République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderrahmane MIRA de Béjaia

جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

Faculté des Sciences Exactes

Département d'Informatique

SUPPORT DE COURS

Préparé par :

Dr Mohamed Essaid KHANOUCHE
Maître de Conférences à l'Université de Béjaia

Module :

BASES DE DONNÉES

Année universitaire 2018 – 2019

Table des matières

Liste des figures

Liste des tableaux

PRÉAMBULE

Le présent document portant sur « Les bases de données » est destiné aux étudiants en **deuxième année Licence Informatique** et pourra être consulté par toute personne voulant comprendre la problématique des bases de données. En effet, les bases de données jouent un rôle central dans le développement technologique depuis plus de quatre décennies. Aujourd'hui, nous pouvons remarquer que toute application informatique utilise de manière directe ou indirecte une base de données. Ce document présente une synthèse des principes et des techniques actuelles en matière de base de données notamment les bases de données relationnelles. Ces paradigmes sont au cœur des systèmes d'information d'aujourd'hui et doivent être connus de tout étudiant à l'université et de tout professionnel de l'informatique. L'objectif de ce document est de former les étudiants à la conception, à la création, et à la gestion des bases de données relationnelles en adoptant un apprentissage progressif et méthodologique des concepts fondamentaux des bases de données basé sur des exemples pratiques. Ce support de cours pourrait être éventuellement consulté par les étudiants en *Master 1 Mathématiques Appliquées* et les étudiants en *troisième année Licence Informatique Recrutement Natio*nal.

La structure générale de ce document comprend cinq chapitres.

CHAPITRE 1 – Bases de données et système de gestion de bases de données

Il a pour objectif d'introduire la problématique des bases de données. Nous introduisons tout d'abord le concept de bases de données qui joue un rôle central dans les dispositifs informatiques de collecte, mise en forme, stockage et utilisation de l'information. Dans la suite du chapitre, nous décrivons les principaux modèles de données et exposons une vue d'ensemble des Systèmes de Gestion de Bases de Données (principe de fonctionnement, objectifs et les trois niveaux de description des données de la norme ANSI/SPARC). Enfin, dans la dernière partie, nous esquissons quelques exemples de SGBD commerciaux, tels que Oracle, et d'autres gratuits, tels que MySQL.

CHAPITRE 2 – Concepts de base sur les bases de données relationnelles

Il se focalise sur l'étude des concepts pivots des bases de données relationnelles. En effet, le modèle relationnel fût introduit en 1970 pour gérer de grandes quantités de données en s'appuyant sur la théorie des ensembles et la logique des prédicats. Dans la première partie du chapitre, nous présentons des notions sur le modèle relationnel constituant aujourd'hui le modèle de données le plus populaire. Ensuite, nous abordons le paradigme de bases de données relationnelles à travers les concepts de schéma de base de données relationnelle et contraintes d'intégrité.

CHAPITRE 3 – Dépendances fonctionnelles et normalisation

Il étudiera la théorie de la normalisation qui constitue un processus permettant le contrôle de la redondance et la préservation de la cohérence de données dans les bases de données relationnelles. L'objectif de ce processus est d'éviter les pertes de données, les incohérences de données, et la dégradation des performances de traitement. La théorie de normalisation repose essentiellement sur le concept de *dépendances fonctionnelles* qui constituent une catégorie principale parmi les contraintes d'intégrité parce qu'elles sont à la base du processus de conception des bases de données relationnelles.

CHAPITRE 4 – Algèbre relationnelle

Il aborde le support mathématique sur lequel repose le modèle relationnel. L'algèbre relationnelle définit un ensemble d'opérateurs formels qu'il est possible d'appliquer sur des relations pour créer de nouvelles relations. On peut considérer que l'algèbre relationnelle est aux relations ce qu'est l'arithmétique aux entiers. Cet ensemble d'opérateurs formels constitue le cœur du langage de requête normalisé SQL.

CHAPITRE 5 – Mise en œuvre d'une base de données relationnelle : Langage SQL

Il est consacré au langage SQL (Structured Query Language) qui se présente sous forme de commandes normalisées de gestion de bases de données relationnelles. Une commande SQL constitue la description d'une opération qu'un SGBD doit exécuter sur une base de données. Les commandes SQL se divisent généralement en quatre catégories : le langage de définition de données (LDD), le langage de manipulation de données (LMD), le langage de protections d'accès et le langage de contrôle de transactions. Dans le contexte de ce document, seulement les LDD et LMD sont présentés.

CHAPITRE 1 – Bases de données et système de gestion de bases de données

Sommaire

1. Introduction

Ce chapitre introduit la problématique des bases de données. Nous commençons par la définition du concept de bases de données qui constitue l'élément fondamental dans les procédures informatiques de collecte, mise en forme, stockage et utilisation de l'information. Nous décrivons ensuite les principaux modèles de données et introduisons les Systèmes de Gestion de Bases de Données (SGBD) en se focalisant sur leur principe de fonctionnement, leurs objectifs et les trois niveaux de description de données de la norme ANSI/SPARC. Enfin, nous donnons quelques exemples de SGBD commerciaux et d'autres gratuits.

2. Notion de base de données

Il existe dans la littérature plusieurs définitions de la notion de base de données. Nous nous limitons ici aux définitions qui nous semblent les plus pertinentes.

Définition 1.1 – Une *base de données* est un ensemble structuré d'informations modélisant les *entités* (ou les *objets*) d'une partie du monde réel et ayant un objectif commun [GAR 2003]. Ces informations sont pertinentes, cohérentes, non redondantes, et accessible par plusieurs utilisateurs de manière sélective. Une *entité* est un élément significatif pour lequel il faut disposer d'informations ou de caractéristiques. Par exemple, dans une base de données de gestion d'une bibliothèque, les entités modélisées sont principalement le *lecteur* et l'*ouvrage*. Les informations enregistrées sur l'entité *Lecteur* peuvent être le matricule, le nom, le prénom, etc. Cependant, on ne s'intéresse pas, par exemple, à la taille et au nombre d'amis d'un lecteur parce que ces informations ne sont pas pertinentes dans le contexte de la gestion d'une bibliothèque. Le mot *« ensemble structuré d'informations »* signifie une collection organisée selon des structures logiques particulières (tables, arbres, etc.).

Quel que soit le support utilisé pour rassembler et stocker les données (papier, fichiers, etc.), dès lors que des données sont rassemblées et stockées d'une manière organisée dans un but spécifique, on parle d'une base de données.

Définition 1.2 – Une *base de données* est un ensemble structuré et organisé permettant le stockage de grandes quantités d'informations pour en faciliter la manipulation (ajout, mise à jour et recherche de données) par de multiples utilisateurs. Dans le contexte de ce document, nous nous intéressons aux bases de données informatisées. En effet, une base de données informatisée est un ensemble structuré de données modélisant des entités du monde réel, enregistré sur des supports accessibles par l'ordinateur et pouvant être mis-à-jour/interrogé par de multiples utilisateurs [AUD 2009].

La gestion et l'accès à une base de données est réalisée par un outil informatique ou *logiciel* spécifique appelé *Système de Gestion de Base de Données* (SGBD). Nous y reviendrons dans la section 4. Un tel système est caractérisé par le modèle de données qu'il supporte. Ce modèle peut être hiérarchique, réseau, relationnel, ou bien objet (cf. section 3). Le modèle qui s'est naturellement imposé depuis plusieurs années est le modèle relationnel (cf. chapitre 2).

3. Modèles de données

Tout comme *les maquettes* qui permettent, en général, aux personnes de conceptualiser plus facilement une idée abstraite, un *modèle de données* décrit la manière dont les données d'une base de données sont représentées logiquement [MAT 2002]. Différents modèles ont été imaginés au cours du temps, tels que le modèle hiérarchique, le modèle en réseau, le modèle relationnel, le modèle relationnel-objet, etc.

3.1. Modèle hiérarchique

Ce modèle fût développé par la société IBM (International Business Machines) pour la gestion des données du programme Apollo de la NASA durant les années 1960. Une base de données hiérarchique lie des enregistrements (un ensemble de champs ou de propriétés) dans une structure arborescente où chaque enregistrement a un seul possesseur, c.-à-d., chaque niveau de la hiérarchie n'a qu'un nœud qui pointe sur lui depuis le niveau supérieur. L'accès aux données commence par la racine et descend l'arborescence jusqu'aux détails recherchés. La **Fig. 1.1** illustre un exemple d'une base de données hiérarchique.

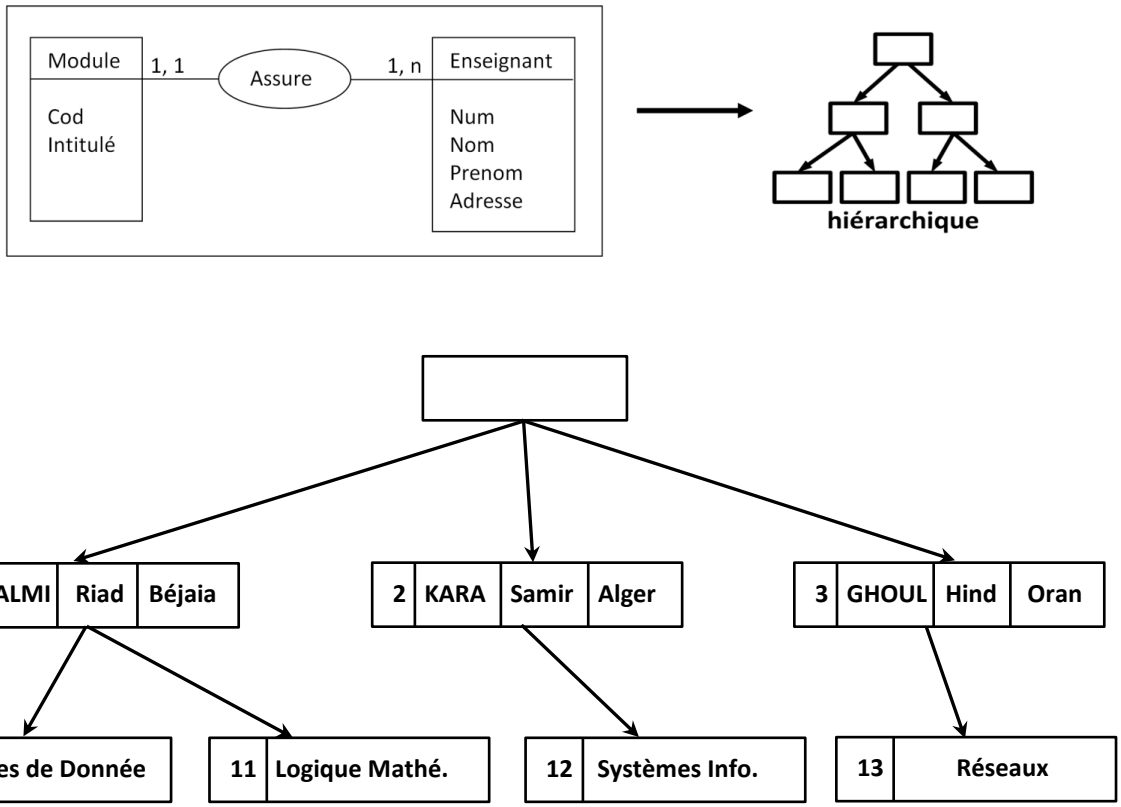


Fig. 1. 1 – Exemple d'une base de données suivant le modèle hiérarchique.

Le modèle hiérarchique possède les caractéristiques suivantes : un modèle logique orienté enregistrement, les données sont représentées par des enregistrements, les enregistrements sont associés par des relations qui sont des liens et les liens n'associent que deux enregistrements à la fois. Un exemple de système de gestion de bases de données hiérarchique est IMS (Information Management System) d'IBM.

Cette représentation hiérarchiques des différents types de données facilite la réponse à certaines questions, mais pourrait rendre difficile la réponse à d'autres. Dans le cas où le principe de relation *« un-à-plusieurs »* n'est pas respecté (par exemple, sur la Fig. 1.1, si les modules *Bases de Données* et *Système d'Information* sont tous les deux assurés par deux enseignants), le modèle hiérarchique ne devient plus adapté pour représenter cette situation en

raison de la redondance de données qu'il engendre. Cela justifie l'introduction d'un modèle de données plus général qui est le *modèle réseau*.

3.2. Modèle réseau

Ce modèle a été inventé par Charles William BACHMAN à la fin des années 1960. Le modèle réseau est une amélioration du modèle hiérarchique dans la mesure où il permet de pallier de nombreuses limitations inhérentes à ce dernier grâce à la possibilité d'établir des liaisons de type ***« plusieurs-à-plusieurs »***, c.-à-d., les liens entre entités peuvent exister sans restriction. L'accès aux données se réalise alors par des chemins divers. La Fig. 1.2 montre que le fait que les modules *Bases de Données* et *Système d'Information* sont tous les deux assurés par deux enseignants, peut être représenté sans redondance.

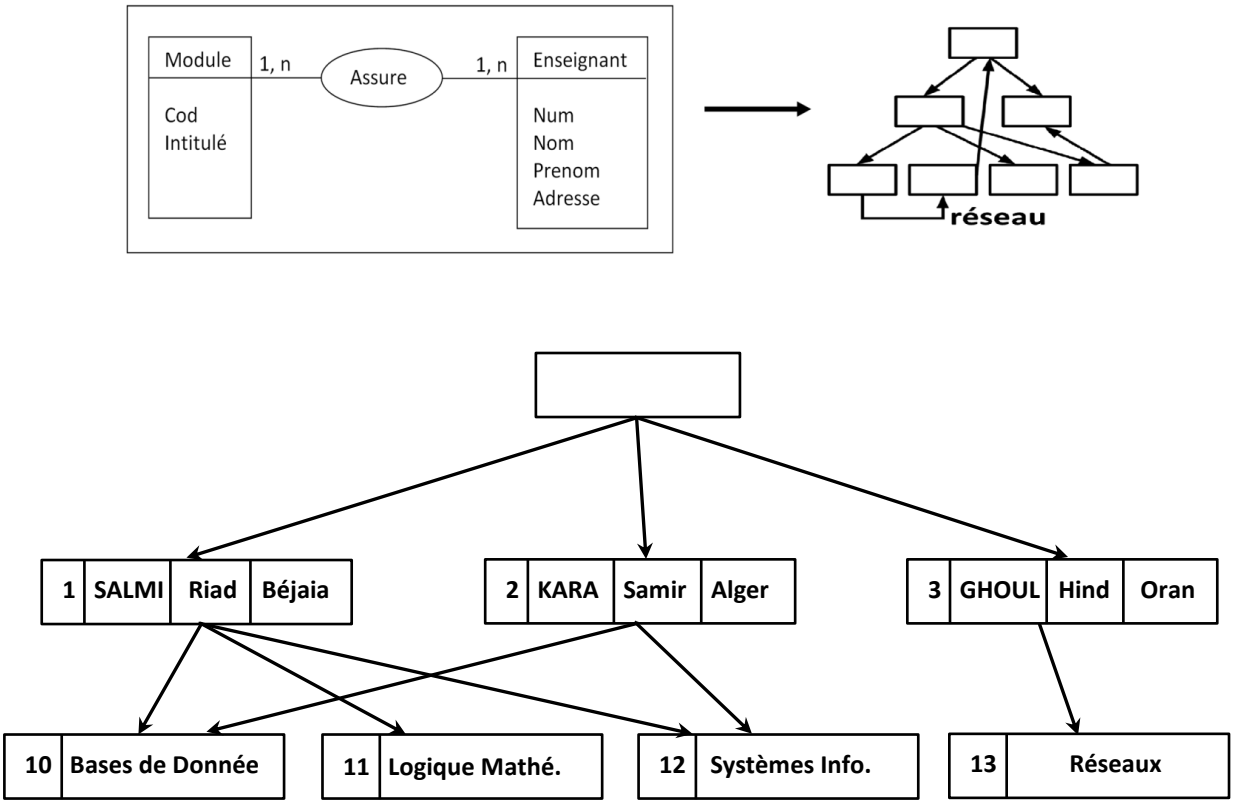


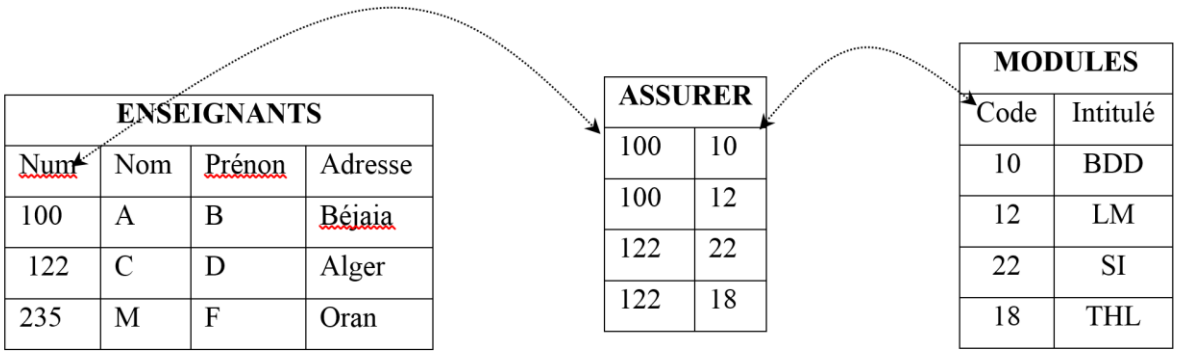
Fig. 1. 2 – Exemple d'une base de données suivant le modèle réseau.

