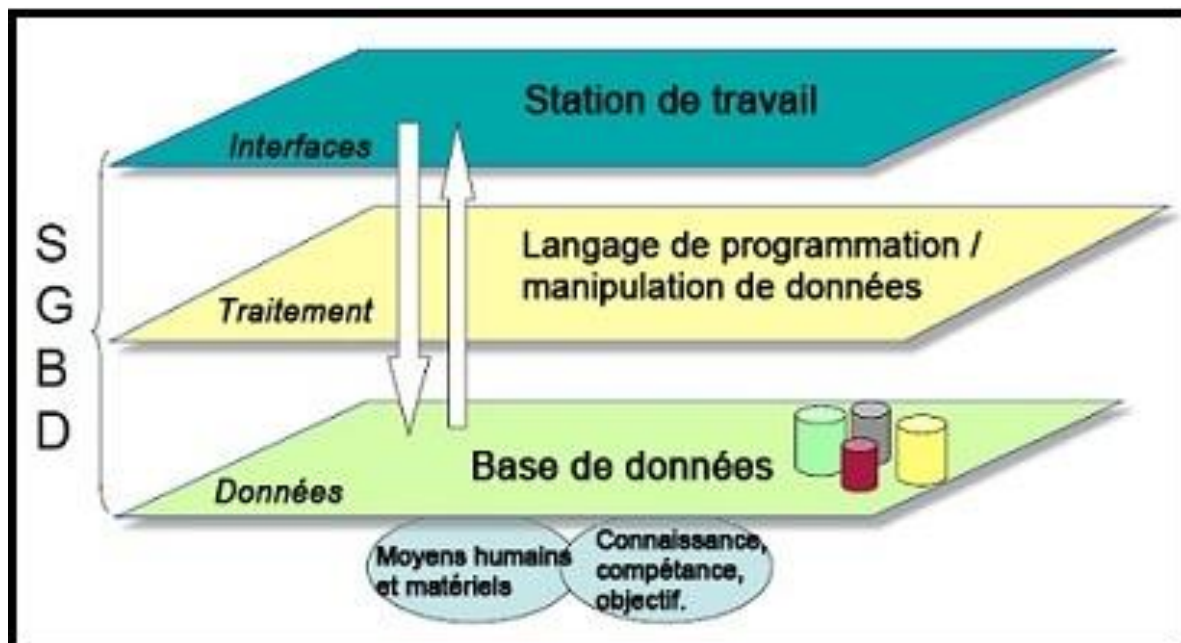


SGBD

Pourquoi et Comment ?

Mooc Fondamentaux pour le Big Data

(Institut Mines-Telecom)



SGBD relationnels classiques

- ▶ Basés sur le **modèle relationnel**
- ▶ Un langage de requêtes standard : **SQL**
- ▶ Données **stockées sur disque**
- ▶ Relations (tables) stockées **ligne par ligne**
- ▶ Système **centralisé**, avec possibilités limitées de distribution

ORACLE®



Microsoft



SYBASE®
An SAP Company



Forces des SGBD relationnels classiques

- ▶ **Indépendance** entre :
 - ▶ modèle de données et structures de stockage
 - ▶ requêtes déclaratives et exécution
- ▶ Requêtes **complexes**
- ▶ **Optimisation** très fine des requêtes, **index** permettant un accès rapide aux données
- ▶ Logiciels **mûrs**, **stables**, **efficaces**, riches en fonctionnalités et en interfaces
- ▶ **Contraintes d'intégrité** permettant d'assurer des invariants sur les données
- ▶ Gestion efficace de **grands volumes de données** (gigaoctet, voire téraoctet)
- ▶ **Transactions** (ensembles d'opérations élémentaires) garantissant la gestion de la concurrence, l'isolation entre utilisateurs, la reprise sur panne

Propriétés ACID

Les **transactions** des SGBD relationnels classiques respectent les propriétés **ACID** :

Atomicité : L'ensemble des opérations d'une transaction est soit exécuté en bloc, soit annulé en bloc

Cohérence : Les transactions respectent les contraintes d'intégrité de la base

Isolation : Deux exécutions concurrentes de transactions résultent en un état équivalent à l'exécution sérielle des transactions

Durabilité : Une fois une transaction confirmée, les données correspondantes restent durablement dans la base, même en cas de panne

Faiblesses des SGBD relationnels classiques









- ▶ Incapable de gérer de **très grands volumes de données** (de l'ordre du péta-octet)
- ▶ Impossible de gérer des **débits extrêmes** (plus que quelques milliers de requêtes par seconde)
- ▶ Le modèle relationnel est parfois peu adapté au stockage et à l'interrogation de **certains types de données** (données hiérarchiques, faiblement structurées, semi-structurées)
- ▶ Les propriétés ACID entraînent de sérieux **surcoûts** en latence, accès disques, temps CPU (verrous, journalisation, etc.)
- ▶ Performances **limitées par les accès disque**

