

Bases de Données

Pierre Senellart

13 mars 2024

Première partie

Base de Données

1 Introduction

1.1 Gestion de Données

Numerous applications (standalone software, Web sites, etc.) need to manage data :

- Structure data useful to the application
- Store them in a persistent manner (data retained even when the application is not running)
- Efficiently query information within large data volumes
- Update data without violating some structural constraints
- Enable data access and updates by multiple users, possibly concurrently

Often, it is desirable to access the same data from several distinct applications, from distinct computers.

A naïve implementation can be proposed with :

- Definition ad-hoc file formats to store data on disk, with regular sync and failure recovery.
- In-Memory storage of app data with structures and algorithms for efficiently finding information
- Data update functions which preserve functions.
- Definition of user access and an authentication process
- Definition of a network communication process

At the foundations on DBMS (Data Base Management Systems) is the Codd Theorem, expressing the equivalence between logical formalisms and algebra.

Définition 1.1: SGBD

Un Système de Gestion de Bases de Données (SGBD) est un logiciel qui simplifie le design d'applications qui manipule des données, en proposant un accès indépendant de l'application aux fonctionnalités requises pour le data management.

Définition 1.2: Base de Données

Une Base de Données (DB) est une collection de données (spécifiques à une application données) gérée par un SGBD.

Proposition 1.1: Features des SGBD

Indépendance Physique L'utilisateur d'un SGBD n'a pas besoin de savoir comment les données sont stockées. Le stockage peut être modifié sans impacter l'accès.

Indépendance Logique Il est possible d'accorder à l'utilisateur une vision partielle des données correspondant à ce dont il a besoin et auquel il a accès.

Facilité d'Accès On utilise un langage déclaratif décrivant les requêtes et modifications sur la data, spécifiant le but d'un utilisateur plutôt que la manière dont c'est implémenté.

Optimisation des Requêtes Les requêtes sont automatiquement optimisées pour être implémentées aussi efficacement que possible sur la DB.

Intégrité Physique La DB doit rester dans un état cohérent et les données doivent être préservées, même en cas d'échec logiciel ou physique.

Intégrité Logique Le SGBD fixe des contraintes sur la structure et rejette les modifications brisant ces contraintes.

Partage de Données Les données doivent être accessibles par plusieurs utilisateurs en même temps, et ces accès ne doivent pas briser l'intégrité physique ou logique des données.

Standardisation L'utilisation d'un SGBD est standardisée, de sorte qu'il est possible de changer de SGBD sans trop changer le code de l'application.

1.2 Types de SGBD

Il existe des dizaines de SGBD qui ne possèdent pas tous toutes ces features. Ils peuvent être différenciés selon

- leur modèle de données
- leur équilibre performance-fonctionnalités
- leur facilité d'utilisation
- leur scalabilité
- leur architecture interne

Proposition 1.2: Types de SGBD

Relationnel (RDBMS) Tables, requêtes complexes (SQL), fonctionnalités nombreuses

XML Arbres, requêtes complexes (XQuery), fonctionnalités semblables au RDBMS

Graph/Triples Données sous forme de graph, requêtes complexes exprimant la navigation dans le graphe.

Objets Modèle de donnée complexe, inspiré par OOP

Documents Modèle complexe, organisé en documents, requêtes et fonctionnalités relativement simples

Key-Value Modèle très basique de données, focus sur la performance.

Column Stores Modèle de données entre Key-Value et RDBMS, focus sur l'itérabilité et l'agglomération de colonnes.

Les RDBMS classiques sont basés sur le modèle relationnel (la décomposition de données en relations ou tables). Les RDBMS utilisent un langage de requêtes standard : le SQL. Les données sont stockées sur un disque et les relations sont stockées lignes par lignes. Ceci amène à un système centralisé avec des possibilités de distributions limitées.

Ils sont utiles pour :

- l'indépendance entre le modèle de donnée et les structures de stockages et les requêtes déclaratives et la manière dont celles-ci sont exécutées.

