BASES DE DONNÉES NOSQL

HISTORIQUE

- **Modèle hiérarchique (1950)**
  - Relations parent-enfants
  - Arbre hiérarchique
  - **Implémentation :** le moteur IMS (Information Management System) d'IBM
- **Codasyl (Conference on Data Systems Languages) (1959)**
  - Etablissement d'un langage d'interrogation Cobol (COmmon Business Oriented Language)
  - Langage navigationnel
  - Etablissement d'un modèle de données : modèle réseau
- **Modèle relationnel d'Edgar Frank**
  - Basé que la théorie des ensembles
  - Isoler l'accès aux données de l'implémentation physique
  - Proposer un langage déclaratif, algébrique indépendant des algorithmes de manipulation des données

QUELQUES RÈGLES DE CODD

- **Règle 0** : Toutes les fonctionnalités du SGBDR doivent être disponibles à travers le modèle relationnel et le langage d'interrogation.
- **Règle 1 :** Toutes les données sont représentées par des valeurs présentes dans des colonnes et des lignes de tables.
- **Règle 3 :** Une cellule peut ne pas contenir de valeur, ou exprimer que la valeur est inconnue, à l'aide du marqueur NULL. Il s'agit d'un indicateur spécial, distinct de toute valeur et traité de façon particulière.
- **Règle 5 :** Le SGBDR doit implémenter un langage relationnel qui supporte des fonctionnalités de manipulation des données et des métadonnées, de définition de contraintes de sécurité et la gestion des transactions.

QUELQUES RÈGLES DE CODD

- **Règle 10 :** *Indépendance d'intégrité :* les contraintes d'intégrité doivent être indépendantes des programmes clients et doivent être stockées dans le catalogue du SGBDR. On doit pouvoir modifier ces contraintes sans affecter les programmes clients.
- **Règle 11 :** *Indépendance de distribution* : la distribution ou le partitionnement des données ne doivent avoir aucun impact sur les programmes clients.
- **Règle 12 :** *Règle de non-subversion :* aucune interface de bas niveau ne doit permettre de contourner les règles édictées. Dans les faits, cette règle implique qu'il n'est possible d'interroger et de manipuler le SGBDR qu'à travers son langage relationnel.

EMERGENCE DU BIG DATA

- Le ***big data***, **mégadonnées**, **données massives**, désigne des ensembles de données devenus si volumineux qu'ils dépassent l'intuition et les capacités humaines d'analyse et même celles des outils informatiques classiques de gestion de base de données ou de l'information,
- 5V du big data
  - Volume
  - Variété
  - Vélocité
  - Valeur
  - Véracité

CARACTÉRISTIQUES DU BIG DATA

Volume

- Le volume des données stockées est en pleine expansion
- Les données numériques créées dans le monde : 1,2 zettaoctet/an en 2010, 1,8 en 2011, 2,8 en 2012 et s'élèveront à 40 en 2020.
- En 2013 : Twitter 7 téraoctets de données chaque jour, Facebook 10 téraoctets.
- En 2014: Facebook Hive générait 4 000 To de data par jour

Variété

- Données relationnelles, semi-structurées, non structurées, texte, images, données géo-référencées, etc.

Vélocité

- Fréquence à laquelle les données sont à la fois générées, capturées, partagées et mises à jour
- **Véracité, Valeur**

SCALABILITÉ FACE AU BIG DATA

- **Verticale**
  - Effectuée en effectuant une évolution hardware (CPU plus rapide, plus de RAM, Disques volumineux, etc.
  - Limitée par le nombre de CPU, la RAM et les capacités maximum des disques configures sur une seule machine
- **Horizontale**
  - Peut être effectuée en ajoutant plus de machines
  - Nécessite une distribution de la BD et la replication
  - Limitée par les mises à jour et les communications réseaux.

EMERGENCE DES BD NOSQL

- Développement des centres de données
- Nouveaux paradigmes de traitement : Map-reduce

→ **Nécessité de nouveaux types de BD : Not Only SQL en 2009**

- Nouvelle génération de BD non relationnelles, distribuées, open source et sclabale horizontalement
- **Quelques exemples de solutions NoSQL**
  - **Clé-valeur** : Riak, Redis
  - **Orientée colonne** : Cassandra, HBase
  - **Orientée document** : MongoDB, CouchDB, ElasticSearch
  - **Orientée graphe** : Neo4j, HypergraphDB