NoSQL

- ▶ No SQL ou Not Only SQL
- ▶ SGBD avec d'autres compromis que ceux faits par les systèmes classiques
- ▶ Écosystème très varié
- ▶ Fonctionnalités recherchées : modèle de données différent, passage à l'échelle, performances extrêmes
- ▶ Fonctionnalités abandonnées : ACID, (parfois) requêtes complexes

Systèmes avec modèle de données différent

Requêtes complexes, modèle de données non relationnel

Type	Organisation	Requêtes	Exemples de systèmes
XML	Données arborescentes, hiérarchiques	XQuery	 
Objet	Données complexes, avec propriétés et méthodes	OQL, VQL	 
Graphe	Graphe avec nœuds, arêtes, propriétés	Cypher, Gremlin	 
Triplets	Triplets RDF du Web sémantique	SPARQL	 

Systèmes clef-valeur

- ▶ Requêtes **très simples** :
 - get** récupère la valeur associée à une clef
 - put** ajoute un nouveau couple clef/valeur
- ▶ Accent mis sur le **passage à l'échelle** transparent, une **faible latence**, un **débit très élevé**
- ▶ Exemple d'implémentation : **table de hachage distribuée**



Chord

MemcacheDB

Systèmes orientés document

- ▶ Requêtes toujours **très simples** :
 - get** récupère le document (JSON, XML, YAML...) associé à une clef
 - put** ajoute un nouveau document associé à une clef
- ▶ Des **index additionnels** permettant de récupérer les documents contenant tel mot-clef, ayant telle propriété, etc.
- ▶ Documents **organisés en collections**, gestion de méta-données (versions, dates), etc.
- ▶ Accent mis sur la **simplicité de l'interface**, la **facilité de manipulation** dans un langage de programmation



Systèmes orientés colonnes

- ▶ Au lieu de stocker les données ligne par ligne, les **stocker colonne par colonne**
- ▶ Organisation **plus riche** que dans les systèmes clef-valeur (plusieurs colonnes par objet stocké)
- ▶ Rend plus efficace l'**agrégation ou le parcours des valeurs d'une même colonne**
- ▶ **Distribution** transparente, **passage à l'échelle** grâce à des arbres de recherche distribués ou des tables de hachages distribuées



NewSQL

- ▶ Certaines applications nécessitent :
 - ▶ des langages de requêtes **riches** (jointure, agrégation)
 - ▶ une conformité aux propriétés **ACID**
 - ▶ mais des **performances supérieures** à celles des SGBD classiques
- ▶ Solutions possibles :
 - ▶ Se débarrasser des **goulots d'étranglement** classiques des SGBD : verrous, journalisation, gestion des caches
 - ▶ Bases de données **en mémoire vive**, avec copie sur disque asynchrone
 - ▶ Une gestion de concurrence **sans verrou** (MVCC)
 - ▶ Une architecture distribuée sans partage d'information (**shared nothing**) et avec **équilibrage de charge** transparent



Dans quels cas choisir un SGBD non classique ?

- ▶ Quand on a des besoins de **latence** ou de **débit extrêmes**
- ▶ Quand on a des **volumes** de données **extrêmes**
- ▶ Quand le modèle relationnel et SQL **se prêtent mal** au stockage et à l'accès aux données (pas si fréquent !)
- ▶ Quand, après tests détaillés, les performances des SGBD classiques se révèlent **insuffisantes**
- ▶ **Savoir ce qu'on perd** : ACID (suivant les cas), possibilité d'interrogations complexes, stabilité de logiciels bien établis. . .
- ▶ Les bases de données NoSQL et NewSQL répondent à de **vrais besoins**. . . mais les besoins sont **souvent surestimés**

Notions de base SQL

```
1 SELECT <liste d'expressions>
2 FROM <liste de tables>
3 WHERE <conditions>
4 GROUP BY <liste d'attributs>
5 HAVING <conditions>
6 ORDER BY <liste d'attributs>
```

Les rôles des trois premières lignes sont les suivants :

- La clause SELECT spécifie le schéma de sortie (projection).
- La clause FROM précise les tables impliquées et leurs liens (produit cartésien et jointures).
- La clause WHERE fixe les conditions que doivent remplir les n-uplets résultats (sélection).

Les trois dernières clauses nous font sortir du calcul et de l'algèbre relationnelle :

- La clause GROUP BY explique comment regrouper des nuplets (agrégation).
- La clause HAVING impose une condition sur les groupes (i.e. permet d'éliminer l'intégralité d'un groupe, en se basant sur la valeur d'un agrégat calculé sur ce groupe).
- Enfin ORDER BY définit les critères de tris des résultats.

Cette requête est déclarative au même sens que l'est le calcul relationnel, c'est-à-dire qu'elle explique ce qu'on souhaite obtenir, et non pas comment l'obtenir.

La clause SELECT est suivie d'une liste d'expressions. Chaque expression peut être un attribut, un littéral entre guillemet ou une expression calculée. Le caractère '*' est un joker signifiant « tous les attributs de la table ».