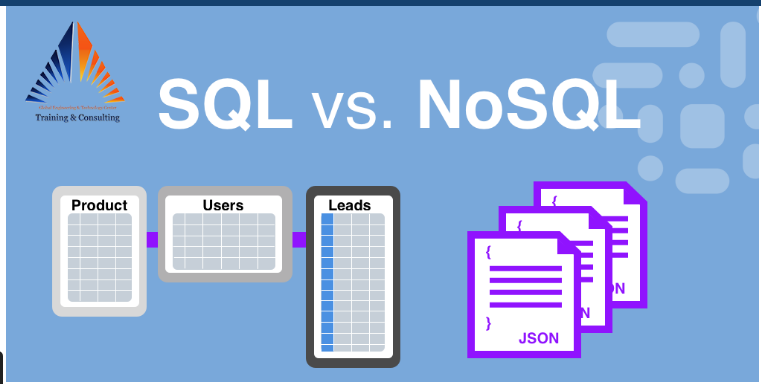
## TITRE : SQL VS NOSQL : QUELLES DIFFÉRENCES ?

1. **Définitions du SQL et du NoSQL**



A. SQL, le langage des données structure

Le SQL, qui signifie Structured Query Language, est un langage informatique normalisé permettant de communiquer avec une base de données.

Les bases de données relationnelles ont longtemps été les plus populaires. Il s’agit de bases de données dans lesquelles l’information est organisée avec des tableaux à deux dimensions nommées « tables ». Les lignes correspondent aux enregistrements. Chaque enregistrement contient un groupe d’informations – les attributs – relatives à un sujet. La plupart des systèmes de gestion de bases de données relationnelles reconnaissent le SQL.

Plus concrètement, le SQL a pour but de stocker, de manipuler et de retrouver ces données. Il permet également d’effectuer des requêtes, de mettre à jour les données ou de les réorganiser, de créer et de modifier le schéma et la structure d’un système de base de données.

### EXPEMPLE **BASES DE DONNÉES SQL :**

**Oracle**: **Oracle Database** est un système de gestion de base de données relationnelle et relationnel-objet, crée par Oracle dans les années 70. Elle est la première **database**conçue pour le grid computing. Le **grid computing** en entreprise est la technique la plus flexible et rentable pour gérer les systèmes informatiques et les applicatifs

**Mysql :** MySQL est un **SGBD**(Système de Gestion de Base de Données) relationnelle, crée par MySQL AB en 1995. Appréciée des professionnels et des particuliers, elle est la base de données la plus utilisée au monde.

### 

B. NoSQL, le SQL des bases de données non relationnelles

NoSQL signifie à la fois « Not only SQL », car certains langages NoSQL comprennent le langage SQL en plus de leur propre capacité, et « Non-relationnel » parce qu’il ne peut pas stocker facilement des données relationnelles.

La particularité des bases de données NoSQL est qu’elles n’utilisent pas le modèle relationnel. Il n’y a donc pas de tableau avec des caractéristiques et nombres d’attributs fixes. Les schémas sont donc absents ou flexibles. Cela permet de regrouper des données ayant des structures différentes. Les bases de données NoSQL peuvent également être distribuées, c’est-à-dire qu’elles peuvent être stockées sur plusieurs systèmes, plusieurs serveurs par exemple. Pour approfondir le sujet, nous vous conseillons l’article NoSQL : Tout comprendre sur les bases de données non relationnelles.

### EXPEMPLE **BASES DE DONNÉES NOSQL :**

**MONGODB :** Il s’agit d’une base de données NoSQL open source orientée document. MongoDB utilise des documents de type JSON pour stocker toutes les données. Il est écrit en C++.

**ORACLE NOSQL :** Oracle NoSQL Database implémente une carte allant des clés définies par l’utilisateur aux éléments de données non structurées.

1. **Les differences entre SQL et No SQL**

# LES TABLES SQL » VS « LES DOCUMENTS NOSQL »

Première différence de taille. SQL organise le stockage de données sur le principe de tables reliées entre elles. La structure et les types des données sont rigides, c’est-à-dire fixés à l’avance avant d’implémenter une logique métier.

noSQL stocke et manipule des documents qui correspondent à des collections d’objets.

Les tables SQL imposent un modèle de données strictes, donc il est difficile de faire des erreurs. NoSQL est plus flexible et pardonnable, mais la possibilité de stocker des données n’importe où peut entraîner des problèmes de cohérence.

# « LE SCHÉMA SQL » VS « LA LOGIQUE NOSQL »

Dans une base de données SQL, il est impossible d’ajouter des données tant que vous ne définissez des tables et des types de champs dans ce que l’on appelle un schéma. De plus, ce schéma SQL contient d’autres informations : Clés primaires – index, contraintes, fonction, procédures stockées …

Votre schéma de données doit être conçu et mis en œuvre avant que toute logique métier puisse être développée pour manipuler des données. Il est possible de faire des mises à jour plus tard, mais de gros changements peuvent être compliqués.

Dans une base de données NoSQL, la logique est toute autre ! Les données peuvent être ajoutées n’importe où, à tout moment. Il n’est pas nécessaire de spécifier une conception de document ou même une collection à l’avance

Il faut noter que la représentation des données en collection et le résultat en flux JSON des requêtes permet de consommer les données très rapidement et facilement par les applications front (web, mobile) qui incluent de plus en plus les appels Ajax.

# « LA NORMALIZATION SQL » VS « LA DENORMALISATION NOSQL »

Par les termes anglo-saxons « Normalization » et « Denormalisation » on veut préciser la façon dont les données sont ou pas dupliquées(noSQL) ou reliées par des clés étrangères (SQL) .

