11-06 matin Cours : Bases de Données - Conception, Modélisation et Technologies Modernes

Bases de Données : Conception, Modélisation et Technologies Modernes

Introduction : L'Évolution des Systèmes de Gestion de Données

Les bases de données constituent l'épine dorsale des systèmes d'information modernes, servant de fondement à pratiquement toutes les applications informatiques contemporaines. L'évolution technologique récente a considérablement élargi le spectre des solutions disponibles, allant des bases de données relationnelles traditionnelles aux systèmes NoSQL spécialisés, en passant par les architectures distribuées à grande échelle.
Cette diversification répond à des besoins croissants de scalabilité, de flexibilité et de performance dans un contexte où les volumes de données atteignent des proportions sans précédent.
Comprendre ces technologies, leurs forces et leurs limitations devient essentiel pour tout professionnel de l'informatique moderne.

Chapitre 1 : Types de Données et Optimisation

1.1 Fondements de l'Encodage des Données

L'encodage des données représente le processus critique de conversion d'informations compréhensibles par l'humain vers un format manipulable par les systèmes informatiques. Cette transformation doit prendre en compte non seulement la **partie entière**des valeurs numériques, mais également leur **partie fractionnaire**, particulièrement cruciale dans certains domaines d'application.

Domaines critiques pour la précision :

- **Applications financières**: où chaque décimale peut représenter des montants considérables
- **Calculs scientifiques**: nécessitant une précision maximale pour éviter les erreurs de propagation
- **Applications temps réel**: où la vitesse de traitement est critique mais ne doit pas compromettre l'exactitude

1.2 Stratégies d'Optimisation des Types Numériques

Le dimensionnement approprié des types de données constitue un enjeu majeur d'optimisation. Le principe fondamental consiste à **adapter la taille du type aux besoins réels**de l'application, évitant ainsi le gaspillage de ressources.

Tailles standard et leurs plages :

- **8 bits**: 0 à 255 (unsigned) ou -128 à 127 (signed)
- **16 bits**: 0 à 65 535 (unsigned) ou -32 768 à 32 767 (signed)
- **32 bits**: 0 à 4 294 967 295 (unsigned)
- **64 bits**: plage considérablement élargie pour les très grandes valeurs
  Cas d'application concret : Base Interpol
  Un exemple emblématique illustre ces enjeux : le développement d'une base de données pour Interpol destinée à stocker les passeports volés, plaques d'immatriculation et personnes recherchées. Cette base devait être répliquée sur des postes portables déployés dans les forces de police mondiales, avec des contraintes particulières :
- **Fonctionnement hors ligne**: autonomie complète nécessaire
- **Synchronisation limitée**: mise à jour quotidienne ou hebdomadaire
- **Connexions dégradées**: débit faible, taux de perte de paquets élevé
- **Intégrité critique**: gestion des checksums et retry
  Dans ce contexte, chaque octet économisé réduisait le temps de transfert, améliorait la fiabilité de synchronisation et diminuait les coûts de communication satellitaire.

1.3 Types de Données Textuelles : CHAR versus VARCHAR

La gestion des chaînes de caractères implique un choix stratégique entre deux approches fondamentalement différentes :

Type CHAR (taille fixe) :

- Réserve toujours le même espace disque
- Performance prévisible grâce à l'accès direct par offset

- Risque de gaspillage si la chaîne est plus courte que l'espace alloué
  Type VARCHAR (taille variable) :
- Adapte l'espace à la longueur réelle du contenu
- Optimise l'utilisation de l'espace disque
- Nécessite un overhead de gestion pour stocker la longueur réelle
  Stratégies de choix :
  Utiliser CHAR lorsque :
- Les données ont toujours la même longueur (codes pays, identifiants fixes)
- Les performances d'accès sont prioritaires
- Le volume de données est relativement faible
  Utiliser VARCHAR lorsque :
- Les données ont des longueurs très variables
- L'économie d'espace est prioritaire
- Le volume total est important (contexte Big Data)

Chapitre 2 : Systèmes de Clés et Relations

2.1 Concept Fondamental des Clés

Une **clé** en base de données fonctionne conceptuellement comme une clé physique : elle permet d'**accéder de manière unique** à une ressource spécifique. Dans le contexte des bases de données, elle garantit l'identification univoque d'un enregistrement parmi potentiellement des millions d'autres.

