

# Bases de données

Pierre Senellart, Michaël Thomazo, Paul Boniol, Serge Abiteboul  
Notes par Antoine Groudiev

Version du 7 février 2024

## Table des matières

<b>1</b>	<b>Gestion de données, modèle relationnel, algèbre et SQL</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.1.1	Gestion de données . . . . .	1
1.1.2	Types de SGBD . . . . .	2
1.2	Modèle relationnel . . . . .	3
1.3	Langages de requêtes . . . . .	4
1.3.1	Algèbre relationnelle . . . . .	4
1.3.2	SQL . . . . .	5

## Introduction

Ce document est l'ensemble non officiel des notes du cours *Bases de données* du Département Informatique de l'ENS Ulm. Elles sont librement inspirées des notes de cours sous forme de présentation rédigées par Pierre Senellart.

## 1 Gestion de données, modèle relationnel, algèbre et SQL

### 1.1 Introduction

#### 1.1.1 Gestion de données

Les bases de données ont de nombreuses applications (logiciels seuls, sites Web, ...), et servent à gérer des données. En particulier, elles sont utilisées pour :

- structurer des données utiles aux applications
- les conserver de manière persistente, même quand l'application ne tourne pas
- lancer des requêtes efficaces sur un large volume de données
- mettre à jour des données sans violer les contraintes structurelles de la base
- autoriser plusieurs utilisateurs à accéder et modifier les données, possiblement simultanément

Il est par ailleurs souvent désirable d'accéder aux mêmes données depuis des applications distinctes, sur des ordinateurs différents.

On peut considérer comme exemple le système d'information d'un hôtel. Il est accessible depuis un logiciel interne (bureau d'accueil), un site web (hôtes), et une suite de logiciels de comptabilité. Il nécessite des données structurées pour les chambres, clients, réservations, ... Les

données doivent rester stockées de manière pérenne, même en cas de coupure de courant. Les requêtes doivent être quasi-instantanées, même avec un long historique de l'hôtel.

L'implémentation naïve nécessite un format de fichier pour stocker les données sur un disque, et un mécanisme de synchronisation, et de récupération en cas d'interruption de la synchronisation. Elle nécessite beaucoup de compétences différentes, de la programmation orientée objet, à des notions de réseau, de maîtrise de structures de données et algorithmes efficaces, de gestion en parallèle, ... Ré-implémenter une telle structure à chaque fois qu'un tel outil est nécessaire est une perte de temps.

On utilise donc souvent un *Système de Gestion de Bases de Données* (SGBD), qui est un logiciel simplifiant le design d'une application manipulant des données, en fournissant un accès unifié aux fonctionnalités nécessaires pour la gestion des données, et ce peu importe l'application. Une *base de donnée* désigne une collection de données, gérée par un SGBD.

Un SGBD contient plusieurs fonctionnalités et caractéristiques :

- Indépendance physique
- Indépendance logique
- Facilité de l'accès aux données
- Optimisation des requêtes
- Intégrité logique
- Intégrité physique
- Partage de données
- Standardisation

### 1.1.2 Types de SGBD

De nombreux SGBD sont largement utilisés et ils ne fournissent pas tous toutes ces fonctionnalités. Les SGBDS peuvent être distingués selon le modèle de données utilisé, les choix techniques pris, la facilité d'utilisation, la possibilité de mise à l'échelle, ou encore l'architecture interne.

Les types de SGBD les plus courants sont les suivants :

- Relationnel : les données sont stockées sous forme de table ; ce modèle permet des requêtes complexes, utilisant le langage SQL
- XML : les données sont stockées sous forme d'arbre
- Graphes/Triplets
- Objets, inspiré de la POO
- Clé-valeur : très proche du fonctionnement d'une table de hachage
- Orienté colonne : modèle de donnée entre le Clé-valeur et Relationnel ; permet de parcourir très rapidement les données à l'intérieur de ce qui correspond à une colonne dans le modèle relationnel

Les SGBD ne suivant pas le modèle relationnel sont appelés **NoSQL**.

Dans le modèle relationnel, les données sont décomposées sous forme de *relation* (i.e. de tables), et utilisent un langage plus ou moins standardisé, le SQL. Les données sont stockées sur le disque, dans des tables de relation elles-mêmes stockées ligne par ligne. C'est un système centralisé, avec des propriétés de distributions limitées. Parmi les systèmes les plus répandus, on pourra citer Oracle, Microsoft SQL Server, MySQL, SQLite, et PostgreSQL. En TP, on utilisera la convention PostgreSQL.

