Introduction à la gestion de données

Cours L3 Bases de Données

Pierre Senellart

12 février 2020

Plan

Gestion de données

De nombreuses applications (logiciels, sites Web, etc.) ont besoin de gérer des données :

- Structurer les données utiles à l'application
- Permettre de les conserver de manière persistante (données conservées même quand l'application ne tourne pas)
- Rechercher efficacement des informations dans de grands volumes de données
- Mise à jour des données sans violer certaines contraintes de structure
- Permettre l'accès et la mise à jour de données à des utilisateurs multiples, parfois de manière concurrente

Souvent, souhaitable d'accéder aux mêmes données depuis plusieurs applications distinctes, depuis des ordinateurs distincts.

Exemple : Système d'information d'un hôtel

Accès depuis un logiciel maison (réception), un site Web (clients), un logiciel de comptabilité. Besoins :

- Données structurées représentant les chambres, les clients, les réservations, les tarifs, etc.
- Pas de données perdues quand ces applications ne sont pas utilisées, ou quand une panne de courant générale survient
- Retrouver quasi-instantanément quelles chambres sont réservées, et par qui, un jour donné, dans un historique de plusieurs années de réservation
- Facilement ajouter une réservation en s'assurant de ne pas réserver deux fois la même chambre le même jour
- Le client, le réceptionniste, le comptable, ne doivent pas la même vue des données (confidentialité, simplicité, etc.) ; différents clients ne peuvent réserver la même chambre au même moment.

Implémentation naïve (1/2)

- Implémentation dans un langage de programmation classique (C++, Java, Python, etc.) de structures de données permettant de représenter l'ensemble des données utiles
- Définition de formats de fichiers ad hoc pour stocker les données sur disques, avec synchronisation régulière et mécanisme de récupération en cas de panne
- Stockage en mémoire des données de l'application, avec structures de données (ABR, tables de hachage) et algorithmes (recherche, tri, agrégation, parcours de graphe...) permettant de retrouver efficacement l'information
- Fonctions de mises à jour des données, avec du code vérifiant au passage qu'aucune contrainte n'est violée

Implémentation naïve (2/2)

- Définition au sein du logiciel de droits utilisateurs différents, d'un mécanisme d'authentification ; utilisation de threads pour répondre en même temps à différentes requêtes, verrous/sémaphores sur les fonctions critiques de manipulation de données
- Définition et implémentation d'un protocole de communication réseau pour connecter ce composant logiciel à un serveur Web, un logiciel Windows, une suite logicielle de comptabilité, etc.

Implémentation naïve (2/2)

- Définition au sein du logiciel de droits utilisateurs différents, d'un mécanisme d'authentification ; utilisation de threads pour répondre en même temps à différentes requêtes, verrous/sémaphores sur les fonctions critiques de manipulation de données
- Définition et implémentation d'un protocole de communication réseau pour connecter ce composant logiciel à un serveur Web, un logiciel Windows, une suite logicielle de comptabilité, etc.

Beaucoup de travail !

Implémentation naïve (2/2)

- Définition au sein du logiciel de droits utilisateurs différents, d'un mécanisme d'authentification ; utilisation de threads pour répondre en même temps à différentes requêtes, verrous/sémaphores sur les fonctions critiques de manipulation de données
- Définition et implémentation d'un protocole de communication réseau pour connecter ce composant logiciel à un serveur Web, un logiciel Windows, une suite logicielle de comptabilité, etc.

Beaucoup de travail ! Demande un programmeur qui maîtrise la POO, la sérialisation, la reprise sur panne, les structures de données, l'algorithmique, la gestion de contraintes d'intégrité, la gestion de rôles, la programmation parallèle, la gestion de concurrence, la programmation réseau...

Implémentation naïve (2/2)

- Définition au sein du logiciel de droits utilisateurs différents, d'un mécanisme d'authentification ; utilisation de threads pour répondre en même temps à différentes requêtes, verrous/sémaphores sur les fonctions critiques de manipulation de données
- Définition et implémentation d'un protocole de communication réseau pour connecter ce composant logiciel à un serveur Web, un logiciel Windows, une suite logicielle de comptabilité, etc.

