Bases de données

Informatique BUT 1re année et L1

Stéphane DEVISMES
Professeur à l'Université de Picardie Jules Verne

Anaïs DURAND
Maîtresse de conférences à l'Université Clermont Auvergne

Pascal LAFOURCADE
Professeur des universités à l'IUT de Clermont Auvergne

DUNOD

Direction et conception graphique de la couverture :
Nicolas Wiel – Pierre André Gualino (graphiste)
Mise en page :

11 rue Paul Bert, 92240 Malakoff
www.dunod.com

ISBN 978-2-10-087890-1

Avant-propos

Le XXI^{e} siècle est le siècle du numérique. Chaque société est visible sur Internet, les démarches administratives sont de plus en plus numérisées et rechercher une information sur son ordinateur ou son téléphone portable en interrogeant son moteur de recherche favori est devenu un réflexe. Derrière ces usages familliers de tous se cachent des années de recherche en informatique, en particulier en Base de Données (BD).

Cet ouvrage permet d'appréhender le fonctionnement d'un Système de Gestion de Base de Données (SGBD). Il s'agit principalement d'une introduction au langage *Structured Query Language* (SQL). Le but général est de donner toutes les bases nécessaires afin d'utiliser convenablement ces outils puissants, avec un vrai point de vue pratique.

Le premier chapitre de cet ouvrage aborde donc les grands principes fondateurs des bases de données. Le second chapitre traite de la création de bases de données. Les requêtes d'interrogation simples sont abordées dans le troisième chapitre. Dans le chapitre suivant, les requêtes plus avancées sont abordées. Ensuite, les notions de jointures entre plusieurs tables, d'agrégation et de sous-requêtes sont abordées. Le chapitre 7 s'attarde sur la conception des base de données. Cet aspect est crucial et détermine la structure des données mais a aussi souvent une influence sur les performances des requêtes. Le chapitre suivant propose des outils de visualisation en Python. Enfin, le dernier chapitre présente les outils d'administration des utilisateurs.

Cet ouvrage couvre le programme de première année de Bachelor Universitaire de Technologie (BUT) Informatique. Il s'adresse donc en priorité aux étudiants et enseignants de cette formation. Il s'adresse également aux étudiants et enseignants de licence et d'école d'ingénieur en informatique, ainsi qu'à toute personne souhaitant découvrir et travailler avec des bases de données.

Les auteurs expriment leur gratitude à Matthieu Daniel, Laurent Desrosiers et Anne Le Duc pour leurs commentaires et suggestions de modifications constructifs, à la suite de leurs relectures assidues.

À Amiens et Clermont-Ferrand, 16 août 2024.

Stéphane Devismes, Anaïs Durand et Pascal Lafourcade

Table des matières VI

1

Introduction

1.1 Base de données

Un *Système de Gestion de Base de Données (SGBD)* est un ensemble de logiciels servant à stocker, modifier et consulter de manière efficace des informations, appelées *données*. L'efficacité d'un tel système s'évalue suivant de nombreux critères, dont voici une liste non-exhaustive :

1. le temps d'exécution des consultations et mises à jour ;
2. la capacité à assurer la cohérence des données ;
3. la capacité à minimiser les redondances ;
4. la disponibilité des données en dépit de pannes ;
5. la sécurité.

Les SGBD sont élaborés à partir de modèles. Historiquement, les premiers SGBD, comme le moteur *Information Management System* (IMS) d'IBM, suivaient un modèle dit « hiérarchique ». En 1970, Edgard Frank Codd a proposé *le modèle relationnel*, un modèle basé sur la théorie des ensembles et l'algèbre relationnelle. En 2024, il s'agit du modèle de loin le plus utilisé pour les bases de données. Ce modèle permet en particulier de définir des contraintes garantissant l'intégrité référentielle et fonctionnelle des données. Un SGBD basé sur le modèle relationel est appelé un SGBD relationnelles (SGBDR).

De façon informelle, dans un SGBDR, les données sont stockées dans des tables, chaque table étant structurée en colonnes.

Exemple 1.1

La table `AUTEURS` ci-dessous contient les données des auteurs de ce livre, organisées en trois colonnes : leur nom, leur prénom, et leur ville.

`nom`	`prénom`	`ville`
Devismes	Stéphane	Amiens
Durand	Anaïs	Clermont-Ferrand
Lafourcade	Pascal	Clermont-Ferrand

Le contenu d'une base de données relationnelle (les tables et leur contenu) n'est bien sûr pas figé. Il est par exemple possible d'ajouter des données, d'en modifier, d'en supprimer. Un langage dédié, SQL, permet de réaliser ces opérations.