- leurs requêtes complexes
- leur optimisation fine des requêtes, avec des annuaires permettant un accès rapide aux données
- leur technologie éprouvée, stable, efficace avec de nombreuses fonctionnalités et interfaces
- des contraintes d'intégrité avec des invariants sur les données
- une gestion efficace de volumes très larges de données (jusqu'aux Terabytes)
- des transactions avec des garanties sur le contrôle de la concurrence, du failure recovery et l'isolement des utilisateurs

Proposition 1.3: ACID

Les transactions (suite d'opérations élémentaires) RDBMS satisfont les propriétés ACID :

Atomicité L'ensemble des opérations d'une transaction est soit exécuté comme un tout soit annulé comme un tout

Consistence Les transactions assurent que les contraintes d'intégrité sont respectées.

Isolement Deux exécutions en parallèle de transactions amènent à un état équivalent à l'exécution en série des transactions

Durabilité Une fois que les transactions sont effectuées, les données correspondantes restent durablement dans la base, même en cas d'échec système

Les RDBMS ont tout de même des faiblesses :

- Ils sont incapables de gérer des volumes extrêmement larges de données (de l'ordre d'un petabyte)
- Il est impossible de gérer un rythme extrême de taux de requête (au delà de plusieurs milliers par seconde)
- Le modèle relationnel est parfois mal adapté au stockage et au requêtage de certains types de données (données hiérarchiques, sans ou partiellement structurées)
- Les propriétés ACID impliquent des coûts majeurs en latence, en accès au disque et en temps de calcul
- Les performances sont limitées par l'accès au disque

Tout ce qui n'est pas un RDBMS est un NoSQL ou NotOnlySQL. Ce sont des DBMS avec des organisations différentes des systèmes classiques. Ils forment un écosystème très divers. Ils utilisent un modèle de données différent, ont de meilleures performances et un meilleur passage à l'échelle, mais, ils ne vérifient pas les propriétés ACID et parfois n'ont pas de requêtes complexes.

2 Modèle Relationnel

On fixe dans la suite des ensembles dénombrables :

- \mathcal{L} d'étiquettes
- \mathcal{V} de valeurs
- \mathcal{T} de types tels que $\forall \tau \in \mathcal{T}, \tau \subseteq \mathcal{V}$

Définition 2.1: Schéma Relationnel

Un schéma relationnel (d'arité n) est un n -uplet (A_1, \dots, A_n) où chaque A_i (appelé attribut) est une L_i, τ_i où $L_i \in \mathcal{L}, \tau_i \in \mathcal{T}$ de sorte que tous les L_i sont distincts.

Définition 2.2: Schéma de Base de Données

Un schéma de DB est défini comme un ensemble fini L d'étiquettes de \mathcal{L} (noms de relations), chaque étiquette de L étant associé à un schéma relationnel.

Définition 2.3: Instance Relationnelle

Une instance de schéma relationnel $((L_1, \tau_1), \dots, (L_n, \tau_n))$ (aussi appelé une relation sur ce schéma) est un ensemble fini $\{t_1, \dots, t_k\}$ de tuples de la forme $t_j = (v_{j,1}, \dots, v_{j,n})$ où $\forall j, \forall i, v_{j,i} \in \tau_i$.

Définition 2.4: Instance de DB

Une instance de schéma de base de données (ou simplement une base de donnée sur ce schéma) est une fonction qui à chaque nom de relation associe une instance du schéma relationnel correspondant.

Le mot « Relation » est utilisé de manière ambiguë pour parler d'un schéma relationnel ou d'une instance d'un schéma relationnel.