Les données du modèle réseau sont représentées par des embles d'enregistrements associés par des relations (liens). Ces enregistrements représentent les entités d'un modèle entité–association et les attributs des entités deviennent les champs des enregistrements. Les enregistrements sont associés par des relations que l'on appelle liens. Les liens n'associent que deux enregistrements à la fois. Un exemple de SGBD réseau est EDMS (Extended Data Management System) de Xerox ou encore IDS2 de Bull.

3.3. Modèle relationnel

Chercheur au centre de recherche d'IBM à la fin des années 1960, Edgar Frank CODD n'était pas satisfait par les modèles de données de l'époque. Il chercha alors de nouveaux modèles, plus simples, permettant la gestion de grandes quantités de données. Mathématicien de formation, CODD était persuadé de pouvoir s'appuyer sur deux branches spécifiques des mathématiques (la théorie des ensembles et la logique des prédicats) pour résoudre le problème de la redondance et de l'incohérence des données.

En 1970, CODD proposait de stocker des données hétérogènes dans des tables ou relations (cf. Fig. 1.3). Il s'agit du modèle relationnel qui est beaucoup plus flexible et facile à utiliser en comparaison aux deux autres modèles précédents parce que l'accès aux données est plus rapide en comparaison aux autres modèles. Depuis les années 80, ce modèle a mûri et a été adoptée par un large nombre d'entreprises.



ENSEIGNANTS

Num	Nom	Prénon	Adresse
100	A	B	Béjaia
122	C	D	Alger
235	M	F	Oran

ASSURER

100	10
100	12
122	22
122	18

MODULES

Code	Intitulé
10	BDD
12	LM
22	SI
18	THL

Fig. 1. 3 – Exemple d'une base de données relationnelle.

En parallèle au développement du modèle relationnel, COOD mit au point SEQUEL (*Structured English QUEry Language*), un langage de manipulation de données, rebaptisé par la suite SQL (*Structured Query Language*) et normalisé dès 1986 par ANSI (*American National Standards Institute*). Le langage SQL fût adopté comme standard par l'ISO (*International Organization for Standardization*) en 1987.

Le modèle relationnel repose sur la théorie mathématique des relations. Chaque relation est Représentée par un tableau à deux dimensions et composée d'un nombre fini de colonnes appelées aussi *attributs*. Chaque attribut possède un nom unique à l'intérieur d'une relation. Une relation ne peut posséder deux enregistrements ou *tuples* identiques.

Il existe d'autres modèles de données, tels que le modèle relationnel objet (modèle relationnel avec intégration des concepts de l'approche orienté objet : héritage, méthodes, types utilisateur, etc.), le modèle XML, etc.

4. Système de gestion de bases de données

4.1. Principes et objectifs

La gestion et l'accès à une base de données sont assurés par un ensemble de programmes qui constituent le Système de Gestion de Base de Données (SGBD). Ce dernier doit permettre l'ajout, la modification, la suppression, et la recherche de données (c.-à-d., interrogation de la base de données) [MAT 2002]. Un SGBD héberge généralement plusieurs bases de données qui sont destinées à des applications différentes (gestion de scolarité, gestion de stock, gestion du personnel, etc.).

Un SGBD est caractérisé principalement par deux aspects : (*i*) la description de la base de données et (*ii*) la manipulation de la base de données. À l'aide d'un langage de description de données, la description d'une base de données permet de définir la structure et les types de données selon un modèle particulier, définir des contraintes d'intégrité sur les données, définir des autorisations d'accès (mot de passe, privilège, etc.). Cette description est appelée *schéma de la base de données*. Une fois la base décrite, on peut y effectuer des insertions, des mises à jour (modification, suppression) et des interrogations (sélectionner, trier, calculer, etc.) ; on parle dans ce cas de la manipulation de la base de données qui est faite par un langage de manipulation de données.

Quel que soit le modèle de données (hiérarchique, réseau, relationnel, etc.), dans le contexte d'un environnement multiutilisateurs impliquant des accès concurrents, un SGBD doit être en mesure de masquer la représentation physique des données, d'assurer la protection et la cohérence de ces données.

Actuellement, la plupart des SGBD fonctionnent selon un mode client/serveur. Le serveur (i.e. la machine qui stocke les données) reçoit des requêtes de plusieurs clients et ceci de manière concurrente. Le serveur analyse la requête, la traite et retourne le résultat au client.

4.2. Les fonctions d'un SGBD

Les principaux objectifs d'un SGBD sont les suivants :

1. **Indépendance physique.** La façon dont les données sont définies doit être indépendante des structures de stockage utilisées.

2. **Indépendance logique.** Un même ensemble de données peut être vu différemment par des utilisateurs différents.

3. **Non redondance (unicité) des données.** Afin d'éviter les problèmes lors des mises à jour, chaque donnée ne doit pas être dupliquée inutilement. Dans le cas où une donnée est dupliquée, laquelle des copies est la bonne ? Est-on sûr d'avoir mis à jour toutes les copies de la donnée dupliquée ?

4. **Cohérence des données.** Dans une base de données, les contraintes d'intégrité permettent de garantir la cohérence des données, c'est-à-dire, les données présentes dans la base sont conformes aux données attendues. Les contraintes d'intégrité sont exprimées dès la création de la base et doivent être vérifiées automatiquement à chaque insertion, modification ou suppression des données. Par exemple, le coefficient d'un module est supérieur à 0.

5. **Efficacité des accès aux données.** Le SGBD doit intégrer des techniques spécifiques pour avoir de bonnes performances lors de l'accès à une base de données. En effet, l'accès aux données se fait en utilisant un langage de manipulation de données. Il est crucial que ce langage permette d'obtenir des réponses aux requêtes en un temps « raisonnable ».

6. **Concurrence et partage des données.** Il s'agit de permettre à de multiples utilisateurs d'accéder aux mêmes données simultanément (c.à.d. au même moment), et de manière transparente. En d'autres termes, chaque utilisateur à l'impression d'être le seul à travailler sur la base de données.

7. **Sécurité des données.** Les données doivent être protégées contre les accès non autorisés. Pour cela, chaque utilisateur a le droit de demander l'exécution de certaines opérations sur certaines données de la base. Ce droit est appelé *privilège*.

8. **Résistance aux pannes.** Que se passe-t-il si une panne survient au milieu d'une modification par exemple ? Il faut pouvoir récupérer la base de données dans un état *sain* (c.à.d. état cohérent le plus récent avant la panne).

4.3. Architecture d'un SGBD

Un SGBD exécute des *services* pour le compte de programmes d'application, dits *programmes clients*. Généralement, un programme d'application **P** s'exécute sur une machine appelée *serveur d'application*, tandis que le SGBD tourne sur une autre machine appelée *serveur de données*. Dans cette architecture, dite *client-serveur*, il existe généralement une troisième machine, la *station cliente*, sur laquelle tourne un navigateur via lequel un utilisateur (humain) interagit avec le programme d'application. On parle alors d'une architecture *à trois couches* ou *trois-tiers*. Il est possible d'installer le programme d'application, le SGBD et le navigateur sur une seule machine. Néanmoins, l'architecture garde sa structure *à trois couches*. Notons que le concept de *client* et de *serveur* correspondent à des rôles : P est un client du SGBD puisqu'il lui demande des services mais P est aussi un serveur pour la station cliente, puisqu'il lui offre des services [HAI 2015].

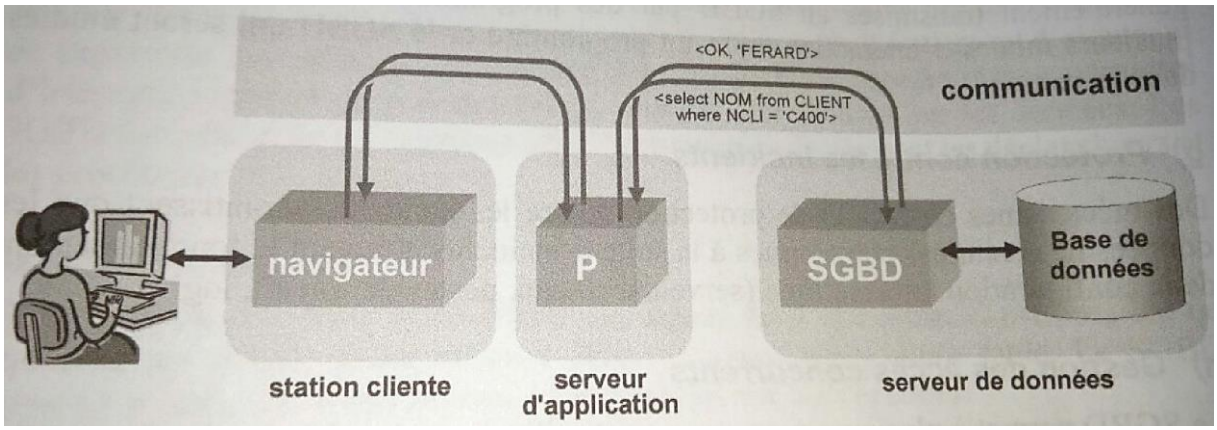


Fig. 1. 4 – Architecture trois-tiers pour un système de gestion de bases de données.

Ces trois composants logiciels dialoguent au travers de canaux de communication de nature quelconque. Chaque programme d'application P envoie aux SGBD des *demandes de services* par exemple *« Quel est l'adresse de l'enseignant ayant le numéro ENS001 ?»* sous la forme d'une requête SQL *« SELECT adresse FROM enseignants WHERE numEns='ENS001'*. En réponse à cette demande, qui doit être correcte et légitime, le SGBD l'exécute et retourne au programme d'application le résultat de cette exécution.

4.4. Niveaux de description des données ANSI/SPARC

Pour atteindre les objectifs d'un système de gestion de bases de données, trois niveaux de description des données ont été définis par la norme ANSI/X3/SPARC [ANS I978].

Le niveau externe : *« Comment différencier entre les utilisateurs avec des privilèges ? »*

Il correspond aux différentes vues des utilisateurs. Chaque schéma externe donne une vue sur le schéma conceptuel à une classe d'utilisateurs. Autrement dit, ce niveau correspond à la perception de tout ou partie de la base par un groupe d'utilisateurs, indépendamment des autres. On appelle cette description le *schéma externe* ou *vue*. Il peut exister plusieurs schémas externes représentant les différentes vues sur la base de données avec des possibilités de recouvrement.

Le niveau conceptuel ou logique : *« Quelle est la structure de la base de données ? »*

Il décrit la structure de toutes les données de la base ainsi que leurs propriétés (*i.e.* les relations qui existent entre elles : leur sémantique inhérente), sans se soucier de l'implémentation physique ni de la façon dont chaque groupe d'utilisateurs voudra s'en servir.

En d'autres termes, ce niveau décrit la structure de la base globalement à tous les utilisateurs. On appelle cette description le *schéma conceptuel*. Ce schéma est produit par une analyse de l'application à réaliser et il décrit la structure de la base de données indépendamment de son implémentation. Dans le cas des SGBD relationnels, il s'agit d'une vision tabulaire où la

sémantique des données est exprimée en utilisant les concepts : relation, attributs, contraintes d'intégrité, etc.

Le niveau interne ou physique : *« Comment les données sont stockées physiquement ? »*

Ce niveau permet de décrire les structures de stockage des données et les méthodes d'accès utilisées pour structurer et retrouver les données sur le support de stockage. En d'autres termes, ce niveau détermine l'emplacement physique des données sur le support de stockage et le nombre d'octets occupés par chaque donnée. Ce niveau s'appuie sur un système de gestion de fichiers pour définir la politique de stockage ainsi que le placement des données. Le niveau physique est donc responsable du choix de l'organisation physique des fichiers ainsi que de l'utilisation de telle ou telle méthode d'accès en fonction de la requête. On appelle cette description le *schéma interne* ; il est fortement dépendant du SGBD utilisé.

4.5. Quelques SGBD

Il existe de nombreux systèmes de gestion de bases de données commerciaux, tels que Oracle (Oracle Corporation), DB2 (IBM), SQL Server (Microsoft), etc. D'autre SGBD sont libres et gratuits (c.à.d. open source), tels que MySQL (Oracle Corporation), PostgreSQL (projet de l'université de Berkeley), etc.

CHAPITRE 2 – Concepts de base sur les bases de données relationnelles

Sommaire

1. Introduction

Le *modèle relationnel* de données a vu le jour avec la publication de l'article « *A relational model of data for large shared data banks* » par Edgar Frank CODD en 1970 [COD 1970]. Ce modèle dont l'objectif est de résoudre le problème de la redondance de données, consiste à organiser les données sous forme de *relations* ou *tables* indépendamment de toutes considérations technologiques. Le modèle relationnel est aujourd'hui la base de nombreux systèmes de gestion de bases de données (SGBD) qui sont appelés *SGBD Relationnels*. Ce chapitre se focalise sur l'étude des concepts pivots des bases de données relationnelles. Nous introduisons des notions sur le modèle relationnel, le schéma d'une base de données relationnelle et les contraintes d'intégrité de données.

2. Notions autour du modèle relationnel

Le modèle relationnel est fondé sur la théorie mathématique des ensembles. Les concepts suivants sont importants pour introduire les bases de données relationnelles : attribut, domaine et relation.

2.1. Attribut

Un attribut A_i est un identificateur (ou nom) décrivant une information stockée dans une base [AUD 2003]. Le nom associé à un attribut doit être porteur de sens. Le *titre* d'un ouvrage, le *nom* d'une personne et le *coefficient* d'un module sont des exemples d'attributs.

2.2. Domaine

Le domaine D_i d'un attribut A_i est l'ensemble dans lequel cet attribut prend valeur [GAR 2003]. Par exemple, l'attribut *coefficient* d'un module a pour domaine un *sous-ensemble de l'ensemble des entiers naturels*. Il est à noter qu'un coefficient négatif ou nul n'a pas de sens. Le domaine de l'attribut *nom* caractérisant une personne est l'ensemble des noms autorisés pour une personne (séquence de caractère alphabétiques accentués ou non et de certains autres symboles tels que l'apostrophe ou le tiret).

