

## Introduction NoSQL

**Impedance mismatch:** décalage modèle relationnel vs objets. Solutions: ORM ou Document (JSON).

**Problème:** multiples jointures. **Solution:** tout en JSON. Relations 1:N faciles, M:N complexes.

**Scalabilité:** Verticale ( $\uparrow$  CPU/RAM, limites) vs Horizontale ( $\uparrow$  serveurs, économique, complexe avec relationnel).

**Caractéristiques:** pas relationnel, scalabilité horizontale, schéma flexible, open-source, réplication, API simples, **eventually consistent**.

**Catégories:** Documents (MongoDB, Couchbase), Colonnes (Cassandra, HBase), Graphes (Neo4j), Clé-valeur (Redis, Riak).

## BD Documents & Couchbase

**Stockage:** JSON natif, tout dans un document. **Schéma flexible:** dynamique, champs variables. **Couchbase:** documents + clé-valeur, langage **N1QL** (SQL-like).

### Modélisation

**Imbrication:** + vitesse, tolérance pannes. - incohérence, docs volumineux.

**Séparation:** + cohérence, cache efficace. - jointures multiples.

**Règles:** 1:1/1:N  $\rightarrow$  imbriqué. M:N  $\rightarrow$  séparés. Lectures parent+enfant  $\rightarrow$  imbriqué. Cohérence prioritaire  $\rightarrow$  séparés.

**Physique:** Bucket (DB), Collection (table), Scope (regroupement), Item (clé + JSON).

### N1QL Mots-clés

**Base:** SELECT, FROM, WHERE, ORDER BY, LIMIT, RAW, DISTINCT.

**Découverte:** INFER. **Accès:** ., [], LIKE, META().id, USE KEYS.

**Collections:** IN/NOT IN, WITHIN/NOT WITHIN, ANY...SATISFIES...END, EVERY...SATISFIES...END.

**Tableaux:** ARRAY...FOR...WHEN...END, FIRST...FOR...WHEN...END.

**Aggrégation:** GROUP BY, HAVING, COUNT, ARRAY\_AGG, ARRAY\_COUNT, ARRAY\_MAX/MIN.

**NULL:** IS VALUED, IS NULL, IS MISSING.

**Index:** CREATE PRIMARY INDEX, CREATE INDEX, composite, couvrant. DISTINCT ARRAY...FOR...END (tableaux).

**Jointures:** INNER JOIN, LEFT/RIGHT OUTER JOIN, ON. NEST (imbrique), UNNEST (aplati).

**Avancé:** Sous-requêtes, LET, LETTING, EXPLAIN.

## BD Graphes

**Property Graph:** Nœuds (propriétés, labels) + Relations (dirigées, nommées, propriétés). Relations stockées (**index-free adjacency**)  $\rightarrow$  temps constant vs JOIN.

**Perf:** constante quelle que soit taille. RDBMS dégrade. Ex: 1M personnes, prof. 5: RDBMS 1543s, Neo4j 2.1s.

**Neo4j:** BD native, langage **Cypher**. Convention: labels CamelCase, relations MAJUSCULES\_TIRETS.

### Cypher

**Nœuds:** (), (p:Person), (p:Person {name: 'Alice'}).

**Relations:** (a)-->(b), (a)-[r:TYPE]->(b), (a)-[r {p: v}]->(b).

**CRUD:** CREATE, MATCH...RETURN, WHERE, WHERE NOT, IS NOT NULL, SET, DELETE, DETACH DELETE, REMOVE, MERGE, ON CREATE/MATCH.

**Index:** CREATE INDEX FOR (a:Actor) ON (a.name), composite. SHOW indexes.

**Contraintes:** CREATE CONSTRAINT...REQUIRE...IS UNIQUE. SHOW constraints.

**Aggrégations:** count(), collect(), avg(), max(), min(), sum(), count(DISTINCT x).

**Avancé:** WITH, UNWIND, ORDER BY, DISTINCT.

**Chemins:** [:TYPE\*] (1+), [:TYPE\*3], [:TYPE\*3..5], [:TYPE\*3..], [:TYPE\*..5], shortestPath().

## Distribution & Cohérence

**Pourquoi distribuer:** évolutivité ( $\uparrow$  charge), haute disponibilité (pannes), latence réduite (centres proches).

**Architectures:** Mémoire partagée (SMP, coûteux), Disques partagés (conflits). Sans partage/Scale Out (nœuds indépendants, matériel standard, coordination logicielle).