2.2 Clés Primaires : Identification et Évolution

La clé primaire constitue l'**identifiant unique** de chaque enregistrement dans une table, devant respecter deux contraintes fondamentales :

- **Unicité** : aucune duplication possible
- **Non-nullité** : toujours présente et définie
  Types de clés primaires :
  Clés naturelles : basées sur les attributs métier existants (numéro de sécurité sociale, email unique)
  Clés artificielles (surrogate keys) : identifiants techniques générés spécifiquement pour l'identification, offrant l'avantage d'être indépendants des données métier mais impliquant des données supplémentaires sans valeur métier directe.

2.3 Évolution vers les Systèmes Distribués

L'approche traditionnelle d'auto-incrémentation (ID : 1, 2, 3, 4..) présente des limitations majeures dans les systèmes distribués :

- **Problèmes de scalabilité** sur systèmes distribués

- **Conflits de synchronisation** entre serveurs multiples
- **Goulots d'étranglement** dans les architectures parallèles
  Solutions modernes :
- **UUID** (Universally Unique Identifier) : identifiants de 128 bits garantissant l'unicité globale
- **GUID** (Globally Unique Identifier) : basés sur des algorithmes cryptographiques et temporels
- **Stratégies de partitionnement** : nombres pairs/impairs par serveur, préfixes géographiques
  Ces approches éliminent les collisions et permettent une génération locale sur chaque nœud sans synchronisation.

2.4 Clés Étrangères et Intégrité Référentielle

Les **clés étrangères** établissent les liens entre tables, formant l'architecture relationnelle. Contrairement aux clés primaires, elles peuvent apparaître plusieurs fois dans une table et référencent obligatoirement la clé primaire d'une autre table.
Problématique des clés métier :
Les clés métier possèdent souvent une signification fonctionnelle qui peut évoluer. Par exemple, dans un système RH :

- Matricules commençant par 8 : cadres
- Matricules commençant par 2 : ETAM (Employés, Techniciens et Agents de Maîtrise)
- Matricules commençant par 1 : externes
  Le problème majeur réside dans leur **mutabilité** : lorsqu'un employé change de statut, son matricule doit évoluer, nécessitant des modifications complexes dans toutes les tables liées.
  Solution : clés de substitution
  Les **clés techniques** résolvent cette problématique en étant :
- Artificielles sans signification métier
- **Immuables** une fois attribuées
- Compatibles avec l'existence simultanée de clés secondaires uniques pour les besoins métier

2.5 Contraintes d'Intégrité Référentielle

La déclaration explicite des contraintes lors de la création des tables est essentielle. SQL propose plusieurs options via la clause `ON DELETE` :
RESTRICT (comportement par défaut) : empêche la suppression tant qu'une référence existe
SET NULL : remplace automatiquement la référence par NULL lors de la suppression
CASCADE : supprime automatiquement tous les enregistrements référençant l'élément supprimé
Ces contraintes offrent une protection automatique par le SGBD et constituent une documentation vivante des relations entre entités.

Chapitre 3 : Transactions et Propriétés ACID

3.1 Contexte Historique et Enjeux

Les systèmes de gestion de bases de données relationnelles ont émergé dans le secteur bancaire, où la gestion simultanée de milliers de transferts d'argent nécessitait des garanties de cohérence absolues. Cette origine explique l'importance accordée aux propriétés transactionnelles.
Exemple illustratif : transfert bancaire
Un transfert de 1000€ implique :

1. Débiter 1000€ du compte source
2. Créditer 1000€ sur le compte destination
   Les problèmes de concurrence peuvent provoquer :

- **Panne système** : argent disparu ou dupliqué si interruption entre les deux opérations
- **Accès concurrent** : lecture simultanée du même solde par deux transactions, causant des pertes

3.2 Propriétés ACID Détaillées

A - Atomicité

Principe "tout ou rien": une transaction est indivisible. Soit toutes les opérations réussissent, soit aucune n'est appliquée. En cas d'échec partiel, le système effectue un **rollback** automatique.