Les SGBD relationnels classiques possèdent de nombreuses forces. Ils introduisent une indépendance entre le modèle de données et les structures de stockage, et les requêtes déclaratives

et la façon dont elles sont exécutées. Ils sont capables d'exécuter des requêtes optimisées, finement optimisées, permettant de traiter jusqu'à des To de données. C'est une technologie mature, stable, et efficace, riche en fonctionnalités et interfaces. Le modèle relationnel permet de maintenir des contraintes d'intégrité et des transactions qui garantissent le contrôle de la concurrence des processus, l'isolation entre les utilisateurs, et la récupération en cas d'échec.

Les SGBD relationnels satisfont les propriétés suivantes, dites **ACID** :

- *Atomisme* : une transaction est soit exécutée en entier, soit pas exécutée du tout
- *Consistence* : les contraintes d'intégrité sont respectées
- *Isolation* : deux transactions concurrentes sont exécutées en série
- *Durabilité* : les données peuvent être récupérées même après un échec système

Ces propriétés viennent néanmoins également avec des faiblesses, comme leur incapacité à gérer des volumes de données extrêmement larges, de certains types, et impliquent un coût élevé des requêtes.

Les modèles **NoSQL** forment un écosystème très divers, abandonnant généralement les propriétés **ACID**, et parfois les requêtes complexes. Des nouveaux systèmes, dits **NewSQL**, essayent d'établir un compromis entre propriétés **ACID** et performances.

## 1.2 Modèle relationnel

On se fixe les ensembles infinis dénombrables suivants :

- $\mathcal{L}$ , les étiquettes
- $\mathcal{V}$ , les valeurs
- $\mathcal{T}$ , les types, tel que  $\forall \tau \in \mathcal{T}, \tau \subseteq \mathcal{V}$

**Définition** (Schéma relationnel d'arité  $n$ ). Un schéma relationnel d'arité  $n$  est un  $n$ -uplet  $(A_1, \dots, A_n)$  où chaque  $A_i$  est un attribut, c'est à dire une paire  $(L_i, \tau_i) \in \mathcal{L} \times \mathcal{T}$  où les  $L_i$  sont tous distincts.

**Définition** (Schéma de base de données). Un schéma de base de données est défini par un ensemble fini d'étiquettes  $L \subseteq \mathcal{L}$  (les noms des relations), où chaque étiquette  $L$  est associée à un schéma de relation.

**Exemple.** On peut par exemple considérer que  $\mathcal{L}$  est l'ensemble des chaînes de caractères alphanumériques commençant par une lettre.  $\mathcal{V}$  est l'ensemble des suites de bits, et  $\mathcal{T}$  est formé des types *INTEGER*, *REAL*, *TEXT*, et *DATA*, ... Un schéma de base de données est formé de deux noms de relations *Client* et *Reservation*. Un *Client* est de la forme :

$$((id, INTEGER), (name, TEXT), (email, TEXT))$$

et une réservation est de la forme :

$$(id, INTEGER, (guest, INTEGER), (room, INTEGER), (arrival, DATE), (nights, INTEGER))$$

**Définition** (Instance d'un schéma de relation). Une *instance d'un schéma de relation* (auss appelé une *relation sur ce schéma*)  $((L_1, \tau_1), \dots, (L_n, \tau_n))$  est un ensemble fini  $\{t_1, \dots, t_k\}$  de tuples de la forme  $t_j = (v_{j1}, \dots, v_{jn})$ , où  $\forall j, \forall i, v_{ij} \in \tau_i$ .

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

**Remarque.** Le terme *relation* est utilisé de manière ambiguë pour parler aussi bien d'un schéma de relation ou d'une instance d'un schéma de relation.

Si  $A = (L, \tau)$  est le  $i$ -ème attribut d'une relation  $R$ , et que  $t$  est le  $n$ -ième tuple d'une instance de  $R$ , on note  $t[A]$  (ou  $t[L]$ ) la valeur de la  $i$ -ème composante de  $t$ . De manière similaire, si  $\mathcal{A}$  est un  $k$ -uplet d'attributs parmi les  $n$  attributs de  $R$ ,  $t[\mathcal{A}]$  est le  $k$ -uplet formé à partir de  $t$  en concaténant les  $t[A]$  pour  $A \in \mathcal{A}$ .

