Bases de Données NoSQL

Comprendre, concevoir et implémenter des solutions NoSQL

Commencer le cours →

SQL : Le Standard Relationnel

Caractéristiques

- Schéma strict et prédéfini
- Relations entre tables
- Jointures complexes
- Transactions ACID
  - Atomicity : Atomicité
  - Consistency : Cohérence
  - Isolation : Isolation
  - Durability : Durabilité

Exemple

```sql
CREATE TABLE users (
  id INT PRIMARY KEY,
  name VARCHAR(255),
  email VARCHAR(255) UNIQUE
);
```

Limitations

- Scalabilité verticale
- Schéma rigide
- Coûts de modification
- Performance avec big data

NoSQL : La Nouvelle Approche

Définition

- "Not Only SQL"
- Schéma flexible
- Scalabilité horizontale
- BASE vs ACID
  - Basically Available
  - Soft state
  - Eventually consistent

Forces

- Flexibilité du schéma
- Scalabilité horizontale
- Haute disponibilité
- Performance élevée

Caractéristiques Clés

```
// Document flexible
{
  "id": "user_123",
  "name": "John Doe",
  "preferences": {
    "theme": "dark",
    "notifications": true
  },
  "devices": [
    {"type": "mobile", "os": "iOS"},
    {"type": "laptop", "os": "Linux"}
  ]
}
```

Pourquoi NoSQL ?

Évolution des Besoins

- Big Data
- Cloud Computing
- Applications temps réel
- Microservices

Avantages Clés

- Développement agile
- Déploiement simplifié
- Coûts optimisés
- Maintenance facilitée

Cas d'Usage

```
// Exemple de données IoT
{
  "device_id": "sensor_789",
  "timestamp": "2024-01-20T14:30:00Z",
  "readings": {
    "temperature": 23.5,
    "humidity": 45,
    "pressure": 1013
  },
  "location": {
    "lat": 48.8566,
    "lng": 2.3522
  }
}
```

4 Types de Bases NoSQL

Découvrons différents types de bases de données NoSQL

Key-Value

Caractéristiques

- Structure la plus simple
- Performance maximale
- Mise en cache
- Scalabilité horizontale

Cas d'Usage

- Sessions utilisateurs
- Cache système
- Files de messages
- Compteurs temps réel

```
// Exemple Redis
SET user:1000 "John Doe"
SET session:abc123 {
  "userId": 1000,
  "lastAccess": "2024-01-20"
}

GET user:1000
```

Exemples d'Entreprises

- **Amazon**: DynamoDB pour les sessions utilisateurs
- **Twitter**: Redis pour le cache temps réel
- **Instagram**: Redis pour le feed d'activités
- **Snapchat**: Redis pour les messages

Document

Caractéristiques

- Documents JSON/BSON
- Schéma flexible
- Requêtes riches
- Index secondaires

Exemples d’Entreprises

- **LinkedIn**: MongoDB pour les profils utilisateurs
- **CISCO**: MongoDB pour IoT et analyses
- **EA**: MongoDB pour les données de jeux
- **Google**: Firestore pour les applications web

Base Orientée Colonne

Caractéristiques

- Optimisé pour les lectures
- Stockage par colonne
- Compression efficace
- Grande échelle

Exemples d'Entreprises

- **Netflix**: Cassandra pour les recommandations
- **Spotify**: Cassandra pour les playlists
- **Instagram**: Cassandra pour les messages
- **Apple**: HBase pour les données de l'App Store

Cas d'Usage chez Netflix

```
CREATE TABLE user_viewing (
  user_id uuid,
  movie_id uuid,
  timestamp timestamp,
  watch_duration int,
  rating float,
  PRIMARY KEY (user_id, movie_id)
);
```

Base de Graphes

Caractéristiques

- Relations complexes
- Parcours de graphe
- Requêtes de traversée
- Données connectées

