Bases de Données

Pierre Senellart

7 février 2024

Table des matières

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

Indépendence 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 standardisé, 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 fonctionnalitées 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 aveec des garanties sur le controle 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 transction est soit éxécuté 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 meilleurs 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 $(A_1, \ldots, A_n)$ 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 $((L_1, \tau_1), \ldots, (L_n, \tau_n))$ (aussi appelé une relation sur ce schéma) est un ensemble fini $\{t_1, \ldots, t_k\}$ de tuples de la forme $t_j = (v_{j,1}, \ldots, v_{j,n})$ 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 ambigüe 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, $t[A]$ 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 référencent 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 abstractise 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	$R \in \mathcal{L}$
$\rho_{A \to B}$	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	
$\bowtie_\varphi$	2	Jointure	φ formule

Relation Expression : `Guest`

Résultat :

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

Renommage Expression : $\rho_{\texttt{id} \to \texttt{guest}}(\texttt{Guest})$

Résultat :

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

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

Résultat :

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

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

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 : $\Pi_{\texttt{id}}(\texttt{Guest}) \times \Pi_{\texttt{name}}(\texttt{Guest})$

Résultat :

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

Union Expression : $\Pi_{\texttt{room}}\left(\sigma_{\texttt{guest=2}}\left(\texttt{Reservation}\right)\right) \cup \Pi_{\texttt{room}}\left(\sigma_{\texttt{arrival=2017-01-15}}\left(\texttt{Reservation}\right)\right)$

Résultat :

room
107
302
504

Cet union aurait pû être écrite

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

Différence Expression : $\Pi_{\texttt{room}}\left(\sigma_{\texttt{guest = 2}}\left(\texttt{Reservation}\right)\right) \setminus \Pi_{\texttt{room}}\left(\sigma_{\texttt{arrival=2017-01-15}}\left(\texttt{Reservation}\right)\right)$

Résultat :

room
107

Cette différence simple aurait pû être écrite

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

Jointure Expression : $\texttt{Reservation} \bowtie_{\texttt{guest=id}} \texttt{Guest}$ 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'attibuts 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 $\bigwedge_{A\in\mathcal{A}\cap\mathcal{B}} A = A$.