Beaucoup de travail ! Demande un programmeur qui maîtrise la POO, la sérialisation, la reprise sur panne, les structures de données, l'algorithmique, la gestion de contraintes d'intégrité, la gestion de rôles, la programmation parallèle, la gestion de concurrence, la programmation réseau... À refaire pour chaque nouvelle application ayant à gérer des données !

Rôle d'un SGBD/DBMS

Système de Gestion de Base de Données (Database Management System)

Logiciel simplifiant la conception d'applications manipulant des données, en fournissant un accès unifié à l'ensemble des fonctionnalités nécessaires à la gestion de données, quelle que soit l'application

Base de données

Ensemble de données (propre à une application spécifique) géré par un SGBD

Fonctionnalités d'un SGBD (1/2)

Indépendance physique. L'utilisateur d'un SGBD n'a pas besoin de savoir comment les données sont stockées (dans un fichier, sur une partition brute, sur un système de fichiers distribués) ; le stockage peut être modifié sans impacter l'accès aux données

Indépendance logique. Il est possible de fournir à l'utilisateur une vue partielle des données, correspondant à ce à quoi il a besoin et droit d'accéder

Accès aisé aux données. Utilisation d'un langage déclaratif décrivant les requêtes et mises à jour sur les données, qui décrit l'intention de l'utilisateur plutôt que la manière dont ce sera implémenté

Optimisation des requêtes. Les requêtes sont automatiquement optimisées pour être implémentées le plus efficacement possible sur la base de données

Fonctionnalités d'un SGBD (2/2)

Intégrité logique. Le SGBD impose des contraintes sur la structure des données ; toute modification violant ces contraintes est refusée

Intégrité physique. La base reste dans un état cohérent et les données sont préservées de manière durable, même en cas de panne

Partage de données. Les données sont accessibles à des utilisateurs multiples, de manière concurrente, sans que ces accès multiples et concurrent ne violent l'intégrité physique et logique des données

Normalisation. L'utilisation du SGBD est standardisée, de sorte qu'il est possible de remplacer un SGBD par un autre sans changer (de manière majeure) le code de l'application

Variété des SGBD

- Dizaines de SGBD existant utilisés de manière non négligeables
- Tous les SGBD ne fournissent pas toutes ces fonctionnalités...
- Mais ceux que nous allons étudier pendant ce cours les fournissent toutes
- Les SGBD se distinguent les uns des autres par le modèle de données utilisé, les compromis réalisés en terme de performances et de fonctionnalités, la facilité d'utilisation, les volumes de données gérés, l'architecture interne, etc.

Grands types de SGBD

Relationnels (SGBDR). Tables, requêtes complexes (SQL), fonctionnalités riches

XML. Arbres, requêtes complexes (XQuery), fonctionnalités similaires aux SGBDR

Graphes/Triplets. Données graphes, requêtes complexes exprimant des parcours de graphe

Objets. Modèle de données très complexe, inspiré de la POO

Documents. Données complexes, organisées en documents, requêtes et fonctionnalités relativement simples

Clef-Valeur. Modèle de données très basique, accent mis sur la performance

Orientés Colonnes. Modèle de données entre celui des clef-valeur et des SGBDR ; accent mis sur le parcours ou l'agrégation efficace de colonnes

Grands types de SGBD

Relationnels (SGBDR). Tables, requêtes complexes (SQL), fonctionnalités riches

NoSQL

- **XML.** Arbres, requêtes complexes (XQuery), fonctionnalités similaires aux SGBDR
- **Graphes/Triplets.** Données graphes, requêtes complexes exprimant des parcours de graphe
- **Objets.** Modèle de données très complexe, inspiré de la POO
- **Documents.** Données complexes, organisées en documents, requêtes et fonctionnalités relativement simples
- **Clef-Valeur.** Modèle de données très basique, accent mis sur la performance
- **Orientés Colonnes.** Modèle de données entre celui des clef-valeur et des SGBDR ; accent mis sur le parcours ou l'agrégation efficace de colonnes

