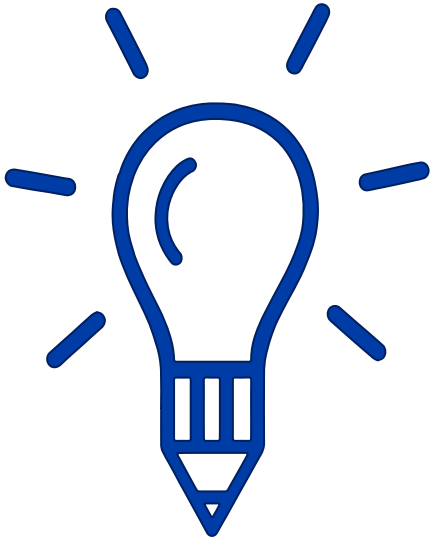
# Formation NoSQL

## *Partie 2 : Caractéristiques NoSQL*

# Caractéristiques NoSQL

# Caractéristiques NoSQL



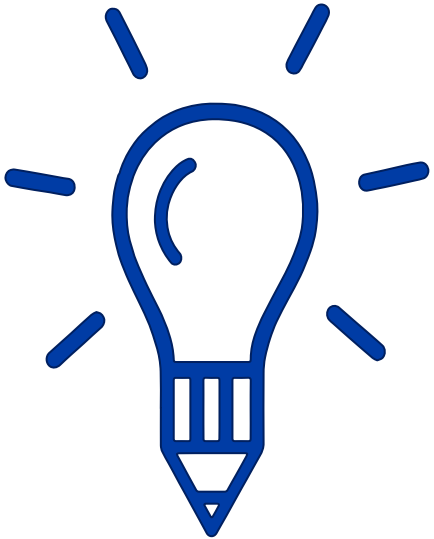
**01** Structure de données proches des utilisateurs, développeurs : sérialisation, tables de hachage, JSON

**02** Priorité au traitement du côté client

**03** Protocoles d'accès aux données, interfaces depuis les langages classiques

**04** Données structurées et non structurées, documents, images

# Caractéristiques NoSQL



**05** Stockage réparti : réplication, sharping, gossip protocl, hachage,...

**06** Parallélisation des traitements : implémentation de MapReduce

**07** Cohérence des données et gestion des accès concurrents :"eventual consistancy" et multi-version concurrency control

# Tendances

## *De plus en plus de clients en ligne*

Mise à l'échelle pour prendre en charge des milliers, voire des millions, d'utilisateurs

Satisfaire aux exigences de l'utilisateur grâce à des performances

élevées et constantes Maintenir une disponibilité 24 heures sur 24,

7 jours sur 7.

# Tendances

***L'internet connecte tout***

Prise en charge de nombreuses choses différentes avec différentes structures de données

Prise en charge des mises à jour de matériel/logiciels, générant des données différentes Prise en charge des flux continus de données en temps réel

# Tendances

***Big data : Données massives***

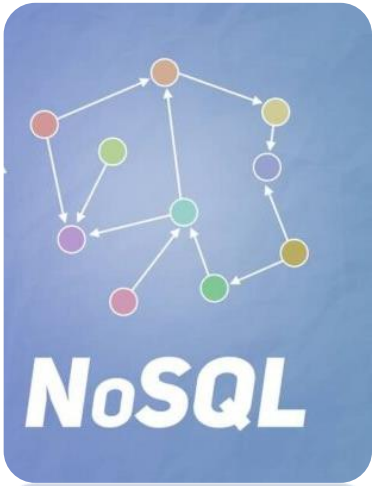
Stockage de données semi-structurées/non structurées générées par les clients Stockage de différents types de données provenant de différentes sources, ensemble Stockage de données générées par des milliers/millions de clients/objets

# Tendances

## *Stockage sur le Cloud*

Évolution à la demande pour prendre en charge davantage de clients et stocker plus de données Exploitation d'applications à l'échelle mondiale - clients dans le monde entier Réduction des coûts d'infrastructure, accélération de la mise sur le marché

# Tendances




CouchDB relax

Redis

Neo4j the graph database

Cassandra

mongoDB

membase

riak

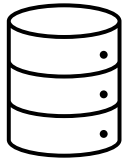
# *NoSQL* = *N*ot *O*nly *SQL*

## *Définition*

- NoSQL stocke les informations dans ***des documents JSON*** au lieu ***des colonnes et des lignes*** utilisées par les bases de données relationnelles.

- NoSQL signifie ***"pas seulement SQL"*** plutôt que ***"pas de SQL"*** du tout. Cela signifie qu'une base de données JSON NoSQL peut stocker et récupérer des données en utilisant littéralement ***"aucun SQL".***

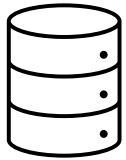
- Combiner ***la flexibilité de JSON avec la puissance de SQL*** pour obtenir le meilleur des deux.



# NoSQL = Not Only SQL

## Définition

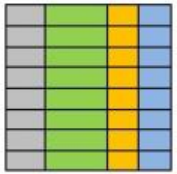
- Flexibles et évolutives.
- Capables de répondre rapidement aux exigences de gestion des données des entreprises modernes.



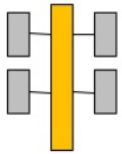
# NoSQL = Not Only SQL

## SQL Databases

Relational

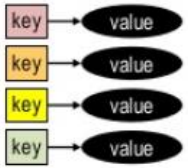


Analytical (OLAP)

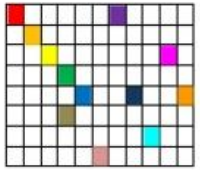


## NoSQL Databases

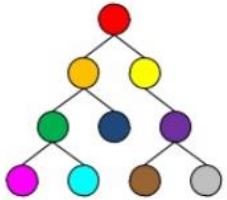
Key-Value



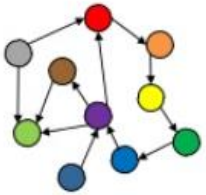

Column-Family



Document



Graph



# NoSQL = Not Only SQL

***Types de bases de données NoSQL les plus populaires***

## Key-value stores (Stockage Clé-Valeur)

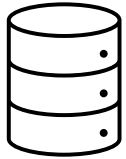
Ils regroupent les données associées dans des collections avec des enregistrements qui sont identifiés par des clés uniques pour une récupération facile.

Les Key-value stores ont juste assez de structure pour refléter la valeur des bases de données relationnelles tout en préservant les avantages du NoSQL.

ORACLE

BERKELEY DB

DynamoDB



amazon web services

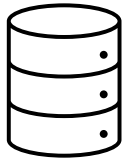
# NoSQL = Not Only SQL

***Types de bases de données NoSQL les plus populaires***

## *Wide-column databases (Les bases de données à colonnes larges)*

Ils utilisent le format tabulaire des bases de données relationnelles, mais permettent **une grande variation** dans la façon dont les données sont nommées et formatées dans chaque ligne, même dans la même table. Comme le stockage clé-valeur, **les bases de données à colonnes larges** ont une certaine structure de base tout en conservant **une grande flexibilité.**



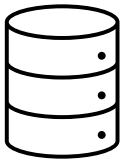
# NoSQL = Not Only SQL

***Types de bases de données NoSQL les plus populaires***

## Documents Databases (Les bases de données de documents )

*Les bases de données de documents* sont principalement construites pour stocker des informations sous forme de documents, y compris, mais sans s'y limiter, des documents JSON. Ces systèmes peuvent également être utilisés pour stocker des documents XML, par exemple.





# NoSQL = Not Only SQL

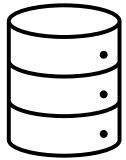
***Types de bases de données NoSQL les plus populaires***

## *Graph databases (Les bases de données graphiques)*

Ils utilisent des structures graphiques pour définir les relations entre les points de données stockés. Les bases de données graphiques sont utiles pour identifier des modèles dans des informations non structurées et semi-structurées.