Les 12 règles de Codd. Plus formellement, Codd énumère 12 règles définissant les caractéristiques nécessaires à un SGBD pour qu'il soit considéré comme un SGBDR :

- *Règle 1 : l'unicité.* Toutes les données sont représentées d'une et une seule manière : ce sont des valeurs dans des colonnes de tables.
- *Règle 2 : la garantie d'accès.* Toute donnée doit être accessible sans ambiguïté. Précisément, toute donnée stockée doit être logiquement accessible à partir du nom de la table contenante, du nom de la colonne contenante et de la valeur de l'identifiant de la ligne contenante.
- *Règle 3 : le traitement des valeurs nulles.* Une valeur particulière, communément appelée la valeur **`NULL`**, doit signifier l'absence d'une donnée. Cette valeur doit pouvoir être affectée, quel que soit le type de la donnée.
- *Règle 4 : le catalogue lui-même relationnel.* Le *catalogue*, c'est-à-dire, la description de la base de données, doit être accessible de la même manière que les données ordinaires.
- *Règle 5 : le sous-langage de données.* Le système doit proposer un langage permettant de (1) définir et mettre à jour les données, (2) définir des représentations logiques d'ensemble de données appelée *vues*, (3) définir des contraintes d'intégrité, (4) contrôler les autorisations d'accès et (5) gérer des transactions (c'est-à-dire, l'exécution atomique d'une suite d'opérations).
- *Règle 6 : la mise à jour des vues.* Le système doit assurer la mise à jour automatique des données des vues.
- *Règle 7 : l'insertion, mise à jour, et effacement de haut niveau.* Le langage du système doit permettre les opérations *par lot* d'insertion, de mise à jour et de suppression.
- *Règle 8 : l'indépendance physique.* Les modifications au niveau physique (c'est-à-dire, comment les données sont stockées) n'implique pas de changement au niveau logique.
- *Règle 9 : l'indépendance logique.* Le niveau logique doit pouvoir être modifié sans remettre en cause le niveau physique.
- *Règle 10 : l'indépendance d'intégrité.* Les contraintes d'intégrités doivent pouvoir être définies et stockées dans le catalogue.
- *Règle 11 : l'indépendance de distribution.* La localisation des données (peut-être sur divers sites) doit être invisible aux utilisateurs de la base de données.

— *Règle 12 : la non-subversion.* Si le système fournit une interface de bas niveau, cette interface ne doit pas permettre de contourner les contraintes définies dans le système.

Une règle 0 est parfois ajoutée. Elle stipule qu'un SGBDR doit être capable de gérer complètement une base de données à l'aide ses caractéristiques relationnelles. Autrement dit, le SGBDR ne peut faire appel à aucune opération non-relationnelle pour réaliser la moindre de ses fonctionnalités.

Pour résumer, les SGBDR incluent les caractéristiques majeures suivantes : une séparation claire entre la vision logique et l'implémentation physique, l'existence d'un langage dédié, la structuration forte des données sous forme tabulaire, la possibilité de définir des contraintes fortes au niveau du moteur, ainsi que la cohérence forte au niveau transactionnel.

1.2 SQL

Un SGBDR doit, entre autres, mettre à disposition un langage de haut niveau dédié (*cf.* règles 5 et 7), basé sur l'algèbre relationnelle, cachant à l'utilisateur le détail de l'implantation physique du système. Pour la majeure partie des SGBDR (par exemple, Oracle, MySQL, PostgreSQL, MariaDB, SQLite), il s'agit du langage *Structured Query Language* (SQL). Ce langage est normalisé depuis 1986, sa dernière norme remontant à 2016. Dans ce cours, le SGBDR *PostgreSQL* est utilisé. Il met à disposition le langage SQL. SQL est en réalité composé de plusieurs langages :

— le *Langage de Manipulation des Données (LMD)* qui permet la sélection, la recherche, l'ajout, la suppression ou la modification des données en base ;
— le *Langage de Définition des Données (LDD)* qui permet de créer et modifier la structure de la base, autrement dit de gérer de l'organisation des données ;
— le *Langage de Contrôle de Transaction (LCT)* pour la gestion des transactions (ensemble d'opérations à gérer d'un seul bloc) ;
— le *Langage de Contrôle des Données (LCD)* pour la gestion des droits d'accès aux données par les utilisateurs de la base.

Généralement, SQL est encapsulé dans un langage de programmation procédural (par exemple, *PL/pgSQL* pour PostgreSQL). Ce langage permet, entre autres, de définir des procédures stockées, des déclencheurs (*triggers*) et des fonctions utilisateurs (UDF, pour *User Define Function*).

1.3 Modèle relationnel et SGBDR

Les SGBDR sont basés sur le modèle relationnel. Le modèle relationnel est lui-même basé sur une algèbre relationnelle, qui relève donc des mathématiques. Cette algèbre consiste à définir des opérations sur les relations. Les corrélations et les différences principales entre le modèle relationnel et les SGBD sont maintenant présentées.

1.3.1 Relations et tables

Soient $D_1, \ldots, D_n$ des ensembles de valeurs. Le *produit cartésien* $D_1 \times \cdots \times D_n$ est l'ensemble de tous les *n-uplets* (*e.g.*, doublet, triplet, etc.) $\langle x_1, \ldots, x_n \rangle$ possibles avec $x_i \in D_i$ pour chaque i. Une *relation* R sur $D_1, \ldots, D_n$ est un sous-ensemble du produit cartésien $D_1 \times \cdots \times D_n$: $R \subseteq D_1 \times \cdots \times D_n$.