SGBD relationnels classiques

- Basés sur le modèle relationnel : décomposition des données en relations, ou tables
- Un langage de requêtes standard : SQL
- Données stockées sur disque
- Relations (tables) stockées ligne par ligne
- Système centralisé, avec possibilités limitées de distribution

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 :

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

NewSQL

- Certaines applications nécessitent :
  - un langage de requêtes riches (SQL)
  - une conformité aux propriétés ACID
  - mais des performances supérieures à celles des SGBD classiques

NewSQL

- Certaines applications nécessitent :
  - un langage de requêtes riches (SQL)
  - 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

Google Spanner Clustrix VOLTDB

Dans ce cours

- Principalement techniques des SBGD relationnels classiques (les plus utilisés et mûrs, de très loin !)
- . . . et la riche théorie sous-jacente

Plan

Schéma relationnel

On fixe des ensembles (généralement infinis dénombrables) :

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

Définition

Un schéma de relation (d'arité n) est un n-uplet $(A_1, \ldots, A_n)$ où chaque A_i (appelé attribut) est un couple (L_i, τ_i) avec $L_i \in \mathcal{L}$, $\tau_i \in \mathcal{T}$ et tels que les A_i sont 2 à 2 distincts.

Définition

Un schéma relationnel est défini par un ensemble fini d'étiquettes $L \subseteq \mathcal{L}$ (les noms de relation) chaque étiquette de L étant associée à un schéma de relation.

Exemple de schéma relationnel

- Univers :
  - $\mathcal{L}$ est l'ensemble des chaînes de caractères alphanumériques commençant par une lettre
  - $\mathcal{V}$ est l'ensemble des suites finies de bits
  - $\mathcal{T}$ est formé de types tels que `INTEGER` (représentations sous formes de bits de nombres entiers entre -2^{31} et $2^{31}-1$), `REAL` (représentations de nombres en virgule flottante simple précision IEEE 754), `TEXT` (représentation en UTF-8 de chaînes de caractères), `DATE` (représentation d'une date au format ISO8601), etc.
- Schéma relationnel formé de 2 noms de relations, `Client` et `Reservation`
- `Client` : ((`id`, `INTEGER`), (`nom`, `TEXT`), (`email`, `TEXT`))
- `Reservation` :
  ((`id`, `INTEGER`), (`client`, `INTEGER`), (`chambre`, `INTEGER`), (`arrivee`, `DATE`), (`nuits`, `INTEGER`))

Base de données

Définition

Une instance d'un schéma de relation $((L_1, \tau_1), \ldots, (L_n, \tau_n))$ (on parle aussi d'une relation sur ce schéma) est un ensemble fini $\{t_1, \ldots, t_k\}$ de n-uplets de la forme $t_j = (v_{j1}, \ldots, v_{jn})$ avec $\forall j \forall i \; v_{ji} \in \tau_i$.

Définition

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

Note : On emploie relation de manière ambiguë soit pour un schéma de relation, soit pour une instance d'un schéma de relation.

Exemple

Client

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Reservation

id	client	chambre	arrivee	nuits
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

Quelques 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 du i-ème composant de t.
- De même, si $\mathcal{A}$ est un k-uplet d'attributs apparaissant parmi les n attributs de R, $t[\mathcal{A}]$ est le k-uplet formé à partir de t en concaténant les $t[A]$ pour A dans $\mathcal{A}$.
- Un tuple est un n-uplet pour un certain n.

Contraintes d'intégrité simples

On peut ajouter au schéma relationnel des contraintes d'intégrité, de différentes natures, pour définir une notion de validité d'instance.