COMPARATIF RELATIONNEL - NOSQL

Les bases de données relationnelles

- **Avantages**
  - La technologie est mature, le SQL est un langage standard et normalisé
  - On a une garantie que les transactions sont atomiques, cohérentes, isolées et durables (principe ACID)
  - La possibilité de mettre en œuvre des requêtes complexes
  - Un large support est disponible et il existe également de fortes communautés.
- **Inconvénients :**
  - La modification du modèle établi peut être couteuse
  - L'évolutivité des performances est privilégiée de manière verticale (augmentation des ressources du serveur) bien qu'une évolutivité horizontale soit possible, cette dernière reste plus coûteuse (environnement type cluster)
  - Sur un très grand volume de données (centaines-milliers de Teraoctets) le modèle peut atteindre des limites en terme de performance
  - Pour certains éditeurs, le prix de licence est élevé.

COMPARATIF BDR - NOSQL

Les bases de données NoSQL

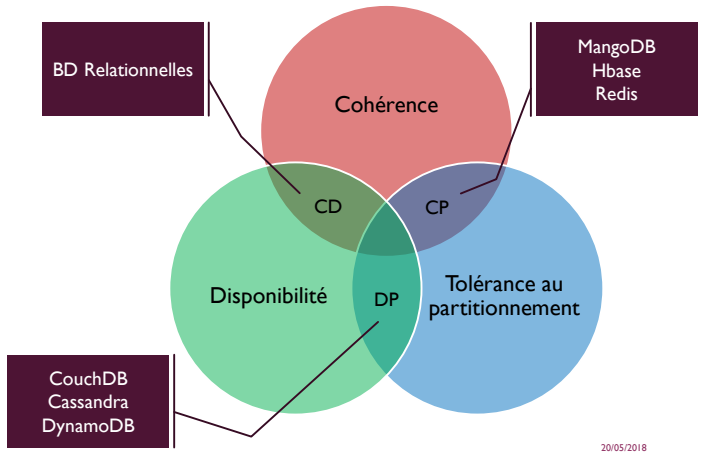
- **Avantages :**
  - L'évolutivité se fait de manière horizontale (pour augmenter les performances on ajoute des nouvelles machines)
  - Les données sont distribuées sur plusieurs machines (sharding) de ce fait on évite les goulets d'étranglements lors de la récupération des données (fortes performances de lecture)
  - La représentation des données est notable par l'absence de schéma (schemaless)
  - La majorité des solutions est Open Source, néanmoins il existe des Support Pro pour répondre aux besoins des entreprises.
- **Inconvénients**
  - Il n'existe pas de langage d'interrogation standardisé : chaque éditeur a mis en place le sien
  - La mise en œuvre d'un environnement fortement transactionnel (fort besoin d'écriture) où le séquencement des écritures est primordial, reste complexe puisque l'architecture est distribuée compliquant l'atomicité et la cohérence des transactions
  - L'écriture de requêtes complexes est difficile à mettre en œuvre
  - L'offre NoSQL est segmentée en plusieurs familles où chacune répond à un besoin précis.

THÉORÈME CAP (ERIC BREWER)

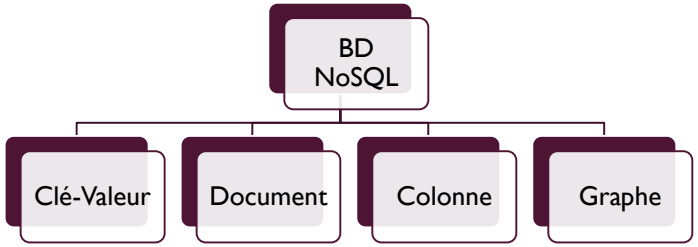
Un système distribué peut supporter uniquement deus caractéristiques parmi les trois suivantes :

- **Cohérence**
  - Tous les nœuds sont à jour sur les données au même moment
  - Tous les nœuds du système voient exactement les mêmes données au même moment
- **Disponibilité**
  - La perte d'un nœud n'empêche pas le système de fonctionner et de servir l'intégralité des données.
  - Garantie que toutes les requêtes reçoivent une réponse
- **Tolérance au partitionnement**
  - Chaque nœud doit pouvoir fonctionner de manière autonome
  - Aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement
  - En cas de morcellement en sous-réseaux, chacun doit pouvoir fonctionner de manière autonome

THÉORÈME CAP



TAXONOMIE DES BD NOSQL



BD NOSQL CLÉS-VALEURS

- La plus simple et flexibles des BD NOSQL
- Associe des clés à des valeurs
- Solution aux limitations des BD relationnelles
- Les valeurs sont identifiées et accédées via la clé

Clé	Valeur
User20	Ami1, Ami2, Ami3

Exemples

- **Réseau social :** à partir d'un utilisateur (la clé), je veux obtenir une liste de ses amis (la valeur).
- **Catalogue de livres :** le numéro ISBN (la clé) donne accès à tous les détails sur le livre (la valeur).
- **Journal d'activités** : la date d'un événement (la clé) indexe les détails de ce qui s'est passé à ce moment (la valeur).

