OBJECTIFS

Objectif Global

Ce cours introduit les concepts fondamentaux des bases de données. Les étudiants découvriront les différents types de bases de données existantes, le théorème CAP, et comment choisir un système de gestion de bases de données (SGBD) adapté aux besoins d'un projet.

Objectifs pédagogiques

- Savoir définir les différents types de bases de données
- Choisir le SGBD adapté
- Installer un environnement de travail pour se connecter à un SGBD
- Protéger un serveur de bases de données

Démarche pédagogique

A partir du kit, des ressources fournies ainsi que de recherches en autonomie, vous allez découvrir les différents types de bases de données, comment s'y connecter et comment en sécuriser l'accès. A partir de ce que vous aurez appris, vous allez réaliser un mémo contenant définitions et procédures. Afin de vous guider dans l'apprentissage, vous aurez quelques exercices à réaliser seul puis vous pourrez débattre de vos choix en groupe/

Compétences

- Créer un mémo
- Définir une base de données SQL
- Définir une base de données noSQL
- Installer son environnement de travail

MODALITÉS

Durée

1,0J

ITÉRATION 1

Comprendre les bases de données

Modalités

- Travail individuel en autonomie

Livrables

Votre mémo comprend les définitions des concepts clés et vous aide à choisir une base de données.

Objectifs

Connaître les différents types de base de données

Étape 1 : Les différents types de bases de données

Objectif : Comprendre la diversité des bases de données

Les bases de données se distinguent par leur structure, leur usage et les technologies qui les soutiennent. Chaque type est conçu pour répondre à des besoins spécifiques dans des contextes variés.

Les bases de données relationnelles (RDBMS), comme MySQL, PostgreSQL et Oracle, utilisent un modèle tabulaire où les données sont organisées en lignes et colonnes. Ce type est idéal pour des applications nécessitant des transactions complexes et une forte consistance des données, comme les systèmes bancaires.

En revanche, les bases de données NoSQL sont adaptées aux données non structurées ou semi-structurées. Par exemple, Redis, un système clé-valeur, est optimisé pour des cas d'utilisation où la rapidité d'accès aux données est cruciale, comme les caches. MongoDB, une base documentaire, est souvent utilisée pour stocker des objets JSON dans des applications web modernes. Cassandra, une base orientée colonnes, offre une scalabilité horizontale exceptionnelle pour les gros volumes de données, tandis que Neo4j, une base de données graphe, excelle dans la gestion de relations complexes, comme les réseaux sociaux.

Certaines bases de données, comme Redis, fonctionnent entièrement en mémoire pour garantir des temps de réponse ultra-rapides, bien qu'elles soient souvent limitées par la taille de la RAM. Les bases orientées objet, moins courantes, intègrent des concepts de programmation orientée objet pour des cas où l'alignement entre données et objets est essentiel.

Enfin, les bases de données distribuées, comme Amazon DynamoDB, sont conçues pour offrir une haute disponibilité et résilience en répartissant les données sur plusieurs serveurs. Elles sont particulièrement utiles pour les applications nécessitant une tolérance aux pannes réseau.

En complément, vous pouvez également explorer SQLite, une base de données intégrée légère, idéale pour les applications embarquées ou le prototypage rapide, via SQLite, ainsi que H2, une base de données Java particulièrement adaptée aux tests et au développement, disponible sur H2 Database.

Pour renforcer votre compréhension, il est recommandé de créer un mémo synthétisant tous les concepts abordés dans cette section. Cela inclut les types de bases de données, leurs caractéristiques principales, les exemples de technologies associées, ainsi que les cas d'utilisation pour chaque type. Ce mémo servira de référence rapide pour vos projets futurs.

Ressources : (à lire uniquement à ce stade)

Voici quelques liens vers des ressources en ligne pour apprendre SQL, MySQL et PostgreSQL :

- W3Schools - Tutoriel PostgreSQL : PostgreSQL Tutorial - Un guide complet pour apprendre PostgreSQL avec des exemples et des exercices pratiques[1].
- W3Schools - Introduction à PostgreSQL : Introduction to PostgreSQL - Une introduction pour débuter avec PostgreSQL[2].
- W3Schools - Tutoriel MySQL : MySQL Tutorial - Un tutoriel pour apprendre MySQL avec des exemples et des exercices[3].
- GeeksforGeeks - Tutoriel PostgreSQL : PostgreSQL Tutorial - Un tutoriel couvrant les concepts de base à avancés de PostgreSQL[4].

