisateurs, posts de blog, produits, etc.).
- **Systèmes de gestion de contenu (CMS)** : Pour stocker du contenu riche (texte, images, vidéos) avec des métadonnées associées.

Exemple :

Dans un site de gestion de contenu, chaque article de blog peut être un document MongoDB avec des champs pour le titre, le contenu, les commentaires, les métadonnées (date de publication, tags, etc.), et ces champs peuvent être indexés pour une recherche rapide.

2.3. Bases de données en colonnes (Column Store)

Les bases de données en **colonnes** organisent les données par colonnes plutôt que par lignes, ce qui permet des lectures rapides pour des requêtes qui ne nécessitent qu'un sous-ensemble des colonnes d'une table. Ce modèle est particulièrement adapté pour l'analyse de grandes quantités de données.

Exemples populaires :

- **Cassandra**
- **HBase**
- **Couchbase (mode colonne)**

Caractéristiques :

- **Optimisation des lectures** : Les colonnes peuvent être lues indépendamment, ce qui est idéal pour des requêtes analytiques sur de grandes quantités de données.
- **Scalabilité horizontale** : Ces bases sont conçues pour être distribuées et permettent une scalabilité horizontale facile.
- **Performance pour les requêtes de type analytique** : Elles sont particulièrement adaptées aux systèmes de type OLAP (Online Analytical Processing).

Cas d'utilisation :

- **Big Data et analyse de données** : Ces bases sont idéales pour les applications de Big Data et d'analyse, comme le suivi de données en temps réel, l'analyse de logs, etc.
- **Gestion de séries temporelles** : Par exemple, pour surveiller des capteurs IoT ou des logs d'événements en continu.

Exemple :

Dans un système de monitoring de serveurs, les données provenant de chaque serveur (comme les températures, les taux d'occupation, les erreurs, etc.) peuvent être stockées dans une base en colonnes, où chaque colonne représente un type de donnée (par exemple, une colonne pour les températures, une autre pour les erreurs).

2.4. Bases de données orientées graphes (Graph Database)

Les bases de données orientées **graphes** sont conçues pour stocker et manipuler des données qui sont fortement connectées, avec des relations complexes entre les éléments.

Exemples populaires :

- **Neo4j**
- **ArangoDB**
- **Titan**

Caractéristiques :

- **Modèle de graphes** : Les données sont stockées sous forme de nœuds, d'arêtes et de propriétés. Ce modèle permet de représenter des relations complexes (par exemple, les réseaux sociaux, les recommandations, etc.).
- **Performances des requêtes relationnelles complexes** : Les bases de données graphes sont optimisées pour les requêtes qui explorent des relations entre les entités, comme les recherches de voisins dans un réseau social.
- **Flexibilité** : Elles permettent une évolution facile du modèle de données et sont idéales pour des applications nécessitant une évolution rapide.

Cas d'utilisation :

- **Réseaux sociaux** : Pour gérer les connexions entre utilisateurs et effectuer des recommandations.
- **Recommandations** : Par exemple, recommander des films sur la base de préférences similaires d'utilisateurs (système de filtrage collaboratif).
- **Gestion de réseaux** : Pour analyser des réseaux de distribution, des réseaux de communication, etc.

---

3. Comparaison avec les bases de données relationnelles (SQL)

Les bases de données **relationnelles** (SQL) utilisent des tables avec des lignes et des colonnes, et elles reposent sur un schéma strict. Les bases de données **NoSQL**, en revanche, offrent plus de flexibilité, de scalabilité horizontale et sont souvent mieux adaptées pour les grandes quantités de données non structurées ou semi-structurées.

Critère	SQL	NoSQL
Modèle de données	Structuré, basé sur des tables	Flexible, clé-valeur, document, graphe
Scalabilité	Scalabilité verticale (ajout de ressources au serveur)	Scalabilité horizontale (ajout de serveurs)
Schéma	Schéma rigide et prédéfini	Schéma flexible, souvent sans schéma strict
Requêtes	Requêtes SQL complexes	Requêtes adaptées au modèle de données (par exemple, JSON, graphes)
Cas d'utilisation	Transactions complexes, intégrité référentielle	Données massives, données non structurées, temps réel

---

Module 10 : La gestion des droits d'accès des utilisateurs ou clients

Objectifs du module :