Quel rapport existe-t-il entre un *attribut* et un *domaine* ? Un domaine désigne l'ensemble de valeurs possibles d'un attribut, ou à l'inverse, un attribut prend valeur dans un domaine et peut à un instant donné comporter seulement certaines valeurs du domaine. Par exemple, si le domaine est l'ensemble des entiers naturels, seules quelques valeurs seront prises à un instant donné par l'attribut *coefficient* d'un module (ex. {1, …, 6}).

Notons que les valeurs des domaines sont élémentaires. Une valeur d'attribut ne peut alors être un ensemble de valeur. Cette caractéristique est appelée atomicité de valeur d'un attribut.

2.3. Produit cartésien

Le produit cartésien d'un ensemble de n domaines $D_1, \dots, D_i, \dots, D_n$ $(1 \leq i \leq n)$ non nécessairement distincts, noté D_1 x … x D_n, est l'ensemble de toutes les combinaisons possible des valeurs de D_i [HAI 2015]. Par exemple, considérons deux domaines AUTEURS={GARDARIN, SOUTOU} et LIVRES={MySQL, Oracle}. Le produit cartésien D_1 x D_2 comporte quatre combinaisons illustrées dans la Table 2.1.

AUTEURS x LIVRES	
GARDARIN	MySQL
GARDARIN	Oracle
SOUTOU	MySQL

SOUTOU	Oracle

Table 2. 1 – Exemple du produit cartésien de deux domaines.

2.4. Relation

Formellement, une relation est un sous-ensemble du produit cartésien d'une liste de domaines caractérisé par un nom [GAR 2003]. Par exemple, la relation *AUTEURS_LIVRES* représentée dans la Table 2.2 est obtenue à partir des domaines *AUTEURS* et *LIVRES*. Il est à noter que la relation **AUTEURS_LIVRES ⊆ AUTEURS x LIVRES**

AUTEURS_LIVRES	
Auteur	Livre
GARDARIN	MySQL
SOUTOU	Oracle

Table 2. 2 – Exemple d'une relation.

D'un point de vue pratique, une relation est représentée sous forme d'un tableau à deux dimensions, appelée *table*. Une relation est alors une abstraction mathématique d'une table, ou à l'inverse, une table est une représentation matérielle d'une relation. Dans le contexte de ce document, les termes tables et relation sont utilisés indifféremment. Une table est constituée d'un ensemble de *colonnes* ou *attributs*. Chaque colonne dispose d'un nom et deux colonnes ne peuvent avoir le même nom. Le *degré* d'une table représente le nombre de ses attributs. Toute valeur qui apparaît dans une colonne doit appartenir au domaine de cette colonne. Dans une table, il est possible que deux colonnes, voire plus, aient le même domaine. Les données d'une table apparaissent comme un ensemble de *lignes* ou *tuples*. Toutes les lignes d'une table ont le même format et représentent des objets ou des associations du monde réel. La *cardinalité* d'une table représente le nombre de tuples présents dans cette table à un instant donné. Le tableau 2.2 illustre la représentation tabulaire de la relation *AUTEURS_LIVRES* ayant deux colonnes *« auteur »* et *« livre »*.

2.5. Schéma de relation

Un schéma de relation indique le nom de la relation ainsi que ses attributs avec leurs domaines respectifs [GAR 2003]. Ce schéma est noté sous la forme :

$$nomSchéma\ (A_1 : D_1, \dots, A_i :\ D_i, \dots, A_n : D_n)$$

où $nomSchéma$ est le nom de la relation (unique dans la base de données), A_i représente le $i^{ème}$ attribut et D_i le domaine de valeurs de l'attribut A_i. Par exemple, le schéma de relation «OUVRAGES» décrivant les caractéristiques d'un ensemble de livres, est défini comme suit :

OUVRAGES (codeOuv : Chaîne de caractères, titreOuv : Chaîne de caractères, année : Entier).

Généralement, lors de la définition du schéma d'une relation, les domaines de valeurs sont omis puisqu'ils sont implicites et découlent directement des noms d'attributs. Le schéma de relation « *OUVRAGES* » devient alors :

OUVRAGES (codeOuv, titreOuv, année).

Le schéma d'une relation est son ***intention***, c.à.d. les propriétés communes des tuples qu'elle contiendra au cours du temps. Il s'agit de la description des données. En revanche, l'ensemble des tuples, qu'une relation contient à un instant donné, représente une ***extension*** ou une ***instance*** de cette relation.

Par exemple, une extension possible du schéma de relation *OUVRAGES* est décrite dans la Table 2.3.

OUVRAGES		
codeOuv	titreOuv	année
INF74/18	Bases de données et systèmes d'information	2017
INF20/11	Modélisation UML	2005
MEC13/61	Mécanique des fluides	2016
BIO34/70	Biologie moderne	2016

Table 2. 3 – Extension du schéma de relation OUVRAGES.

En désignant par t_i $(1 \leq i \leq 4)$ le *i-ème* tuple de la relation, le tuple t_1 est égal à **(INF74/18, Bases de données et systèmes d'information, 2017)**, alors que $t_1(codeOuv)$ représente la valeur du tuple t_1 associé à l'attribut *codeOuv* qui est égale à **(INF74/18)**.

On dit que deux tuples t et t' sont égaux sur un sous ensemble d'attributs A ($A \subseteq \{A_1, \dots, A_n\}$), s'ils ont les mêmes valeurs pour chacun des attributs de A. Par exemple, les deux tuples t_3 et t_4 sont égaux sur l'attribut *année,* c.à.d. $t_3(\text{année}) = t_4(\text{année})$. En revanche, deux tuples t et t' sont différents sur un sous ensemble d'attributs A ($A \subseteq \{A_1, \dots, A_n\}$), s'ils ont au moins une valeur différente pour les attributs de A. Par exemple, les deux tuples t_1 et t_2 sont différents parce que $t_1(\text{codeOuv}) \neq t_2(\text{codeOuv})$.

2.6. Différentes clés d'une relation

Par définition, une relation est un ensemble de tuples n'ayant pas d'éléments en double, c.à.d. il ne peut exister deux fois le même tuple dans une relation. Pour identifier les tuples d'une relation sans donner les valeurs de tous les attributs et d'assurer simplement l'unicité des tuples, la notion de *clé de relation* est utilisée. Une *clé de relation* est un sous-ensemble minimal d'attributs appartenant au schéma de la relation et dont la connaissance des valeurs permet d'identifier un tuple unique de la relation considérée [GAR 2003]. Une clé est constituée d'un unique attribut ou peut en comporter plusieurs. Formellement, une clé d'une relation R(A_1, …, A_n) est un sous-ensemble d'attributs $C \subseteq \{A_1, \dots, A_n\}$ tel que, quels que soient deux tuples t_1 et t_2 d'une instance de R, $t_1(C) \neq t_2(C)$, c.-à-d. les deux tuples t_1 et t_2 ont des valeurs de C différentes.

Par exemple, pour le schéma de relation *OUVRAGES*, l'attribut *codeOuv* représente une clé parce que chaque livre est différencié par un code unique qui lui est propre. En revanche, l'attribut *titeOuv* ne peut pas être une clé de la relation *OUVRAGES* parce que deux ouvrages peuvent avoir le même titre (par exemple, deux éditions différentes d'un même ouvrage ont le même titre).

Toute relation possède au moins une clé et peut en avoir plusieurs. Lorsqu'une relation admet plusieurs clés, celles-ci sont appelées *clés candidates*. Les clés candidates d'une relation n'ont pas forcément le même nombre d'attributs. Par exemple, pour le schéma de relation :

EMPLOYÉS (matricule, nom, prénom, dateNaissance, numSécuritéSociale)

les attributs *matricule* et *numSécuritéSociale* représentent deux clés candidates parce que tous les employées ont des valeurs différentes de ces attributs.

Une *clé primaire* d'une relation est une clé candidate choisie en général arbitrairement. Pour signaler la clé primaire d'une relation, ses attributs sont généralement soulignés dans le schéma de relation. Étant donné que la clé primaire identifie de manière unique les tuples d'une relation, aucun de ses attributs ne doit avoir la valeur NULL. Dans une relation, une valeur NULL représente des données manquantes ou inconnues et non pas la valeur zéro. Par exemple, dans le cas où l'attribut *matricule* est choisi comme clé primaire, le schéma de relation *EMPLOYÉS* sera représenté comme suit :

EMPLOYÉS (<u>matricule</u>, nom, prénom, dateNaissance, numSécuritéSociale)

Un ensemble d'attributs contenant une clé de relation est appelée *super-clé* [ULL 1988]. Pour le schéma de relation *OUVRAGES*, l'ensemble *{codeOuv, titreOuv}*, par exemple, constitue une super-clé.

2.7. Clé étrangère

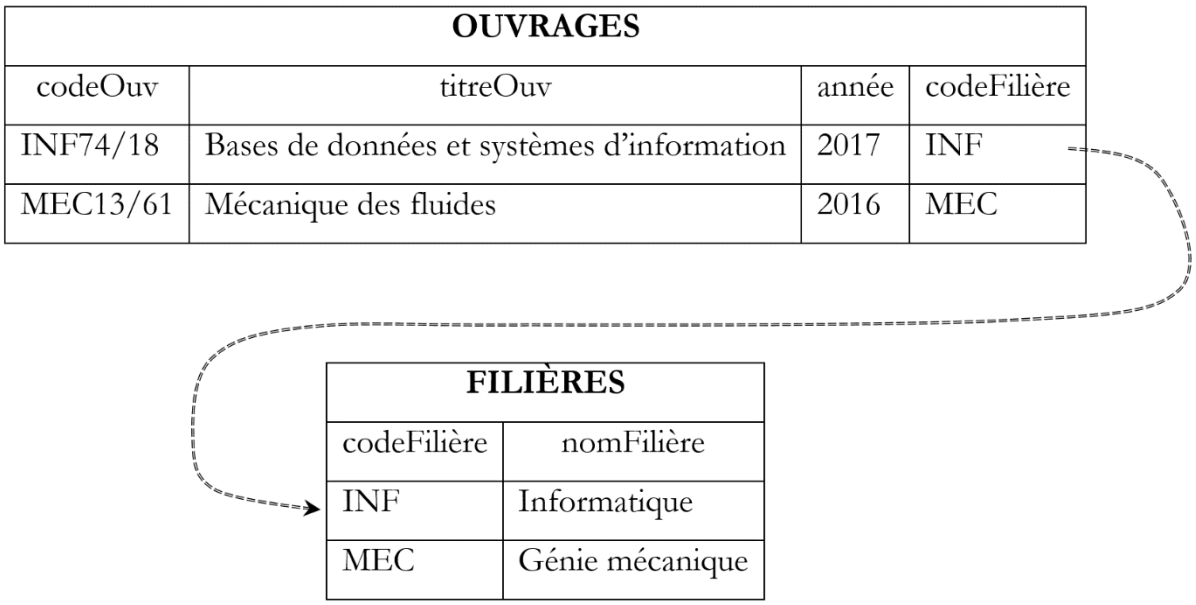
Une *clé étrangère* dans un schéma de relation représente un sous-ensemble d'attributs qui constitue une clé candidate d'un autre schéma de relation [BEN 2017].

Considérons les deux schémas de relations *OUVRAGES* et *FILIÈRES* où nous supposons qu'un ouvrage est classé dans une seule filière :

OUVRAGES (codeOuv, titreOuv, année, codeFilière)

FILIÈRES (codeFilière, nomFilière).

L'attribut *codeFilière* est une clé étrangère dans le schéma de relation *OUVRAGES* parce que cet attribut constitue une clé candidate du schéma *FILIÈRES.* Dans les extensions suivantes des schémas de relations *OUVRAGES* et *FILIÈRES* :



OUVRAGES

codeOuv	titreOuv	année	codeFilière
INF74/18	Bases de données et systèmes d'information	2017	INF
MEC13/61	Mécanique des fluides	2016	MEC

FILIÈRES

codeFilière	nomFilière
INF	Informatique
MEC	Génie mécanique

Fig. 2. 1 – Exemple illustratif du concept de la clé étrangère.

D'un point de vue pratique, la clé étrangère *codeFilière* dans la table *FILIÈRES* matérialise un lien vers la table *OUVRAGES*. En effet, pour chaque ouvrage, la valeur de la clé étrangère permet de récupérer les informations relatives à sa filière. Par exemple, pour avoir les informations de la filière du livre INF74/18, il suffit de trouver la valeur de la clé étrangère (ici INF) dans la clé candidate ; le tuple est *(INF, Informatique).*

Le nom d'une clé étrangère dans un schéma de relation et celui de la clé candidate correspondante peuvent être différents, mais ils doivent avoir le même domaine et la même signification. Par exemple, l'attribut clé étrangère *codeFilière* dans le schéma de relation *OUVRAGES* pourrait être appelé *Filière* :

OUVRAGES (codeOuv, titreOuvr, année, Filière).

FILIÈRES (codeFilière, nomFilière).

L'attribut *Filière* du schéma de relation *OUVRAGES* fait référence à l'attribut *codeFilière* du schéma de relation *FILIÈRES*.

3. Le paradigme de bases de données relationnelles

3.1. Schéma d'une base de données relationnelle

Le schéma d'une base de données est composé d'un ensemble de schémas de relations décrivant un contexte d'étude particulier et liés les uns aux autres à travers des clés étrangères. Par exemple, le schéma de relations suivant décrit une base de données relative à la gestion des emprunts d'ouvrage au sein d'une bibliothèque universitaire.

PERSONNES (numPers, nomPers, prénPers, âge, numSS, ville).

FILIÈRES (codeFilière, nomFilière).

OUVRAGES (codeOuv, titreOuv, année, prix, *codeFilière*).

EMPRUNTS *(Lecteur, Ouvrage,* dateEmprunt, dateRetour*)*.

Les clés étrangères dans les schémas de relations sont mises en *italique-gras*.

3.2. Les contraintes d'intégrité

Une contrainte d'intégrité est une condition qui doit être vérifiée par les données contenues dans une base. Les contraintes d'intégrité sont définies dès la création d'une base et permettent de garantir la cohérence de données lors des mises à jour de la base, c'est-à-dire, les données présentes dans la base sont conformes aux données attendues. En effet, avant d'effectuer une mise à jour (ajout, modification ou suppression), le SGBD vérifie qu'aucune contrainte d'intégrité n'est violée. Dans le cas où au moins une contrainte d'intégrité n'est pas respectée, l'opération de mise à jour est refusée.

3.2.1. Les contraints imposé à un attribut

Définir les valeurs possibles d'un attribut. Il s'agit ici de la spécification du domaine d'un attribut ou d'un ensemble de valeurs spécifiques dans lequel un attribut peut prendre valeurs. Pour le schéma de relation :

PERSONNES (numPers, nomPers, prénPers, âge, numSS, sexe, ville).

La contrainte imposée, par exemple, à l'attribut *numPers* pourrait être *entier* alors que celle imposée à l'attribut *sexe* pourrait être l'une des valeurs dans l'ensemble {homme, femme}.

Valeur requise d'un attribut. Cette contrainte exprime la spécificité qu'un attribut doit avoir explicitement une valeur lors d'une opération de mise à jour de la base de données. Les autres attributs peuvent avoir ou non de valeur, c.-à-d., la valeur de ces attributs peut être manquante. Dans une base de données, il est couramment admis que les valeurs manquantes ou inconnues

représentent des valeurs NULL. Pratiquement, une valeur requise d'un attribut est implémentée par la contrainte NOT NULL en langage SQL.

Par exemple, les attributs *numAdh* et *nomAdh* doivent avoir des valeurs pour chaque tuple présent dans la base. Cependant, nous pouvons avoir des personnes dont l'âge est inconnu.

Valeur par défaut d'un attribut : lors de l'insertion d'un tuple, si la valeur d'un attribut n'est pas spécifiée explicitement, l'attribut recevra une valeur prédéfinie.