- Pas de schéma
- La valeur stockée est gérée au niveau applicatif. Elle peut être
  - Entier, Chaine de caractère, JSON, XML, HTML, Image, vidéo, etc.

CLÉS-VALEURS - AVANTAGES

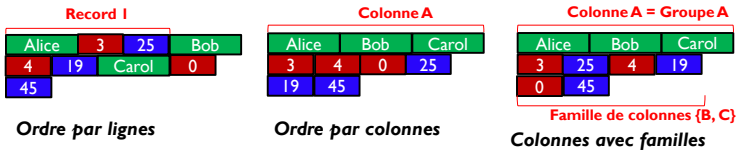
- **Modélisation flexible :** ne nécessite aucune structuration spécifique des données. Utiliser n'importe quel structure qui satisfait les besoins de l'application.
- **Haute performance :** opérations couteuses non nécessaires (jointure, union, etc.), la recherche se fait uniquement sur la clé.
- **Scalabilité :** mise à l'échelle en utilisant des commodity hardwares. La mise à l'échelle ne nécessite aucune re-conception de la BD.
- **Haute disponibilité :** implémentée sur une architecture distribuée (cluster) avec une haute tolérance aux pannes
- **Inconvénients** : plusieurs objets difficilement représentés par une paire clé-valeur.

EXEMPLES DE BD CLÉ-VALEUR

- DynamoDB : une brique dans la solution d'Amazon
- Azure Table Storage : Microsoft
- Riak
- Redis

BD ORIENTÉES COLONNES

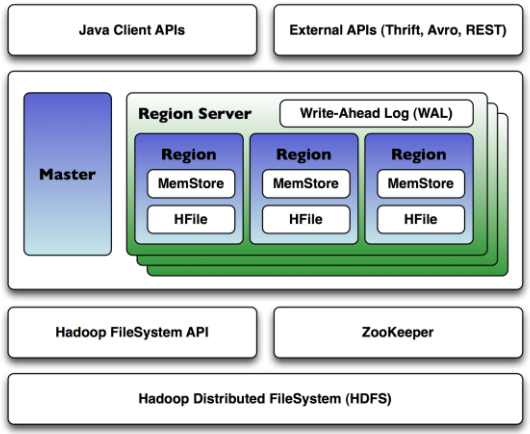
- Les bases de données colonnes sont hybrides entre BD relationnels et clés-valeurs
- Les valeurs sont stockées dans des groupes de plusieurs colonnes mais ordonnés selon les colonnes (en opposition à l'ordre par ligne)



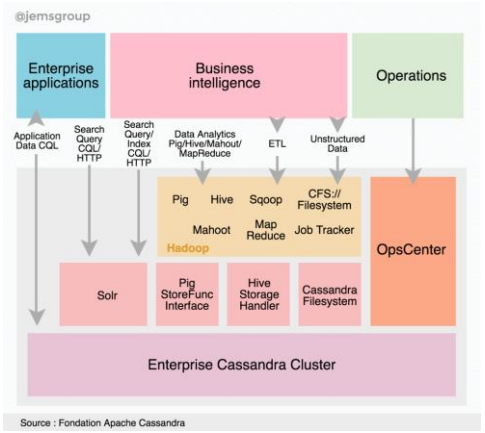
Ordre par lignes ***Ordre par colonnes*** ***Colonnes avec familles***

- Les valeurs sont interrogées par correspondance de clé
- Exemple de BD colonnes : Hbase, Vertica

EXEMPLE : HBASE ARCHITECTURE



EXEMPLE - CASSANDRA



Source : Fondation Apache Cassandra

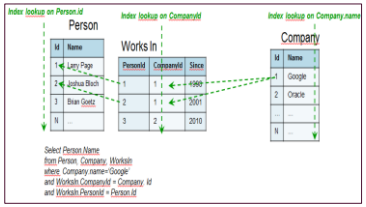
BD ORIENTÉES GRAPHE

- Une base de données graphe est une **base de données spécifiquement dédiée au stockage de structures de données de type graphe**.
- Concepts de base : Nœud, arc
- une base graphe correspond à **tout système de stockage fournissant une adjacence entre éléments voisins sans indexation**
  - Tout voisin d'une entité est accessible directement par un pointeur physique
- Types de graphes modélisés
  - Homogène, hétérogène
  - Orienté, non-orienté
  - Simple, Hypergraphe, etc.