On peut prendre une DB formée de deux noms de relations **Guest** et **Reservation** où :

- **Guest**: ((id, INTEGER), (name, TEXT), (email, TEXT))
- **Reservation**: ((id, INTEGER), (guest, INTEGER), (room, INTEGER), (arrival, DATE), (nights, INTEGER))

On peut alors représenter une instance par ses deux tables :

Guest		
id	name	email
1	John Smith	john.smith@gmail.com
2	Alice Black	alice@black.name
3	John Smith	john.smith@ens.fr

Reservation				
id	guest	room	arrival	nights
1	1	504	2017-01-01	5
2	2	107	2017-01-10	3
3	3	302	2017-01-15	6
4	2	504	2017-01-15	2
5	2	107	2017-01-30	1

Définition 2.5: Notations

- Si $A = (L, \tau)$ est le i -ème attribut d'une relation R et t un n -uplet d'une instance de R , on note $t[A]$ ou $t[L]$ la valeur de la i -ème composante de t .
- De même, si \mathcal{A} est un k -uplet d'attributs, $t[\mathcal{A}]$ est la k -uplets concaténé des $t[A]_{A \in \mathcal{A}}$.

On peut de plus définir des contraintes d'intégrités pour définir la notion de validité d'une instance :

Définition 2.6: Contrainte simples d'intégrité

Clef Un tuple d'attributs \mathcal{A} d'un schéma relationnel R est une clef s'il n'existe pas deux tuples distincts dans une instance de R ayant les même valeurs sur \mathcal{A} .

Clef Étrangère Un k -uplet d'attributs \mathcal{A} d'un schéma R est une clef étrangère référençant un k -uplet \mathcal{B} d'attributs d'un schéma S si pour toutes instances I^R et I^S de R et S , pour tout tuple t de I^R , il existe un unique tuple t' de I^S avec $t[\mathcal{A}] = t'[\mathcal{B}]$

Check Une condition arbitraire sur les valeurs des attributs d'une relation

Définition 2.7: Variante : Perspectives Nommée et Non-Nommées

Perspective Nommée On oublie la position de l'attribut et on considère qu'ils sont uniquement identifiés par leurs noms.

Perspective Non Nommée On oublie le nom des attributs et on considère qu'ils sont uniquement identifiés par leur position.

La perspective n'a pas d'impact majeur, seulement sur la praticité. On utilise plus généralement et le nom et la position.

Définition 2.8: Variante : Sémantique Multi-Ensemble

Une instance relationnelle est défini comme un ensemble (fini) de tuples. On peut aussi considérer une sémantique multi-ensemble du modèle relationnel, où une instance relationnelle est un multi-ensemble de tuples.

C'est ce qui fonctionne le mieux pour définir comment les RDBMS fonctionnent, mais la plupart de la théorie est établie pour la sémantique à ensembles.

Définition 2.9: Variante : Non-Typée

Dans les modèles et les résultats théoriques, on abstraitise souvent les types des attributs et on considère que chaque attribut à un type universel \mathcal{V} .

3 Langages de Requêtes

3.1 Algèbre Relationnel

Définition 3.1: Algèbre Relationnel

Une expression de l'algèbre relationnel produit une nouvelle relation des relations de la DB à partir d'opérateurs prenant 0, 1 ou 2 sous-expressions :

Op.	Arité	Description	Condition
R	0	Nom de Relation	$R \in \mathcal{L}$
$\rho_{A \rightarrow B}$	1	Renommage	$A, B \in \mathcal{L}$
$\Phi_{A_1 \dots A_n}$	1	Projection	$A_1, \dots, A_n \in \mathcal{L}$
σ_φ	1	Sélection	φ formule
\times	2	Produit	
\cup	2	Union	
\setminus	2	Différence	
\bowtie_φ	2	Jointure	φ formule

Relation Expression : **Guest**

	id	name	email
Résultat :	1	John Smith	john.smith@gmail.com
	2	Alice Black	alice@black.name
	3	John Smith	john.smith@ens.fr

Renommage Expression : $\rho_{id \rightarrow guest}(\text{Guest})$

	guest	name	email
Résultat :	1	John Smith	john.smith@gmail.com
	2	Alice Black	alice@black.name
	3	John Smith	john.smith@ens.fr

Projection Expression : $\Pi_{\text{email}, \text{id}}(\text{Guest})$

	email	id
Résultat :	john.smith@gmail.com	1
	alice@black.name	2
	john.smith@ens.fr	3

Selection Expression : $\sigma_{\text{arrival} > 2017-01-12 \wedge \text{guest}=2}(\text{Reservation})$

Résultat :

La formule utilisé dans la sélection peut contenir n'importe quelle combinaison avec des opérateurs booléens de comparaisons d'attributs à attributs ou de constantes.