C - Cohérence

La base doit respecter toutes les règles de cohérence définies avant et après chaque transaction. En environnement distribué, cette propriété devient particulièrement complexe car elle doit considérer l'ensemble du système, non chaque nœud isolément.
Problématique distribuée : Si deux serveurs (Paris et Lyon) tentent simultanément de débiter 10 000€ d'un compte contenant exactement cette somme, chaque serveur peut valider individuellement l'opération, résultant en un solde final de -10 000€.

I - Isolation

Les transactions concurrentes ne doivent pas interférer entre elles. Cette propriété présente un dilemme entre **performance** et **exactitude**, résolu par différents niveaux d'isolation :
Niveaux d'isolation (du plus strict au plus permissif) :

1. **Serializable** : isolation maximale, performance minimale
2. **Repeatable Read** : lectures cohérentes dans une transaction

3. **Read Committed** : lecture des données committées uniquement
4. **Read Uncommitted** : performance maximale, risques de dirty reads

D - Durabilité

Une fois validée (commit), une transaction doit persister même en cas de panne système ultérieure. Cette propriété nécessite une écriture physique sur disque, pas seulement en cache.
Défis techniques : Les multiples couches de cache (OS, contrôleur RAID, virtualisation) peuvent compromettre la durabilité si une coupure survient avant l'écriture physique effective.

3.3 Stratégies de Verrouillage

Pessimistic Locking : verrouillage préventif des ressources, correspondant au niveau Serializable
Optimistic Locking : pari sur l'absence de conflits avec détection a posteriori. Problématique dans les applications à fort trafic où les collisions deviennent fréquentes.

Chapitre 4 : Fonctionnement du Moteur et Optimisation des Requêtes

4.1 Architecture Physique et Logique

Une base de données constitue fondamentalement un **espace de stockage sur disque** contenant des milliards d'octets organisés en bits. Le SGBD agit comme intermédiaire intelligent entre l'utilisateur et ces données brutes, gérant le chargement depuis le stockage permanent vers la mémoire vive selon les besoins.
Cette opération de chargement détermine les performances de toute requête, le moteur utilisant des **pointeurs** pour naviguer dans les structures et ne chargeant initialement que les métadonnées nécessaires.

4.2 Ordre d'Exécution Logique des Requêtes

Contrairement à l'intuition naturelle, l'ordre d'exécution ne suit pas l'ordre syntaxique. La séquence logique est :

1. **FROM** : identification et chargement des tables sources
2. **JOIN** : création de tables temporaires par produit cartésien filtré
3. **WHERE** : filtrage ligne par ligne des données
4. **GROUP BY** : regroupement logique des données
5. **HAVING** : filtrage des groupes après agrégation
6. **SELECT** : sélection et calcul des colonnes finales
7. **UNION** : combinaison de jeux de résultats
8. **ORDER BY** : tri final

4.3 Mécanismes des Jointures

Les jointures créent des **tables temporaires**en mémoire via un produit cartésien conceptuel entre les tables impliquées. Le moteur optimise ce processus en :

- Choisissant intelligemment la table de base
- Appliquant les critères de jointure pour éliminer les combinaisons non pertinentes
- Utilisant les index disponibles pour accélérer les opérations
  La taille de ces tables temporaires impacte directement les performances, particulièrement dans le contexte Big Data où joindre des tables de millions de lignes peut générer des structures de plusieurs gigaoctets.

4.4 Optimisations et Recommandations

Méthodologie de construction : commencer par FROM et identifier toutes les sources de données, puis progresser selon l'ordre logique d'exécution.
Optimisation mémoire : l'ajout de RAM peut diviser les temps d'exécution par 10 en permettant l'utilisation de stratégies de jointure en mémoire plutôt que sur disque.
Standards de portabilité : SQL est normalisé depuis ANSI SQL 92 (1992), couvrant environ 80% des besoins courants. Les extensions spécifiques (fonctions spatiales, analytiques) varient selon les SGBD mais suivent une logique similaire.