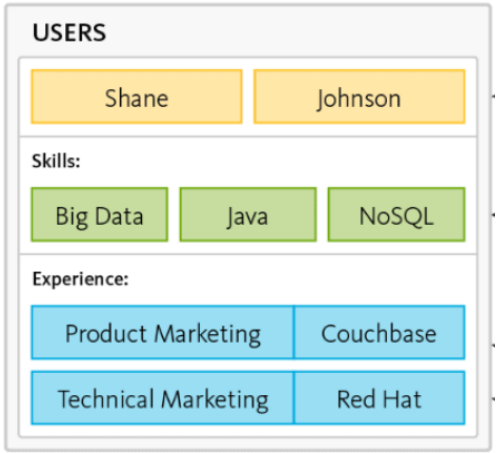
neo4j

FlockDB



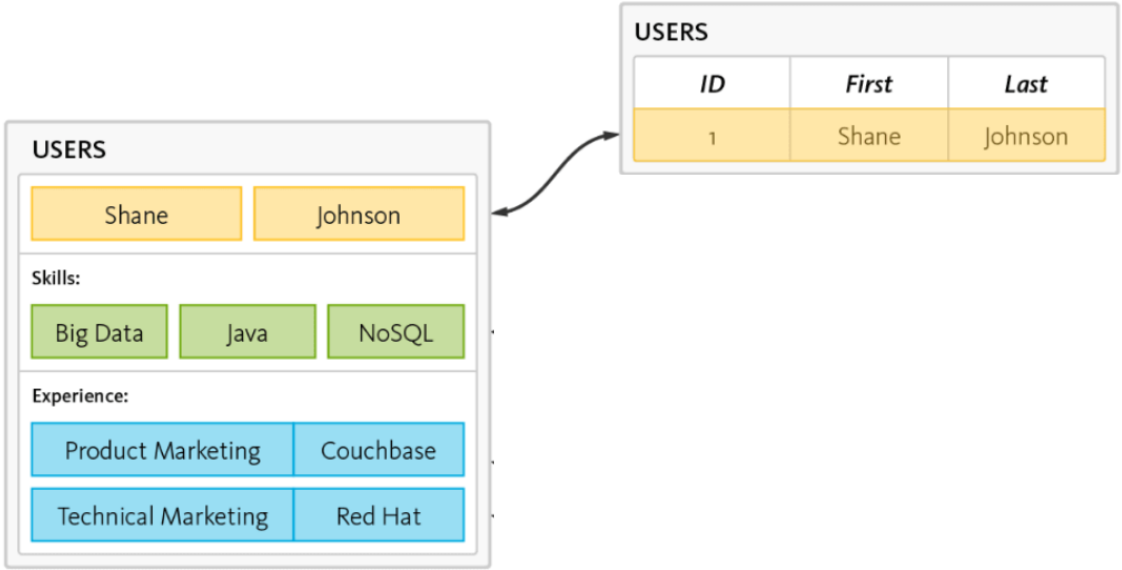
# NoSQL = Not Only SQL

***Comment fonctionne le NoSQL ?*** *Exemple*



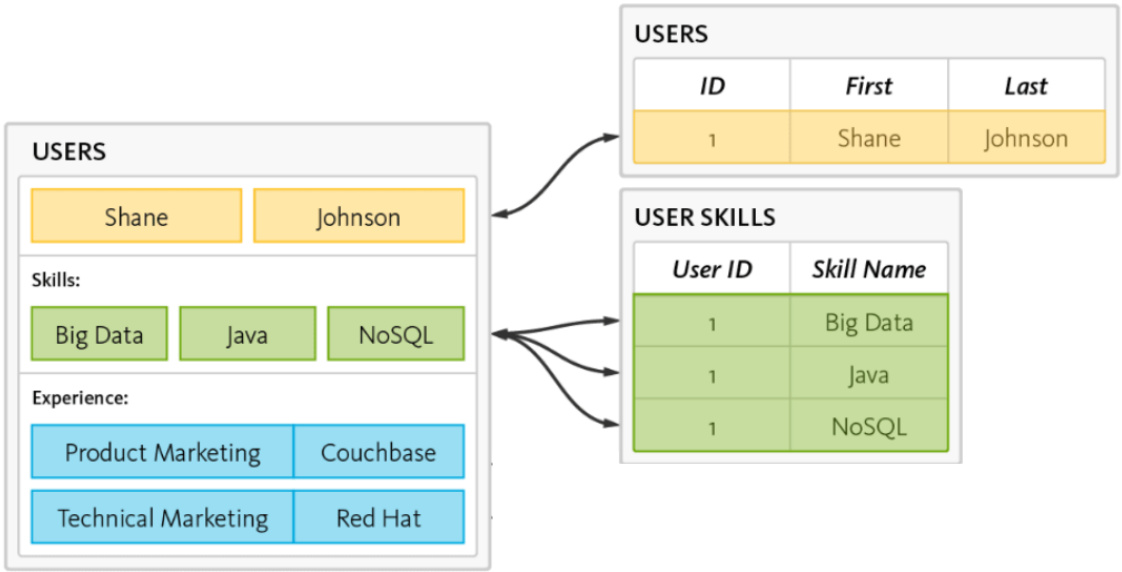
# NoSQL = Not Only SQL

***Comment fonctionne le NoSQL ? Exemple***



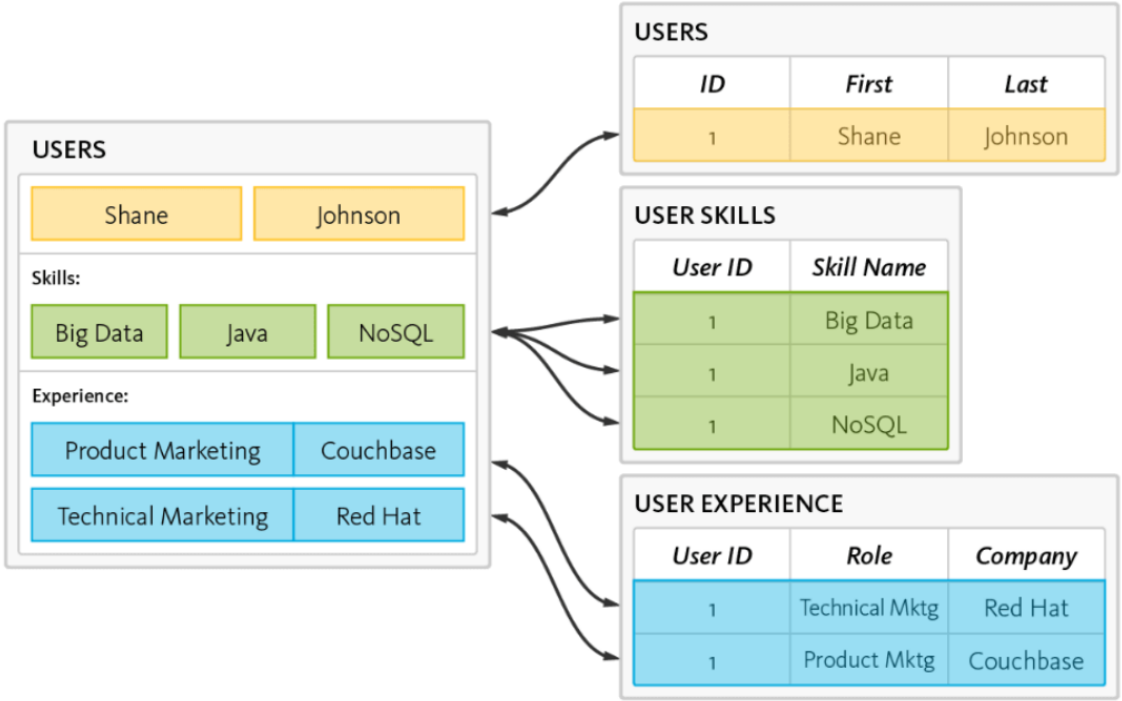
# NoSQL = Not Only SQL

***Comment fonctionne le NoSQL ? Exemple***



# NoSQL = Not Only SQL

***Comment fonctionne le NoSQL ? Exemple***




Ajouter 6 lignes !
Dans trois table

# NoSQL = Not Only SQL

***Comment fonctionne le NoSQL ?** Exemple*

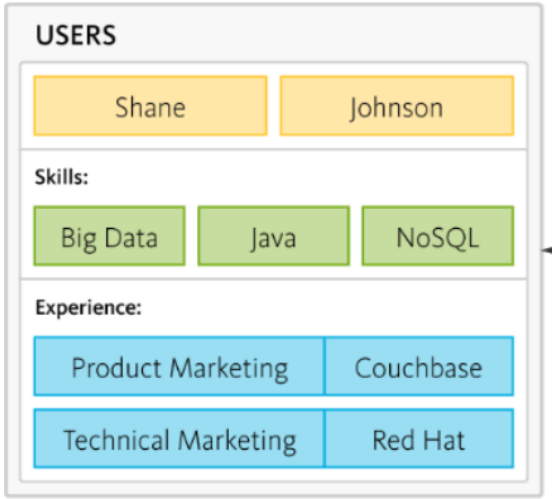
| | | | | |
|---|---|---|---|---|
| Shane | Johnson | Big Data | Product Marketing | Couchbase |
| Shane | Johnson | Big Data | Technical Marketing | Red Hat |
| Shane | Johnson | Java | Product Marketing | Couchbase |
| Shane | Johnson | Java | Technical Marketing | Red Hat |
| Shane | Johnson | NoSQL | Product Marketing | Couchbase |
| Shane | Johnson | NoSQL | Technical Marketing | Red Hat |

Redondance!