Produit Expression : $\Pi_{\text{id}}(\text{Guest}) \times \Pi_{\text{name}}(\text{Guest})$

	id	name
	1	Alice Black
	2	Alice Black
Résultat :	3	Alice Black
	1	John Smith
	2	John Smith
	3	John Smith

Union Expression : $\Pi_{\text{room}}(\sigma_{\text{guest}=2}(\text{Reservation})) \cup$

$\Pi_{\text{room}}(\sigma_{\text{arrival}=2017-01-15}(\text{Reservation}))$

	room
Résultat :	107
	302
	504

Cet union aurait pû être écrite

$\Pi_{\text{room}}(\sigma_{\text{guest}=2 \vee \text{arrival}=2017-01-15}(\text{Reservation}))$

Différence Expression : $\Pi_{\text{room}}(\sigma_{\text{guest} = 2}(\text{Reservation})) \setminus$

$\Pi_{\text{room}}(\sigma_{\text{arrival}=2017-01-15}(\text{Reservation}))$

	room
Résultat :	107

Cette différence simple aurait pû être écrite

$\Pi_{\text{room}}(\sigma_{\text{guest}=2 \wedge \text{arrival} \neq 2017-01-15}(\text{Reservation}))$

Jointure Expression : $\text{Reservation} \bowtie_{\text{guest}=\text{id}} \text{Guest}$ Résultat :

id	guest	arrival	nights	name	email
1	1	504	2017-01-01	5	John Smith john.smith@gmail.com
2	2	107	2017-01-10	3	Alice Black alice@black.name
3	3	302	2017-01-15	6	John Smith john.smith@ens.fr
4	2	504	2017-01-15	2	Alice Black alice@black.name
5	2	107	2017-01-30	1	Alice Black alice@black.name

La formule utilisée dans la jointure peut être n'importe quelle combinaison booléenne de comparaisons d'attributs de la table de gauche et de la table de droite.

La jointure n'est pas une opération élémentaire de l'algèbre relationnelle. Elle peut être vue comme une combinaison de Renommage, de Produit, de sélection et de projection.

Si R et S ont pour attributs \mathcal{A} et \mathcal{B} , on note $R \bowtie S$ la jointure naturelle de R et S où la formule de jointure est $\bigwedge_{A \in \mathcal{A} \cap \mathcal{B}} A = A$.

Deuxième partie

Calcul Relationnel

4 Calcul Relationnel

4.1 Définitions

Définition 4.1: Calcul Relationnel

Le calcul relationnel est un langage logique pour exprimer les requêtes, construit avec de la FOL, sans symboles de fonctions et avec comme symboles de relation les étiquettes du schéma de base de données.

Dans la suite on se fixe :

- Un ensemble \mathcal{X} de variables
- Un ensemble \mathcal{V} de valeurs
- Un schéma de BDD \mathcal{S} .

Proposition 4.1: Syntaxe

- Pour toute relation $R \in \mathcal{S}$ d'arité n , pour tout $(\alpha_1, \dots, \alpha_n) \in (\mathcal{X} \cup \mathcal{V})^n$, $R(\alpha_1, \dots, \alpha_n) \in FO$.
- $\alpha = \alpha' \in FO$ pour $(\alpha, \alpha') \in (\mathcal{X} \cup \mathcal{V})^2$
- On autorise la quantification existentielle et universelle, la négation, la conjonction et la disjonction.

On écrit alors une requête du calcul relationnel sous la forme $Q(x_1, \dots, x_m) = \varphi$ où x_1, \dots, x_m sont des variables libres de φ .