Chapitre 5 : SQL Avancé et Agrégations

5.1 La Clause HAVING : Filtrage Post-Agrégation

La clause **HAVING** constitue l'un des mécanismes les plus puissants pour le filtrage de données après agrégation. Sa distinction temporelle avec WHERE est cruciale : **WHERE filtre avant agrégation, HAVING filtre après**.
Exemple pratique d'analyse salariale :

```
SELECT annee_embauche,
MAX(salaire) as salairemax,
   AVG(salaire) as salairemoyen
FROM employes
GROUP BY annee_embauche
HAVING MAX(salaire) > 2 * AVG(salaire);
```

Cette requête identifie les années présentant des disparités salariales importantes, révélant des politiques de rémunération potentiellement déséquilibrées.
Règle fondamentale : HAVING ne peut exister sans GROUP BY, cette interdépendance reflétant la logique de traitement séquentiel des données.

5.2 Ordre Logique et Optimisation Conceptuelle

L'approche séquentielle optimise naturellement les performances :

1. Filtrage des données non pertinentes (WHERE)
2. Regroupement sur un ensemble réduit
3. Calculs uniquement sur les données finales
   Cette méthode évite des calculs coûteux (prix TTC = prix × quantité × TVA) sur des lignes destinées à être supprimées.

5.3 Opérations d'Union et Tri Global

UNION vs UNION ALL :

- **UNION** : élimine automatiquement les doublons
- **UNION ALL** : conserve toutes les lignes, plus performant
  L'ORDER BY s'applique au résultat consolidé après toutes les unions, nécessitant des parenthèses explicites pour trier avant fusion.

Chapitre 6 : Introduction au NoSQL

6.1 Philosophie et Définition

NoSQL signifie "**Not Only SQL**" et non "No SQL", reflétant une approche complémentaire plutôt qu'antagoniste. Cette nuance traduit une philosophie d'ouverture reconnaissant que d'autres approches peuvent être plus adaptées dans certains contextes.

6.2 Motivations et Avantages

Scalabilité horizontale : contrairement à l'approche verticale (ajout de puissance), l'approche horizontale permet d'ajouter de nouveaux nœuds pour absorber la charge croissante, crucial dans le contexte Big Data.
Relaxation des contraintes ACID : échange d'une partie des garanties contre des performances accrues lorsque les exigences fonctionnelles le permettent.
Flexibilité schématique : évolution dynamique des structures, adaptation rapide aux changements de besoins, contrairement à la rigidité relationnelle.

6.3 Modèles de Données et Applications

Document (MongoDB, Elasticsearch) : stockage de documents structurés, optimal pour les catalogues hétérogènes
Clé-valeur (Redis, DynamoDB) : dictionnaire géant optimisé pour la performance, idéal pour les caches et sessions
Colonnes (Cassandra, HBase) : optimisation pour l'analytique avec stockage columnar

permettant des compressions efficaces
Graphes (Neo4j, ArangoDB) : gestion des relations complexes, excellent pour les réseaux sociaux et recommandations
Time Series (InfluxDB, Prometheus) : optimisation temporelle avec architecture immutable "Write-Once, Read-Many"

Chapitre 7 : NoSQL Spécialisé et Théorème CAP

7.1 Problématiques des Schémas Variables

Les systèmes NoSQL permettent une **flexibilité schématique** remarquable. Un document JSON peut contenir des attributs absents dans d'autres documents de la même collection, avantage crucial pour les marketplaces hétérogènes où chaque catégorie possède des caractéristiques spécifiques.
Exemple e-commerce : un produit livre aura `{titre, auteur, ISBN, pages}` tandis qu'un vêtement aura `{nom, couleur, taille, matière}`. En relationnel, cela nécessiterait une table avec tous les champs possibles, générant majoritairement des valeurs NULL.

7.2 Architecture des Data Lakes

L'approche Data Lake consiste à :

1. **Enregistrer** chaque événement sous forme brute
2. **Stocker** sans transformation préalable
3. **Traiter** a posteriori selon les besoins analytiques

Cette philosophie s'intègre parfaitement avec les capacités NoSQL de gestion de données semi-structurées.