3.2.2. Les contraintes référentielles

Une contrainte d'intégrité référentielle impose que la valeur non nulle d'une clé étrangère doit également apparaître dans la clé candidate associée. Les contraintes d'intégrité référentielles sont implémentées par la définition des clés étrangères dans une relation : une clé étrangère est un groupe d'attributs qui doit apparaître comme clé candidate dans une (autre) relation. On peut avoir une relation qui fait référence à elle-même (dans le cas d'une association réflexive).

Dans l'exemple de la section 2.7, la contrainte d'intégrité référentielle est vérifiée parce que les valeurs INF et MEC de la clé étrangère du schéma de relation OUVRAGES apparaissent toutes dans la clé candidate correspondante (c.-à-d., la clé primaire du schéma de relation FILIÈRES).

3.2.3. Les contraintes d'entité

Cette contrainte impose les deux conditions suivantes : (*i*) un schéma de relation doit obligatoirement avoir une clé primaire et (*ii*) tout attribut appartenant à cette clé doit avoir des valeurs non nulles [DAT 1981].

CHAPITRE 3 – Dépendances fonctionnelles et normalisation.

Sommaire

1. Introduction

Les dépendances fonctionnelles constituent une catégorie principale parmi les contraintes d'intégrité parce qu'elles sont à la base du processus de conception des bases de données relationnelles. Ce chapitre définira tout d'abord les dépendances fonctionnelles, leurs propriétés, la notion de clé d'une relation, la couverture minimale, le graphe de dépendances et la fermeture transitive. Ensuite, nous étudions les différentes formes normales dont l'objectif est d'effectuer la décomposition d'une relation sans perdre d'informations en se basant sur la notion de dépendance fonctionnelle. Enfin, nous présentons un algorithme de normalisation par synthèse permettant d'avoir un schéma de base de données non redondant.

2. Le phénomène de redondance

Considérons l'extension du schéma de relation ***LIVRES*** où sont enregistrées les informations sur les livres disponibles dans une bibliothèque[1]. Ces informations inclues le numéro, le titre, l'auteur, le code ISBN et la date d'achat. Un livre qui fait l'objet d'une demande importante de la part des lecteurs, peut être acquis en plusieurs exemplaires, qui font chacun l'objet d'une ligne distincte de la table.

LIVRES				
Numero	Titre	Auteur	ISBN	DateAchat
INF74/18.1	Bases de données et SI	S. BENAÏCHOU	978 2340 0 1782 5	14/10/2018
INF74/18.2	Bases de données et SI	S. BENAÏCHOU	978 2340 0 1782 5	23/01/2019
MEC13/61.1	Mécanique des fluides	R. GHERNAOUT	978 9961 0 1327 4	14/10/2018
BIO10/25.1	Biologie moléculaire	A. MEFTAH	978 2100 0 7344 3	22/2/2009
MEC13/61.2	Mécanique des fluides	R. GHERNAOUT	978 9961 0 1327 4	24/2/2019
MEC13/61.3	Mécanique des fluides	R. GHERNAOUT	978 9961 0 1327 4	24/2/2019

Table 3. 1 – Exemple illustrant le phénomène de la redondance.

Cette représentation pose le problème suivant : lorsqu'un livre existe en plusieurs exemplaires, les tuples décrivant ceux-ci auront les mêmes valeurs du titre, de l'auteur et du code ISBN. Ce phénomène est appelé *redondance d'information* puisqu'une même information est enregistrée plus d'une fois. Sur le plan pratique, en plus de l'occupation d'un espace mémoire inutile, cette redondance engendre des inconvénients, tels que :

- Si l'ajout d'un premier exemplaire d'un livre peut se faire librement, celui des exemplaires suivants doit être conforme aux informations déjà présentes, ce qui complique considérablement la procédure.
- La modification du titre ou de l'auteur d'un livre exigera la même modification des lignes de tous les exemplaires de ce livre, à défaut de quoi les données deviendraient incohérentes.
- La suppression de l'unique (ou de tous les) exemplaire(s) d'un livre entraînerait la perte définitive des informations sur son titre et son auteur.

Les données contenues dans la relation *LIVRES* représentent des informations sur deux types d'entités : les *ouvrages* et les *exemplaires* de ces ouvrages. En effet, on distingue entre

[1] Cet exemple est inspiré de [HAI 2015]

l'ouvrage ***Mécanique des fluides*** écrit par ***R. GHERNAOUT*** ayant l'ISBN ***978 9961 0 1327 4*** de ses trois exemplaires ayant les numéros **MEC13/61.1**, **MEC13/61.2** et **MEC13/61.3** respectivement. Les données spécifiques à un ouvrage sont communes à tous ces exemplaires.

La résolution de ce problème passe par la décomposition de la table LIVRES en deux tables distinctes : OUVRAGES et EXEMPLAIRES. La table OUVRAGES contiendra les informations sur les ouvrages, tels que le code ISBN, le titre et l'auteur. La table EXEMPLAIRES décrit les exemplaires de ces ouvrages. Pour chacun des exemplaires on retiendra le numéro d'exemplaire, le code ISBN de l'ouvrage et la date d'achat de l'exemplaire.

OUVRAGES		
Titre	Auteur	ISBN
Bases de données et SI	S. BENAÏCHOU	978 2340 0 1782 5
Mécanique des fluides	R. GHERNAOUT	978 9961 0 1327 4
Biologie moléculaire	A. MEFTAH	978 2100 0 7344 3

EXEMPLAIRES		
Numero	ISBN	DateAchat
INF74/18.1	978 2340 0 1782 5	14/10/2018
INF74/18.2	978 2340 0 1782 5	23/01/2019
MEC13/61.1	978 9961 0 1327 4	14/10/2018
BIO10/25.1	978 2100 0 7344 3	22/2/2009
MEC13/61.2	978 9961 0 1327 4	24/2/2019
MEC13/61.3	978 9961 0 1327 4	24/2/2019

Table 3. 2 – Décomposition de la relation LIVRES sans redondance d'information.

Ce processus de décomposition en vue d'éliminer ou de limiter la redondance est appelé la *normalisation*. L'objectif est d'éviter la perte de données, les incohérences de données, et l'effondrement des performances de traitements. La normalisation est basée sur le concept de *dépendance fonctionnelle*.

3. Dépendance fonctionnelle

3.1. Définition d'une dépendance fonctionnelle

Soit R (A_1, …, A_n) un schéma de relation avec X et Y deux sous-ensembles de ses attributs. On dit que *X détermine fonctionnellement Y dans R*, ou que *Y dépend fonctionnellement de X dans R*, noté X→Y, si pour toute extension de R, pour tous tuples t_1 et t_2 de R, à chaque fois que $t_1(X) = t_2(X)$ alors $t_1(Y) = t_2(Y)$; c'est-à-dire que si *X détermine fonctionnellement Y dans R* alors lorsque deux tuples de toute extension de R ont des valeurs identiques de X, leurs valeurs de Y doivent être les mêmes [MAT 2002]. Les sous-ensembles d'attributs X et Y sont appelés respectivement *déterminant* et *conclusion* de la dépendance fonctionnelle.

Exemple 3.1

Considérons l'extension du schéma de relation **LIVRES** décrite dans la Table 3.1. Les dépendances fonctionnelles (*i*) Titre→Auteur et (*ii*) ISBN→ DateAchat sont-elles vérifiées par cette extension ?

Solution 3.1

Il n'y a qu'un seul auteur associé à un titre donné. La dépendance fonctionnelle Titre→Auteur est alors satisfaite pas la relation LIVRES.

Considérons les deux tuples t_1 = (INF74/18.1, Bases de données et SI, S. BENAÏCHOU, 978 2340 0 1782 5, 14/10/2018) et t_2 = (INF74/18.2, Bases de données et SI, S. BENAÏCHOU, 978 2340 0 1782 5, 23/01/2019). Nous avons, t_1(ISBN) = t_2(ISBN), mais t_1(DateAchat) ≠ t_2(DateAchat).

Comme ces deux tuples ont les mêmes valeurs de l'attribut *ISBN*, mais leurs valeurs de l'attribut *DateAchat* sont différentes, nous dirons que ISBN ↛ DateAchat.

Il est essentiel de noter que les dépendances fonctionnelles sont des assertions sur le monde réel qui doivent être satisfaites par toutes les extensions d'une relation et non pas par les valeurs actuelles de la relation. La seule manière de déterminer une dépendance fonctionnelle consiste alors d'analyser soigneusement ce que signifient ses attributs. La notation X→Y signifie que *« la relation R satisfait la dépendance fonctionnelle X→Y »* ou que *« la dépendance fonctionnelle X→Y est satisfaite par la relation R »*. L'algorithme 3.1 illustre les étapes à suivre pour déterminer si une extension donnée d'une relation $\mathcal{R}$ satisfait une dépendance fonctionnelle A→B [MAT 2002].

Algorithme 3.1 – Satisfaction d'une dépendance fonctionnelle par l'extension d'une relation.

Entrée : Une extension d'une relation $\mathcal{R}$ composée d'un ensemble de tuples.

Une dépendance fonctionnelle A→B.

DÉBUT

1. Trier les tuple de la relation $\mathcal{R}$ sur les attributs de A tels que les tuples ayant des valeurs égales pour A soient près l'un de l'autre.
2. Vérifier que les tuples ayant des valeurs égales pour les attributs de A ont aussi des valeurs identiques pour B.
3. **Si** au moins un couple de tuples obtenu dans **1.** ne satisfait pas la condition **2. Alors**
4. L'extension de $\mathcal{R}$ ne satisfait pas la dépendance A→B.
5. **Sinon** L'extension de $\mathcal{R}$ satisfait la dépendance A→B.
6. **Fin Si**

FIN

Sortie : La dépendance A→B est satisfaite par l'extension de $\mathcal{R}$ ou non.

Exemple 3.2

Considérons l'extension du schéma de relation **LIVRES** décrite dans la Table 3.1. En utilisant l'algorithme de satisfaction d'une dépendance, monter que la dépendance fonctionnelle *DateAchat→ ISBN* n'est pas satisfaite par cette extension.

Solution 3.2

Au moins deux des valeurs de l'attribut *DateAchat* sont identiques. Cependant, au moins deux des valeurs de l'attribut ISBN sont différentes l'une de l'autre pour des valeurs identiques de l'attribut *DateAchat*. Alors, la dépendance *DateAchat→ ISBN* n'est pas satisfaite pas l'extension de la relation LIVRES.

3.2. Axiomes de déduction des dépendances fonctionnelles

Les *axiomes de déduction* constituent un ensemble de règles permettant de déduire de nouvelles dépendances fonctionnelles à partir d'un ensemble donné de dépendances fonctionnelles satisfaites par une relation. Considérons $\mathcal{R}$ une relation avec X, Y, Z et W des sous-ensembles de ses attributs. L'ensemble des axiomes de déduction, appelés *règles d'inférence d'Armstrong*, est présenté ci-dessous [RAM 2002].

(1) *Réflexivité* : tout ensemble d'attributs détermine fonctionnellement lui-même ou n'importe laquelle de ses parties. Cet axiome produit ce qu'on appelle des *DF triviales*.

Si Y ⊆ X, alors X→Y.

(2) *Augmentation :* le côté gauche d'une dépendance fonctionnelle ou ses deux côtés peuvent être enrichis par un ou plusieurs attributs. L'axiome ne permet pas d'accroître uniquement le côté droit.

Si X→Y, alors XZ→YZ et/ou XZ →Y.

(3) *Transitivité :* lorsqu'un attribut détermine fonctionnellement un second attribut qui, à son tour, détermine un troisième attribut, le premier attribut détermine alors le troisième.

Si X→Y et Y→Z, alors ⇒ X→Z ;

(4) *Pseudo-transitivité :* est une généralisation de l'axiome de transitivité.

Si X→Y et YW→Z, alors XW→Z.

L'axiome de pseudo-transitivité peut être démontré en utilisant les axiomes d'augmentation et de transitivité. En effet, on a X→Y, alors l'axiome d'augmentation nous permet d'avoir XW→YW. À partir de XW→YW et comme on a YW→Z, alors l'axiome de transitivité nous donne W→Z.

(5) *Union :* lorsque deux dépendances fonctionnelles ont le même déterminant, il est possible de former une nouvelle dépendance qui préserve le déterminant et dont la conclusion est obtenue par l'union des conclusions des deux dépendances fonctionnelles.

Si X→Y et X→Z, alors X→YZ.

L'axiome d'union peut être démontré en utilisant les axiomes d'augmentation et de transitivité. En effet, on a X→Y, alors l'axiome d'augmentation permet d'avoir XX→XY. Comme la concaténation représente l'union, c.-à-d., XX = X U X = X, alors X→XY. En outre, on a X→Z, alors l'axiome d'augmentation permet d'avoir XY→ZY. À partir de X→XY et XY→ZY, nous obtenons X→ZY par l'application de l'axiome de transitivité.

(6) *Décomposition :* est l'inverse de l'axiome d'union.

X→YZ ⇒ X→Y et X→Z.

Exemple 3.3

Soit $\mathcal{F}$={A→B, C→D, BD→E} un ensemble de dépendances satisfaites par la relation $\mathcal{R}(A, B, C, D, E)$. En appliquant les axiomes de déduction, monter que la dépendance fonctionnelle AC→E peut être déduite de $\mathcal{F}$.

Solution 3.3

On a A→B, alors l'application de l'axiome d'augmentation permet d'avoir AC→BC. En outre, on a C→D et BD→E, alors nous pouvons avoir la dépendance BC→E par l'application de l'axiome de pseudo-transitivité.

À partir des dépendances AC→BC et BC→E, la dépendance AC→E est obtenue par l'application de l'axiome de transitivité. Alors la dépendance fonctionnelle AC→E est déduite de l'ensemble $\mathcal{F}$.

3.3. La fermeture d'un ensemble d'attributs

Considérons X un ensemble d'attributs et $\mathcal{F}$ un ensemble de dépendances fonctionnelles qui sont définis sur le même schéma de relation $\mathcal{R}$. La *fermeture* de l'ensemble d'attributs X sous $\mathcal{F}$, notée X^+, est l'ensemble des attributs déterminés par X en utilisant les dépendance de $\mathcal{F}$ [BEN 2017]. L'algorithme 3.2 décrit les étapes du calcul de la fermeture d'un ensemble d'attributs[2].

Algorithme 3. 2 – Fermeture d'un ensemble d'attributs.

Entrée : Un ensemble d'attributs X.

Un ensemble de dépendances fonctionnelles $\mathcal{F}$.

DÉBUT

1. Initialiser X^+ à X ;
2. Initialiser un ensemble $\mathcal{F}'$ avec les dépendances fonctionnelles de $\mathcal{F}$;
3. Chercher une dépendance fonctionnelle $A \rightarrow B \in \mathcal{F}'$ telle que $A \subseteq X^+$;
4. **Si** la dépendance $A \rightarrow B$ existe **Faire**
5. **Si** $B \notin X^+$
6. $X^+ = X^+ \cup \{B\}$;
7. **Finsi**
8. $\mathcal{F}' = \mathcal{F}' - \{ A \rightarrow B \}$;
9. Aller à l'instruction **3.**
10. **Finsi**

FIN

Sortie : X^+ la fermeture de X.

[2] Cet algorithme a été adapté de [RAM 2003]

Exemple 3.4

Considérons $\mathcal{F}$={A→B ; A→C ; BC→A ; B→D} un ensemble de dépendances fonctionnelles. En appliquant l'algorithme du calcul de la fermeture d'un ensemble d'attributs, trouver A^+ et $(BC)^+$ dans $\mathcal{F}$.