Clef. Un tuple d'attributs $\mathcal{A}$ d'un schéma de relation R est une clef s'il ne peut exister deux tuples distincts t_1 et t_2 dans une instance de R avec $t_1[\mathcal{A}] = t_2[\mathcal{A}]$

Clef étrangère. Un k-uplet d'attributs $\mathcal{A}$ d'un schéma de relation R est une clef étrangère référençant un k-uplet d'attributs $\mathcal{B}$ d'une relation S si pour toute 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 $t[\mathcal{A}] = t'[\mathcal{B}]$

Contrainte Check. Condition arbitraire sur les valeurs des attributs d'une relation (s'appliquant à chacun des tuples des instances de cette relation)

Exemples de contraintes

- `id` est une clef de `Client`
- `email` est une clef de `Client`
- `id` est une clef de `Reservation`
- (`chambre`, `arrivee`) est une clef de `Reservation`
- (`client`, `arrivee`) est une clef de `Reservation` (?)
- `client` est une clef étrangère de `Reservation` référençant `id` de `Client`
- Dans `Client`, `email` doit contenir un « @ »
- Dans `Reservation`, `chambre` doit être entre 1 et 650
- Dans `Reservation`, `nuits` doit être positif

Impossible d'exprimer des contraintes plus complexes (p. ex., une chambre ne peut pas être occupée deux fois la même nuit, ce qui dépend de la date et du nombre de nuits pour chaque chambre)

Variantes : perspectives nommées et non-nommées

La version présentée considère que les attributs d'une relation sont ordonnés, et ont un nom. C'est ce qui correspond le mieux à ce que font les SGBDR, mais ce n'est pas forcément le plus agréable pour raisonner sur le modèle relationnel.

Perspective nommée. On oublie la position des attributs, on considère qu'ils sont identifiés uniquement par leur nom.

Perspective non-nommée. On oublie le nom des attributs, on considère qu'ils sont identifiés uniquement par leur position. On utilise des notations telles que $t(2)$ pour accéder à la valeur du deuxième attribut d'un tuple.

Pas d'impact majeur, on utilisera l'une ou l'autre perspective suivant ce qui est le plus agréable.

Variante : sémantique multi-ensembliste

- On a défini une instance de relation comme un ensemble fini de tuples. On peut aussi considérer une sémantique multi-ensembliste du modèle relationnel, où une instance de relation est un multi-ensemble fini de tuples.
- C'est ce qui correspond le mieux à ce que font les SGBDR...
- ... mais la plupart de la théorie des bases de données relationnelles est fait pour la sémantique ensembliste, plus agréable à manipuler
- On utilisera principalement la sémantique ensembliste dans les résultats théoriques
- Les SGBDR implémentent une sémantique multi-ensembliste, mais on peut forcer une sémantique ensembliste avec le mot-clef **DISTINCT**

Variante : version non typée

- Dans les implémentations, les attributs sont toujours typés
- Dans les modèles et résultats théoriques, on fait souvent abstraction du type des attributs et on considère que chaque attribut a un type universel $\mathcal{V}$
- On omettra donc parfois le type des attributs

Plan

Algèbre relationnelle

- Langage algébrique permettant de manipuler des relations
- Une expression de l'algèbre relationnelle produit une nouvelle relation à partir des relations de la base de données
- Chaque opérateur prend 0, 1, ou 2 sous-expressions
- Principaux opérateurs :

Op.	Arité	Description	Condition
R	0	Nom de relation	$R \in \mathcal{L}$
$\rho_{A \to B}$	1	Renommage	$A, B \in \mathcal{L}$
$\Pi_{A_1 \ldots A_n}$	1	Projection	$A_1 \ldots A_n \in \mathcal{L}$
σ_φ	1	Sélection	φ formule
$\times$	2	Produit cartésien	
$\cup$	2	Union	
$\setminus$	2	Différence	
$\bowtie_\varphi$	2	Jointure	φ formule

Nom de relation

Client

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Reservation

id	client	chambre	arrivee	nuits
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

Expression : `Client`

Résultat :

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Renommage

Client

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Reservation

id	client	chambre	arrivee	nuits
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

Expression : $\rho_{\texttt{id}\to\texttt{client}}(\texttt{Client})$

Résultat :

client	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Projection

Client

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Reservation

id	client	chambre	arrivee	nuits
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

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

Résultat :

email	id
jean.dupont@gmail.com	1
alice@dupuis.name	2
jean.dupont@ens.fr	3

Sélection

Client

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Reservation

id	client	chambre	arrivee	nuits
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

Expression : $\sigma_{\texttt{arrivee}>\texttt{2017-01-12}\wedge\texttt{client}=\texttt{2}}(\texttt{Reservation})$

Résultat :

id	client	chambre	arrivee	nuits
4	2	504	2017-01-15	2
5	2	107	2017-01-30	1

La formule utilisée dans la sélection peut être n'importe quelle combinaison booléenne de comparaisons des valeurs d'attributs entre elles et avec des constantes.