# NoSQL = Not Only SQL

***Comment fonctionne le NoSQL ? Exemple***

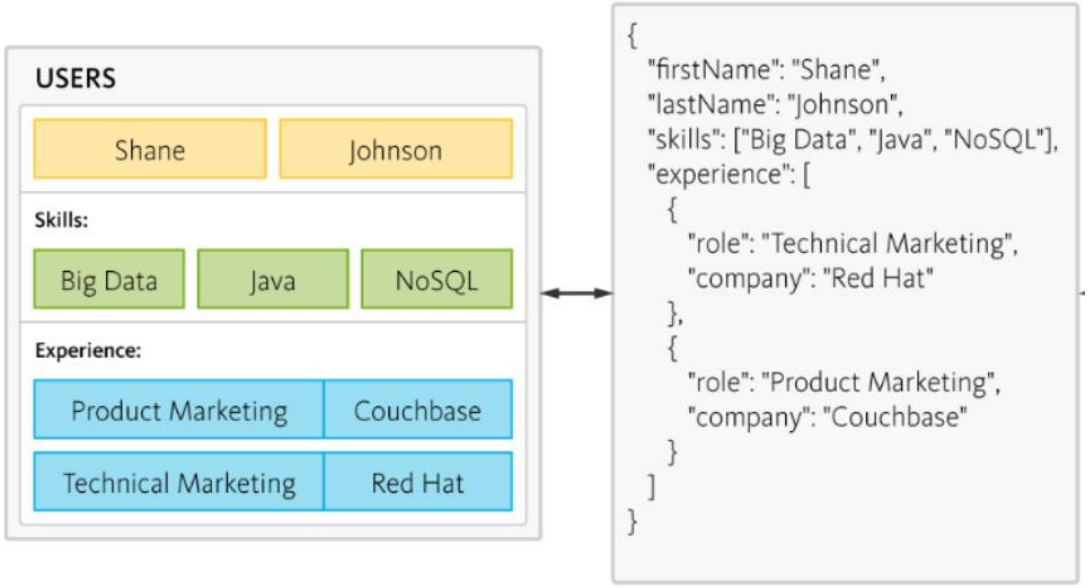
Une base de données orientée document définie comme NoSQL, JSON est **le bon format** pour le stockage des données de manière utile, c'est également **la bonne méthode** pour la consommation et la production de données pour les applications Web, mobiles et IoT.




# NoSQL = Not Only SQL

***Comment fonctionne le NoSQL ? Exemple***

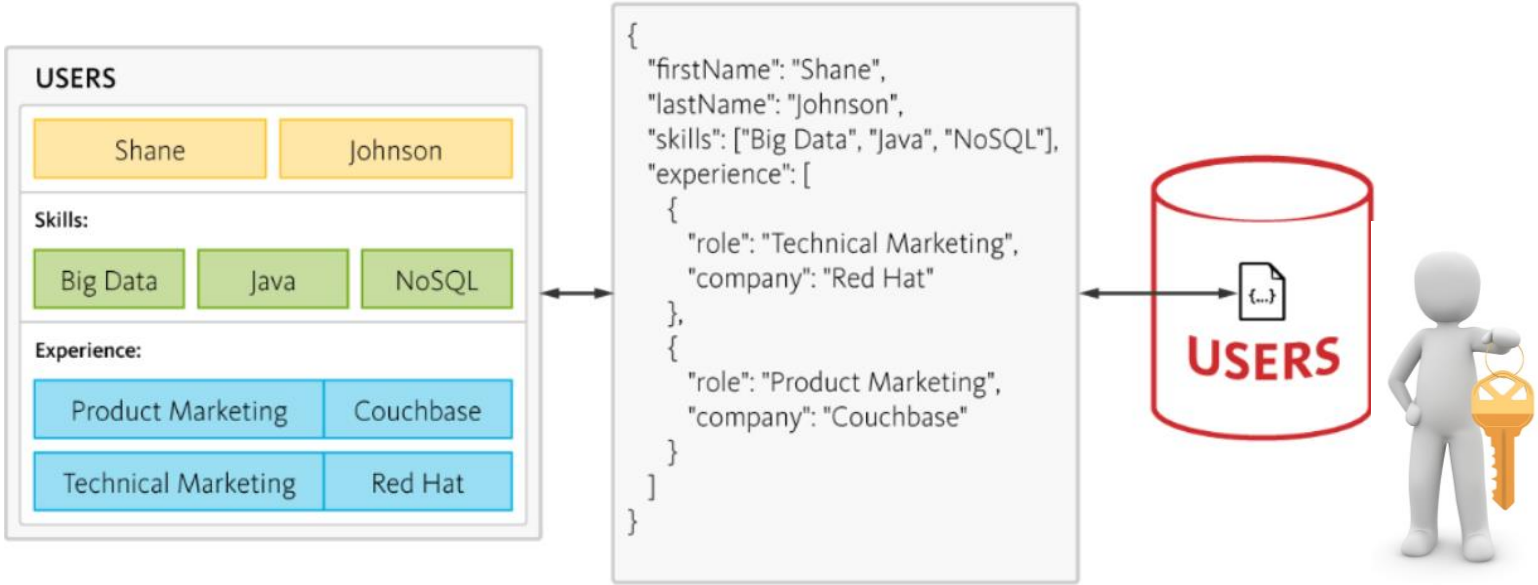
Une base de données orientée document définie comme NoSQL, JSON est **le bon format** pour le stockage des données de manière utile, c'est également **la bonne méthode** pour la consommation et la production de données pour les applications Web, mobiles et IoT.




# NoSQL = Not Only SQL

***Comment fonctionne le NoSQL ? Exemple***

Une base de données orientée document définie comme NoSQL, JSON est **le bon format** pour le stockage des données de manière utile, c'est également **la bonne méthode** pour la consommation et la production de données pour les applications Web, mobiles et IoT.



# *NoSQL* = *N*ot *O*nly *SQL*

***Qu'en est-il des requêtes et du SQL ?***

Certains ***pensent qu'il est plus difficile*** d'interroger les bases de données NoSQL, mais c'est ***une idée fausse.*** La flexibilité inhérente aux bases de données NoSQL orientées documents permet de traiter aussi bien les données structurées que les données non structurées, et les nouveaux outils permettent des requêtes plus rapides que jamais.

*Couchbase Server 4.0* a introduit N1QL, un puissant langage de requête qui étend SQL à JSON, permettant aux développeurs d'exploiter à la fois la *puissance de SQL* et *la flexibilité de JSON*. Il prend non seulement en charge les instructions standard **SELECT / FROM / WHERE**, mais aussi l'agrégation **(GROUP BY),** le **tri (SORT BY),** les jointures **(LEFT OUTER / INNER),** ainsi que l'interrogation de tableaux et de collections imbriqués.

# NoSQL = Not Only SQL

***Performance à grande échelle***

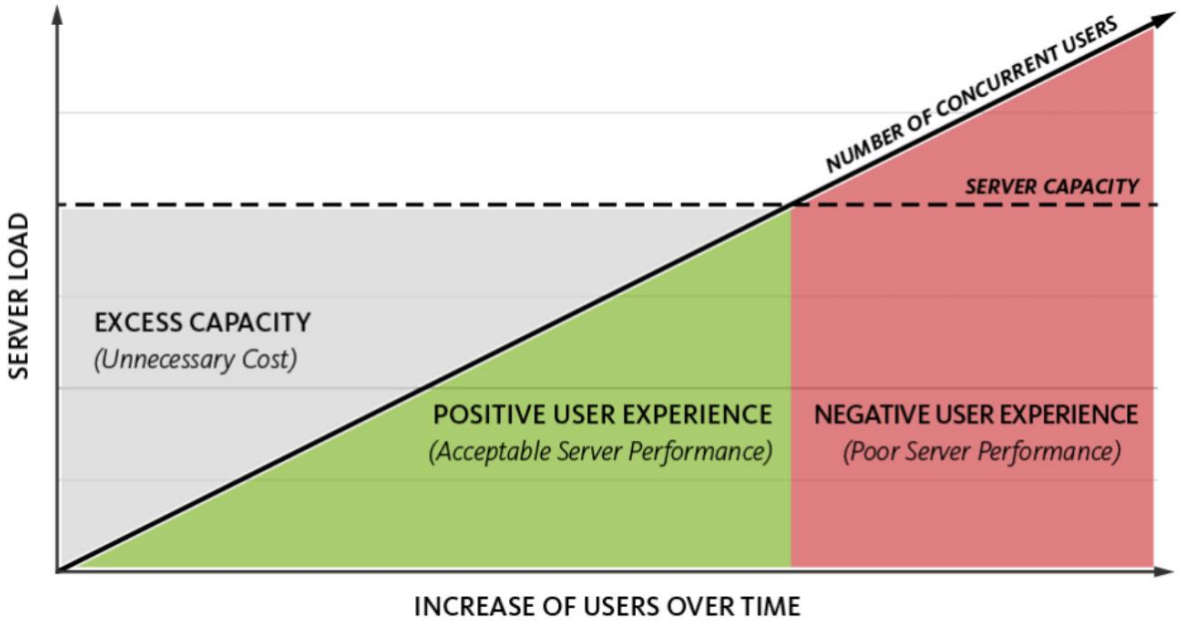
Les applications et les services doivent prendre en charge un nombre toujours croissant d'utilisateurs et de données.

La base de données doit être capable de faire évoluer les lectures, les écritures et le stockage.

C'est un problème pour les bases de données relationnelles qui sont limitées à la mise à l'échelle (c'est-à-dire à l'ajout de processeurs, de mémoire et de stockage sur un seul serveur physique). Par conséquent, la capacité à évoluer efficacement, et à la demande, est un défi.

# NoSQL = Not Only SQL

***Performance à grande échelle***



# NoSQL = Not Only SQL

***Performance à grande échelle***

Une base de données NoSQL distribuée, exploite le matériel de base pour évoluer, c'est-à-dire pour ajouter des ressources en ajoutant simplement des serveurs.