Voici quelques liens vers des ressources en ligne pour apprendre les bases de données NoSQL, notamment Redis, MongoDB, Cassandra et Neo4j :

- NoSql Databases: Cassandra vs Mongo vs Redis DB Comparison - Java Code Geeks - Une comparaison détaillée des bases de données NoSQL populaires.
- System Properties Comparison Apache Cassandra vs. Neo4j vs. Redis - Une vue détaillée côte à côte de Cassandra, Neo4j et Redis.
- Free Video: NoSQL Databases - Neo4j, MongoDB, REDIS, Column Store Databases from YouTube | Class Central - Explorez les bases de données NoSQL comme Neo4j, MongoDB, Redis et les bases de données en colonnes.

Pour aller plus loin, quelques ressources qui peuvent vous être utiles dans le futur :

- MySQL
- MongoDB
- Redis
- Cassandra
- Neo4j

Etape 2 : Comment choisir un Système de Gestion de Base de Données (SGBD)

Lorsqu'il s'agit de sélectionner un système de gestion de base de données (SGBD), plusieurs facteurs doivent être pris en compte pour garantir que le choix est adapté aux besoins du projet. Le choix doit reposer sur une analyse détaillée des exigences techniques et fonctionnelles, ainsi que sur les contraintes opérationnelles.

Une approche possible est de s'appuyer sur le **CAP Theorem** :

Le théorème CAP est un concept fondamental dans le domaine des bases de données distribuées. Il a été formulé par Eric Brewer en 2000 et stipule qu'il est impossible pour un système de base de données distribuée de garantir simultanément les trois propriétés suivantes :

1. Consistency (Cohérence)** : Tous les nœuds voient les mêmes données au même moment. Une lecture garantit de retourner le résultat de la dernière écriture.
2. Availability (Disponibilité)** : Chaque requête reçoit une réponse, sans garantie que cette réponse soit la plus récente. Le système est opérationnel et répond aux requêtes même en cas de panne de certains nœuds.
3. Partition Tolerance (Tolérance aux partitions)** : Le système continue de fonctionner malgré des partitions réseau. Les nœuds peuvent être divisés en plusieurs groupes qui ne peuvent pas communiquer entre eux.

Selon le théorème CAP, un système distribué ne peut garantir que deux de ces trois propriétés en même temps. Voyons cela avec des exemples :

Exemples d'Utilisation

1. CA (Cohérence et Disponibilité) :
- Exemple : Un système de base de données traditionnelle comme MySQL en mode non-distribué.
- Description : Dans un environnement non distribué, où il n'y a pas de partition réseau, un système peut garantir à la fois la cohérence et la disponibilité. Toutes les lectures renvoient les données les plus récentes, et le système est toujours disponible pour les requêtes.
2. CP (Cohérence et Tolérance aux Partitions) :
- Exemple : HBase, un système de gestion de base de données orienté colonnes.
- Description : HBase privilégie la cohérence et la tolérance aux partitions. En cas de partition réseau, le système peut devenir indisponible pour garantir que toutes les lectures renvoient des données cohérentes.
3. AP (Disponibilité et Tolérance aux Partitions) :**
- Exemple : Cassandra, une base de données NoSQL orientée colonnes.

- Description : Cassandra est conçue pour être hautement disponible et tolérante aux partitions. En cas de partition réseau, le système continue de répondre aux requêtes, mais il peut y avoir des incohérences temporaires entre les nœuds. Les données seront éventuellement cohérentes une fois la partition résolue.

Le théorème CAP est crucial pour comprendre les compromis inhérents aux systèmes de bases de données distribuées. Lors de la conception d'un système, il est essentiel de déterminer quelles propriétés sont les plus importantes pour l'application spécifique. Par exemple, une application bancaire pourrait privilégier la cohérence et la disponibilité (CA), tandis qu'un système de messagerie instantanée pourrait privilégier la disponibilité et la tolérance aux partitions (AP).