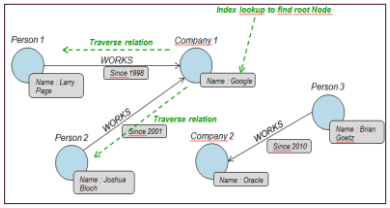
BD GRAPHE

- Caractéristiques
  - Traiter des **données fortement connectées**
  - Gérer facilement un **modèle complexe et flexible**
  - Offrir des performances exceptionnelles pour les lectures locales, par **parcours de graphe**.

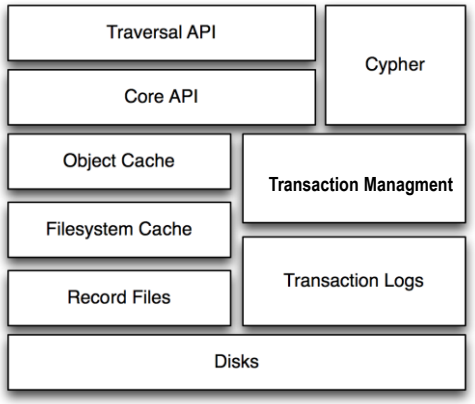
Modélisation relationnelle
Trois jointures



Modélisation graphe
Parcours de liens



EXEMPLE NEO4J



BD ORIENTÉE DOCUMENT

- La base de données orientée documents est une évolution de la base de données clé-valeur
- chaque clé n'est plus associée à une valeur sous forme de bloc binaire mais à un document dont la structure reste libre : XML, JSON
- Rendre la base de données consciente de la structure de la valeur qu'elle stocke
- On peut définir un document comme un ensemble de couples propriété/valeur, dont la seule contrainte est de respecter le format de représentation
- Autres fonctionnalités
  - Ajout, modification, lecture ou suppression de seulement certains champs dans un document
  - Indexation de champs de document permettant ainsi un accès rapide sans avoir recours uniquement à la clé
  - Requêtes élaborées pouvant inclure des prédicats sur les champs

EXEMPLE- DOCUMENT

```
{
  "titre": « BD NoSQL",
  "datePublication" : Date("20/05/2018"),
  "auteur": « K. B.",
  "tags": [ "bigdata", "nosql" ],
  "commentaires": [ {
      "auteur": « Said",
      "commentaire": « Un très bon cours"
    }, {
      "auteur": « Karim",
      "commentaire": « A améliorer"
    }
  ]
}
```

BD MANGODB

MANGODB FONCTIONNALITÉS

- Stockage orienté Document
- Supporte les index
- Replication & haute disponibilité
- Partitionnement automatique des données
- Langage de requêtes
- Rapidité des Mises à jour
- Supporte Map-Reduce

DOCUMENT STORE

BR		MongoDB
Base de donnée	➡	Base de données
Table, Vue	➡	Collection
Tuple	➡	Document (JSON, BSON)
Colonne	➡	Champs
Index	➡	Index
Jointure	➡	Document imbriqués
Clé étrangère	➡	Référence
Partition	➡	Shard

```
> db.user.findOne({age:39})
{
        "_id" :
ObjectId("5114e0bd42…"),
        "first" : "John",
        "last" : "Doe",
        "age" : 39,
        "interests" : [
                "Reading",
                "Mountain Biking ]
        "favorites": {
                "color": "Blue",
                "sport": "Soccer"}
}
```


MISE À JOUR

- Create
  - db.collection.insert(<document>)
  - db.collection.save(<document>)
  - db.collection.update(<query>, <update>, { upsert: true })
- Read
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)
- Update
  - db.collection.update(<query>, <update>, <options>)
- Delete
  - db.collection.remove(<query>, <justOne>)

EXAMPLE

```
> db.user.insert({
        first: "John",
        last : "Doe",
        age: 39
})
```

```
> db.user.find ()
{
        "_id" : ObjectId("51…"),
        "first" : "John",
        "last" : "Doe",
        "age" : 39
}
```

```
> db.user.update(
        {"_id" : ObjectId("51…")},
        {
            $set: {
                age: 40,
                salary: 7000}
        }
)
```

```
> db.user.remove({
        "first": /^J/
})
```