La capacité d'extension permet aux entreprises de s'adapter plus efficacement :

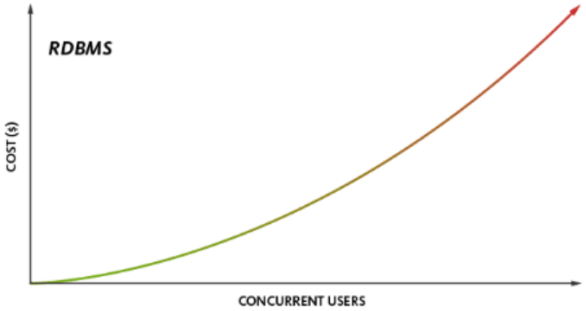
**a)** en ne déployant pas plus de matériel que nécessaire pour répondre à la charge actuelle ;

**b)** en exploitant du matériel moins coûteux et/ou une infrastructure en nuage ;
**c)** en s'adaptant à la demande et sans temps d'arrêt

# NoSQL = Not Only SQL

***Performance à grande échelle***

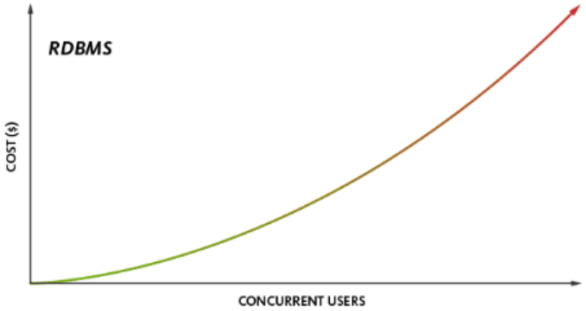
Une base de données NoSQL distribuée, exploite le matériel de base pour évoluer, c'est-à-dire pour ajouter des ressources en ajoutant simplement des serveurs.

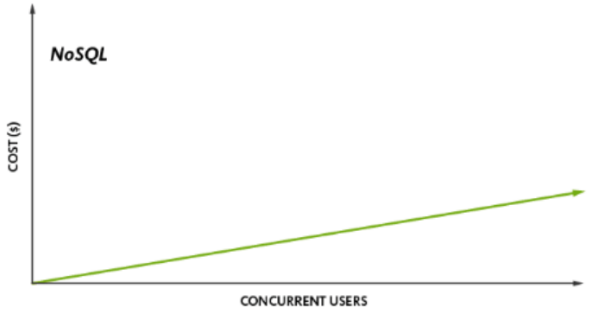



# NoSQL = Not Only SQL

***Performance à grande échelle***

Une base de données NoSQL distribuée, exploite le matériel de base pour évoluer, c'est-à-dire pour ajouter des ressources en ajoutant simplement des serveurs.




# NoSQL = Not Only SQL

***Disponibilité pour un déploiement global et permanent***

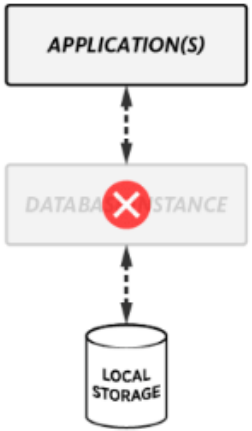
Comme de plus en plus d'engagements des clients se font en ligne via des applications web et mobiles, la disponibilité devient une préoccupation majeure, voire primordiale. Ces applications critiques doivent être disponibles 24 heures sur 24, 7 jours sur 7, sans exception.

Assurer une disponibilité 24 heures sur 24 et 7 jours sur 7 est un défi pour les bases de données relationnelles qui sont déployées sur un seul serveur physique ou qui reposent sur un cluster avec un stockage partagé.

Si elles sont déployées sur un seul serveur et que celui-ci tombe en panne, ou sur un cluster et que le stockage partagé tombe en panne, la base de données devient indisponible.

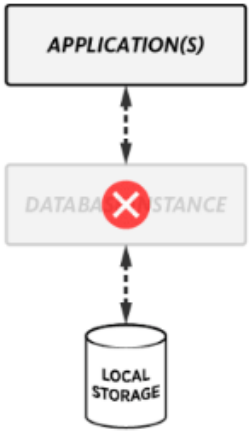
# NoSQL = Not Only SQL

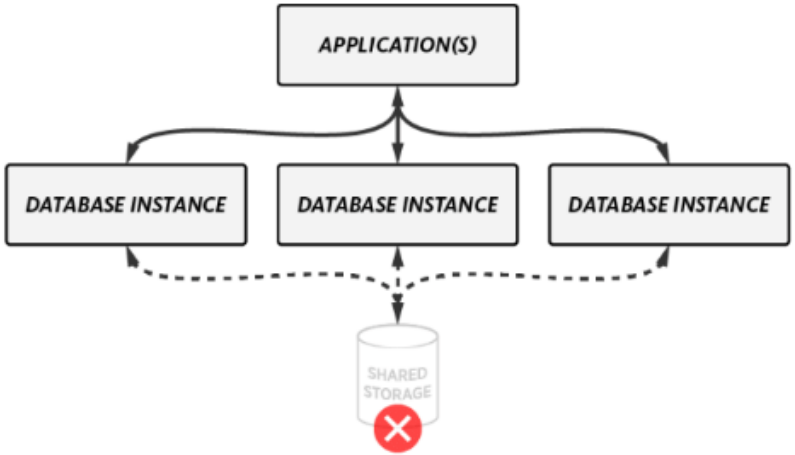
***Disponibilité pour un déploiement global et permanent***




# NoSQL = Not Only SQL

***Disponibilité pour un déploiement global et permanent***




# NoSQL = Not Only SQL

***Disponibilité pour un déploiement global et permanent***

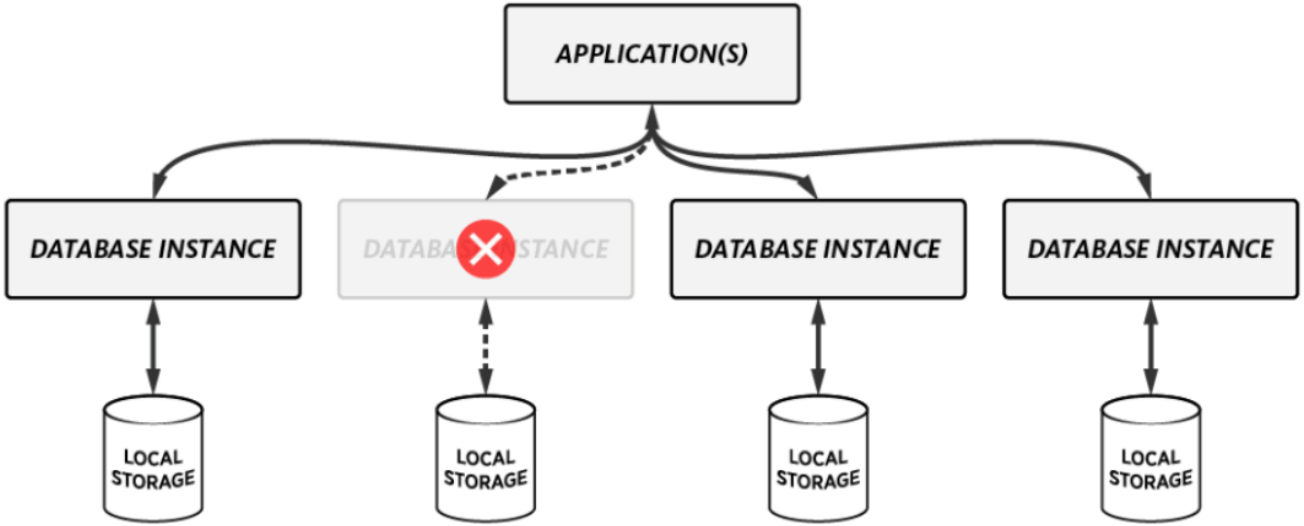
Contrairement à la technologie relationnelle, une base de données distribuée, NoSQL, partitionne et distribue les données à plusieurs instances de base de données sans ressources partagées.

En outre, les données peuvent être répliquées sur une ou plusieurs instances pour une haute disponibilité (réplication inter-clusters). *Alors que les bases de données relationnelles comme Oracle nécessitent un logiciel distinct pour la réplication (par exemple, Oracle Active Data Guard),* les bases de données NoSQL n'en ont pas besoin elle est intégrée et automatique

En outre, le basculement automatique garantit que si un nœud tombe en panne, la base de données peut continuer à effectuer des lectures et des écritures en envoyant les demandes à un autre nœud.

# NoSQL = Not Only SQL

***Disponibilité pour un déploiement global et permanent***




# NoSQL = Not Only SQL

***Disponibilité pour un déploiement global et permanent***

À mesure que les engagements des clients se déplacent en ligne, la nécessité d'être disponible dans plusieurs pays et/ou régions devient critique. Si le déploiement d'une base de données dans plusieurs centres de données augmente la disponibilité et facilite la reprise après sinistre, il a également l'avantage d'augmenter les performances, car toutes les lectures et écritures peuvent être exécutées dans le centre de données le plus proche, ce qui réduit la latence.

