Un exposé de 15 minutes sur la différence entre SQL et NoSQL en termes de **sécurité** peut être structuré pour couvrir à la fois les bases et des aspects techniques intéressants. Voici un plan que tu peux suivre pour ton exposé :

Plan de l'exposé

Introduction (2-3 minutes)

- **Définition des bases :**
  - **SQL (Structured Query Language)** : Base de données relationnelle (RDBMS), comme MySQL, PostgreSQL, ou SQL Server. Structure tabulaire avec des relations définies.
  - **NoSQL (Not Only SQL)** : Base de données non relationnelle, comme MongoDB, Cassandra, ou DynamoDB. Structure flexible (documents, clés-valeurs, colonnes, graphes).
- **Pourquoi la sécurité est essentielle ?**
  - Les bases de données stockent des informations sensibles comme des mots de passe, des informations financières, ou des données clients.

Différences majeures en termes de sécurité (8-10 minutes)

1. Modèle de sécurité intégré :

- **SQL :**
  - Sécurité basée sur un modèle utilisateur-privilege :
    - Contrôle d'accès via **Rôles et Permissions** (GRANT, REVOKE).
    - Niveau de granularité élevé : droits par table, vue, colonne.
  - Gestion de la sécurité grâce à l'intégrité transactionnelle (ACID).
  - Intégration native avec des protocoles comme SSL/TLS.
- **NoSQL :**
  - Moins standardisé : dépend des bases spécifiques.
    - MongoDB utilise des rôles, mais la granularité peut être limitée.
  - Certaines bases (comme Redis) ne proposent pas de gestion native avancée des permissions.
    - Exemple : accès souvent basé sur une simple clé d'authentification.

2. Chiffrement des données :

- **SQL :**
  - Offre souvent des solutions intégrées de chiffrement :
    - **Chiffrement au repos** (TDE – Transparent Data Encryption).
    - **Chiffrement en transit** (via TLS/SSL).
  - Intégration native avec des normes comme **AES-256**.
- **NoSQL :**
  - Le chiffrement n'est pas toujours activé par défaut.

        - MongoDB supporte le chiffrement au repos depuis les versions récentes.
        - Certaines bases nécessitent des configurations manuelles ou des services tiers pour le chiffrement.

3. Gestion des données sensibles (Compliance) :

- **SQL :**
    - Respecte les normes comme **GDPR, HIPAA, PCI-DSS**, car les outils sont matures.
    - Supporte des fonctionnalités comme le masquage dynamique des données et l'audit.
- **NoSQL :**
    - Moins mature sur certains standards de compliance.
    - Complexité dans l'implémentation de certaines politiques d'audit et de masquage, car NoSQL favorise la flexibilité.

4. Sécurité des requêtes (Prévention des attaques) :

- **SQL :**
    - **Vulnérabilité connue : Injection SQL.**
        - Solution : Préparer les requêtes via des **statements paramétrés**.
        - Les outils de SQL modernes intègrent des protections natives contre ce type d'attaque.
- **NoSQL :**
    - Pas d'injections SQL traditionnelles, mais vulnérabilités spécifiques :
        - **NoSQL Injection**, notamment dans MongoDB ou DynamoDB, lorsqu'on ne vérifie pas les entrées utilisateur.
        - Solution : Valider et assainir les entrées.

5. Authentification et autorisation :

- **SQL :**
    - Protocoles robustes : intégration avec LDAP, Active Directory, Kerberos.
    - Authentification basée sur plusieurs niveaux (utilisateur, application).
- **NoSQL :**
    - Variable selon la base :
        - Par exemple, MongoDB supporte les rôles utilisateur, mais d'autres bases comme Redis s'appuient souvent sur une clé partagée.

6. Haute disponibilité et résilience (Sécurité des données) :

- **SQL :**
    - Sécurité accrue grâce aux mécanismes traditionnels : sauvegardes automatiques, réplication synchrone, et journaux de transactions.
- **NoSQL :**
    - Très performant pour la disponibilité : réplication asynchrone, sharding.
    - Le compromis : risque d'incohérence si le modèle CAP (Consistency, Availability, Partition tolerance) n'est pas bien géré.

Comparaison (Tableau récapitulatif)

Critère	SQL	NoSQL
Contrôle des accès	Granularité élevée, rôles intégrés	Variable, dépend de la base
Chiffrement	Intégré (au repos et en transit)	Parfois nécessitant une configuration manuelle
Compliance	Maturité élevée	Moins mature sur certains standards
Vulnérabilités communes	Injection SQL	Injection NoSQL, configurations par défaut vulnérables
Gestion des sauvegardes	Transactions et journaux robustes	Axé sur la réplication et la tolérance aux pannes

---

Conclusion (2 minutes)

- **SQL :**
  - Plus sécurisé dans les environnements critiques grâce à sa standardisation et à ses outils robustes.
- **NoSQL :**
  - Flexible, mais nécessite souvent une configuration manuelle pour atteindre le même niveau de sécurité.
- **Message clé :** Le choix dépend des **besoins spécifiques** de l'application (performance vs complexité) et de l'effort de sécurisation que tu es prêt à mettre en place.

SQL (Structured Query Language)

- **Définition :**
  SQL est un langage utilisé pour interagir avec des bases de données relationnelles (RDBMS). Ces bases de données organisent les données en tables avec des lignes et des colonnes, selon un schéma prédéfini.
- **Caractéristiques principales :**
  - Utilise un **schéma fixe** (structure rigide).
  - Les relations entre les tables sont définies par des clés primaires et des clés étrangères.
  - Supporte les **transactions ACID** (Atomicité, Cohérence, Isolation, Durabilité) pour garantir la fiabilité des données.
  - Exemples : MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server.

NoSQL (Not Only SQL)

- **Définition :**
  NoSQL désigne une famille de bases de données non relationnelles qui offrent une flexibilité dans la structure des données. Ces bases sont conçues pour gérer de grandes quantités de données non structurées ou semi-structurées.
- **Caractéristiques principales :**
  - Pas de **schéma fixe** (structure flexible).
  - Données stockées sous différentes formes : documents (JSON), paires clé-valeur, colonnes larges, ou graphes.
  - Conçu pour la **scalabilité horizontale** et la haute performance.
  - Exemples : MongoDB (documents), Redis (clé-valeur), Cassandra (colonnes), Neo4j (graphes).