Proposition 4.2: Sémantique

Une requête peut être vue comme une fonction ayant pour input une base de donnée D sur \mathcal{S} et produisant une relation. On note \models la relation définie par induction ci-dessous :

- $D \models (R(u_1, \dots, u_m)) \iff R(u_1, \dots, u_m) \in D$
- $D \models \varphi_1 \wedge \varphi_2 \iff (D \models \varphi_1) \wedge (D \models \varphi_2)$
- $D \models \varphi_1 \vee \varphi_2 \iff (D \models \varphi_1) \vee (D \models \varphi_2)$
- $D \models \neg \varphi_1 \iff D \not\models \varphi_1$
- $D \models \forall x \varphi_1 \iff \forall v \in \text{adom}(D), D \models \varphi_1[x \leftarrow v]$
- $D \models \exists x \varphi_1 \iff \exists v \in \text{adom}(D), D \models \varphi_1[x \leftarrow v]$

Alors, une requête Q s'écrit :

$$Q(D) \{(v_1, \dots, v_m) \in \text{adom}(D)^n \mid D \models \varphi[x_1 \leftarrow v_1, \dots, x_m \leftarrow v_m]\}$$

Définition 4.2: Sous-Classes de Requêtes

Requête Conjonctive Calcul Relationnel sans \vee, \neg, \forall

Requête Positive Calcul relationnel sans \neg, \forall

Requête Union de Conjonctives Cas de requête positive où les \vee et \wedge forment une formule sous FNC.

Requête Booléenne Requête sans variable libre

4.2 Théorème de Codd

Théorème 4.1: Codd

L'algèbre relationnelle et le calcul relationnel sont équivalents :

- Pour toute requête q de l'algèbre relationnelle sur un schéma S il existe une requête Q du calcul relationnel sur S telle que pour toute base de donnée D sur S , $q(D) = Q(D)$
- Pour toute requête Q du calcul relationnel sur un schéma S il existe une requête q de l'algèbre relationnelle sur S telle que pour toute base de donnée D sur S , $q(D) = Q(D)$

De plus, la traduction d'un formalisme à l'autre s'effectue en temps polynômial.

Démonstration. Calcul → Algèbre. Montrer que chaque opérateur de l'algèbre peut être réécrit dans le calcul

Calcul → Algèbre. Construire une requête permettant de renvoyer toutes les valeurs du domaine actif puis réécrire par induction toute expression du calcul en une expression algébrique. ■

Ce théorème nous autorise à utiliser un formalisme déclaratif pour spécifier nos requêtes : la logique, ou le SQL. On peut alors les compiler par la transformation de Codd en un formalisme algébrique. Les requêtes algébriques sont alors optimisées, puis évaluées.

5 Complexité de l'évaluation de Requêtes

5.1 Définitions

On cherche à évaluer la complexité de l'évaluation de $Q(D)$ où Q est une formule dans un certain langage de requêtage (ici la FOL) et D une base de donnée finie. Pour simplifier l'étude, on va supposer que Q est une requête booléenne.

Proposition 5.1: Complexité

On a :

$$\text{AC}^0 \subsetneq \text{PTIME} \subseteq \text{NP} \subseteq \text{PSPACE}$$

Définition 5.1: Complexité Descriptive

Un langage de requêtes \mathcal{Q} capture une classe de complexité \mathcal{C} si :

- Pour tout $Q \in \mathcal{Q}$, l'évaluation de la requête Q est dans \mathcal{C}
- Pour tout problème $P \in \mathcal{C}$, il existe une requête $Q \in \mathcal{Q}$ dont l'évaluation résout exactement P (sans réduction).

Si \mathcal{Q} capture \mathcal{C} et si \mathcal{C} a des problèmes complets pour \mathcal{C} il existe une requête \mathcal{C} -complète dans \mathcal{Q} , mais la réciproque est fausse.

5.2 Calcul Relationnel

Théorème 5.1: Évaluation FOL

L'évaluation de la FOL est PTIME en la taille des données.

Démonstration. On réécrit en forme normale prénexe et on évalue naïvement. ■

Théorème 5.2: Capture de FO

On ne peut pas calculer dans FOL qu'une relation contenant un ordre total a un nombre pair d'éléments

Théorème 5.3: Évaluation FOL mais Mieux

L'évaluation de la FOL est AC^0 en la taille de la base de données.