Solution 3.4

Calculer A^+ dans $\mathcal{F}$

A+={A} ; $\mathcal{F}' = \{A \rightarrow B\ ;\ A \rightarrow C\ ;\ BC \rightarrow A\ ;\ B \rightarrow D\}$;

A→B et $B \notin A^+ \Rightarrow A^+ = \{A,\ B\}$; $\mathcal{F}' = \{A \rightarrow C\ ;\ BC \rightarrow A;\ B \rightarrow D\}$;

A→C et $C \notin A^+ \Rightarrow A^+ = \{A,\ B,\ C\}$; $\mathcal{F}' = \{BC \rightarrow A\ ; B \rightarrow D\}$;

BC→A et $A \in A^+ \Rightarrow A^+ = \{A,\ B,\ C\}$; $\mathcal{F}' = \{B \rightarrow D\}$;

B→D et $D \notin A^+ \Rightarrow A^+ = \{A,\ B,\ C, D\}$; $\mathcal{F}' = \{\}$.

Il n'y a plus de dépendances dans $\mathcal{F}'$, l'algorithme s'arrête et $A^+ = \{A,\ B,\ C, D\}$;

Calculer $(BC)^+$ dans $\mathcal{F}$

(BC)+={B, C} ; $\mathcal{F}' = \{A \rightarrow B\ ;\ A \rightarrow C\ ;\ BC \rightarrow A\ ;\ B \rightarrow D\}$;

$BC \rightarrow A$ et $A \notin (BC)^+ \Rightarrow (BC)^+ = \{A,\ B,\ C\}$; $\mathcal{F}' = \{A \rightarrow B\ ;\ A \rightarrow C\ ;\ B \rightarrow D\}$;

A→B et $B \in (BC)^+ \Rightarrow (BC)^+ = \{A,\ B,\ C\}$; $\mathcal{F}' = \{A \rightarrow C\ ;\ B \rightarrow D\}$;

A→C et $C \in (BC)^+ \Rightarrow (BC)^+ = \{A,\ B,\ C\}$; $\mathcal{F}' = \{B \rightarrow D\}$;

B→D et $D \notin (BC)^+ \Rightarrow (BC)^+ = \{A,\ B,\ C, D\}$; $\mathcal{F}' = \{\}$.

Il n'y a plus de dépendances dans $\mathcal{F}'$, l'algorithme d'arrête et $(BC)^+ = \{A,\ B,\ C, D\}$.

3.4. Typologie des dépendances fonctionnelles

Dépendance fonctionnelle élémentaire

Une dépendance fonctionnelle X→Y est dite *élémentaire*, ou *réduite à gauche*, si pour tout X'⊂X, la dépendance fonctionnelle X'→Y n'est pas vraie, c.-à-d., ∀ X'⊂X, X'↛Y [BOU 1999]. En d'autres termes, Y ne dépend pas fonctionnellement d'une partie de X. Il est à noter que la question de l'élémentarité d'une dépendance fonctionnelle n'est posée que lorsque le déterminant de cette dépendance est constitué de plusieurs attributs.

Exemple 3.5

Dans le schéma de relation PERSONNES (numPers, nomPers, préPers, âge, sexe, ville), la dépendance fonctionnelle *« numPers, nomPers→ préPers »* n'est pas élémentaire parce qu'il suffit du numéro de la personne (numPers) pour déterminer son prénom (préPers).

Dépendance fonctionnelle canonique

Une dépendance fonctionnelle X→Y est dite *canonique* si Y ne comporte qu'un seul attribut et un ensemble $\mathcal{F}$ de dépendances fonctionnelles est dit canonique lorsque toutes ses dépendances fonctionnelles sont canoniques [BOU 1999].

Toute dépendance fonctionnelle non canonique peut être transformée en un ensemble de dépendances canoniques en utilisant l'axiome de décomposition.

Exemple 3.6

Dans le schéma de relation PERSONNES (numPers, nomPers, préPers, âge, sexe, ville), la dépendance fonctionnelle *« numPers → nomPers, préPers »* n'est pas canonique. Toutefois, cette dépendance peut être décomposée en deux dépendances canoniques : *« numPers→nomPers »* et *« numPers→préPers »*.

Dépendance fonctionnelle redondante (déduite)

Étant donné $\mathcal{F}$ un ensemble de dépendances fonctionnelles, on dit que la dépendance fonctionnelle X→Y de $\mathcal{F}$ est *redondante* lorsqu'elle peut être obtenue de l'ensemble de dépendances $\mathcal{F} - \{\mathbf{X} \rightarrow \mathbf{Y}\}$ [MAT 2002]. En d'autres termes, la dépendance X→Y est redondante si on peut avoir Y à partir de X en utilisant les dépendances $\mathcal{F} - \{\mathbf{X} \rightarrow \mathbf{Y}\}$.

Le processus qui consiste à déterminer si une dépendance X→Y est redondante pour un ensemble $\mathcal{F}$ de dépendances peut être fastidieux et long en particulier lorsque cet ensemble comporte un grand nombre de dépendances fonctionnelles. L'algorithme 3.3 présente les étapes à suivre pour déterminer si une dépendance fonctionnelle X→Y est redondance.

Algorithme 3. 3 : Déterminer la redondance d'une dépendance fonctionnelle.

Entrée : Un ensemble $\mathcal{F}$ de dépendances fonctionnelles.

Une dépendance fonctionnelle X→Y dont la redondance est à vérifier.

DÉBUT

1. Calculer X^+ dans $\mathcal{F} - \{\mathbf{X} \rightarrow \mathbf{Y}\}$ en utilisant l'algorithme 3.2.
2. **Si** $Y \in X^+$ **Alors** ;
3. La dépendance X → Y est redondante.
4. **Sinon**
5. La dépendance X → Y n'est pas redondante.
6. **Finsi**

FIN

Sortie : La dépendance X→Y est redondante dans $\mathcal{F}$ ou non.

Exemple 3.7

Considérons le schéma de relation *Scolarité (Cours, Prof, Heure, Salle, Etudiant, Note)* satisfaisant l'ensemble $\mathcal{F}$ de dépendances fonctionnelles tel que :

$\mathcal{F}$={Cours → Prof ; Heure Salle → Cours ; Heure Prof → Salle ; Cours Etudiant → Note ; Heure Etudiant → Salle ; Heure Etudiant → Note}.

Les dépendances « Heure Salle→Cours » et « Heure Etudiant→Note » sont-elles redondantes ?

Solution 3.7

L'application de l'algorithme 3.2 nous permet de calculer (Heure Salle)+ dans $\boldsymbol{\mathcal{F}} -$ {Heure Salle → Cours}. En effet, (Heure Salle)+ ={ Heure, Salle}. On remarque que Cours ∉ (Heure Salle)+. La dépendance « Heure Salle → Cours » n'est pas donc redondante.

L'application de l'algorithme 3.2 nous permet de calculer (Heure Etudiant)+ dans $\boldsymbol{\mathcal{F}} -$ {Heure Etudiant→Note}. En effet, (Heure Etudiant)+ ={Cours, Heure, Salle, Etudiant, Note}. On remarque que Notes ∈ (Heure Etudiant)+. Par conséquent, la dépendance « Heure Etudiant→Note » est redondante.

3.5. Retour sur la notion de clé d'une relation

Considérons R (A_1, …, A_n) un schéma de relation et C un sous-ensemble de ses attributs. De manière formelle, on dira que C est une *clé* du schéma de relation R si C→A_1, …, A_n, c.-à-d., C+ = {A_1, …, An} [BOU 1999].

Le sous-ensemble C est une *clé minimale* du schéma de relation R si les deux conditions suivantes sont satisfaites : (1) X est une clé de R ; (2) ∄ X'⊂X / X'→ A_1, …, A_n [TAH 2016].

La relation R peut avoir plusieurs clés, elles sont dites ***clés candidates***. On en choisit une qui sera appelée ***clé primaire***.

Soit X une clé minimale d'une relation R. On appelle *super clé* de la relation R, un sous-ensemble d'attributs X' de R, telle que : X⊂X', c.-à-d. que X' est une clé ne garantissant pas la contrainte de minimalité.

Exemple 3.8

Considérons le schéma de relation *Scolarité (Cours, Prof, Heure, Salle, Etudiant, Note)* satisfaisant l'ensemble $\mathcal{F}$ de dépendances fonctionnelles tel que :

$\mathcal{F}$={Cours → Prof ; Heure Salle → Cours ; Heure Prof → Salle ; Cours Etudiant → Note ; Heure Etudiant → Salle}.

Montrer que (Heure Etudiant) est une clé minimale de la relation *scolarité*.

Solution 3.8

L'application de l'algorithme 3.2 nous permet de calculer (Heure Etudiant)+ dans $\mathcal{F}$. En effet, (Heure Etudiant)+ = {Cours, Prof, Heure, Salle, Etudiant, Note}.
Par conséquent Heure Etudiant→Cours, Prof, Heure, Salle, Etudiant, Note. L'ensemble d'attribut « Heure Etudiant » est alors une clé du schéma de relation scolarité.

Examinons à présent si la clé « Heure Etudiant » est minimale, c.-à-d., une partie de « Heure Etudiant » détermine les attributs Cours, Prof, Heure, Salle, Etudiant et Note.

En utilisant l'algorithme 3.2, nous trouvons :
Heure+ = {Heure}. Alors, Heure ↛ Cours, Prof, Heure, Salle, Etudiant, Note.
Etudiant+ = {Etudinat}. Alors, Etudiant ↛ Cours, Prof, Heure, Salle, Etudiant, Note

Aucune des parties de la clé « Heure Etudiant » ne détermine tous les attributs de scolarité. Par conséquent, « Heure Etudiant » est une clé minimale.

3.6. Couverture minimale d'un ensemble de dépendances fonctionnelles

Une couverture minimale CM d'un ensemble de dépendances fonctionnelles $\mathcal{F}$ est un sous-ensemble *minimum* de dépendances *élémentaires* à partir duquel il est possible de reconstituer l'ensemble $\mathcal{F}$ par l'application des axiomes d'Armstrong [HAI 2015]. L'algorithme 3.4 décrit les étapes à suivre pour déterminer la couverture minimale CM d'un ensemble $\mathcal{F}$ de dépendances fonctionnelles [TAH 2016].

Algorithme 3.4 – Couverture minimale d'un ensemble de dépendances

Entrée : Un ensemble de dépendances fonctionnelles $\mathcal{F}$

DÉBUT

1. Initialiser CM à $\mathcal{F}$;
   // **Étape 1** : mettre les dépendances de CM sous forme canonique.
2. Toute dépendance $X \rightarrow A_1, \ldots, A_n$ est remplacée par *n* dépendances $X \rightarrow A_i$.
   // **Étape 2** : Réduire à gauche les dépendances non élémentaires.
3. **Pour** toute dépendance $X \rightarrow Y \in CM$ **faire**
4. **Si** $\exists\ Z \subset X$ / $Y \in Z+$ **alors**
5. $CM = CM - \{X \rightarrow Y\} \cup \{Z \rightarrow Y\}$;
6. **Finsi**

7. **Finpour**
   // **Étape 2** : Éliminer les dépendances redondantes.
8. **Pour** toute dépendance X→Y ∈ CM **faire**
9. Calculer $(X)^+$ dans CM – {X→Y} ;
10. **Si** Y ∈ $(X)^+$ **alors**
11. X→Y est redondante ;
12. CM = CM − {X→Y} ;
13. **Finsi**
14. **Finpour**

FIN.

Sortie : La couverture minimale CM.

Tout ensemble de dépendances fonctionnelles a une couverture minimale, mais il peut y'en avoir plusieurs ; cela dépendra de l'ordre des réductions que l'on effectuera.

Exemple 3.9

Considérons un schéma de relation R (A, B, C, D, E, F) satisfaisant l'ensemble $\mathcal{F}$ de dépendances fonctionnelles suivant :

$$\mathcal{F} = \{A \rightarrow B ; CD \rightarrow A ; CB \rightarrow D; AE \rightarrow F; CE \rightarrow D; CDE \rightarrow F \}.$$

Calculer $\mathcal{F}'$ la couverture minimale de $\mathcal{F}$.

Solution 3.9

L'application de l'algorithme 3.4 nous permettra de trouver $\mathcal{F}'$ la couverture minimale de $\mathcal{F}$.

$\mathcal{F}'$={A → B ; CD → A ; CB → D; AE → F; CE → D; CDE → F }.

<u>Étape 1</u> : Forme canonique des $\mathcal{F}$

Les dépendances de l'ensemble $\mathcal{F}'$ sont déjà sous forme canonique.

<u>Étape 2</u> : Réduction à gauche des DF non-élémentaires

La dépendance A → B est élémentaire parce que son déterminant est constitué d'un seul attribut. Vérifions les dépendances CD → A, CB → D, AE → F, CE → D et CDE → F.

On utilisant l'algorithme de calcul de la fermeture d'un ensemble d'attributs, nous aurons :
A+ = {A, B} ; B+={B} ; C+={C} ; D+={D} ; E+={E}.

CD → A est-elle élémentaire ?

A ∉ C+ et A ∉ D+, alors CD→A est élémentaire.

CB → D est-elle élémentaire ?

D ∉ C+ et D ∉ B+, alors CB→D est élémentaire.

AE → F est-elle élémentaire ?

F ∉ A+ et F ∉ E+, alors AE→F est élémentaire.

CE → D est-elle élémentaire ?

D ∉ C+ et D ∉ E+, alors CE → D est élémentaire.

CDE → F est-elle élémentaire ?

(CD)+ = {A, B, C, D} ; (CE)+ = {A, B, C, D, E, F} ; (DE)+ = {D, E} ;

F ∈ (CE)+, alors CDE → F sera remplacée par CE → F.

$\mathcal{F}'$ = {A → B ; CD → A ; CB → D; AE → F; CE → D; CE → F }.

Examinons à présent si la dépendance CE → F est redondante.

F ∉ C+ et F ∉ E+, alors CE → F est élémentaire.

$\mathcal{F}'$ = {A → B ; CD → A ; CB → D; AE → F; CE → D; CE → F }.

<u>Étape 3</u> : éliminer les dépendances redondantes

A→B est-elle redondante ?

On calcule A+ dans $\mathcal{F}'$– {A → B}.

A+= {A}. On remarque que B ∉ A+, alors la dépendance A → B n'est pas redondante.

CD → A est-elle redondante ?

On calcule (CD)+ dans $\mathcal{F}'$ – {CD → A}.

(CD)+= {C, D}. On remarque que A ∉ (CD)+, alors CD→A n'est pas redondante.

CB → D est-elle redondante ?

On calcule (CB)+ dans $\mathcal{F}' - \{CB \rightarrow D\}$.

(CB)+= {B, C}. On remarque que D ∉ (CB)+, alors CB → D n'est pas redondante.

AE → F est-elle redondante ?

On calcule (AE)+ dans $\mathcal{F}' - \{AE \rightarrow F\}$.

(AE)+= {A, B, E}. On remarque que F ∉ (AE)+, alors AE→F n'est pas redondante.

CE → D est-elle redondante ?

On calcule (CE)+ dans $\mathcal{F}' - \{CE \rightarrow D\}$;

(CE)+= {C, E, F}. On remarque que D ∉ (CE)+, alors CE→D n'est pas redondante.

CE → F est-elle redondante ?

On calcule (CE)+ dans $\mathcal{F}' - \{CE \rightarrow F\}$;

(CE)+= { A, B, C, D, E, F}. On remarque que F ∈ (CE)+, alors CE → F est redondante.

Enfin, la couverture minimale de $\mathcal{F}$ est :

$\mathcal{F}' = \{A \rightarrow B ; CD \rightarrow A ; CB \rightarrow D; AE \rightarrow F; CE \rightarrow D\}$.

3.7. Fermeture transitive d'un ensemble de DF

La *fermeture transitive* d'un ensemble $\boldsymbol{\mathcal{F}}$ de dépendances fonctionnelles est l'ensemble $\boldsymbol{\mathcal{F}}^+$ de dépendances fonctionnelles obtenues en ajoutant à $\boldsymbol{\mathcal{F}}$ les dépendances fonctionnelles obtenues par l'application de l'axiome de transitivité [BOU 1999].