- **Comprendre l'importance de la gestion des droits d'accès dans un système de gestion de bases de données.**
- **Explorer les différents types de droits d'accès et leur mise en œuvre dans les systèmes de bases de données.**
- **Mettre en place des stratégies de gestion des utilisateurs et des permissions dans une base de données.**
- **Apprendre à gérer l'accès des utilisateurs dans un environnement multi-utilisateur tout en garantissant la sécurité des données.**

1. Introduction à la gestion des droits d'accès

La gestion des droits d'accès dans une base de données est essentielle pour garantir la **sécurité**, l'**intégrité** et la **confidentialité** des données. Cela consiste à définir qui peut accéder à la base de données, quelles opérations un utilisateur peut effectuer (lecture, écriture, modification, suppression, etc.), et sous quelles conditions.

La **gestion des droits d'accès** se fait généralement via des **rôles** et des **permissions**. Ces mécanismes permettent de contrôler l'accès des utilisateurs aux ressources sensibles et aux fonctionnalités spécifiques du système.

Concepts clés :

- **Utilisateur** : Une personne ou un programme qui se connecte à la base de données.
- **Rôle** : Un ensemble de permissions attribuées à un utilisateur. Par exemple, un rôle "administrateur" pourrait inclure des permissions pour gérer les utilisateurs et les configurations de la base de données.
- **Permission** : Les actions qu'un utilisateur est autorisé à effectuer sur les données (par exemple, `SELECT`, `INSERT`, `UPDATE`, `DELETE`).
- **Accès basé sur des rôles (RBAC)** : Modèle qui attribue des permissions à des rôles plutôt qu'à des utilisateurs individuels.

2. Types de droits d'accès dans une base de données

Les droits d'accès dans une base de données sont généralement associés à un ensemble d'actions que les utilisateurs peuvent effectuer sur des objets spécifiques dans la base de données (tables, vues, procédures, etc.).

Voici quelques exemples des types de droits d'accès les plus courants dans une base de données relationnelle (comme MySQL, PostgreSQL, ou SQL Server) :

2.1. Droits de lecture (SELECT)

- Permet à un utilisateur de **lire les données** dans une table ou une vue.
- Utilisé pour consulter des informations sans les modifier.

2.2. Droits d'écriture (INSERT)

- Permet à un utilisateur **d'ajouter de nouvelles données** dans une table.

2.3. Droits de mise à jour (UPDATE)

- Permet à un utilisateur de **modifier les données** existantes dans une table.

2.4. Droits de suppression (DELETE)

- Permet à un utilisateur de **supprimer** des données d'une table.

2.5. Droits d'exécution (EXECUTE)

- Permet à un utilisateur d'**exécuter une procédure stockée ou une fonction** définie dans la base de données.

2.6. Droits d'administration (ALL PRIVILEGES ou ADMIN)

- Permet à un utilisateur de **gérer la base de données**, y compris la création de tables, la gestion des utilisateurs et des permissions, et la configuration des paramètres de la base de données.
- Ces droits sont généralement accordés à un **administrateur de base de données** (DBA).

2.7. Droits de création (CREATE)

- Permet à un utilisateur de **créer des objets** dans la base de données, comme des tables, des vues, des index, des procédures stockées, etc.

2.8. Droits de modification de structure (ALTER)

- Permet à un utilisateur de **modifier la structure** d'une table ou d'un autre objet, comme l'ajout ou la suppression de colonnes.

2.9. Droits de contrôle des données (GRANT/REVOKE)

- Permet à un utilisateur d'**accorder ou révoquer** des permissions pour d'autres utilisateurs.

---

3. Contrôle d'accès basé sur les rôles (RBAC)

Dans de nombreux systèmes de gestion de bases de données, la gestion des permissions se fait par le biais de **rôles**. Un **rôle** est un groupe de permissions qui peuvent être attribuées à un ou plusieurs utilisateurs. Cela simplifie la gestion des droits d'accès, car au lieu d'attribuer des permissions à chaque utilisateur individuellement, vous pouvez attribuer un rôle et ce rôle inclura toutes les permissions nécessaires.

Exemple :

- **Rôle Administrateur** : Accès complet à toutes les opérations sur toutes les tables.
- **Rôle Lecteur** : Seulement l'accès en lecture aux tables de données.
- **Rôle Éditeur** : Autorisé à insérer et modifier des données, mais pas à supprimer.