Produit cartésien

Client

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Reservation

id	client	chambre	arrivee	nuits
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

Expression : $\Pi_{\texttt{id}}(\texttt{Client}) \times \Pi_{\texttt{nom}}(\texttt{Client})$

Résultat :

id	nom
1	Alice Dupuis
2	Alice Dupuis
3	Alice Dupuis
1	Jean Dupont
2	Jean Dupont
3	Jean Dupont

Union

Client

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Reservation

id	client	chambre	arrivee	nuits
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

Expression :

$$\Pi_{\texttt{chambre}}(\sigma_{\texttt{client}=2}(\texttt{Reservation})) \cup \Pi_{\texttt{chambre}}(\sigma_{\texttt{arrivee}=\text{2017-01-15}}(\texttt{Reservation}))$$

Résultat :

chambre
107
302
504

Union

Client

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Reservation

id	client	chambre	arrivee	nuits
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

Expression : $\Pi_{\texttt{chambre}}(\sigma_{\texttt{client}=2}(\texttt{Reservation})) \cup \Pi_{\texttt{chambre}}(\sigma_{\texttt{arrivee}=\text{2017-01-15}}(\texttt{Reservation}))$

Résultat :

chambre
107
302
504

Cette simple union aurait pu être écrite $\Pi_{\texttt{chambre}}(\sigma_{\texttt{client}=2 \vee \texttt{arrivee}=\text{2017-01-15}}(\texttt{Reservation}))$.

Pas toujours possible.

Différence

Client

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Reservation

id	client	chambre	arrivee	nuits
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

Expression : $\Pi_{\texttt{chambre}}(\sigma_{\texttt{client}=2}(\texttt{Reservation})) \setminus \Pi_{\texttt{chambre}}(\sigma_{\texttt{arrivee}=\text{2017-01-15}}(\texttt{Reservation}))$

Résultat :

chambre
107

Différence

Client

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Reservation

id	client	chambre	arrivee	nuits
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

Expression : $\Pi_{\texttt{chambre}}(\sigma_{\texttt{client}=2}(\texttt{Reservation})) \setminus \Pi_{\texttt{chambre}}(\sigma_{\texttt{arrivee}=\text{2017-01-15}}(\texttt{Reservation}))$

Résultat :

chambre
107

Cette simple différence aurait pu être écrite $\Pi_{\texttt{chambre}}(\sigma_{\texttt{client}=2 \wedge \texttt{arrivee} \neq \text{2017-01-15}}(\texttt{Reservation}))$.
Pas toujours possible.

Jointure

Client

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Reservation

id	client	chambre	arrivee	nuits
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

Expression : $\texttt{Reservation} \bowtie_{\texttt{client}=\texttt{id}} \texttt{Client}$

Résultat :

id	client	chambre	arrivee	nuits	nom	email
1	1	504	2017-01-01	5	Jean Dupont	jean.dupont@gmail.com
2	2	107	2017-01-10	3	Alice Dupuis	alice@dupuis.name
3	3	302	2017-01-15	6	Jean Dupont	jean.dupont@ens.fr
4	2	504	2017-01-15	2	Alice Dupuis	alice@dupuis.name
5	2	107	2017-01-30	1	Alice Dupuis	alice@dupuis.name

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

Note sur la jointure

- La jointure n'est pas une opération élémentaire de l'algèbre relationnelle (même si très utile)
- Elle peut être vue comme une combinaison de renommage, produit cartésien, sélection, projection
- Ainsi :

$$\begin{aligned} & \texttt{Reservation} \bowtie_{\texttt{client}=\texttt{id}} \texttt{Client} \\ \equiv\ & \Pi_{\texttt{id},\texttt{client},\texttt{chambre},\texttt{arrivee},\texttt{nuits},\texttt{nom},\texttt{email}}(\\ & \quad \sigma_{\texttt{client}=\texttt{temp}}(\texttt{Reservation} \times \rho_{\texttt{id}\to\texttt{temp}}(\texttt{Client}))) \end{aligned}$$

- (On note $\equiv$ pour indiquer que deux expressions de l'algèbre relationnelle sont équivalentes : elles donnent le même résultat sur toute base de données)
- Si R et S ont pour attributs $\mathcal{A}$ et $\mathcal{B}$, on note $R \bowtie S$ pour la jointure naturelle de R et de S, où la formule de jointure est $\bigwedge_{A \in \mathcal{A} \cap \mathcal{B}} A = A$.