Démonstration. On réécrit vers l'algèbre relationnel. ■

Théorème 5.4: Évaluation FOL Combinée

L'évaluation de la FOL est PSPACE-complète en la taille de la requête de la base de données

Démonstration. Le caractère PSPACE-difficile découle de l'existence de QSAT. ■

6 Analyse Statique de Requêtes

6.1 Équivalence et Contenu

On cherche à trouver des requêtes équivalentes à d'autres mais plus rapides à évaluer. On ne considère ici que des optimisations globales, c'est à dire considérant une requête en entier.

Définition 6.1: Équivalence, Minimalité et Contenant

On dit que :

- Une requête q est contenue dans une requête q' , noté $q \sqsubseteq q'$ si pour toute base de données D , $q(D) \subseteq q'(D)$.
- Une requête q est équivalente à une requête q' , noté $q \equiv q'$ si pour toute base de données D , $q(D) = q'(D)$.
- Une requête q' est minimale si q' est la meilleure requête équivalente à $q \in \mathcal{Q}$

6.2 Calcul Relationnel

Définition 6.2: Satisfiabilité

Une requête booléenne du calcul relationnel q est satisfiable s'il existe une base de données (finie) D telle que $D \models q$.

Théorème 6.1: Trakhtenbrot - Satisfiabilité du Calcul Relationnel

La satisfiabilité du calcul relationnel (dans le cas fini) est indécidable et récursivement énumérable.

Démonstration. **Difficulté** Réduction depuis le problème de correspondance de POST.

R-E Énumérer les BDD. ■

Théorème 6.2: Satisfiabilité de la FOL

La satisfiabilité de la FOL sur des modèles infinis est indécidable et co-récursivement énumérable.

Démonstration. **Difficulté** Coder le non-arrêt d'une machine de Turing dans une formule de la FOL

Co-R-E Par théorème de complétude de Gödel, chaque formule valide a une preuve, il suffit de les énumérer. ■

Théorème 6.3: Contenance et Équivalence

La contenance et l'équivalence de requête du calcul relationnel sont indécidables et co-récurсивement énumérable.

Démonstration. L'indécidabilité est par réduction directe de l'indécidabilité de la satisfiabilité. La co-récurсивement énumérabilité est montré directement en énumérant les contre-exemples possibles. ■

Troisième partie

Contraintes et Poursuites

7 Dépendances

7.1 Définitions

Définition 7.1: Dépendance Fonctionnelle

Une dépendance fonctionnelle σ sur une table R est une formule de la forme $A \rightarrow B$ où $A, B \subseteq \text{attr}(R)$.
 σ est vérifiée par une instance de R si

$$\forall t, t' \in R, t[A] = t'[A] \implies t[B] = t'[B]$$

Définition 7.2: Dépendance Multivariée

Une dépendance multivariée $\sigma = A \twoheadrightarrow B$ sur R est vérifiée par une instance si :

$$\forall t, t' \in R, t[A] = t'[A] \implies (\exists t'', t''[B] = t[B] \wedge t''[A] = t[A] \wedge t''[C] = t[C])$$

où $C = R \setminus A \cup B$

7.2 Implication

Définition 7.3: Implication

Si on a un ensemble Σ de dépendances sur une table R et une dépendance σ sur cette table Σ, σ , on dit que $\Sigma \models \sigma$ si toute instance de R vérifiant Σ alors l'instance vérifie σ .

On vérifie une implication en vérifiant si une table minimale vérifiant les dépendances de Σ vérifie σ . On part de deux tuples et on ajoute des éléments pour vérifier les dépendances de Σ .

Définition 7.4: Tuple Generated Dependency

Une TGD vérifie :

$$\forall \bar{x}, \forall \bar{y}, (B(\bar{x}, \bar{y}) \rightarrow \exists \bar{z} H(\bar{y}, \bar{z}))$$

avec B et H des conjonctions d'atomes.

C'est une généralisation des dépendances multivariées.

Définition 7.5: EGD

Un EGD qui vérifie :

$$\forall \bar{x}, y_1, y_2, B(\bar{x}, y_1, y_2) \rightarrow y_1 = y_2$$

avec B une conjonction d'atomes.