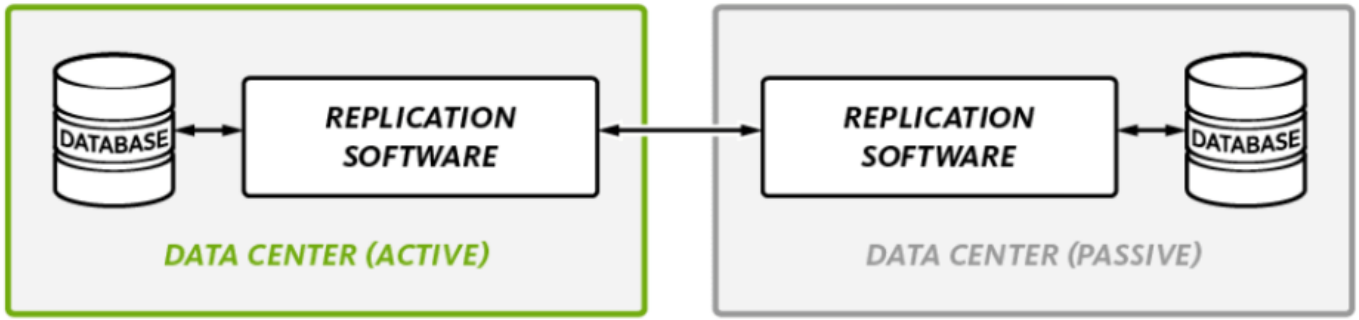
# NoSQL = Not Only SQL

***Disponibilité pour un déploiement global et permanent***

Il est difficile d'assurer une disponibilité globale pour les bases de données relationnelles dans les cas où l'exigence de modules complémentaires distincts accroît la complexité (par exemple, Oracle a besoin d'Oracle GoldenGate pour déplacer les données entre les bases de données) - ou lorsque la réplication entre plusieurs centres de données ne peut être utilisée que pour le basculement parce qu'un seul centre de données est actif à la fois. En outre, lors de la réplication entre les centres de données, les applications construites sur des bases de données relationnelles peuvent subir une dégradation des performances ou constater que les centres de données sont fortement désynchronisés.

# NoSQL = Not Only SQL

***Disponibilité pour un déploiement global et permanent***



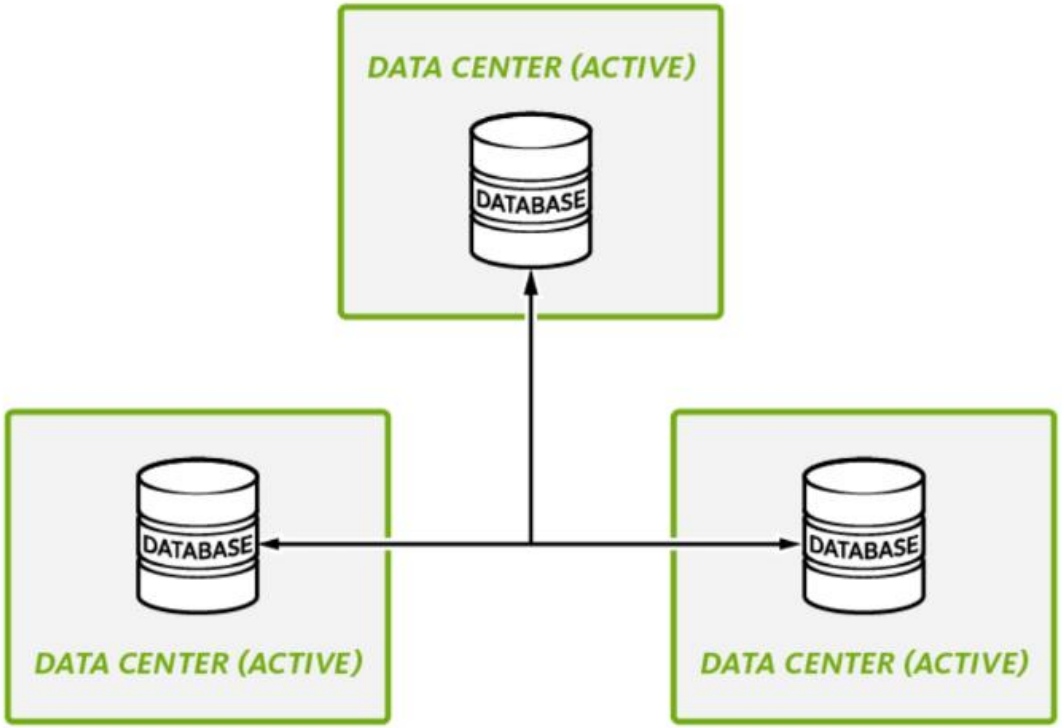

# NoSQL = Not Only SQL

***Disponibilité pour un déploiement global et permanent***

Une base de données NoSQL distribuée comprend une réplication intégrée entre les centres de données - **aucun logiciel distinct n'est nécessaire**. En outre, certaines incluent une réplication unidirectionnelle et bidirectionnelle permettant des déploiements actifs-actifs complets vers plusieurs centres de données, ce qui permet de déployer la base de données dans plusieurs pays et/ou régions et de fournir un accès local aux données aux applications locales et à leurs utilisateurs. Cela permet non seulement d'améliorer les performances, mais aussi d'assurer un basculement immédiat via des routeurs matériels : les applications ne doivent pas attendre que la base de données détecte la défaillance et effectue son propre basculement.

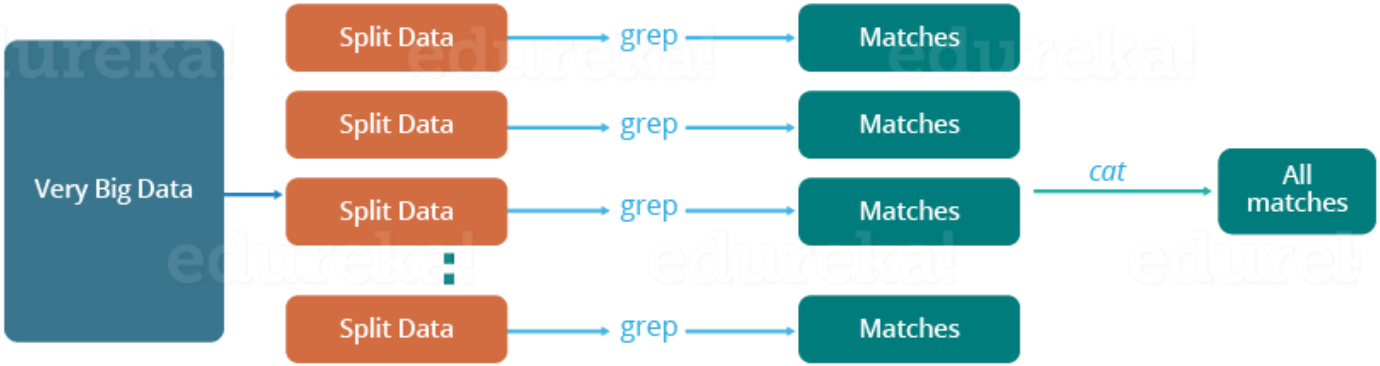
# NoSQL = Not Only SQL

***Disponibilité pour un déploiement global et permanent***




# NoSQL = Not Only SQL

***Méthode traditionnelle du traitement parallèl et distribué***




# NoSQL = Not Only SQL

***Méthode traditionnelle du traitement parallèl et distribué***

Supposons qu'un journal météo contenant la température moyenne quotidienne des années 2000 à 2015. Si on veux calculer le jour où la température est la plus élevée chaque année.

Donc, comme dans la méthode traditionnelle:

- Nous devons diviser les données en petites parties ou blocs et les stocker dans différentes machines.
- Ensuite, nous allons trouver la température la plus élevée dans chaque partie stockée dans la machine correspondante.
- Enfin, nous combinerons les résultats reçus de chacune des machines pour obtenir le résultat final.

# NoSQL = Not Only SQL

***Examinons les défis associés à cette approche traditionnelle :***

**Problème de chemin critique :** il s'agit du temps nécessaire pour terminer le travail sans retarder l'étape suivante ou la date d'achèvement réelle. Ainsi, si l'une des machines retarde le travail, l'ensemble du travail est retardé.

**Problème de fiabilité :** Que se passe-t-il si l'une des machines qui travaillent avec une partie des données tombe en panne ? La gestion de ce basculement devient un défi.

**Problème de division égale :** Comment vais-je diviser les données en plus petits morceaux afin que chaque machine obtienne une partie égale des données à travailler. En d'autres termes, comment diviser les données de manière égale afin qu'aucune machine individuelle ne soit surchargée ou sous-utilisée.