Exemple 3.10

Considérons le schéma de relation ENSEIGNANTS (idEns, nomEns, Grade, Salaire) muni de l'ensemble $\mathcal{F}$ de dépendances fonctionnelles, tel que :

$\mathcal{F}$ = {idEns→nomEns ; idEns →préEns ; idEns →Grade, Grade →Salaire} ;

Trouver $\mathcal{F}^+$ la fermeture transitive de $\mathcal{F}$.

Solution 3.10

La fermeture transitive de $\mathcal{F}$ est : $\mathcal{F}^+ = \mathcal{F} \cup$ {idEns→Salaire}.

3.8. Graphe des attributs et des dépendances fonctionnelles

Un ensemble canonique $\mathcal{F}$ de dépendance fonctionnelles satisfaites par une relation peut être représenté à l'aide d'un graphe, appelé *graphe des attributs et des dépendances fonctionnelle*, tel que : (*i*) les somment représentent les attributs impliqués dans les dépendances et (*ii*) les arcs représentent les dépendances elles-mêmes [BOU 1999].

Les arcs sont orientés des sommets représentant les attributs formant le déterminant de la dépendance fonctionnelle vers le sommet représentant l'attribut appartenant à la conclusion de la dépendance. L'origine d'un arc peut être multiple alors que sa cible doit être un sommet unique.

Exemple 3.11

Considérons le schéma de relation ENSEIGNEMENTS (codeModule, intitulé, coefficient, matricule, nomEtud, préEtud, adresse) munie de l'ensemble $\boldsymbol{\mathcal{F}}$ de dépendances fonctionnelles suivant :

$\mathcal{F}$ = {codeModule → intitulé ; codeModule → coefficient ; matricule→ nomEtud ; matricule→préEtud ; matricule → adresse ; matricule codeModule → note}.

Donner le graphe des attributs et des dépendances fonctionnelles du schéma de relation ENSEIGNEMENTS.

Solution 3.11

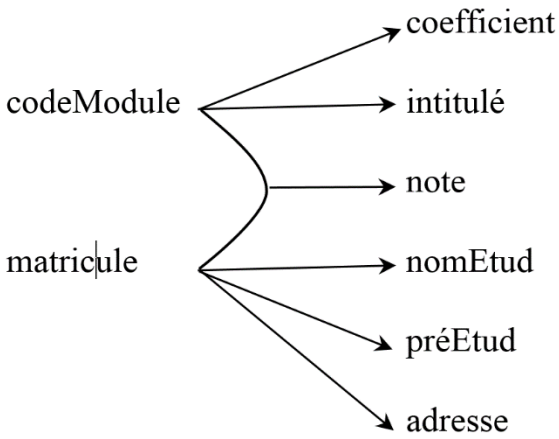


Fig. 3. 1 – Graphe des dépendances fonctionnelles de la relation ENSEIGNEMENTS.

4. Les formes normales

L'objectif des formes normales est d'effectuer la décomposition d'une relation sans perdre d'informations, à partir de la notion de dépendance fonctionnelle [Cod 1972]. Cette décomposition se rapporte à un processus itératif dans lequel une relation est remplacée par un

ensemble de relations dont la structure est plus simple et plus régulière en vue d'éliminer les problèmes de redondance interne dont la relation initiale est éventuellement le siège.

Initialement, E. F. COOD a proposé trois formes normales appelées *première* (abrégée 1NF pour Normal Form), *deuxième* (abrégée 2NF) et *troisième* (abrégée 3NF) formes normales. En plus de ces formes normales initiales, il en existe d'autres telles que, la forme normale de BOYCE-COOD (BCNF – Boyce-Cood Normal Form), la quatrième et la cinquième formes normales. Dans le contexte de ce cours, nous abordons seulement 1NF, 2NF, 3NF et BCNF. Ces formes normales sont représentées sur la figure 3.2 en allant de la moins normalisée à l'extérieur vers la plus normalisée à l'intérieur [MAT 2002].

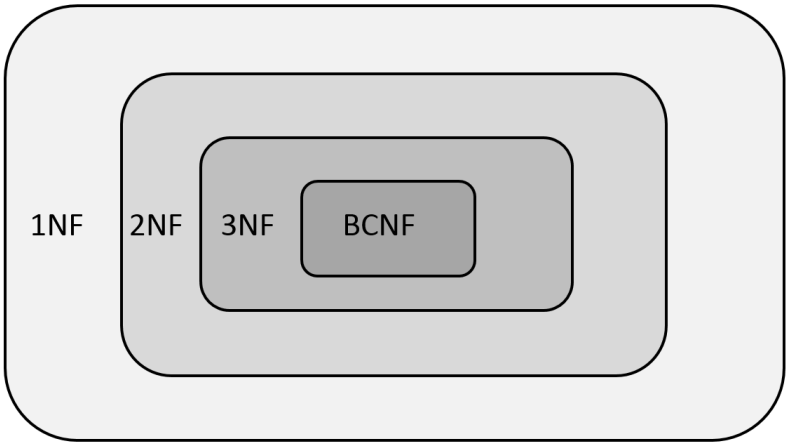


Fig. 3. 2 – Le rapport entre les quatre formes normales.

4.1. Première forme normal

La première forme normale permet simplement d'éviter les attributs multivalués et d'imposer à tout schéma de relation d'avoir des attributs dont le type de données est élémentaire. Cette forme normale exige une certaine présentation pour le schéma d'une relation sans se soucier du problème de redondance inhérent à cette présentation.

> **Définition 3.1.** Un schéma de relation est en *première forme normale* si chacun de ses attributs comporte (au plus[3]) une valeur unique (non multiple) et atomique (non décomposable) pour chaque tuple de la relation [BEN 2017].

Il existe deux approches de normalisation pour transformer une relation qui n'est pas en 1NF en une ou plusieurs relations en 1NF : (*i*) normalisation par aplanissement et (*ii*) normalisation par décomposition.

[3] Un attribut pourrait avoir la valeur NULL pour un tuple donné.

L'approche par aplanissement consiste à retirer les attributs multivaleurs correspondant à des occurrences uniques de l'identifiant de la relation en remplissant les entrées manquantes de chaque tuple incomplet de la relation par des copies des attributs correspondants non répétitifs.

L'approche par décomposition consiste, quant à elle, à scinder la relation initiale en deux nouvelles relations. L'une des deux relations contient la clé primaire de la relation initiale et tous les attributs multivaleurs alors que l'autre relation comportera une copie de la clé primaire de la relation initiale et tous les attributs restants.

Exemple 3.12

Considérons le schéma de relation FACULTÉS (numFac, nonFac, Départements) représentant une partie des différentes facultés de l'université de Béjaia où chacune d'entre elles est composée de plusieurs départements.

FACULTÉS		
numFac	nonFac	Départements
100	Sciences Exactes	{Informatique, Mathématiques, Physique}
101	Technologies	{Génie Civil, Hydraulique}
102	Sciences de la nature et de la vie	{Sciences Alimentaires, Microbiologie}

Table 3. 3 – Exemple d'une relation « FACULTÉS » qui est non en 1NF.

Ce schéma de relation n'est pas en 1NF parce que l'attribut *Départements* comporte un ensemble de valeurs. En utilisant l'approche de normalisation par aplanissement, le schéma de relation initiale sera transformé en un nouveau schéma de relation en 1NF.

FACULTÉS		
numFac	nonFac	Départements
100	Sciences Exactes	Informatique
100	Sciences Exactes	Mathématiques
100	Sciences Exactes	Physique
101	Technologies	Génie Civil
101	Technologies	Hydraulique

102	Sciences de la nature et de la vie	Sciences Alimentaires
102	Sciences de la nature et de la vie	Microbiologie

Table 3. 4 – Approche de normalisation par aplanissement de la relation « FACULTÉS ».

Dans ce nouveau schéma de relation *FACULTÉS*, tout attribut contient une seul valeur atomique pour chaque tuple. Il est à noter que l'attribut *numFac* n'identifie plus de manière unique un tuple de la relation et donc n'est pas une clé primaire de la nouvelle relation *FACULTÉS*. Une clé primaire de cette nouvelle relation est « numFac, Départements »

L'approche de normalisation par décomposition permet de scinder le schéma de relation initiale *FACULTÉS* en deux nouveaux schémas de relation en 1NF comme suit :

FACULTÉS	
numFac	nonFac
100	Sciences Exactes
101	Technologies
102	Sciences de la nature et de la vie

numFac	Départements
100	Informatique
100	Mathématiques
100	Physique
101	Génie Civil
101	Hydraulique
102	Sciences Alimentaires
102	Microbiologie

Table 3. 5 – Approche de normalisation par décomposition de la relation « FACULTÉS ».

4.2. Deuxième forme normale

La deuxième forme normale permet d'éliminer certaines redondances en garantissant qu'aucun attribut n'est déterminé seulement par une partie de la clé.

> **Définition 3.2.** Un schéma de relation est en *deuxième forme normale* si et seulement si (*i*) il est en 1FN et que (*ii*) tout attribut n'appartenant pas à la clé ne dépend pas d'une partie de la clé de ce schéma [AUD 2009].

Dans le cas où un schéma de relation possède plusieurs clés candidates, la définition doit être vérifiée pour chacune d'elles successivement. Un schéma de relation peut alors être en 2NF par rapport à une de ses clés candidates, et ne pas l'être par rapport à une autre.

La représentation typique d'un schéma de relation R(K1, K2, X, Y) non en 2NF est illustré dans la Fig. 3.3 où K1 et K2 désignent deux parties de la clé de R [GAR 2003]. Le problème est que K2→Y, c.à.d. Y dépend d'une partie de la clé, ce qui fait que ce schéma n'est pas en 2NF. Un tel schéma de relation doit être décomposé en deux schémas de relations R1(K1, K2, X) et R2(K2, Y) qui sont tous les deux en 2NF.

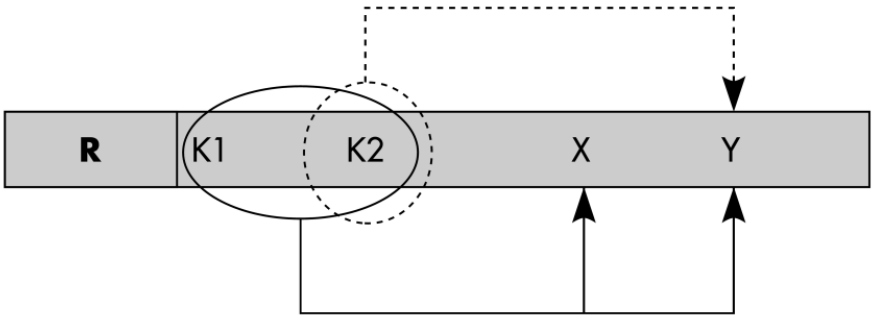


Fig. 3. 3 – Représentation typique d'un schéma de relation non en 2NF.

Exemple 3.13

Considérons le schéma de relation

ÉCRIRE (ISBN, idAuteur, Titre, nomAuteur)

où une personne peut être l'auteur de plusieurs ouvrages et un ouvrage peut être écrit par plusieurs auteurs.

Ce schéma n'est pas en 2NF parce que, par exemple, il suffit de l'ISBN de l'ouvrage pour déterminer son titre ou encore *nomAuteur* ne dépend que de *idAuteur*. En d'autres termes, il y a des attributs n'appartenant pas à la clé du schéma qui dépendent d'une partie de la clé de ce schéma. Pour normaliser ce schéma, il faut le décomposer comme suit :

OUVRAGES (ISBN, Titre)

AUTEURS (idAuteur, nomAuteur)

ÉCRIRE (ISBN, idAuteur)

Les schémas de relations OUVRAGES, AUTEURS et ÉCRIRE sont tous en 2NF.

4.3. Troisième forme normale

La troisième forme normale permet d'éliminer des redondances dues aux dépendances fonctionnelles transitives.

Définition 3.3. Un schéma de relation est en troisième forme normale si et seulement si (*i*) il est en 2FN et (*ii*) tout attribut n'appartenant pas à la clé ne dépend pas d'un autre attribut non clé. Un attribut *non-clé* est un attribut qui n'appartient pas à la clé.

Dans le cas où un schéma de relation possède plusieurs clés candidates, la définition doit être vérifiée pour chacune d'elles successivement. Un schéma de relation peut alors être en 3NF par rapport à une de ses clés candidates, et ne pas l'être par rapport à une autre.

La représentation typique d'un schéma de relation R(K, X, Y, Z) non en 3NF est illustré dans la Fig. 3.4 où K désigne la clé de R [GAR 2003]. Le problème est que X→Y, c.à.d. Y dépend d'un attribut non-clé, ce qui fait que ce schéma n'est pas en 3NF. Un tel schéma de relation doit être décomposé en deux schémas de relations R1(<u>K,</u> X, Y) et R2(<u>X</u>, Y) qui sont tous les deux en 3NF.

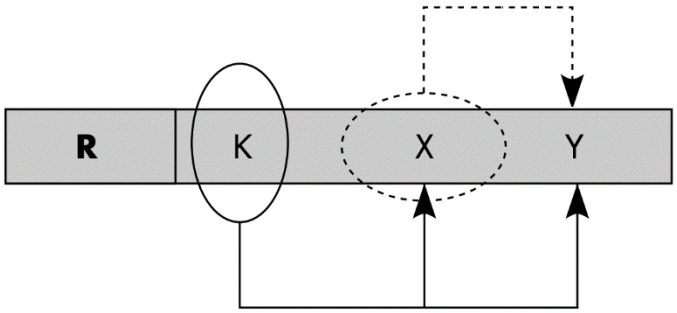


Fig. 3. 4 – Représentation typique d'un schéma de relation non en 3NF.

Exemple 3.14

Considérons le schéma de relation ENSEIGNANTS (<u>numEns</u>, nomEns, préEns, Grade, Salaire) satisfaisant l'ensemble $\mathcal{F}$ de dépendances fonctionnelles suivantes :

$\mathcal{F}$ = {numEns→nomEns ; numEns→préEns ; numEns→Grade ; numEns→Salaire ; Grade→Salaire}.

Ce schéma de relation est en 2NF parce qu'il est en 1NF et tout attribut non-clé dépend totalement de la clé qui est *numEns*. Néanmoins, il n'est pas en 3NF parce que l'attribut non-clé *Grade* détermine *Salaire* qui est aussi un attribut non-clé. Ce schéma est soumis nécessairement au problème de la redondance de données comme le montre la table suivante :

numEns	nomEns	préEns	Grade	Salaire
10	SADOUNE	Lamine	MAA	50000
20	OUALI	Riad	MCB	80000
30	KASSA	Salim	MCB	80000
40	SELMI	AbdErrezak	MAA	50000

Table 3. 6 – Exemple d'extension d'une relation « ENSEIGNEMENTS » non en 3NF.

Pour éliminer cette redondance, le schéma de relation ENSEIGNANTS doit être décomposé en deux schémas de relations en se basant sur la dépendance Grade→Salaire :

GRADES (Grade, Salaire).

ENSEIGNANTS (numEns, nomEns, préEns, *Grade*)

où *Grade* est une clé étrangère qui fait référence au schéma de relation GRADES.

La décomposition a donnée naissance à deux schémas de relations en 3NF où la redondance de données a complétement disparu.

GRADES	
Grade	Salaire
MCB	80000
MAA	50000

ENSEIGNANTS			
numEns	nomEns	préEns	Grade
10	SADOUNE	Lamine	MAA
20	OUALI	Riad	MCB
30	KASSA	Salim	MCB
40	SELMI	AbdErrezak	MAA

Table 3. 7 – Exemple de la relation « ENSEIGNEMENTS » en 3NF.

4.4. Forme normale de Boyce-Codd

Pour éliminer les redondances créées par des dépendances fonctionnelles entre des attributs non-clés et des parties de clés, les chercheurs BOYCE et CODD ont introduit la forme normale qui porte leur nom (en abrégé BCNF) [COD 1974].