Avantages du RBAC :

- **Gestion simplifiée** : Il est plus facile de gérer les permissions par rôle plutôt que par utilisateur.
- **Réduction des erreurs humaines** : Moins de risques d'attribuer des permissions excessives à des utilisateurs.
- **Audit facilité** : Les actions d'un utilisateur peuvent être tracées facilement par rôle, ce qui permet un meilleur suivi des activités.

4. Implémentation des droits d'accès dans une base de données SQL

4.1. Création d'un utilisateur et attribution de permissions

Dans MySQL, par exemple, la création d'un utilisateur et l'attribution de permissions se fait avec les commandes suivantes :

Création d'un utilisateur :

```
CREATE USER 'username'@'hostname' IDENTIFIED BY 'password';
```

Ici, `username` est le nom de l'utilisateur, et `hostname` peut être une adresse IP ou un nom de domaine. Le mot de passe est attribué à l'utilisateur.

Attribution des permissions :

Pour accorder des droits d'accès à un utilisateur, utilisez la commande `GRANT` :

```
GRANT SELECT, INSERT ON my_database.* TO 'username'@'hostname';
```

Cela accorde les droits **SELECT** et **INSERT** sur toutes les tables de la base de données `my_database` à l'utilisateur.

Révocation de permissions :

Pour révoquer des permissions, vous pouvez utiliser la commande `REVOKE` :

```
REVOKE DELETE ON my_database.* FROM 'username'@'hostname';
```

Cela supprime le droit **DELETE** pour l'utilisateur spécifié.

Appliquer les changements :

Pour appliquer les changements et rendre effectives les nouvelles permissions :

```
FLUSH PRIVILEGES;
```

4.2. Création et gestion des rôles

Dans certaines bases de données, comme PostgreSQL ou Oracle, vous pouvez également gérer les rôles directement.

Exemple dans PostgreSQL :

Création d'un rôle :

```
CREATE ROLE admin_role;
```

Attribution de permissions au rôle :

```
GRANT SELECT, INSERT, UPDATE ON ALL TABLES IN SCHEMA public TO admin_role;
```

Attribution du rôle à un utilisateur :

```
GRANT admin_role TO username;
```

5. Bonnes pratiques pour la gestion des droits d'accès

1. **Principe du moindre privilège** : Accordez aux utilisateurs uniquement les permissions nécessaires pour accomplir leurs tâches. Cela réduit les risques d'accès non autorisé ou de suppression accidentelle des données.
2. **Audit des permissions** : Effectuez régulièrement des audits des permissions des utilisateurs pour garantir qu'elles sont toujours appropriées.
3. **Gestion des rôles et groupes** : Utilisez des rôles pour simplifier la gestion des permissions. Assurez-vous que les utilisateurs ont des rôles adaptés à leurs responsabilités.
4. **Révocation des accès inutiles** : Révoquez rapidement les accès des utilisateurs qui n'en ont plus besoin (par exemple, lors de la fin d'un contrat de travail).
5. **Utilisation de connexions sécurisées** : Assurez-vous que les connexions à la base de données sont sécurisées, en utilisant des mécanismes comme SSL/TLS.
6. **Politique de mot de passe solide** : Imposer une politique de mot de passe stricte pour les utilisateurs afin de renforcer la sécurité.

---

6. Gestion des droits d'accès dans des environnements distribués et cloud

Les bases de données dans des environnements distribués ou sur des plateformes cloud (par exemple, Amazon RDS, Google Cloud SQL, ou Microsoft Azure SQL) permettent également la gestion des utilisateurs et des rôles, mais avec des spécificités propres à chaque plateforme :

- **Contrôle d'accès basé sur des rôles (RBAC)** dans les services cloud.
- **IAM (Identity and Access Management)** dans AWS pour gérer les permissions au niveau des services.
- **Gestion des accès via des groupes et des politiques d'authentification dans les services managés**.

Ces plateformes permettent de gérer les permissions non seulement pour la base de données elle-même, mais aussi pour les autres services associés, garantissant ainsi une sécurité renforcée dans des architectures distribuées.

---

7. Conclusion

La gestion des droits d'accès est un aspect fondamental de la sécurité des bases de données. En utilisant des rôles et des permissions, vous pouvez contrôler l'accès aux données sensibles tout en offrant aux utilisateurs un accès approprié pour leurs tâches spécifiques. Ce contrôle permet de protéger les données contre les accès non autorisés, les erreurs humaines et d'assurer que seules les personnes appropriées ont accès aux informations critiques.