Opérations illégales

- Toutes les expressions de l'algèbre relationnelle ne sont pas valides
- La validité d'une expression dépend en général du schéma relationnel
- Par exemple :
  - On ne peut pas faire référence à un nom de relation qui n'existe pas dans le schéma relationnel
  - On ne peut pas faire référence (dans un renommage, projection, sélection, jointure) à un attribut qui n'existe pas dans le résultat de la sous-expression
  - On ne peut pas unir deux relations avec des attributs différents
  - On ne peut pas produire (produit cartésien, jointure, renommage) une table avec deux attributs de même nom
- Les systèmes implémentant l'algèbre relationnelle peuvent faire de la vérification statique ou dynamique de ces règles, ou les ignorent parfois

Sémantique multi-ensembliste

En sémantique multi-ensembliste (ce qui est vraiment utilisée par les SGBDR) :

- Toutes les opérations renvoient des multi-ensembles
- En particulier, projection et union peuvent introduire des multi-ensembles même quand les relations initiales sont des ensembles

Extension : Agrégation

- Des extensions variées ont été proposées à l'algèbre relationnelle pour capturer des fonctionnalités supplémentaires
- En particulier, agrégation et regroupement [Klug, 1982, Libkin, 2003] des résultats
- Avec une syntaxe inspirée de [Libkin, 2003] :

$$\sigma_{\texttt{avg}>3}(\gamma^{\texttt{avg}}_{\texttt{chambre}}[\lambda x.\texttt{avg}(x)](\Pi_{\texttt{chambre},\texttt{nuits}}(\texttt{Reservation})))$$

calcule le nombre moyen de nuits par réservation pour chaque chambre ayant une moyenne supérieure à 3

chambre	avg
302	6
504	3,5

Plan

SQL

- Structured Query Language, langage normalisé (ISO/IEC 9075, plusieurs versions [ISO, 1987, 1999]) pour interagir avec un SGBDR
- Malheureusement, implémentation du standard très variable d'un SGBDR à l'autre
- Beaucoup de petites choses (p. ex., types disponibles) varient d'un SGBDR à l'autre plutôt que de suivre le standard
- Différences plus syntaxiques que fondamentales
- Où cela fait une différence, la version PostgreSQL est présentée
- Deux parties principales : DDL (Data Definition Language) pour définir le schéma et DML (Data Manipulation Language) pour interroger et mettre à jour les données
- Langage déclaratif : on décrit ce que l'on veut, on laisse au système le soin de transformer en un plan d'exécution

Syntaxe de SQL

- Assez verbeux, conçu pour être presque lisible comme de l'anglais [Chamberlin and Boyce, 1974]
- Mots-clefs insensibles à la casse, conventionnellement écrits en majuscule
- Identifiants souvent insensibles à la casse (dépend du SGBDR), souvent avec une majuscule initiale pour les noms de table, en minuscule pour les noms d'attributs
- Commentaires introduits par --
- Ordres SQL terminés par un « ; » dans certains contextes mais le « ; » ne fait pas strictement partie de l'ordre SQL

NULL

- En SQL, NULL est une valeur spéciale que peut prendre un attribut dans un tuple
- Dénote l'absence de valeur
- Différent de 0, de la chaîne vide, etc.
- Logique tri-valuée bizarre : Vrai, Faux, NULL
- Une comparaison normale (égalité, non-égalité...) avec NULL renvoie toujours NULL
- **IS NULL**, **IS NOT NULL** peuvent être utilisées pour tester si une valeur est NULL
- NULL est ultimement converti en Faux
- Conséquences bizarres, s'intègre mal au modèle relationnel formel