Définition 3.4. Un schéma de relation est en forme normale de Boyce-Codd (BCNF) si et seulement si toutes les dépendances fonctionnelles sont de la forme *Clé→Attribut non-clé* [GAR 2003].

Toute relation a une décomposition en BCNF qui est sans perte. Par contre, une décomposition en BCNF ne préserve en général pas les dépendances. La représentation typique d'un schéma de relation qui n'est pas en BCNF est illustrée dans la figure 3.5 où K est la clé de R et Y un attribut non-clé qui détermine une partie de la clé. Un tel schéma de relation doit être décomposé en R1 (K1, K2, X) et R2(Y, K1).

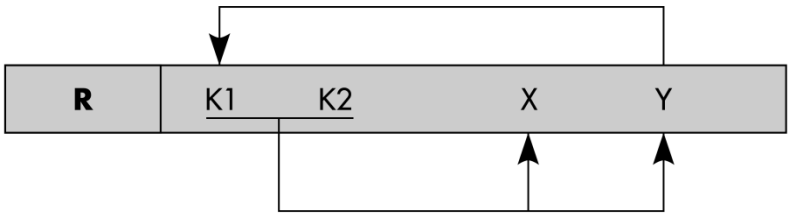


Fig. 3. 5 – Représentation typique d'un schéma de relation non en BCNF.

5. Algorithme de normalisation par synthèse

L'algorithme de normalisation par synthèse permet de passer d'un schéma de relation R(U, F) non en 3NF à une collection de schémas de relations $R_i(U_i, F_i)$ en 3FN. Le schéma R est défini sur un ensemble $U=\{A_1, \ldots A_n\}$ d'attribut et muni d'un ensemble F de dépendances fonctionnelles. Les étapes du processus de normalisation par synthèse sont décrites dans l'algorithme 3.5 [BEN 2017].

Algorithme 3.5. Processus de normalisation par synthèse.

Entrées : $R(A_1, \ldots A_n)$ un schéma de relation.

F un ensemble de dépendances fonctionnelles.

DÉBUT

1. Trouver la couverture minimale F' de F en utilisant l'algorithme 3.4.
2. Calculer les clés minimales de R.
3. Partitionner l'ensemble F' en sous-ensembles $F_1, \ldots, F_p$ tels que les dépendances d'un sous-ensemble ont le même déterminant.
4. **Pour** chaque sous-ensemble F_i **Faire**.
5. Construire une relation R_i (U_i, F_i) dont la clé est le déterminant de F_i.
6. **Finpour**
7. **Si** certains attributs ne figurent dans aucune dépendance de F' **alors**
8. Créer un schéma de relation formé de tous ces attributs.
9. **Finsi**
10. Si K la clé candidate de R ne figure dans aucune des relations R_i obtenues **alors**

11. Ajouter un schéma de relation contenant les attributs de K.

END

Sorties : une collection de schémas R_i (U_i, F_i) en 3NF.

CHAPITRE 4 – Algèbre relationnelle

Sommaire

1. Introduction

L'algèbre relationnelle joue un rôle capital dans les SGBD relationnels dans la mesure où elle représente un support mathématique sur lequel repose le modèle relationnel de données. L'algèbre relationnelle a été introduite par le mathématicien Edgar Frank CODD en 1970. Elle définit un ensemble d'opérateurs formels qu'il est possible d'appliquer aux relations pour créer de nouvelles relations. On peut considérer que l'algèbre relationnelle est aux relations ce qu'est l'arithmétique aux entiers. Dans ce chapitre, nous introduisons à travers des exemples pratiques les principaux opérateurs ainsi que leurs propriétés.

2. Classification des opérateurs algébriques

Il existe plusieurs classifications des opérateurs relationnels qui varient selon le critère utilisé. La classification peut être faite selon le nombre de relations impliquées dans l'opération ou selon le type de l'opérateur.

2.1. Nombre de relations impliquées

Selon le nombre de relations sur lequel ils opèrent, les opérateurs relationnels peuvent être divisés en trois catégories : (*i*) les opérateurs *unaires*, (*ii*) les opérateurs *binaires* et (*iii*) les opérateurs *n-aires* [AUD 2009]. En effet, les opérateurs *unaires* permettant de produire une nouvelle relation à partir d'une seule relation (exemple, *sélection, projection*, etc.). Les opérateurs *binaires* permettent de produire une nouvelle relation à partir de deux relations de même degré et de même domaine (exemple, *union, intersection, etc.)*. Les opérateurs *n-aires* permettent de produire une nouvelle relation à partir de deux ou plusieurs autres relations (exemple, *produit cartésien*, *jointure*, etc.).

2.2. Types d'opérateurs relationnels

En se basant sur le type de l'opérateur, les opérateurs relationnels peuvent être classés en : (*i*) opérateurs *basique* et (*ii*) opérateurs *dérivés* [GAR 2003]. Les opérateurs basiques peuvent être divisés en deux catégories : les opérateurs *ensemblistes* et les opérateurs *spécifiques*. Les opérateurs ensemblistes sont des opérations binaires permettant d'avoir une relation à partir de deux autres relations. Il s'agit de l'union, la différence et le produit cartésien. Les opérations spécifiques sont les opérations unaires de projection et de restriction permettant de créer une nouvelle relation à partir d'une seul relation, et l'opération binaire de jointure. Les opérateurs dérivés peuvent être en général obtenues par combinaison des opérateurs basiques. Parmi ces opérateurs nous pouvons citer l'intersection, la division, le complément, la semi-jointure, la jointure externe et l'éclatement. Dans le contexte de ce cours, nous présentons les opérateurs relationnels suivant cette deuxième classification.

Les notations ne sont pas standardisées en algèbre relationnelle. Nous utilisons ici des notations courantes mais donc pas forcément universelles.

3. Les opérateurs basique

3.1. Les opérations ensemblistes

3.1.1. Union

L'union de deux relations $R(A_1, \ldots, A_n)$ et $S(A_1, \ldots, A_n)$ ayant le même schéma, notée $R \cup S$, est une troisième relation $T(A_1, \ldots, A_n)$ constituée des tuples appartenant à l'une ou l'autre des deux relations R et S sans doublon, c.-à-d., dans le cas où un même tuple existe dans R et S, il

n'apparaîtra qu'une seule fois dans la relation T. L'union est une opération ensembliste commutative, cela signifie que $R \cup S = S \cup R$.

Exemple 4.1

À partir des deux relations PERSONNEL1 et PERSONNEL2. La relation PERSONNEL1 ∪ PERSONNEL2 permet d'avoir les personnes qui sont dans PERSONNEL1 ou celle se trouvant dans PERSONNEL2.

PERSONNEL1			
idPers	nomPers	prePers	depAff
115	SALMI	Mounir	10
123	ADI	Ahlem	20

PERSONNEL2			
idPers	nomPers	prePers	depAff
115	SALMI	Mounir	10
203	DIB	Samir	40

PERSONNEL1 ∪ PERSONNEL2			
idPers	nomPers	prePers	depAff
115	SALMI	Mounir	10
123	ADI	Ahlem	20
203	DIB	Samir	40

3.1.2. Différence

La différence de deux relations $R(A_1, \ldots, A_n)$ et $S(A_1, \ldots, A_n)$ ayant le même schéma, notée $R - S$, est une troisième relation $T(A_1, \ldots, A_n)$ constituée des tuples appartenant à la relations R et ne se trouvant pas dans la relation S. La différence est une opération ensembliste qui n'est pas commutative, cela signifie que $R - S \neq S - R$.

Exemple 4.2

À partir des deux relations PERSONNEL1 et PERSONNEL2 de l'exemple précédent, nous remarquons que PERSONNEL1−PERSONNEL2 est bien différent de PERSONNEL1–PERSONNEL2.

PERSONNEL1 – PERSONNEL2			
idPers	nomPers	prePers	depAff
123	ADI	Ahlem	20

PERSONNEL2 – PERSONNEL1			
idPers	nomPers	prePers	depAff
203	DIB	Samir	40

3.1.3. Produit cartésien

Le produit cartésien de deux relations $R(A_1, \ldots, A_n)$ et $S(B_1, \ldots, B_m)$, n'ayant pas nécessité le même schéma, notée $R \times S$, génère une troisième relation $T(A_1, \ldots, A_n, B_1, \ldots, B_m)$ regroupant toutes les possibilités de combinaison des tuples des relations R et S.

Le degré de la relation T est la somme des degrés de relations R et S. La cardinalité de T, quant à elle, est le produit des cardinalités de relations R et S.

Exemple 4.3

Considérons la relation PERSONNEL comportant les informations du personnel d'un ensemble de départements.

PERSONNEL				
idPers	nomPers	prePers	Ville	depAff
115	SALMI	Mounir	Béjaia	10
123	ADI	Ahlem	Alger	30

DEPARTEMENT		
idDept	nomDept	Lieu
10	Comptabilité	Béjaia
30	Informatique	Oran

PERSONNEL × DEPARTEMENT							
idPers	nomPers	prePers	Ville	depAff	idDept	nomDept	Lieu
115	SALMI	Mounir	Béjaia	10	10	Comptabilité	Béjaia
115	SALMI	Mounir	Béjaia	10	30	Informatique	Oran
123	ADI	Ahlem	Alger	30	10	Comptabilité	Béjaia
123	ADI	Ahlem	Alger	30	30	Informatique	Oran

Le produit cartésien contient des tuples qui n'ont pas de sens. Par exemple, à partir du premier tuple, nous pouvons voir que la personne 115 est affectée au département de comptabilité sis à Béjaia. Par contre, quelle est la signification du second tuple ?

3.2. Les opérations spécifiques

3.2.1. La projection

La projection est un opérateur spécifique aux relations permettant de supprimer un sous-ensemble de ses. Formellement. la projection d'une relation $R(A_1, \ldots, A_n)$ sur un sous-ensemble $A_1, \ldots, A_k$ $(k<n)$ de ses attributs, notée $\pi_{A_1,\ldots,A_k}(R)$, produit une nouvelle relation $S(A_1, \ldots, A_k)$ dans laquelle seuls les attributs $A_1, \ldots, A_k$ sont considérés et les tuples en double sont supprimés.

Exemple 4.4

Considérons la relation PERSONNEL suivante :

PERSONNEL				
<u>idPers</u>	nomPers	prePers	Ville	depAff
115	SALMI	Mounir	Béjaia	10
123	ADI	Ahlem	Alger	20
170	GASMI	Riad	Oran	10
203	DIB	Samir	Béjaia	40

$\pi_{idPers,\ nomPers}(PERSONNEL)$	
<u>idPers</u>	nomPers
115	SALMI
123	ADI
170	GASMI
203	DIB

$\pi_{villePers}(PERSONNEL)$
villePers
Béjaia
Alger
Oran

3.2.2. La sélection

Lorsque la sélection est appliquée à une relation $R(A_1, \ldots, A_n)$, notée $\sigma_{Exp}(R)$, elle génère une nouvelle relation $S(A_1, \ldots, A_n)$ ayant le même schéma que la relation R et regroupant tous les tuples de la R qui satisfont l'expression logique Exp. La sélection est aussi appelée parfois *restriction*.

L'expression logique Exp comporte une ou plusieurs conditions reliées par des opérateurs logiques ($\wedge$ ou $\vee$). Chaque condition est de la forme « $A_i\ operateur\ val_i$ » où « A_i » est un attribut de R, « $operateur$ » un prédicat de comparaison ($>$, $<$, $=$, $\geq$, $\leq$, $\neq$) et val_i une valeur appartenant au domaine de l'attribut A_i.

Exemple 4.5

Considérons l'exemple suivant :

PERSONNEL				
idPers	nomPers	prePers	Ville	depAff
115	SALMI	Mounir	Béjaia	10
123	ADI	Ahlem	Alger	20
170	GASMI	Riad	Oran	10
203	DIB	Samir	Béjaia	40

DEPARTEMENT		
idDept	nomDept	Lieu
10	Comptabilité	Béjaia
20	Commercial	Setif
30	Informatique	Oran
40	Production	Alger

$\sigma_{\text{depAff}=10}(\text{PERSONNEL})$				
idPers	nomPers	prePers	Ville	depAff
115	SALMI	Mounir	Béjaia	10
170	GASMI	Riad	Oran	10

$\sigma_{\text{Lieu}='\text{Alger}'\vee\text{Lieu}='Béjaia'}(\text{DEPARTEMENT})$		
idDept	nomDept	Lieu
10	Comptabilité	Béjaia
40	Production	Alger

3.2.3. La jointure

L'opérateur de jointure permet de combiner deux relations à l'aide d'une condition de jointure. La jointure de deux relations $R(A_1, \ldots, A_n)$ et $S(B_1, \ldots, B_m)$, notée par $R \infty_{Exp} S$, consiste à rapprocher selon une expression logique Exp les tuples de R et S pour former une relation $T(A_1, \ldots, A_n, B_1, \ldots, B_m)$ regroupant toutes les possibilités de combinaison des tuples de R et S vérifiant l'expression Exp.

La jointure peut être vue comme un cas particulier du produit cartésien où une condition ou plusieurs conditions permettent de comparer des attributs. Elle représente alors un produit cartésien suivi d'une sélection :

$$R \infty_{Exp} S = \sigma_{Exp}(R \times S).$$

L'expression logique Exp est un ensemble de conditions reliées par des opérateurs logiques ($\wedge$, $\vee$). Chaque condition est de la forme « A_i $operateur$ B_j » où « A_i » est un attribut de R, « B_j » un attribut de S et « $operateur$ » un prédicat de comparaison (>, <, =, $\geq$, $\leq$, $\neq$).

Bien que les attributs de jointure puissent ne pas avoir le même nom, ils doivent avoir le même domaine et la même signification sous-jacente.

Selon A_i, B_j et $operateur$ on distingue : i) la *theta-jointure*, ii) l'*équi-jointure* et iii) la *jointure naturelle* [AUD 2009].

Theta-jointure : Une theta-jointure est une jointure dans laquelle l'expression logique Exp est une simple comparaison entre un attribut « A_i » de la relation R et un attribut « B_j » de la relation S [AUD 2009].

Exemple 4.6

Considérons les deux relations PERSONNEL et DEPARTEMENT suivante :

PERSONNEL				
idPers	nomPers	prePers	Ville	idDept
115	SALMI	Mounir	Béjaia	10
123	ADI	Ahlem	Alger	20

DEPARTEMENT		
idDept	nomDept	Lieu
10	Comptabilité	Béjaia
20	Commercial	Alger
30	Informatique	Oran

Du fait que les deux relations PERSONNEL et DEPARTEMENT ont un attribut appelé *idDept*, pour éviter de confondre l'attribut *idDept* de la relation PERSONNEL avec celui de la relation DEPARTEMENT, il est nécessaire de qualifier chaque attribut en le faisant précéder du nom de la relation correspondante avant d'effectuer l'opération de jointure. Le produit cartésien des deux relations PERSONNEL $\times$ DEPARTEMENT donne le résultat suivant :

PERSONNEL × DEPARTEMENT							
idPers	nomPers	prePers	Ville	PERSONNEL.idDept	DEPARTEMENT.idDept	nomDept	Lieu
115	SALMI	Mounir	Béjaia	10	10	Comptabilité	Béjaia
115	SALMI	Mounir	Béjaia	10	20	Commercial	Alger
115	SALMI	Mounir	Béjaia	10	30	Informatique	Oran
123	ADI	Ahlem	Alger	20	10	Comptabilité	Béjaia
123	ADI	Ahlem	Alger	20	20	Commercial	Alger
123	ADI	Ahlem	Alger	20	30	Informatique	Oran

La jointure des deux relations suivant la condition « $Ville \neq Lieu$ » permet d'avoir les tuples illustré dans la relation PERSONNEL $\infty_{(Ville \neq Lieu)}$ DEPARTEMENT.