7.3 Théorème CAP : Compromis Fondamentaux

Le théorème CAP établit qu'un système distribué ne peut garantir simultanément :
C (Consistency) : tous les nœuds voient les mêmes données simultanément
A (Availability) : le système répond toujours aux requêtes
P (Partition Tolerance) : fonctionnement malgré les pannes réseau
Trois combinaisons possibles :
CP : sacrifice de la disponibilité pour maintenir cohérence (bases relationnelles distribuées)
AP : acceptation d'inconsistance temporaire pour rester opérationnel (réseaux sociaux)
CA : fonctionnement parfait sans pannes réseau (bases traditionnelles mono-site)

7.4 Eventual Consistency

L'**eventual consistency** représente le compromis pragmatique des systèmes AP : toutes les écritures seront propagées et la convergence surviendra "éventuellement", sans garantie de délai précis.
Illustration : publication Facebook immédiatement sauvegardée (availability) mais propagation progressive vers les timelines (eventual consistency).

Chapitre 8 : Choix Technologique et Architectures Hybrides

8.1 Critères de Décision

Le choix entre SQL et NoSQL dépend fondamentalement du **pattern d'accès** aux données :
Favoriser SQL :

- Requêtes imprévisibles
- Jointures complexes fréquentes
- Propriétés ACID critiques
- Données fortement structurées
  Favoriser NoSQL :
- Patterns d'accès prévisibles
- Volume nécessitant scalabilité horizontale
- Performance prioritaire sur consistance immédiate
- Données semi-structurées ou variables

8.2 Cas d'Usage Sectoriels

Système de réservation (aviation) : base relationnelle pour propriétés transactionnelles critiques, cohérence forte, relations complexes
Catalogue e-commerce : architecture hybride avec MongoDB pour le catalogue (attributs variables) et PostgreSQL pour les transactions
Application IoT : base Time Series (InfluxDB) pour volume d'écriture élevé, données horodatées, agrégations temporelles

8.3 Architectures Polyglotte

Dans la pratique, les grandes applications utilisent **plusieurs types de bases** simultanément :

- **Base relationnelle** : données de référence, transactions financières
- **Base documentaire** : catalogue produits, contenu utilisateur
- **Cache clé-valeur** : sessions, données temporaires
- **Base time series** : métriques, logs, monitoring
  Cette approche nécessite des **pipelines ETL** pour synchroniser les données entre systèmes, maintenir la cohérence globale, mais ajoute une complexité administrative et

opérationnelle considérable.

Chapitre 9 : Modélisation et Conception

9.1 Universalité de la Modélisation

La modélisation constitue une étape fondamentale **indépendamment du type de base** choisi. Même les bases NoSQL nécessitent des décisions cruciales : pour Elasticsearch, définir un champ "titre" implique des choix de tokenisation, indexation, recherche et sémantique impactant directement performances et pertinence.

9.2 Niveaux d'Abstraction

Niveau conceptuel (QUOI) : concepts métiers et relations, indépendance technologique totale, langage accessible aux non-techniques
Niveau logique (COMMENT) : traduction en structures de données, indépendant de l'implémentation spécifique
Niveau physique (AVEC QUOI) : spécifications techniques liées au SGBD choisi

9.3 Transformation et Règles

Règles fondamentales :

- Entité → Table
- Association 1..N → Clé étrangère côté N
- Association N..N → Table de jointure
- Optimisations 1..1 pour performance
  La compréhension des mécanismes de stockage (chargement par blocs de 32-64 Ko) guide les décisions de modélisation, justifiant l'externalisation des données volumineuses.