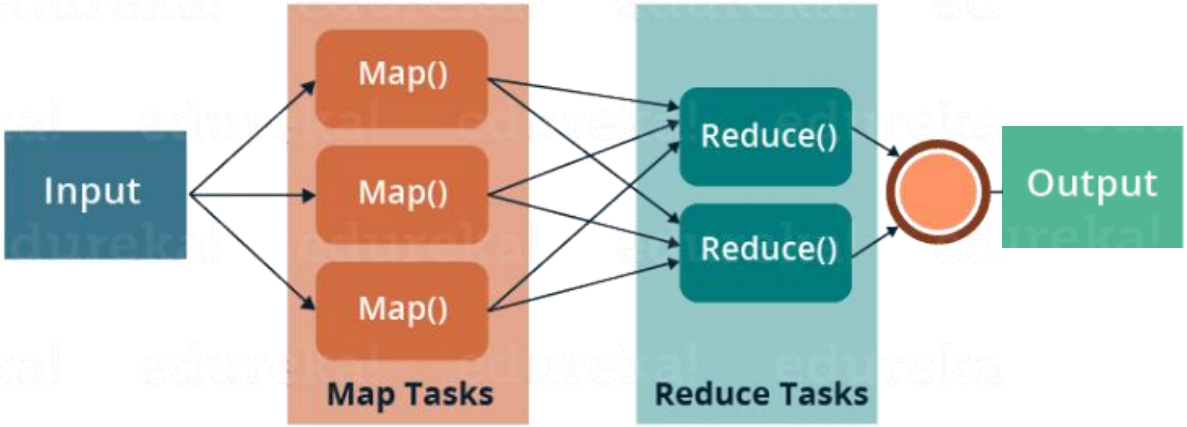
**La répartition unique peut échouer :** Si l'une des machines ne fournit pas la sortie, je ne serai pas en mesure de calculer le résultat. Il doit donc y avoir un mécanisme pour assurer cette capacité de tolérance aux pannes du système.

**Agrégation du résultat :** Il devrait y avoir un mécanisme pour agréger le résultat généré par chacune des machines pour produire la sortie finale.

# NoSQL = Not Only SQL

***Solutions***

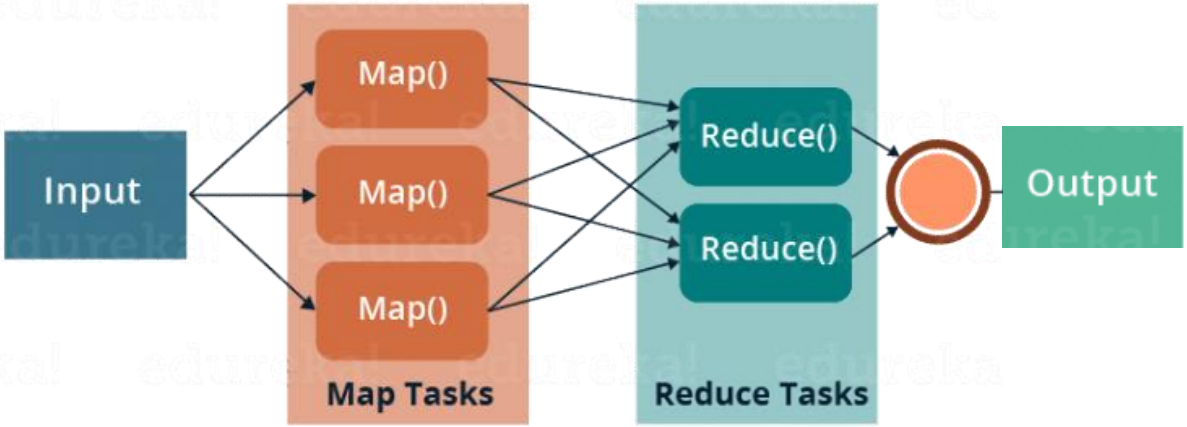
Pour surmonter ces problèmes, nous disposons du cadre **MapReduce** qui nous permet d'effectuer de tels calculs parallèles sans nous soucier de questions telles que **la fiabilité, la tolérance aux pannes, etc.** Par conséquent, MapReduce vous donne la flexibilité d'écrire la logique du code sans vous soucier des questions de conception du système.




# NoSQL = Not Only SQL

***MapReduce : Définition***

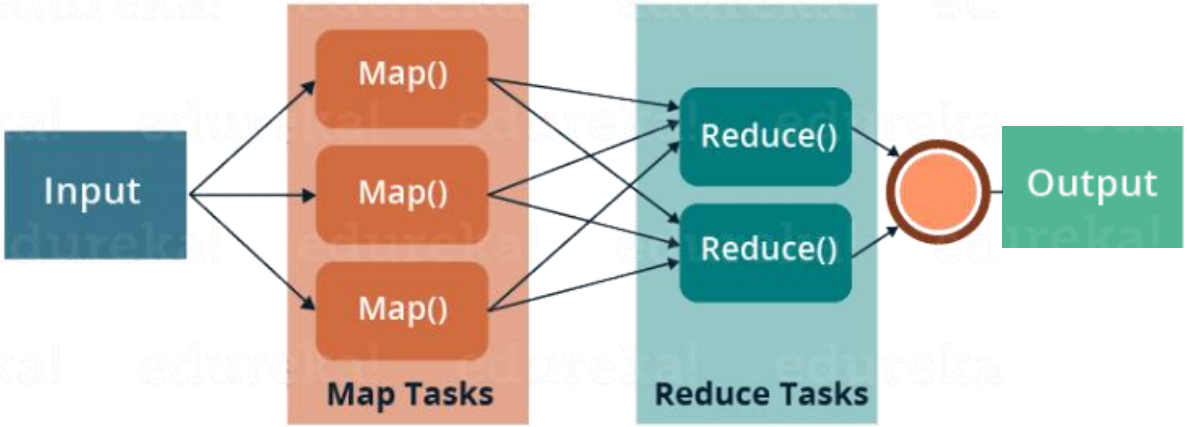
MapReduce est un framework de programmation qui nous permet d'effectuer des traitements distribués et parallèles sur de grands ensembles de données dans un environnement distribué.




# NoSQL = Not Only SQL

## MapReduce : Définition

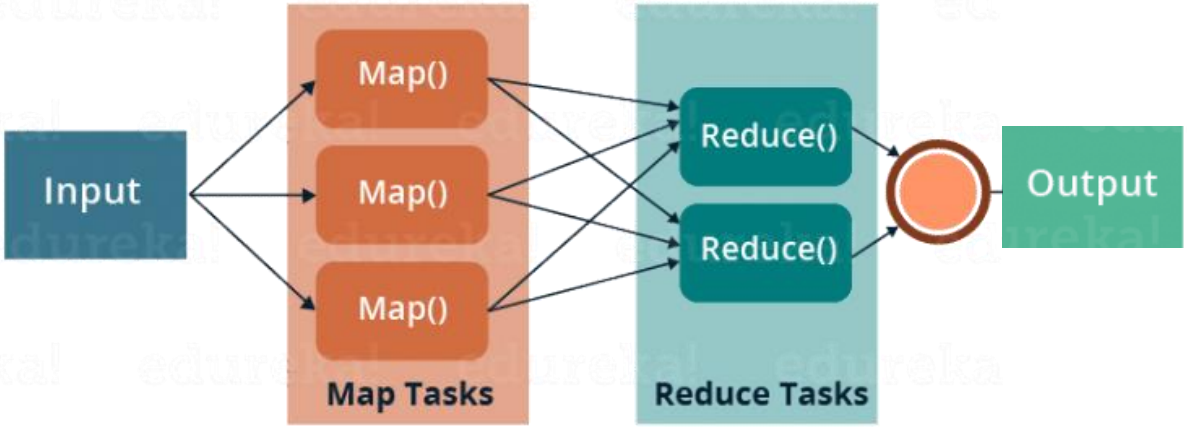
MapReduce se compose de deux tâches distinctes : **Map et Reduce.**
Comme le nom MapReduce le suggère, la phase de réduction a lieu après que la phase de mappage ait été réalisée.
Ainsi, la première tâche est le map, où un bloc de données est lu et traité pour produire des paires clé-valeur comme sorties intermédiaires.




# NoSQL = Not Only SQL

***MapReduce : Définition***

La sortie d'un mappeur ou d'un map job (paires clé-valeur) est une entrée pour le réducteur.
Le réducteur reçoit les paires clé-valeur de plusieurs travaux de mappage.
Ensuite, le réducteur agrège ces tuples de données intermédiaires (paires clé-valeur intermédiaires) en un ensemble plus petit de tuples ou de paires clé-valeur qui constitue la sortie finale.