Data Definition Language

```
CREATE TABLE Client(id INTEGER, nom TEXT, email TEXT);
CREATE TABLE Reservation(id INTEGER, client INTEGER,
  chambre INTEGER, arrivee DATE, nuits INTEGER);
```

Mais aussi :

- `DROP TABLE Client;` pour supprimer une table
- `ALTER TABLE Client RENAME TO Client2;` pour renommer une table
- `ALTER TABLE Client ALTER COLUMN id TYPE TEXT;` pour changer le type d'une colonne

Contraintes (1/2)

Spécifiées à la création de la table, ou ajoutées ensuite (avec **ALTER TABLE**)

PRIMARY KEY pour la clef primaire ; une seule par table, c'est une clef qui sera aussi utilisée pour l'organisation physique des données ; implique **NOT NULL**

UNIQUE pour les autres clefs

REFERENCES pour les clefs étrangères

CHECK pour les contraintes Check

NOT NULL pour indiquer que les valeurs ne peuvent être NULL

Contraintes (2/2)

```
CREATE TABLE Client(
  id INTEGER PRIMARY KEY,
  nom TEXT NOT NULL,
  email TEXT UNIQUE CHECK (email LIKE '%@%')
);

CREATE TABLE Reservation(
  id INTEGER PRIMARY KEY,
  client INTEGER NOT NULL REFERENCES Client(id),
  chambre INTEGER NOT NULL CHECK (chambre>0
    AND chambre<651),
  arrivee DATE NOT NULL,
  nuits INTEGER NOT NULL CHECK (nuits>0),
  UNIQUE(chambre, arrivee),
  UNIQUE(client, arrivee)
);
```

Mises à jours

- Insertions :

```
INSERT INTO Client(id,nom) VALUES (5,'John');
```

- Suppressions :

```
DELETE FROM Reservation WHERE id>4;
```

- Modifications :

```
UPDATE Reservation
  SET chambre=205
  WHERE chambre=204;
```

Insertions de plusieurs valeurs

```
INSERT INTO Client VALUES
  (1,'Jean Dupont','jean.dupont@gmail.com'),
  (2,'Alice Dupuis','alice@dupuis.name'),
  (3,'Jean Dupont','jean.dupont@ens.fr');

INSERT INTO Reservation VALUES
  (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);
```

Requêtes

Forme générale suivante :

SELECT ... **FROM** ... **WHERE** ...
GROUP BY ... **HAVING** ...
UNION SELECT ... **FROM** ...

SELECT projection, renommage, agrégation

FROM produit cartésien

WHERE sélection (optionnel)

GROUP BY regroupement (optionnel)

HAVING sélection sur le regroupement (optionnel)

UNION union (optionnel)

D'autres mots clefs : ORDER BY pour réordonner, LIMIT pour limiter aux k premiers résultats, DISTINCT pour forcer une sémantique ensembliste, EXCEPT pour la différence...

Renommage

Client

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Reservation

id	client	chambre	arrivee	nuits
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

$$\rho_{\texttt{id}\to\texttt{client}}(\texttt{Client})$$

```
SELECT id AS client, nom, email
FROM Client;
```

Projection

Client

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Reservation

id	client	chambre	arrivee	nuits
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

$$\Pi_{\texttt{email},\texttt{id}}(\texttt{Client})$$

```
SELECT DISTINCT email, id
FROM Client;
```

Sélection

Client

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Reservation

id	client	chambre	arrivee	nuits
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

$$\sigma_{\texttt{arrivee}>\texttt{2017-01-12}\wedge\texttt{client}=2}(\texttt{Reservation})$$

```
SELECT *
FROM Reservation
WHERE arrivee>'2017-01-12' AND client=2;
```

Produit cartésien

Client

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Reservation

id	client	chambre	arrivee	nuits
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

$$\Pi_{\texttt{id}}(\texttt{Client}) \times \Pi_{\texttt{nom}}(\texttt{Client})$$

```
SELECT *
FROM
  (SELECT DISTINCT id FROM Client) AS temp1,
  (SELECT DISTINCT nom FROM Client) AS temp2
ORDER BY nom, id;
```