Voici quelques liens vers des ressources en ligne pour mieux comprendre le théorème CAP :

1. CAP Theorem Explained - by Ashish Pratap Singh - Un article détaillé expliquant le théorème CAP et ses implications dans la conception de systèmes distribués.
2. What is CAP Theorem? Definition & FAQs | ScyllaDB - Une explication du théorème CAP et de ses implications pour les bases de données distribuées.
3. CAP theorem - Wikipedia - Une page Wikipédia détaillée sur le théorème CAP, son histoire et ses implications.
4. What Is the CAP Theorem? | IBM - Un article d'IBM expliquant le théorème CAP et son importance dans la conception de systèmes distribués.
5. The CAP Theorem in DBMS | GeeksforGeeks - Un article expliquant les compromis inhérents au théorème CAP dans les systèmes de gestion de bases de données distribuées.

Pour compléter l'approche, voici quelques éléments à réaliser étape par étape afin de choisir quel système de gestion de base de données choisir.

La première étape consiste à comprendre la nature des données à manipuler. Si les données sont fortement structurées et nécessitent des relations complexes, un système relationnel tel que PostgreSQL ou MySQL sera pertinent. En revanche, pour des données semi-structurées ou non structurées, comme des documents JSON ou des flux de données, des bases NoSQL comme MongoDB ou Cassandra peuvent être plus adaptées.

Ensuite, il est essentiel de considérer les exigences de performance et de scalabilité. Les bases de données en mémoire comme Redis sont idéales pour les applications en temps réel, tandis que les bases distribuées comme Amazon DynamoDB permettent de gérer des charges importantes grâce à leur scalabilité horizontale.

Les besoins spécifiques de disponibilité et de tolérance aux pannes doivent également être évalués. Par exemple, pour des systèmes critiques nécessitant une haute disponibilité, il est préférable d'opter pour des bases distribuées tolérantes aux partitions.

Enfin, les contraintes budgétaires et les compétences techniques disponibles dans l'équipe influencent également le choix. Les bases de données open source comme PostgreSQL ou SQLite sont économiques et disposent d'une communauté active, tandis que des solutions propriétaires comme Oracle peuvent offrir un support avancé moyennant des coûts plus élevés.

Exemples de cahiers des charges adaptés :

- **Application Web classique :** Utilisation d'une base relationnelle comme MySQL pour la gestion des utilisateurs et des transactions.
- **Plateforme de données massives :** Adoption de Cassandra pour stocker et interroger de grandes quantités de données scalables horizontalement.
- **Application en temps réel :** Implémentation de Redis pour gérer des sessions utilisateur ou des statistiques en temps réel.
- **Projet embarqué ou prototype :** Usage de SQLite pour une solution légère et facile à intégrer.

Ces exemples montrent comment aligner un SGBD aux besoins spécifiques d'un projet. Pour chaque projet, il est conseillé de rédiger un cahier des charges précis décrivant les besoins, les contraintes et les cas d'utilisation pour faciliter la prise de décision.

Exercices pratiques :

Pour mettre en pratique les concepts appris, voici deux exercices qui vous permettront de vous familiariser avec l'analyse des besoins et la sélection d'un SGBD adapté dans un contexte lié aux datasciences :

1. **Exercice 1 : Analyse des besoins pour une plateforme d'analyse prédictive**
   - **Contexte :** Une entreprise souhaite développer une plateforme pour analyser les données clients et prédire les comportements futurs à partir de jeux de données structurés et non structurés.
   - **Cahier des charges :**
     - Les données incluent des transactions historiques (structurées) et des commentaires clients issus des réseaux sociaux (non structurés).
     - Le système doit permettre des requêtes rapides pour générer des rapports et supporter une croissance importante des données.
     - L'analyse des données sera effectuée en temps différé (batch).
     - Budget limité, préférence pour une solution open source.
   - **Question :** Quel type de SGBD recommanderiez-vous ? Justifiez votre choix.
2. **Exercice 2 : Gestion en temps réel d'un projet de visualisation de données IoT**
   - **Contexte :** Une start-up développe une application pour visualiser en temps réel les données provenant de capteurs IoT installés dans des usines.
   - **Cahier des charges :**
     - Les capteurs envoient des données toutes les secondes, nécessitant un traitement et une visualisation instantanée.
     - Le système doit supporter des pics de charge lors des périodes de forte activité.
     - Haute disponibilité requise pour assurer un suivi constant des données.
     - Les relations entre les différentes entités (capteurs, usines, machines) doivent être modélisées pour certaines analyses spécifiques.