Exemples d'Entreprises

- **Facebook**: GraphQL pour les relations sociales
- **NASA**: Neo4j pour la gestion des connaissances
- **Walmart**: Neo4j pour les recommandations
- **eBay**: Neo4j pour le service client

Cas d'Usage chez Facebook

```
// Exemple de graphe social
CREATE (user:Person {name: 'Alice'})
CREATE (friend:Person {name: 'Bob'})
CREATE (post:Content {type: 'photo'})
CREATE (user)-[:FRIENDS_WITH]->(friend)
CREATE (user)-[:POSTED]->(post)
CREATE (friend)-[:LIKED]->(post)

// Trouver les amis qui ont aimé mes posts
MATCH (me)-[:POSTED]->(post)<-[:LIKED]-(friend)
WHERE me.name = 'Alice'
RETURN friend.name
```

Théorémee CAP

Consistency (Consistance)

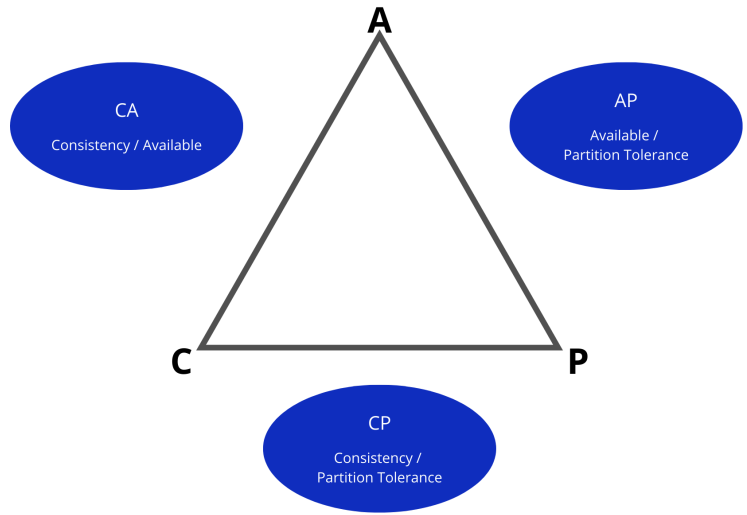
- Toutes les données sont à jour
- Toutes les requêtes retournent les mêmes résultats

Availability (Disponibilité)

- Toutes les requêtes reçoivent une réponse
- Les requêtes peuvent être lues ou écrites

Partition Tolerance (Distribution)

- Le système continue de fonctionner malgré les pannes
- Les données sont réparties sur plusieurs serveurs



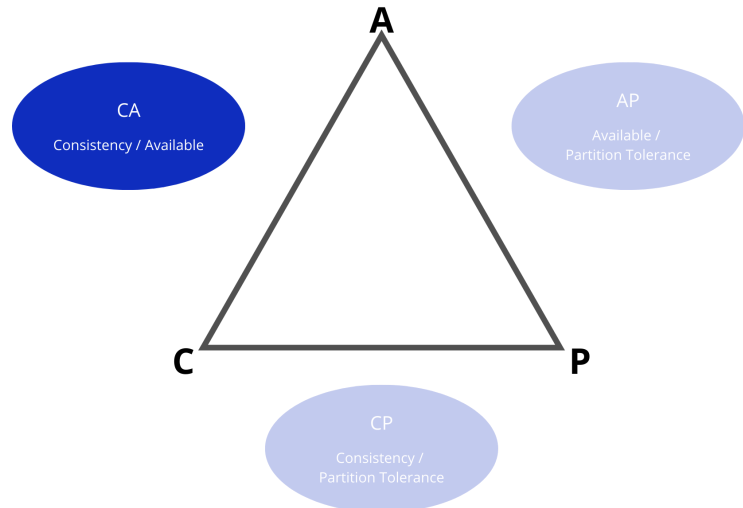
Théorémee CAP

CA (Consistency-Availability)

- Système traditionnel SQL
- Consistance forte
- Haute disponibilité
- Risque de perte de données