Union

Client

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Reservation

id	client	chambre	arrivee	nuits
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

$$\Pi_{\texttt{chambre}}(\sigma_{\texttt{client}=2}(\texttt{Reservation}))$$
$$\cup\, \Pi_{\texttt{chambre}}(\sigma_{\texttt{arrivee}=\texttt{2017-01-15}}(\texttt{Reservation}))$$

```
SELECT chambre
FROM Reservation
WHERE client=2
UNION
SELECT chambre
FROM Reservation
WHERE arrivee='2017-01-15';
```

Différence

Client

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Reservation

id	client	chambre	arrivee	nuits
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

$$\Pi_{\texttt{chambre}}(\sigma_{\texttt{client}=2}(\texttt{Reservation}))$$
$$\setminus \Pi_{\texttt{chambre}}(\sigma_{\texttt{arrivee}=\texttt{2017-01-15}}(\texttt{Reservation}))$$

```
  SELECT chambre
  FROM Reservation
  WHERE client=2
EXCEPT
  SELECT chambre
  FROM Reservation
  WHERE arrivee='2017-01-15';
```

Jointure

Client

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Reservation

id	client	chambre	arrivee	nuits
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

$$\texttt{Reservation} \bowtie_{\texttt{client}=\texttt{id}} \texttt{Client}$$

```
SELECT Reservation.*, nom, email
FROM Reservation JOIN Client ON client=Client.id;
```

```
SELECT Reservation.*, nom, email
FROM Reservation, Client
WHERE client=Client.id;
```

Agrégation

Client

id	nom	email
1	Jean Dupont	jean.dupont@gmail.com
2	Alice Dupuis	alice@dupuis.name
3	Jean Dupont	jean.dupont@ens.fr

Reservation

id	client	chambre	arrivee	nuits
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

$$\sigma_{\mathtt{avg}>3}(\gamma^{\mathtt{avg}}_{\mathtt{chambre}}[\lambda x.\mathtt{avg}(x)](\Pi_{\mathtt{chambre},\mathtt{nuits}}(\mathtt{Reservation})))$$

```
SELECT chambre, AVG(nuits) AS avg
FROM Reservation
GROUP BY chambre
HAVING AVG(nuits)>3
ORDER BY chambre;
```

Plan

Références

- Généralités sur la gestion de données [Benedikt and Senellart, 2012, Abiteboul, 2012]
- Cours sur le programme de prépa en bases de données [Abiteboul et al., 2014]
- Modèle relationnel, algèbre relationnelle : chapitres 3 et 4 de [Abiteboul et al., 1995]
- Détails de SQL : les standards sont non-publics et peu utiles à l'utilisateur final ; consulter la documentation du SGBDR
- Pour PostgreSQL, `https://www.postgresql.org/docs/` et `\h` dans le client de ligne de commande

Bibliographie I

Serge Abiteboul. *Sciences des données : de la logique du premier ordre à la Toile*. Collège de France, 2012. `http://books.openedition.org/cdf/529`.

Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

Serge Abiteboul, Benjamin Nguyen, and Yannick Le Bras. Introduction aux bases de données relationnelles. `http://abiteboul.com/Lili/bdrelationnelles.pdf`, 2014.

Michael Benedikt and Pierre Senellart. Databases. In Edward K. Blum and Alfred V. Aho, editors, *Computer Science. The Hardware, Software and Heart of It*, pages 169–229. Springer-Verlag, 2012.

Donald D. Chamberlin and Raymond F. Boyce. SEQUEL : A structured english query language. In *Proc. SIGFIDET/SIGMOD Workshop*, volume 1, 1974.

Bibliographie II

ISO. *ISO 9075:1987: SQL*. International Standards Organization, 1987.

ISO. *ISO 9075:1999: SQL*. International Standards Organization, 1999.

Anthony C. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *J. ACM*, 29(3) :699–717, 1982.

Leonid Libkin. Expressive power of SQL. *Theor. Comput. Sci.*, 296(3) :379–404, 2003.