- **Question :** Quel type de SGBD serait le plus adapté à ce projet ? Fournissez une explication détaillée de votre choix.

3. **Exercice 3 : Analyse des logs pour la détection d'anomalies**

   - **Contexte :** Une entreprise souhaite développer un système permettant d'analyser des logs générés par ses applications pour détecter des anomalies et prévenir les incidents techniques.

   - **Cahier des charges :**

     - Les logs sont volumineux et semi-structurés, avec un besoin de traitement rapide.
     - Le système doit permettre des analyses en temps différé et en temps réel selon les besoins.
     - Une intégration avec des outils d'analyse comme Apache Spark est prévue.

   - **Question :** Quel type de SGBD choisiriez-vous pour cette tâche et pourquoi ? Justifiez votre recommandation.

4. **Exercice 4 : Base de données pour un tableau de bord analytique en temps réel**

   - **Contexte :** Une plateforme en ligne veut offrir à ses clients un tableau de bord interactif affichant des métriques clés, comme les ventes et les performances des campagnes marketing, en temps réel.

   - **Cahier des charges :**

     - Les données sont alimentées en continu par des systèmes externes via des API.
     - Les utilisateurs doivent pouvoir effectuer des requêtes complexes sur les données récentes et historiques.
     - Les temps de réponse doivent être inférieurs à une seconde.
     - L'architecture doit pouvoir évoluer rapidement avec la croissance des utilisateurs.

   - **Question :** Quel type de SGBD recommanderiez-vous et quelles technologies spécifiques utiliseriez-vous ? Expliquez votre choix.

Ces exercices vous permettront de réfléchir aux spécificités des différents SGBD dans des cas pratiques et de mieux comprendre leur application dans des projets liés aux datasciences.

Pour compléter votre apprentissage, mettez à jour votre mémo en y intégrant les apprentissages récents. Assurez-vous d'y inclure les critères de choix d'un SGBD, les exemples de cahiers des charges ainsi que vos réponses aux exercices proposés. Cela vous permettra de consolider vos connaissances et de disposer d'une référence utile pour des projets futurs.

ITÉRATION 2

Configurer son environnement de travail

Modalités

- Travail individuel en autonomie

Livrables

Votre poste de travail permet de se connecter à une base de données Postgresql

Objectifs

Installer les logiciels nécessaires au travail sur une base de données

Partie 3 : Outils, logiciels et langages pour interagir avec les bases de données

Clients SQL pour Windows et Linux

Pour manipuler des bases de données SQL, vous pouvez installer des outils graphiques ou en ligne de commande qui facilitent l'interaction avec les bases de données.

Pour choisir lequel correspond le mieux à vos attentes, n'hésitez pas à travailler en ilot afin d'explorer les possibilités de chacun des outils.

Installation pour Windows

1. **DBeaver :** Téléchargez et installez DBeaver Community, un client SQL multiplateforme riche en fonctionnalités.
2. **MySQL Workbench :** Installez cet outil à partir du site officiel de MySQL, idéal pour travailler avec les bases MySQL.
3. **pgAdmin :** Téléchargez pgAdmin pour administrer vos bases PostgreSQL facilement.
4. **DataGrip :** Une solution avancée proposée par JetBrains, idéale pour gérer des bases de données complexes et multi-SGBD. Téléchargez-le sur DataGrip.

Installation pour Linux

1. **DBeaver :** Téléchargez la version Linux depuis DBeaver Community. L'installation peut également se faire via des gestionnaires de paquets comme snap ou apt.
2. **MySQL Workbench :** Installez-le à l'aide des gestionnaires de paquets (sudo apt install mysql-workbench).
3. **pgAdmin :** Utilisez apt ou yum pour l'installation (sudo apt install pgadmin4).
4. **DataGrip :** Une solution avancée proposée par JetBrains pour la gestion des bases de données complexes. Téléchargez-le depuis DataGrip et suivez les instructions de configuration spécifiques à Linux.