Exemple de CA

- Oracle
- MySQL
- PostgreSQL



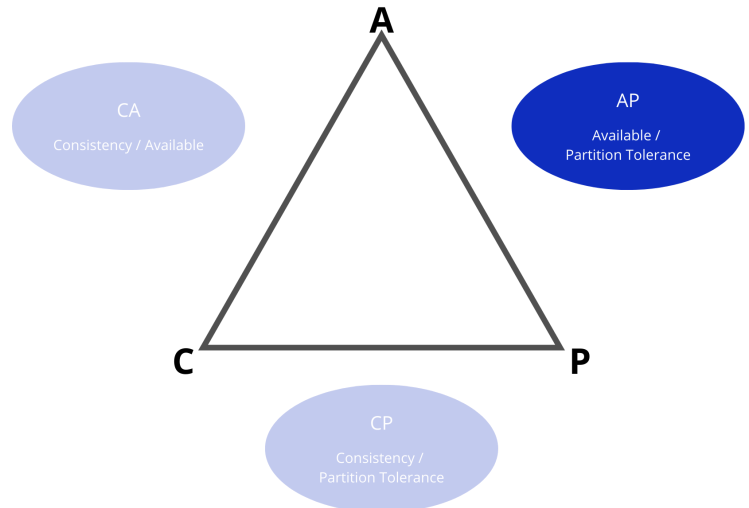
Théorémee CAP

AP (Availability-Partition Tolerance)

- Système NoSQL
- Haute disponibilité
- Tolérance aux pannes
- Risque de données obsolètes

Exemple de AP

- Cassandra
- Couchbase
- Riak



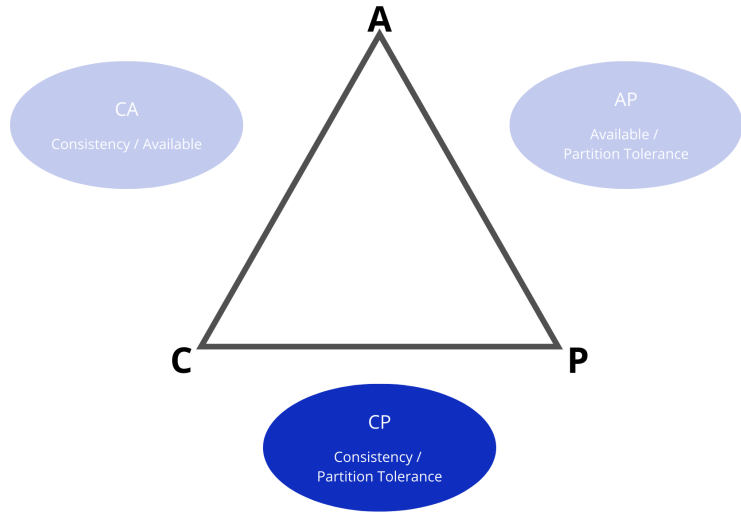
Théorémee CAP

CP (Consistency-Partition Tolerance)

- Système NoSQL
- Consistance forte
- Tolérance aux pannes
- Risque de disponibilité limitée

Exemple de CP

- MongoDB
- HBase
- Redis



Théorémee CAP

AP et CP

- Les systèmes NoSQL sont souvent AP ou CP
- Choisir en fonction des besoins métier
- Certains systèmes peuvent basculer entre AP et CP

  (ex: Couchbase)
- Mais pas les deux en même temps

Scalabilité et Performance

Comprendre les concepts de scalabilité et de performance

- Scalabilité verticale vs horizontale
- Partitionnement des données
- Réplication des données