# « LA LOGIQUE DE JOINTURE SQL » VS « PAS DE JOINTURE DANS NOSQL »

Les requêtes SQL offrent une puissante clause JOIN. Nous pouvons obtenir des données reliées dans plusieurs tables en utilisant une seule instruction SQL. Cela renvoie tous les titres de livres, auteurs et noms d’éditeurs associés (en supposant que l’un a été défini).

NoSQL n’a pas toujours d’équivalent de JOIN, et cela peut étonner ceux qui ont une expérience SQL.

Il faut noter que d’autres SGDB ont implémenté un langage de requête proche du SQL autorisant l’usage de jointure. C’est le cas, par exemple de couchBase qui a créé le N1QL permettant de requêter une base noSQL suivant des principes proches du SQL.

# « INTÉGRITÉ SQL » VS « NOSQL DATA INTEGRITY »

La plupart des bases de données SQL vous permettent d’appliquer des règles d’intégrité de données à l’aide de contraintes de clés étrangères. Dans notre exemple, cela empêche de supprimer des Editeurs si un ou plusieurs livres leur sont toujours attribués.

Le schéma SQL applique ces règles qui prévient la création de données invalides ou d’enregistrements orphelins.

Ces mêmes options d’intégrité de données ne sont pas disponibles dans les bases de données NoSQL. Vous pouvez stocker ce que vous voulez indépendamment de tout autre document. Idéalement, un seul document sera la seule source de toutes les informations sur un élément. Une des premières choses que vous devez faire avec MongoDB est d’écrire une couche de données très claire et de s’assurer que « tout le monde » l’utilise correctement. Sinon, vous risquez vraiment de détruire « silencieusement » des données.

# « TRANSACTION SQL » VS « TRANSACTION NOSQL »

Dans les bases de données SQL, plusieurs mises à jour peuvent être exécutées au sein d’une même transaction afin de garantir le succès ou l’échec de l’exécution du code SQL. On n’imagine pas la vente d’un livre (ajout au journal de vente) dans décroitre le stock (décrémenter le stock du livre qui vient d’être vendu).

Dans une base de données NoSQL, la modification d’un document unique est atomique (si vous mettez à jour trois valeurs dans un document, les trois sont mis à jour avec succès ou ils restent inchangés°. Cependant, il n’y a pas vraiment d’équivalent de la transaction SQL pour les mises à jour de plusieurs documents. Il semble alors clair qu’il faille gérer la notion d’atomicité dans le code.

# SYNTAXE « CRUD SQL » VS « CRUD NOSQL »

La création, la lecture de mise à jour et la suppression de données sont à la base de tous les systèmes de base de données (CRUD Create, Read, Uodate, Delete).

SQL est un langage déclaratif qui est devenu une norme internationale (même si la plupart des sgbd implémentent des syntaxes subtilement différentes).

Les bases de données NoSQL utilisent des appels javascrip-looking avec des arguments JSON. Les opérations de base sont simples, mais cela peut très vite devenir compliqué pour des requêtes plus complexes.

On notera également que l’architecture orientée « document » de noSQL se prêterait plutôt bien à l’utilisation d’ORM car les documents qu’elle stocke sont essentiellement des objets eux-mêmes. Malheureusement, il n’y a pas beaucoup de bibliothèques Java ORM disponibles. Notons l’existence pour MongoDB de morphia (une bibliothèque Java type-safe pour MongoDB) et Spring-Data (la mise en œuvre MongoDB au sein du framework Spring) ou encore Mongoose si vous programmez en NodeJs.

# PERFORMANCE SQL VS NOSQL

C’est sûrement la comparaison la plus controversée ! NoSQL est régulièrement cité comme étant plus rapide que SQL. Et ce n’est pas surprenant. Le principe de « denormalization » induit une représentation plus simple et permet donc de récupérer toutes les informations sur un élément spécifique dans une seule requête. Il n’y a donc pas besoin de liens JOIN ou de requêtes SQL complexes. Mais la redondance des informations alourdie considérablement les opérations de mise à jour.

En résumé, les bases de données « orientées document » ne sont pas intrinsèquement plus rapides. Par exemple, MongoDB est inutilement lent dans de nombreux cas où SQL est particulièrement rapide. En particulier sous-performant sur les requêtes d’agrégation ou sans index.

# SQL VS NOSQL SCALABILITÉ

Il est souvent constaté que les problématiques liées à la répartition de charge posent de réels challenges pour les DSI des entreprises (Théorème de CAP Consistency Availability Partition tolerance) . Le « load balancing » sur un serveur SQL est complexe car dû à la nature même dont les données sont stockées, organisées et reliées entre elles. Pour régler les problèmes de montées en charge, on admettra le plus souvent le principe d’un puissant serveur adossé à une infra de type failover (exemple « Allways On » Ms Sql Server) plutôt que de « clusterer » les instances SQL. Cela a un impact direct sur les licences et donc le coût de ce type d’infrastructure.

Les modèles de données NoSQL peuvent rendre le processus plus facile et beaucoup d’entre eux ont été conçus nativement avec des fonctionnalités de « scalabilité élastique ». L’organisation des données en documents et la « denormalization » des collections permettent le partitionnement et autorise une montée en charge de la base de données sur le matériel courant déployé sur site ou dans le Cloud. Cela permet une croissance pratiquement illimitée.

Pour synthétiser, le but recherché est de simplifier l’architecture tout en décuplant les capacités de stockage. Cela implique le principe de base de données distribuée taillée pour la répartition de charge préférant la gestion d’une table gigantesque (cf. Bigtable de Google) à celle de nombreuses tables interdépendantes (modèle relationnel). A ce titre, les bases de données noSQL sont résolument orientée “Big Data”.