**Modèles:** Partitionnement (sharding: diviser données) + Réplication (copies multiples). Souvent combinés.

### Réplication

3 types:

- **Leader unique** (PostgreSQL, MySQL): leader écrit, followers répliquent. Lecture: leader ou followers
- **Multi-leader**: plusieurs acceptent écritures. Cas: offline, multi-DC, collaborative editing. Problème: conflits, vecteurs de version
- **Sans leader** (Dynamo: Cassandra, Riak): toute réplique accepte écritures

**Sync vs Async:** Synchrone (lent, données garanties) vs Asynchrone (rapide, risque perte). **Semi-synchro:** 1 follower sync, autres async (compromis courant).

**Failover** (panne leader): timeout (30s)  $\rightarrow$  élection nouveau leader (consensus: Raft, Paxos)  $\rightarrow$  reconfiguration clients.

### Logs réplication:

- Statement-based (SQL): problème fonctions non-déterministes (NOW(), RAND())
- WAL shipping (octets disque): couplage moteur, incompatibilité versions
- Logical/row-based (lignes): découplage, rétrocompatible
- Trigger-based: flexible, coûteux

**Replication lag:** délai leader  $\rightarrow$  follower. **Cohérence éventuelle:** incohérence temporaire. Problèmes:

- Read-your-owns-writes (ne pas voir ses modifs): lire du leader pour données modifiables, suivre timestamp
- Lectures monotones (régression): même réplique par utilisateur (hash ID)

**Sans leader - Quorums:** n répliques, w écritures confirmées, r lectures interrogées. **w + r > n** garantit données à jour. Ex: n=5, w=3, r=3  $\rightarrow$  tolérance 2 nœuds. Ajustement: w=n, r=1 (lectures rapides, écritures bloquées si panne).

### Partitionnement

**Objectif:** diviser données volumineuses/débit élevé. Terminologies: shard (MongoDB), region (HBase), tablet (Bigtable), vnode (Cassandra), vBucket (Couchbase).

**Partitionnement équitable** évite hotspots (charge disproportionnée). Chaque partition: leader + followers.

### Stratégies:

- **Intervalle de clé** (range): plages continues, données triées, range queries OK. Risque: hotspots (ex: timestamp  $\rightarrow$  partition aujourd'hui)
- **Hachage de clé:** répartition équitable, pas de range queries (toutes partitions interrogées)

**Rééquilibrage:** Ne PAS utiliser hash(key) mod N (changement N  $\rightarrow$  tout bouge). **Partitions fixes:** nombre  $\gg$  nœuds, déplacement partitions entières, nombre constant. Ex: Riak, Elasticsearch, Couchbase. Automatique vs manuel (spectre, ex: Couchbase suggère, admin approuve).

### Service discovery:

1. Client contacte n'importe quel nœud (round-robin), transmet si besoin
2. Couche routage (load balancer conscient partitions)
3. Client conscient (connexion directe)

**Défi:** apprendre changements affectations. **ZooKeeper:** coordination, mappage partitions  $\leftrightarrow$  nœuds, nœuds s'enregistrent, routage s'abonne.

## Cohérence

**Cohérence:** absence contradiction. SGBDR centralisé: forte cohérence. NoSQL distribué: assouplissement  $\rightarrow$  cohérence à terme.

### Problèmes SGBD centralisé:

- **Dirty Write:** 2 transactions MAJ simultanées (ex: Alice facture, Bob listing  $\rightarrow$  incohérent). Solutions: verrous (pessimiste) ou détection (optimiste)
- **Dirty Read:** lire écritures non-validées (ex: message inséré, compteur pas encore incrémenté)

**Niveaux isolation:** read uncommitted (permet dirty reads), read committed, repeatable reads, serializable (complet).

**Transactions Nosql:** Centralisé: journalisation (log modifications, rollback). Distribué: plus délicat (logs séparés).

**Cohérence réplication:** même valeur sur répliques différentes. Cohérence à terme (propagation temps), données périmées (stale). **Read-your-writes:** sticky session (affinité nœud).

**Théorème CAP:** système distribué ne peut avoir les 3:

- **Consistency:** tous voient mêmes données
- **Availability:** chaque requête  $\rightarrow$  réponse
- **Partition tolerance:** fonctionne malgré isolation

Face au partitionnement réseau: choisir cohérence (rejeter requêtes) OU disponibilité (données périmées). Compromis cohérence  $\leftrightarrow$  temps de réponse selon opérations.