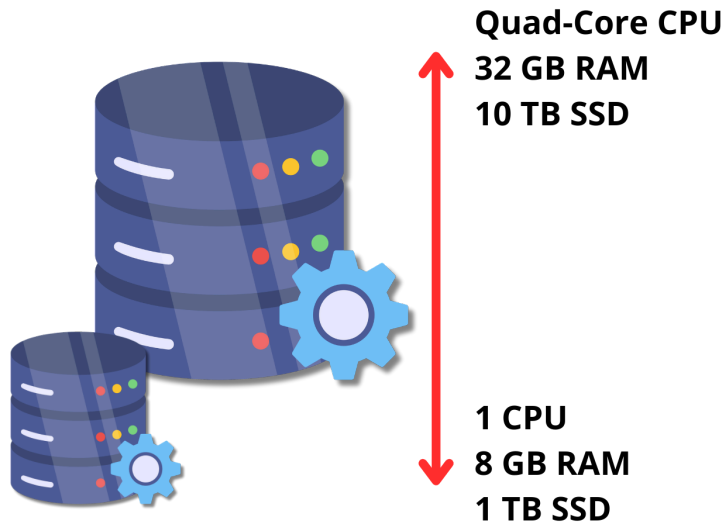
Scalabilité et Performance

Scalabilité Verticale

- Ajouter plus de ressources à un serveur unique
- Limité par la capacité matérielle
- Coûteux et difficile à maintenir

Exemple

- Ajouter plus de RAM à un serveur
- Augmenter la capacité de stockage
- Améliorer les performances CPU



Scalabilité et Performance

Scalabilité Horizontale

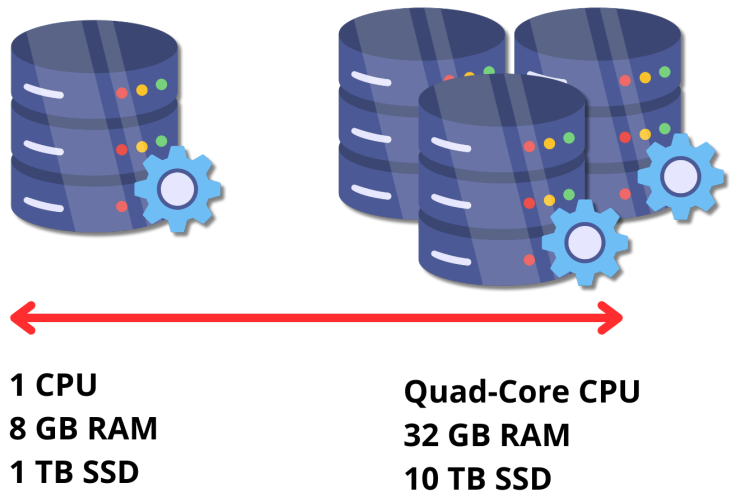
- Ajouter plus de serveurs à un cluster
- Facile à mettre en œuvre
- Coût-efficace et hautement disponible

Limitations

- Complexité de la gestion de cluster
- Consistance et disponibilité
- Partitionnement des données

Exemple

- Ajouter plus de RAM à un serveur
- Augmenter la capacité de stockage
- Améliorer les performances CPU



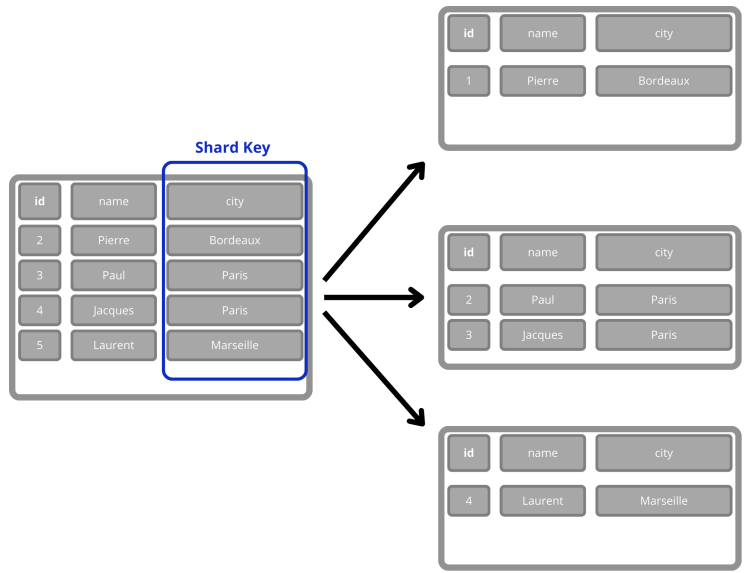
Partitionnement des Données (Sharding)