Chapitre 10 : Normalisation Théorique et Pratique

10.1 Objectifs Fondamentaux

La normalisation vise à :

- **Éliminer la redondance** : éviter la duplication d'informations
- **Prévenir les anomalies** : insertion, suppression, mise à jour
- **Optimiser la structure** : maintenir la cohérence et l'intégrité

10.2 Formes Normales Essentielles

Première Forme Normale (1NF) : atomicité du contenu, exclusion des listes de valeurs dans une colonne
Deuxième Forme Normale (2NF) : dépendance fonctionnelle complète de la clé primaire, particulièrement importante pour les clés composites
Troisième Forme Normale (3NF) : élimination des dépendances transitives, toutes les informations concernent directement l'entité identifiée par la clé primaire

10.3 Formes Normales Avancées (4NF et 5NF)

Quatrième Forme Normale (4NF): S'applique aux dépendances multi-valuées, suggérant de séparer les attributs non-clés qui dépendent de plusieurs autres attributs non-clés (ex: listes de compétences et de langues d'un employé). Son application stricte peut complexifier l'usage par la multiplication des tables et jointures.
Cinquième Forme Normale (5NF): Concerne les dépendances de jointures, visant à optimiser la reconstruction des données via des jointures. Elle est considérée comme très théorique et rarement appliquée en pratique.
Pertinence pratique: Ces formes avancées sont peu utilisées explicitement dans la modélisation quotidienne, les professionnels visant intuitivement la 3NF et n'envisageant la dénormalisation que de manière stratégique.

10.4 Dénormalisation Stratégique

La **dénormalisation**consiste à introduire volontairement de la redondance pour optimiser les performances. Justifiée dans les contextes :

- **Big Data et analytique** : tables de faits pré-jointes
- **Données principalement en lecture** : optimisation des requêtes fréquentes
- **Performance critique** : réduction des jointures nécessaires
  Cette approche nécessite d'accepter consciemment la redondance, complexité de mise à jour et risques d'incohérence en échange des bénéfices escomptés.

10.5 Réalité Professionnelle

Dans la pratique, l'application des formes normales est souvent **intuitive**, guidée par les bonnes pratiques de programmation orientée objet. La référence explicite à la théorie reste rare, l'accent étant mis sur le pragmatisme et les résultats business.

Schéma Récapitulatif

https://archive.qoyri.fr/mindmaps/cours06.11.25.html

Concepts Principaux

Types de Données et Optimisation :

- Dimensionnement selon besoins réels
- CHAR (fixe) vs VARCHAR (variable)
- Impact performance/stockage

Système de Clés :

- Clé primaire : identification unique
- Clés artificielles : indépendance métier
- UUID/GUID : solutions distribuées
- Intégrité référentielle : contraintes ON DELETE

Transactions et ACID :

- Atomicité : tout ou rien
- Cohérence : respect des règles
- Isolation : niveaux et compromis performance
- Durabilité : persistance garantie

SQL et Moteur :

- Ordre logique : FROM → JOIN → WHERE → GROUP BY → HAVING → SELECT → UNION → ORDER BY
- Optimisation : tables temporaires, jointures
- HAVING : filtrage post-agrégation

NoSQL et Alternatives :

- Définition : "Not Only SQL"
- Types : Document, Clé-valeur, Colonnes, Graphes, Time Series
- Théorème CAP : Consistency, Availability, Partition Tolerance
- Eventual consistency vs cohérence forte

Modélisation :

- Niveaux : Conceptuel, Logique, Physique
- Règles de transformation
- Normalisation vs Dénormalisation

Mots-Clés Essentiels

Technique : ACID, UUID, Sharding, Réplication, Index, Jointures, Produit cartésien, Table temporaire, ETL
Conceptuel : Entité, Association, Cardinalité, Clé primaire/étrangère, Atomicité, Cohérence,

Isolation, Durabilité
Architectures : Scalabilité horizontale/verticale, Architecture polyglotte, Data Lake, Eventual consistency, CAP, NoSQL, Time Series
Modélisation : Normalisation, Dénormalisation, Forme normale, Redondance, Anomalie, Dépendance fonctionnelle, Clé de substitution

Cette synthèse constitue un fondement solide pour comprendre l'évolution des technologies de bases de données et maîtriser les choix architecturaux adaptés aux contextes applicatifs modernes. La compréhension de ces concepts permet de naviguer efficacement entre solutions relationnelles traditionnelles et alternatives NoSQL, en fonction des contraintes spécifiques de chaque projet.

Revision #2
Created 6 November 2025 13:26:07 by qoyri
Updated 6 November 2025 13:53:14 by qoyri