Ces outils offrent des interfaces graphiques intuitives pour effectuer des requêtes, administrer des bases de données et visualiser des schémas.

En complétant cette partie avec ces outils et bibliothèques, vous disposerez d'un arsenal puissant pour vos projets de bases de données.

Partie 4 : Connexion aux bases de données SQL depuis un client et sécurité d'accès

Pour travailler efficacement avec les bases de données SQL, il est important de savoir comment se connecter à une base de données depuis un client installé, tout en garantissant la sécurité des accès. Voici les étapes clés et les bonnes pratiques :

Pour commencer, il est important d'installer une base de données PostgreSQL pour pratiquer. Voici quelques ressources utiles :

- Guide d'installation de PostgreSQL pour Windows
- Tutoriel pour l'installation de PostgreSQL sous Linux

Ces guides vous fourniront les étapes nécessaires pour configurer PostgreSQL sur votre système d'exploitation.

Connexion aux bases de données SQL depuis un client

Pour approfondir, voici quelques ressources pratiques pour vous guider dans la configuration des clients SQL :

- Guide officiel de configuration pour PostgreSQL
- Documentation de DBeaver
- Tutoriel MySQL Workbench
- Introduction à pgAdmin

Ces ressources détaillent les étapes pour configurer et optimiser vos connexions, tout en offrant des exemples concrets pour différents environnements.

1. **Configurer le client SQL :**
   - Après l'installation du client (par exemple, DBeaver, DataGrip, ou MySQL Workbench), ouvrez le logiciel et sélectionnez l'option "Créer une nouvelle connexion".
   - Renseignez les informations nécessaires :
     - **Adresse du serveur** : Fournissez l'adresse IP ou le nom de domaine du serveur où la base de données est hébergée.

- **Port** : Généralement, 3306 pour MySQL, 5432 pour PostgreSQL, etc.
- **Nom de la base de données** : Spécifiez le nom de la base à laquelle vous voulez vous connecter.
- **Identifiants d'accès** : Entrez le nom d'utilisateur et le mot de passe.

- Testez la connexion pour vérifier que tout est correctement configuré.

2. **Interaction avec la base de données :**
   - Une fois connecté, utilisez l'interface graphique pour écrire et exécuter des requêtes SQL.
   - Vous pouvez également importer/exporter des données et gérer les schémas de la base.

Bonnes pratiques en matière de sécurité d'accès

1. **Utiliser des connexions sécurisées :**
   - Configurez votre serveur pour utiliser SSL/TLS afin de chiffrer les communications entre le client et la base de données.
   - Vérifiez que votre client SQL prend en charge cette option.
2. **Gestion des utilisateurs et des permissions :**
   - Assurez-vous que chaque utilisateur dispose uniquement des permissions nécessaires à ses tâches. Par exemple, un utilisateur chargé de lire des données ne devrait pas avoir accès aux commandes de modification.
   - Évitez d'utiliser le compte "root" ou administrateur pour des opérations courantes.
3. **Protéger les identifiants :**
   - Ne stockez pas les mots de passe en clair dans vos applications ou fichiers de configuration. Utilisez des gestionnaires de mots de passe ou des fichiers de configuration sécurisés.
   - Changez régulièrement les mots de passe des utilisateurs critiques.
4. **Surveiller l'accès :**

- Activez les journaux d'audit pour surveiller les connexions et les activités sur la base de données.
- Configurez des alertes pour détecter des connexions suspectes ou des tentatives de violation.

En suivant ces étapes et ces bonnes pratiques, vous pourrez interagir avec vos bases de données SQL de manière efficace et sécurisée.

Pour consolider vos apprentissages, prenez le temps de mettre à jour votre mémo en incluant les étapes d'installation de PostgreSQL, les configurations essentielles pour les clients SQL, et les bonnes pratiques de sécurité abordées dans cette partie. Ce mémo servira de guide pratique pour vos projets futurs et vous aidera à retenir les points clés.