Définition

- Diviser les données en partitions
- Distribuer les partitions sur plusieurs serveurs
- Équilibrer la charge de travail

Stratégies

- **Hashing**: Partitionnement basé sur une clé
- **Range**: Partitionnement basé sur une plage
- **Geo**: Partitionnement basé sur la géolocalisation



Réplication des Données (Replication)

Définition

- Dupliquer les données sur plusieurs serveurs
- Améliorer la disponibilité
- Tolérance aux pannes
- Lecture et écriture distribuées
- Consistance des données

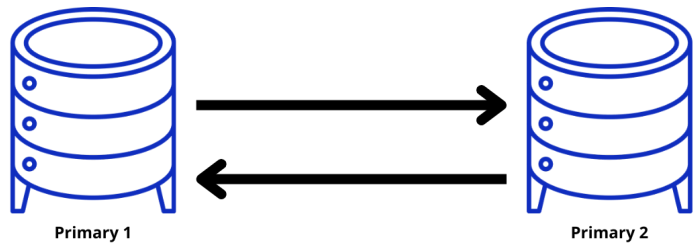
Réplication des Données (Replication)

Réplication Multi-Primaires (Master-Master)

- Plusieurs serveurs pour les écritures
- Réplication synchrone ou asynchrone

Inconvénients

- Risque de conflits de données
- Complexité de la synchronisation
- Consistance des données



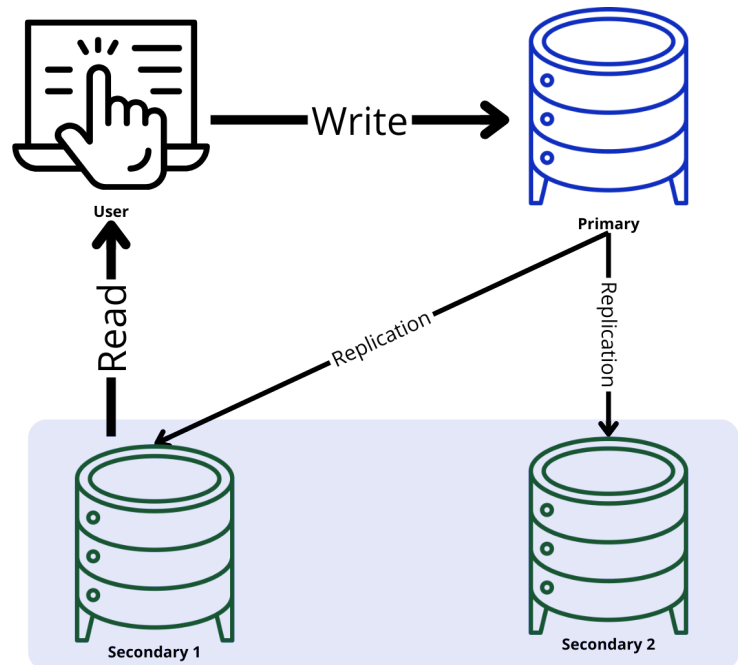
Réplication des Données (Replication)

Primary-Secondary Replication (Master-Slave)

- Un nœud principal pour les écritures
- Des nœud secondaires pour les lectures
- Lecture possible sur le nœud principal selon la configuration

Inconvénients

- Latence de réplication
- Risque de surcharge du nœud principal



Conclusion

Points Clés

- Choisir le bon type de NoSQL
- Considérer les besoins métier
- Planifier la scalabilité