Exemple 1.2

Soient $E_1 = \{\text{Anne}, \text{Bernard}, \text{Cyril}\}$ et $E_2 = \{8, 12\}$ deux ensembles. Le produit cartésien $E_1 \times E_2$ est égal à

$$\{\langle \text{Anne},8\rangle, \langle \text{Anne}, 12\rangle, \langle \text{Bernard}, 8\rangle, \langle \text{Bernard}, 12\rangle, \langle \text{Cyril}, 8\rangle, \langle \text{Cyril}, 12\rangle\}$$

R_1, R_2 et R_3 définies ci-dessous sont des relations sur E_1 et E_2 :
$R_1 = \{\langle \text{Anne}, 8\rangle, \langle \text{Bernard}, 12\rangle, \langle \text{Cyril}, 8\rangle\}$
$R_2 = \{\langle \text{Anne},8\rangle, \langle \text{Anne}, 12\rangle, \langle \text{Bernard}, 8\rangle, \langle \text{Bernard}, 12\rangle, \langle \text{Cyril}, 8\rangle, \langle \text{Cyril}, 12\rangle\}$
$R_3 = \emptyset$

Par définition, les éléments de la relation R sont des *n-uplets*$\langle x_1, \ldots, x_n \rangle$ (avec $x_i \in D_i$ pour chaque i). Pour chaque i de 1 à n, un nom A_i, appelé *attribut*, est choisi pour nommer le i-ème élément de chaque n-uplet de R : x_i est la valeur de l'attribut A_i dans le n-uplet $\langle x_1, \ldots, x_n \rangle$.

Exemple 1.3

Pour $R_1 = \{\langle \text{Anne}, 8\rangle, \langle \text{Bernard}, 12\rangle, \langle \text{Cyril}, 8\rangle\}$, il est, par exemple, possible de choisir de nommer les attributs `prénom` et `âge`. La valeur de l'attribut `prénom` dans le doublet $\langle \text{Anne}, 8\rangle$ est ainsi Anne.

Une relation sur $D_1, \ldots, D_n$ est représentée en SQL par une *table*. Cette table a n *colonnes*, la colonne i a pour nom l'attribut A_i et elle contient des éléments de D_i. La table a une *ligne* pour chaque n-uplet $\langle x_1, \ldots, x_n \rangle$ de la relation correspondante.

Exemple 1.4

Pour $R_1 = \{\langle \text{Anne}, 8\rangle, \langle \text{Bernard}, 12\rangle, \langle \text{Cyril}, 8\rangle\}$, la table a les trois lignes et les deux colonnes suivantes :

prénom	âge
Anne	8
Bernard	12
Cyril	8

Une relation et une table diffèrent en plusieurs points. Une relation est un *ensemble* de n-uplets, donc :

— il ne peut pas y avoir deux n-uplets égaux et ;
— il n'y a pas d'ordre sur les n-uplets.

Par contre une table est une *liste de lignes*, donc :

— une table peut comporter plusieurs lignes égales et ;

— les lignes d'une table sont dans un certain ordre.

Cependant, il est possible d'enlever les *doublons* dans une table, et il est possible d'imposer un *ordre* précis pour l'affichage des n-uplets. Mais, ces opérations sont coûteuses, il ne faut les utiliser qu'en cas de nécessité.

1.3.2 Les types

Dans une relation sur $D_1, \ldots, D_n$ avec comme attributs $A_1, \ldots, A_n$, l'ensemble D_i est le *domaine* ou le *type* de l'attribut A_i.

Les domaines se déclinent en trois types principaux : les *nombres*, les *chaînes de caractères* et le *temps*.

Exemple 1.5

Dans la relation R_1, `prénom` est de type chaîne de caractères et `âge` est de type nombre.

Les types en SQL sont plus précis : il y a plusieurs types de nombres et de chaînes de caractères, afin de préciser la place maximale nécessaire pour stocker leurs éléments en mémoire. De plus, en PostgreSQL, il y a trois sous-types pour gérer le temps : les dates (`Date`), les horaires (`Time`) ou les deux combinés (`Timestamp`).

1.3.3 Identifiants et clés

Un *identifiant* ou une *clé* d'une relation R est un sous-ensemble K d'attributs de R tel que les valeurs de ces attributs déterminent (de manière unique) les valeurs de tous les autres attributs. Autrement dit, si deux lignes ont le même identifiant, alors elles sont égales.

De plus, cet ensemble d'attributs doit être minimal :

— ou bien K est formé d'un seul attribut,
— ou bien K est formé de plusieurs attributs et dès qu'un attribut est enlevé la propriété d'identifiant n'est plus vérifiée.

Il peut y avoir plusieurs identifiants pour une relation. Dans ce cas, il faut en choisir un qui est appelé *identifiant primaire* ou *clé primaire*. Attention, la valeur de chaque attribut d'une clé primaire doit être renseignée, autrement dit elle ne peut être nulle. Si la relation a d'autres clés, ces dernières sont appelées *clés secondaires*.

Exemple 1.6

Dans R_1, `prénom` peut faire office de clé primaire.

En SQL, la contrainte de minimalité n'est pas exigée, seule la contrainte d'unicité est vérifiée.