***MapReduce : Exemple***

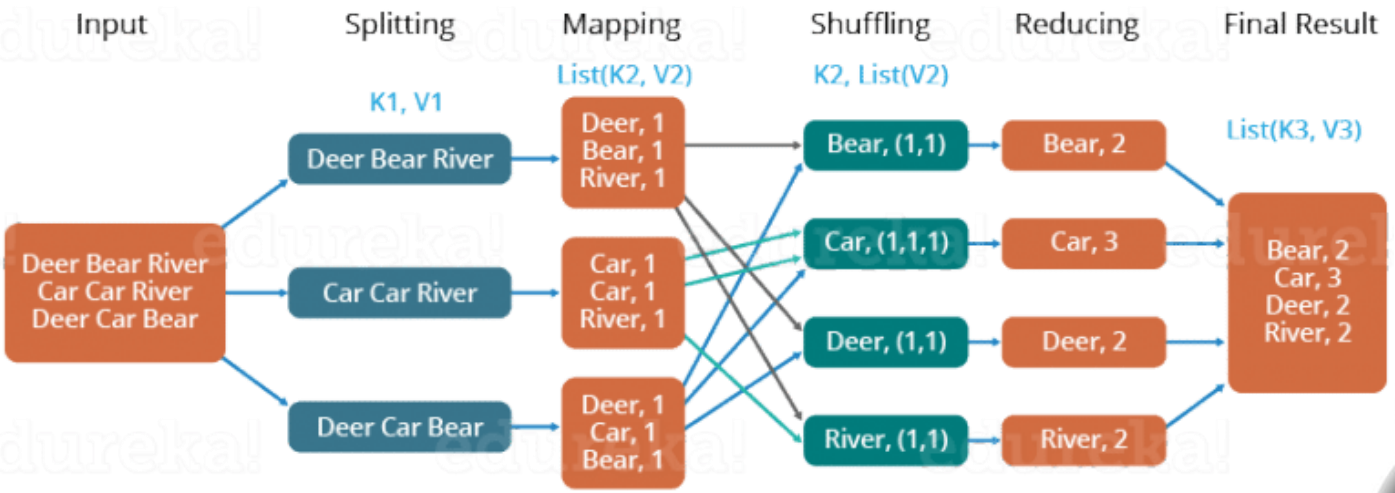
Supposons un fichier texte appelé exemple.txt dont le contenu est le suivant :

Dear, Bear, River, Car, Car, River, Deer, Car and Bear

Maintenant, supposons que nous devions effectuer un comptage de mots sur le fichier exemple.txt en utilisant MapReduce. Nous allons donc trouver les mots uniques et le nombre d'occurrences de ces mots uniques.

# NoSQL = Not Only SQL

***MapReduce : Exemple***




## MapReduce : Exemple

- Tout d'abord, nous divisons l'entrée en trois parties comme indiqué sur la figure. Cela permettra de répartir le travail entre tous les nœuds de cartographie.

- Ensuite, nous tokenisons les mots dans chacun des mappeurs et donnons une valeur codée en dur (1) à chacun des tokens ou mots. Le raisonnement qui sous-tend l'attribution d'une valeur codée en dur égale à 1 est que chaque mot, en soi, n'apparaîtra qu'une fois.

- Maintenant, une liste de paires clé-valeur sera créée où la clé n'est rien d'autre que les mots individuels et la valeur est un. Ainsi, pour la première ligne (Dear Bear River) nous avons 3 paires clé-valeur - Dear, 1 ; Bear, 1 ; River, 1. Le processus de mappage reste le même sur tous les noeuds.

## MapReduce : Exemple

- Après la phase de mappage, un processus de partitionnement a lieu au cours duquel un tri et un brassage sont effectués afin que tous les tuples ayant la même clé soient envoyés au réducteur correspondant.
- Ainsi, après la phase de tri et de brassage, chaque réducteur aura une clé unique et une liste de valeurs correspondant à cette même clé. Par exemple, Bear, [1,1] ; Car, [1,1,1]..., etc.
- Maintenant, chaque réducteur compte les valeurs qui sont présentes dans cette liste de valeurs. Comme le montre la figure, le réducteur obtient une liste de valeurs qui est [1,1] pour la clé Bear. Ensuite, il compte le nombre de uns dans la même liste et donne la sortie finale comme - Bear, 2.
- Enfin, toutes les paires clé/valeur de sortie sont rassemblées et écrites dans le fichier de sortie.

## MapReduce : Avantages

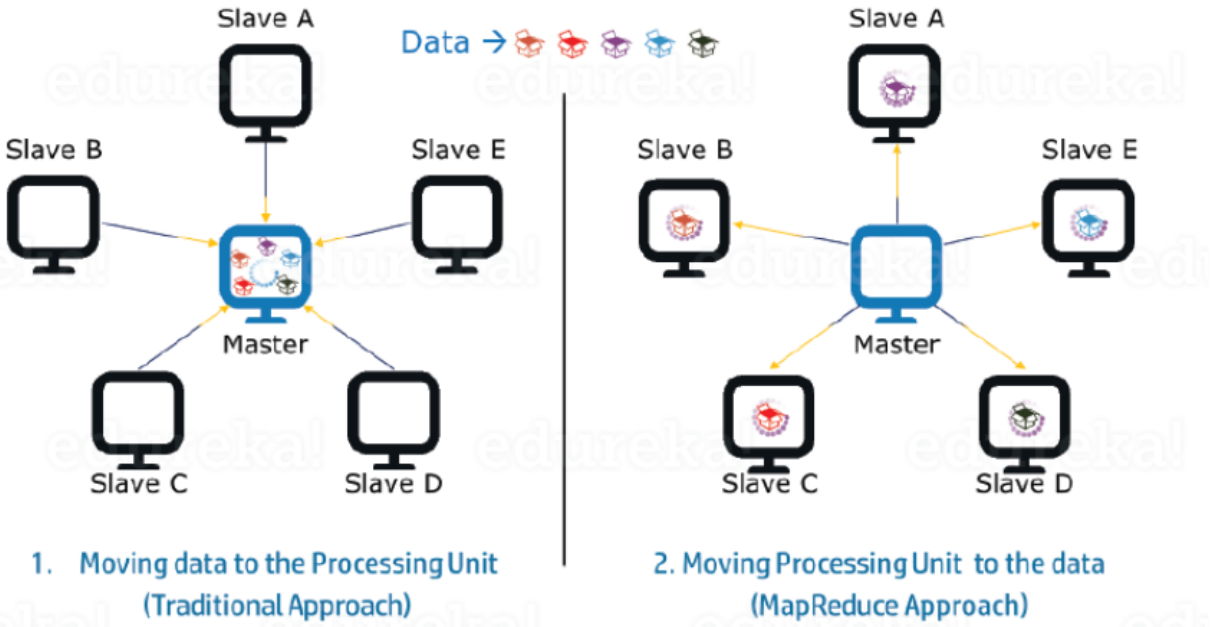
### Traitement Parallèle

- Dans MapReduce, nous divisons la tâche entre plusieurs nœuds et chaque nœud travaille avec une partie de la tâche simultanément. Ainsi, MapReduce est basé sur le paradigme Diviser et Conquérir qui nous aide à traiter les données en utilisant différentes machines. Comme les données sont traitées par plusieurs machines au lieu d'une seule en parallèle, le temps nécessaire pour traiter les données est réduit de façon considérable, comme le montre la figure ci-dessous (2).

# NoSQL = Not Only SQL

## MapReduce : Avantages

### Localité des données




# NoSQL = Not Only SQL

***Eventual Consistency**: Généralités*

Une garantie que lorsqu'une mise à jour est effectuée dans une base de données distribuée, cette mise à jour sera finalement répercutée dans tous les nœuds qui stockent les données, ce qui se traduira par une réponse identique à chaque fois que les données seront interrogées.

La cohérence fait référence à une requête de base de données qui renvoie les mêmes données chaque fois que la même demande est faite.

- La cohérence forte
- La cohérence éventuelle

# NoSQL = Not Only SQL

***Eventual Consistency & Strong Consistency***
*Cohérence forte & Cohérence forte*

La cohérence forte signifie que les données les plus récentes sont renvoyées, mais, en raison des méthodes de cohérence interne, elle peut entraîner une latence ou un retard plus important.

Avec la cohérence éventuelle, les résultats sont moins cohérents au début, mais ils sont fournis beaucoup plus rapidement avec une faible latence.

Les premiers résultats des requêtes de données à cohérence éventuelle peuvent ne pas contenir les mises à jour les plus récentes, car il faut du temps pour que les mises à jour atteignent les répliques dans un cluster de bases de données.

# NoSQL = Not Only SQL

## ***Eventual Consistency :*** *Avantages*

- L'un des principaux avantages d'une base de données cohérente à terme est qu'elle prend en charge le modèle de haute disponibilité de NoSQL.

- Les bases de données cohérentes à terme donnent la priorité à la disponibilité sur la cohérence forte.

- La cohérence finale dans les microservices peut prendre en charge une API toujours disponible qui doit être réactive, même si les résultats de la requête peuvent occasionnellement manquer le dernier commit.

# NoSQL = Not Only SQL

***Eventual Consistency** : Récapitulatif*

La cohérence éventuelle désigne en général la capacité d'une base de données à traiter des transactions simultanément tout en préservant l'intégrité des données.

En termes simples, l'incohérence désigne la garantie qu'un READ renvoie le résultat du dernier WRITE réussi. Cela semble simple, mais une telle garantie est incroyablement difficile à fournir dans une topologie de base de données globalement distribuée, qui implique plusieurs clusters contenant chacun de nombreux nœuds.

# NoSQL = Not Only SQL

## ACID vs. BASE : Avantages et limitations

Les bases de données relationnelles qui prennent en charge la "cohérence forte" offrent des "**garanties ACID**".