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

**Définition (Clé).** Un uplet d'attributs  $\mathcal{A}$  sur un schéma de relation  $R$  est une clé s'il ne peut pas exister deux uplets distincts  $t_1$  et  $t_2$  dans une instance de  $R$  tels que  $t_1[\mathcal{A}] = t_2[\mathcal{A}]$ .

**Définition (Clé étrangère).** Un  $k$ -uplet d'attributs  $\mathcal{A}$  d'un schéma de relation  $R$  est une *clé étrangère* référençant un  $k$ -uplet d'attributs  $\mathcal{B}$  d'un schéma de relation  $S$  si pour toutes les instances  $I^R$  et  $I^S$  de  $R$  et  $S$ , et pour tout uplet  $t$  de  $I^R$ , il existe un *unique* tuple  $t'$  de  $I^S$  avec  $t[\mathcal{A}] = t'[\mathcal{B}]$ .

On peut également définir des contraintes vérifiées, c'est à dire une condition arbitraire sur les valeurs des attributs d'une relation (en appliquant à chaque tuple de l'instance cette relation).

**Exemple.** En reprenant l'exemple précédent, on peut donner les contraintes suivantes :

## 1.3 Langages de requêtes

### 1.3.1 Algèbre relationnelle

On utilise des *langages algébriques* pour exprimer des requêtes. Une expression d'algèbre relationnelle produit une nouvelle relation à partir des relations de la base de données. Chaque opérateur prend 0, 1 ou 2 sous-expressions. Les opérateurs principaux sont les suivants :

**Opérateur "Nom de relation"** Retourne la relation spécifiée en argument.

**Opérateur "Renommage"** Prend une sous-expression, et renomme une colonne.

**Opérateur "Projection"** Prend en argument une séquence d'attributs, et conserve en sortie simplement les attributs spécifiés de la relation donnée.

**Opérateur "Sélection"** Prend une formule de sélection pouvant être n'importe quelle combinaison booléenne de comparaisons d'attributs et de constantes, et conserve uniquement les tuples où la condition booléenne est satisfaite.

**Opérateur "Produit cartésien"** Retourne le produit cartésien ensembliste des deux ensembles obtenus par les sous-expressions.

**Opérateur "Union"** Retourne l'union des ensembles obtenus par les sous-expressions.

**Opérateur "Différence"** Retourne l'ensemble obtenu par la première sous-expression, privé de l'ensemble obtenu par la seconde sous-expression. Cet opérateur ajoute de l'expressivité aux requêtes, par exemple avec des requêtes non-monotones.

**Opérateur "Jointure"** Prends deux tables et une condition de jointure, et retourne les tuples où la condition de jointure est vérifiée. C'est un opérateur crucial dans la perception et l'usage du modèle relationnel, mais est en fait redondant. Il peut être vu comme une combinaison des opérateurs de renommage, produit cartésien, sélection et projection.

Toutes les expressions de l'algèbre relationnel ne sont pas valides, et la validité d'une requête dépend en général du schéma de la base de données. Par exemple, si un nom de relation n'existe pas, si l'on essaye d'unifier deux relations avec des attributs différents, ou encore si l'on essaye de produire une table par produit, join, ou renommage qui contient deux attributs avec le même nom. L'implémentation de la gestion de ces erreurs par les SGBD est très variable.

### 1.3.2 SQL

Le *Structured Query Language* (SQL), est un langage plus ou moins standardisé pour interagir avec un SGBD. Seules quelques différences syntaxiques sont remarquables d'une version à une autre.

C'est un langage *déclaratif*, c'est à dire qu'il n'est pas nécessaire de penser à la complexité de l'opération, simplement à ce que l'on veut. C'est le SGBD qui prend en charge l'optimisation de la requête.

Les mots-clés et identifiants sont généralement insensibles à la casse. En pratique, les mots clés sont écrits en majuscule, et les identifiants commencent par une majuscule. Les commentaires sont introduits par `--`. Dans certains contextes, on termine les commandes par `;`, mais le point-virgule ne fait pas partie de la commande, il est simplement demandé par l'environnement (par exemple, pour un client en ligne de commande).

**NULL** La valeur spéciale NULL signifie l'absence de valeur, et introduit une logique tri-valuée étrange : `True`, `False`, et `NULL`. La comparaison normale avec NULL renvoie toujours NULL. On peut vérifier la présence de valeur avec `IS NULL` ou `IS NOT NULL`.

## Langage de définition de données

### Contraintes

### Mises à jour

### Requêtes