ACID est un acronyme conçu pour capturer les éléments essentiels d'une base de données à forte cohérence. Les composants de l'ACID sont les suivants :

- *Atomicité - si la transaction échoue à un moment donné, l'opération entière est annulée.*
- *Cohérence - la base de données reste structurellement saine à chaque transaction.*
- *Isolation - chaque transaction est indépendante de toute autre transaction.*
- *Durabilité - tous les résultats des transactions sont préservés de façon permanente*

# NoSQL = Not Only SQL

***ACID vs. BASE : Avantages et limitations***

- Les bases de données conformes à la norme ACID **sont généralement lentes**, **difficiles à faire évoluer et coûteuses à exploiter et à maintenir**.
- Un SGBD utilisant le modèle ACID s'attend à ce qu'une unité de travail soit **atomique.** Elle est tout ou rien. Pendant que cette unité de travail est effectuée, les enregistrements ou les tables concernés **peuvent être verrouillés.**
- Les bases de données distribuées avec un modèle BASE **offrent une haute disponibilité.** Les enregistrements restent **disponibles**, mais une fois la transaction terminée sur une majorité de nœuds, la transaction est considérée **comme réussie**.
- La réplication des données sur tous les nœuds **peut prendre un peu plus de temps,** mais les données de tous les nœuds finiront par être **cohérentes**.

# NoSQL = Not Only SQL

## *ACID vs. BASE : Avantages et limitations*

Contrairement aux garanties ACID de SQL, les bases de données NoSQL fournissent des garanties dites **BASE**. Une BASE permet la disponibilité et assouplit la cohérence stricte. L'acronyme BASE désigne :

- **Basic Availability** - les données sont disponibles la plupart du temps, même pendant une panne partielle du système.

- **Soft state** - les répliques ne sont pas cohérentes tout le temps.

- **Eventual consistency** - les données deviendront cohérentes à un moment donné, sans garantie de date.

# NoSQL = Not Only SQL

***ACID vs. BASE : Avantages et limitations***

En tant que telles, les bases de données NoSQL sacrifient un certain degré de cohérence afin **d'augmenter la disponibilité**. Plutôt que de fournir une cohérence forte, elles fournissent une cohérence éventuelle. Cela signifie qu'un datastore qui fournit des garanties BASE peut occasionnellement ne pas retourner le résultat du dernier WRITE

# NoSQL = Not Only SQL

***Théorème de Brewer dit "théorème de CAP"***

Apparu en 2000 (Eric A. Brewer).
Formaliser un théorème reposant sur 3 propriétés fondamentales pour caractériser les bases de données :

**- Relationnelles,**
**- NoSQL,**
**- Autres …**

# NoSQL = Not Only SQL

***Théorème de Brewer dit "théorème de CAP"***

**Consistency** (Cohérence) : Une donnée n'a qu'un seul état visible quel que soit le nombre de réplicas

**Availability** (Disponibilité) : Tant que le système tourne (distribué ou non), la donnée doit être disponible

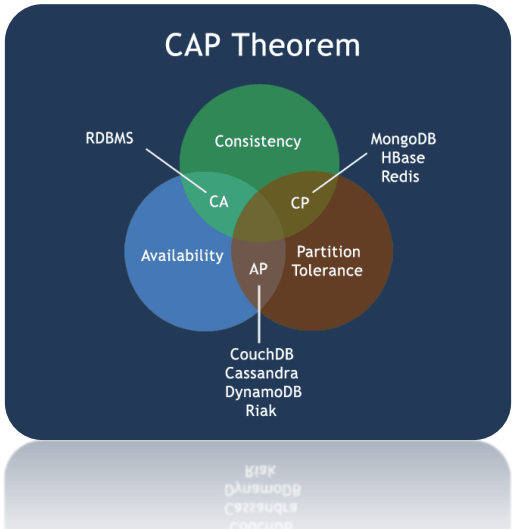
**Partition Tolerance** (Distribution) : Quel que soit le nombre de serveurs, toute requête doit fournir un résultat correct

# NoSQL = Not Only SQL

***Théorème de Brewer dit "théorème de CAP"***

## Le théorème de CAP dit :

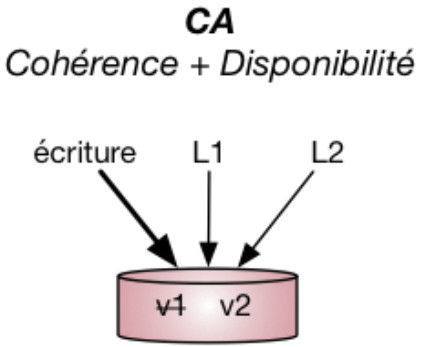
Dans toute base de données, vous ne pouvez respecter au plus que 2 propriétés parmi la cohérence, la disponibilité et la distribution.




# NoSQL = Not Only SQL

***Théorème de Brewer dit "théorème de CAP"***

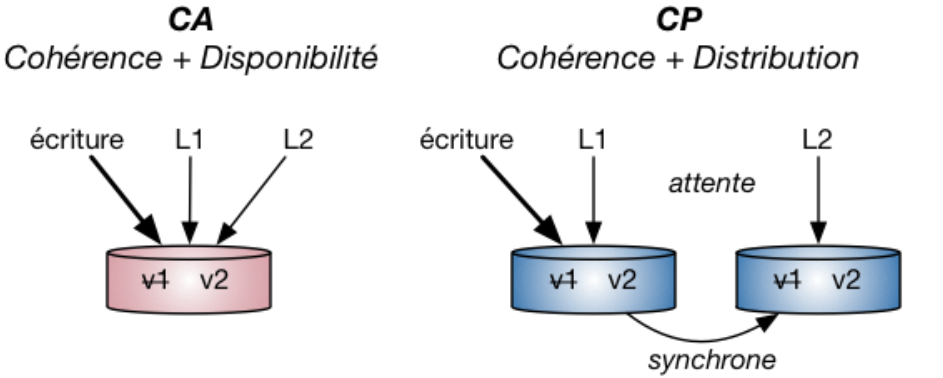
Le couple **CA** (Consistency-Availability), il représente le fait que lors d'opérations concurrentes sur une même donnée, Les requêtes L1 et L2 retournent la nouvelle version (v2) et sans délai d'attente. Cette combinaison n'est possible que dans le cadre de bases de données transactionnelles telles que les SGBDR.




# NoSQL = Not Only SQL

***Théorème de Brewer dit "théorème de CAP"***

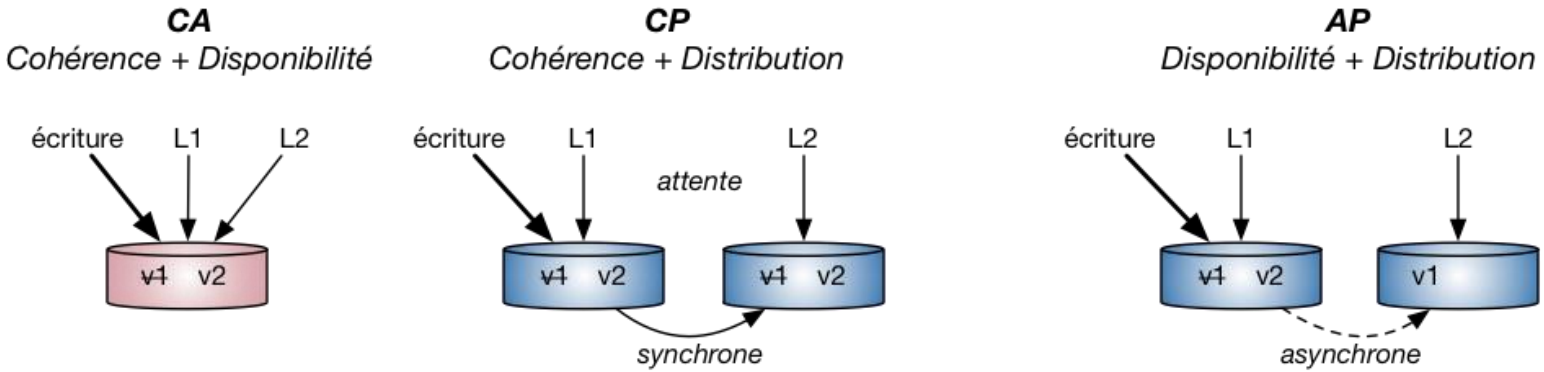
Le couple CP (Consistency-Partition Tolerance) propose maintenant de distribuer les données sur plusieurs serveurs en garantissant la tolérance aux pannes (réplication). En même temps, il est nécessaire de vérifier la cohérence des données en garantissant la valeur retournée malgré des mises à jour concurrentielles. La gestion de cette cohérence nécessite un protocole de synchronisation des réplicas, introduisant des délais de latence dans les temps de réponse (L1 et L2 attendent la synchronisation pour voir v2). C'est le cas de la base NoSQL MongoDB.




# NoSQL = Not Only SQL

***Théorème de Brewer dit "théorème de CAP"***

Le couple AP (Availability-Partition Tolerance) s'intéresse à fournir un temps de réponse rapide tout en distribuant les données et les réplicas. De fait, les mises à jour sont asynchrones sur le réseau, et la donnée est "Eventually Consistent" (L1 voit la version v2, tandis que L2 voit la version v1). C'est le cas de Cassandra dont les temps de réponses sont appréciables, mais le résultat n'est pas garanti à 100% lorsque le nombre de mises à jour simultanées devient important.



# NoSQL = Not Only SQL

### *Le triangle de CAP et les bases de données*

L'avantage de ce triangle CAP est de fournir un critère de choix pour votre application. Vous pouvez ainsi vous reposer sur vos besoins en terme de fonctionnalité, de calculs et de cohérence. Le triangle servira de filtre sur les myriades de solutions proposées.