PERSONNEL $\infty_{(Ville \neq Lieu)}$ DEPARTEMENT							
idPers	nomPers	prePers	Ville	PERSONNEL.idDept	DEPARTEMENT.idDept	nomDept	Lieu
115	SALMI	Mounir	Béjaia	10	20	Commercial	Alger
115	SALMI	Mounir	Béjaia	10	30	Informatique	Oran
123	ADI	Ahlem	Alger	20	10	Comptabilité	Béjaia
123	ADI	Ahlem	Alger	20	30	Informatique	Oran

L'équi-jointure : L'*équi-jointure*, appelée aussi *jointure d'égalité*, est une jointure dans laquelle l'expression logique Exp est un test d'égalité entre un attribut « A_i » de la relation R et un attribut « B_j » de la relation S n'ayant pas le même non d'attribut. L' equi-jointure est notée $R \infty_{A_i = B_j} S$.

Exemple 4.7

La jointure des deux relations de l'exemple 4.6 selon la condition « $Ville = Lieu$ » représente une équi-jointure. Elle permet d'avoir les tuples illustrés dans la relation PERSONNEL $\infty_{(Ville = Lieu)}$ DEPARTEMENT.

PERSONNEL$\infty_{(Ville=Lieu)}$ DEPARTEMENT							
idPers	nomPers	prePers	Ville	PERSONNEL.idDept	DEPARTEMENT.idDept	nomDept	Lieu
115	SALMI	Mounir	Béjaia	10	10	Comptabilité	Béjaia
23	ADI	Ahlem	Alger	20	20	Commercial	Alger

La jointure naturelle : La jointure naturelle est une jointure où l'expression logique Exp est un test d'égalité entre les attributs qui portent les mêmes noms dans les relations R et S. Dans la relation résultante, ces attributs ne sont pas dupliqués, mais fusionnés en une seule colonne par couple d'attributs.

La jointure naturelle entre R et S sur un attribut A_1 commun à R et S, est notée $R \infty_{A_1} S$ ou bien $R \infty S$.

Exemple 4.8

La jointure des deux relations de l'exemple 4.6 selon l'attribut «*idDept*» ayant le même nom dans les deux relations représente une jointure naturelle. Elle permet d'avoir les tuples illustrés dans la relation PERSONNEL ∞_{idDept} DEPARTEMENT

PERSONNEL ∞_{idDept} DEPARTEMENT						
idPers	nomPers	prePers	Ville	idDept	nomDept	Lieu
115	SALMI	Mounir	Béjaia	10	Comptabilité	Béjaia
123	ADI	Ahlem	Alger	20	Commercial	Alger

Si R et S possèdent deux attributs portant des noms communs, A_1 et A_2, $R \infty_{A_1} S$ est bien une jointure naturelle sur l'attribut A_1, mais $R \infty S$ est une jointure naturelle sur le couple d'attributs A_1, A_2, ce qui produit un résultat très différent.

PERSONNEL ∞ DEPARTEMENT est une jointure entre les deux relation sur les attributs *ayant le même nom* dans les deux relations.

4. Les opérateurs dérivés

4.1. Intersection

L'intersection de deux relations $R(A_1, ..., A_n)$ et $S(A_1, ..., A_n)$ ayant le même schéma, notée par $R \cap S$, est une troisième relation $T(A_1, ..., A_n)$ constituée des tuples présents dans les deux relations R et S, c.à.d. les tuples communs aux deux relations. L'intersection est une opération ensembliste commutative, cela signifie que $R \cap S = S \cap R$.

Exemple 4.9

À partir des deux relations PERSONNEL1 et PERSONNEL2, la relation PERSONNEL1 ∩ PERSONNEL2 nous permettra d'avoir les personnes qui sont à la fois dans la relation PERSONNEL1 et PERSONNEL2.

PERSONNEL1			
idPers	nomPers	prePers	depAff
115	SALMI	Mounir	10
123	ADI	Ahlem	20

PERSONNEL2			
idPers	nomPers	prePers	depAff
115	SALMI	Mounir	10
203	DIB	Samir	40

PERSONNEL1 ∩ PERSONNEL2			
idPers	nomPers	prePers	depAff
115	SALMI	Mounir	10

4.2. Semi jointure

Lors de l'exécution d'une jointure, il n'est pas nécessaire dans certains cas de conserver tous les attributs des deux relations en résultat, mais seuls les attributs d'une des deux relations sont conservés. L'opérateur spécifique de semi-jointure, très utile pour optimiser l'évaluation des questions, a été introduit [BER 81].

La *semi-jointure* entre R et S, notée $R \propto_{Exp} S$, représente la jointure entre R et S suivie par une projection du résultat sur les attributs de la relation R [DAR 2003]. Il est à noter que Exp a la même définition que pour l'opérateur de jointure.

Exemple 4.10

Considérons les deux relations PERSONNEL et DEPARTEMENT. Quelles sont les personnes habitant une ville dans laquelle se trouve au moins un département ?

PERSONNEL				
idPers	nomPers	prePers	Ville	idDept
115	SALMI	Mounir	Béjaia	10
123	ADI	Ahlem	Alger	20
203	DIB	Samir	Sétif	20

DEPARTEMENT		
idDept	nomDept	Lieu
10	Comptabilité	Béjaia
20	Commercial	Alger
30	Informatique	Oran

PERSONNEL $\propto_{Ville=Lieu}$ DEPARTEMENT				
idPers	nomPers	prePers	Ville	idDept
115	SALMI	Mounir	Béjaia	10
123	ADI	Ahlem	Alger	20

4.3. La division

L'opérateur de division permet de rechercher dans une relation les sous-tuples qui sont complétés par tous ceux d'une autre relation. Elle permet ainsi d'élaborer la réponse à des questions de la forme «quel que soit x, trouver y» de manière simple [GAR 2003].

La division d'une relation R(A_1, …, A_p, A_{p+1},…, An) sur une relation S(A_1, …, A_p), notée par $R \div S$, permet d'avoir une relation T(A_{p+1},…, An) dont les tuples sont ceux qui concaténés à tout tuple de S donnent un tuple de R.

Exemple 4.11

Considérons le schéma d'une base de données modélisant la réalisation d'un ensemble de projets par des personnes. Quel est l'identifiant des personnes ayant participé dans la réalisation de tous les projets ?

PERSONNEL			
idPers	nomPers	prePers	Ville
115	SALMI	Mounir	Béjaia
123	ADI	Ahlem	Alger
170	GASMI	Riad	Oran

PROJET		
idProjet	Intitulé	Localité
10	Réseau Gaz	Oran
11	Pont	Sétif
12	Complexe sportif	Alger

RÉALISER		
idPers	idProjet	nbHeures
115	10	150
123	10	400
170	12	300
123	11	750
123	12	200

Pour trouver tous les projets, on doit effectuer une projection sur la table PROJETS.

$\pi_{idProjt}$(PROJET)
idProjets
10
11
12

Ensuite, on doit trouver pour chaque personne, les projets dans lesquels elle participe.

$\pi_{idPrs,\ ipProjet}$(RÉALISER)	
idPers	idProjet
115	10
123	10
170	12
123	11
123	12

Enfin, on effectue la division des deux résultats précédents :

$\pi_{idPrs,\ ipProjet}$(RÉALISER) $\div$ $\pi_{idProjt}$(PROJET)
idPers
123

4.4. Le complément

Le **complément** est un opérateur permettant de trouver les tuples qui n'appartiennent pas à une relation. Cet opérateur suppose a priori que les domaines des attributs sont finis sinon on obtient des relations infinies.

Le complément d'une relation R(A_1, …, An), noté *NOT(R)*, permet d'avoir une relation de même schéma contenant les tuples du produit cartésien des domaines des attributs de la relation R n'appartenant pas à cette relation.

Exemple 4.12

Considérons la relation PERSONNEL. Les domaines des attributs de cette relation sont : idP = {115, 123, 170} et nomPers = {SALMI, ADI, GASMI}.

PERSONNEL	
idPers	nomPers
115	SALMI
123	ADI
170	GASMI

NOT(PERSONNEL)	
idPers	nomPers
115	ADI
115	GASMI
123	SALMI
123	GASMI
170	SALMI
170	ADI

Il existe d'autres opérateurs dérivés qui sont peu utilisés en pratique, tels que la semi-jointure et l'éclatement. Ces opérateurs ne feront pas l'objet de ce cours.

CHAPITRE 5 – Mise en œuvre d'une base de données relationnelle : Langage SQL.

Sommaire

1. Introduction

Le langage SQL (Structured Query Language) est le langage informatique standard de gestion de bases de données. Ce langage a été initialement introduit au début des années 1970 sous le nom SEQUEL (Structured English QUEry Language). Il a fait ensuite l'objet de plusieurs normes ANSI/ISO dont la plus connue aujourd'hui est la norme SQL2 qui a été définie en 1992. Le langage SQL est le langage structuré de requêtes qui s'est imposé de façon naturelle dans le monde des bases de données. Une *requête* SQL constitue la description d'une opération que le SGBD doit exécuter. Ce chapitre présente le langage SQL notamment ses deux parties principales : (*i*) le Langage de Définition de Données (LDD) et (*ii*) le Langage de Manipulation de Données (LMD).

2. Catégories des instructions SQL

En fonction de leur utilité et des entités manipulées, les instructions du langage SQL se divisent en deux catégories principales ou sous-langage de données : (*i*) le Langage de Définition de Données (LDD) et (*ii*) le Langage de Manipulation de Données (LMD) [MAT 2003]. Toutefois, il existe deux autres catégories de commandes SQL : le L*angage de Protections d'Accès* et le *Langage de Contrôle de Transaction* [AUD 2009].

2.1. Langage de Définition de Données

Les commandes du LDD (Data Definition Language) sont utilisées pour manipuler la structure d'une base de données. Elles permettent de créer, de modifier et de supprimer des objets, tels que des bases de données, une table, une colonne, une contrainte, etc. En d'autres termes, le sous-langage *LDD* inclus les commandes de définition et de modification des objets de la base de données, tels que des tables, des indexes, des vues, etc. Parmi les commandes LDD nous citons CREATE, ALTER, MODIFY, CHANGE, RENAME, DROP, etc.

2.2. Langage de Manipulation de Données

Les commandes du *LMD* (ou Data Manipulation Language) sont utilisées pour manipuler le contenu d'une base de données. Elles permettent l'ajout, la modification, la suppression et l'interrogation des données de la base. En d'autres termes, le sous-langage *LMD* comporte des commandes permettant de manipuler le contenu d'une base de données. Parmi les commandes LMD, nous citées : INSERT INTO, UPDATE, DELETE et SELECT … FROM.

2.3. Langage de Protections d'Accès

Les commandes du LPA (ou Data Control Language) s'occupent de gérer les droits d'accès aux tables. Les instructions du LPA sont : GRANT et REVOKE.

2.4. Langage de Contrôle de Transaction

Les commandes du TCL (Transaction Control Language) gèrent les modifications faites par les langages LDD et LMD (c.-à-d. la validation et l'annulation des modifications). Les instructions du TCL sont : COMMIT, SAVEPOINT, ROLLBACK, SET TRANSACTION.

Dans le présent document, nous nous focalisons d'avantage sur les deux premières catégories d'instructions (LDD et LMD).

3. Les types de données

Les types de données offerts par le langage SQL sont généralement divisés en quatre catégories : les types numériques, les types chaînes de caractères, les types temporels et les

types bits [BEN 2017]. Ces types sont à utiliser lors de la déclaration des colonnes d'une table. Nous citons ici les types principaux parmi les quatre catégories précitées.

- *INTEGER* (ou *INT*) définit des entiers signés long de 4 octets (ou 4 Bytes) ;
- *DECIMAL (m, n)* pour stocker un nombre décimal de *m* chiffres dont *n* est le nombre de chiffres après la virgule.
- *FLOAT(p)* (ou *REAL*) représente les nombres à virgule flottante d'au moins *p* bits significatifs.
- *CHAR(p)* permet de stocker une chaîne de longueur fixe *p*. Cette longueur est inférieur à 255, sa valeur par défaut est 1. Une chaîne de caractères est une suite quelconque de caractères (nombre, lettre ou tout autre caractère spécial).
- *VARCHAR(p)* permet de stocker une chaîne de longueur variable d'au plus *p* caractères. La longueur de la chaîne doit être inférieur à 2000, il n'y a pas de valeur par défaut.
- *DATE* permet de stocker des dates (année, mois et jour).
- *BIT(p)* chaîne de bits de taille fixe d'une longueur *p*.

4. Base de données exemple

Considérons la base de données ***« Bibliothèque »*** décrivant la gestion des ouvrages dans une bibliothèque universitaire. Nous utilisons cette base tout au long de ce chapitre pour expliquer les différentes commandes du langage SQL.

PERSONNES (numPers, nomPers, préPers, âge, numSS, ville).

FILIÈRES (codeFilière, nomFilière).

OUVRAGES (codeOuv, titreOuvr, année, prix, *codeFilière*).

ÉCRIRE *(Auteur, Ouvrage,* position).

EMPRUNTS *(Lecteur, Ouvrage,* dateEmprunt, dateRetour*).*

Le schéma de relation PERSONNES renseigne les informations d'une personne pouvant être un lecteur ou un auteur. Une personne est décrite par un numéro qui lui est propre, un nom devant être renseigné, un prénom, un âge devant être compris entre 17 et 60 ans, un numéro de sécurité sociale qui est unique et une ville.

Le schéma de relation FILIÈRES inclut les informations de la filière d'un ouvrage (informatique, biologie, génie civil, etc.). On y trouve un code un nom de filière.

Le schéma de relation OUVRAGES renseigne le code de l'ouvrage, son titre qui doit être unique, l'année de sortie de l'ouvrage et son prix de vente. L'attribut *codeFilière* est une clé étrangère faisant référence au schéma de relation FILIÈRES.

Le schéma de relation ÉCRIRE lie les ouvrages à leurs auteurs en définissant une position pour chaque auteur (1 : auteur principal, 2 : deuxième auteur, etc.). Les attributs *Auteur* et *Ouvrage* font référence respectivement aux schémas de relations PERSONNES et OUVRAGES.

Le schéma de relation EMPRUNTS comprend les informations sur les emprunts effectués par les lecteurs. Il est à noter qu'un lecteur peut emprunter le même livre à deux dates différentes. Pour un emprunt, on enregistre la date d'emprunt et la date de retour. Les attributs *Lecteur* et *Ouvrage* font référence respectivement aux schémas de relations PERSONNES et OUVRAGES.

5. Langage de définition de données

5.1. Création/Suppression d'une base de données

La requête CREATE DATABASE permet la création d'une base de données relationnelle. La syntaxe de cette requête est la suivante :

CREATE DATABASE [IF NOT EXISTS] nomBD ;

L'option « IF NOT EXISTS » est facultative. Elle permet d'éviter une erreur si la base de données *nomBD* existe déjà. Si tel est le cas, cette base sera supprimée et créée à nouveau.

Exemple 5.1

Pour créer la base de données « Bibliothèque », on exécute la requête suivante :

CREATE DATABASE Bibliothèque ;

La requête DROP DATABASE permet de supprimer une base de données existante. Sa syntaxe se présente sous la forme :

DROP DATABASE [IF EXISTS] nomBD ;

L'option « IF EXISTS » est facultative et permet d'éviter une erreur si la base de données *nomBD* n'existe pas.

Exemple 5.2

Pour supprimer la base de données « Bibliothèque » créée précédemment, on exécute la requête :

DROP DATABASE Bibliothèque ;

5.2. Contraintes sur les colonnes d'une table

Les *contraintes d'intégrité* imposent des conditions qui doivent être satisfaites par les valeurs stockées dans les colonnes d'une table. Par exemple, lors de la création d'une table *OUVRAGES*, il est en toute logique de préciser que la colonne décrivant le prix d'un ouvrage

soit un entier positif. Les contraintes d'intégrité sous SQL sont au nombre de six : PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL et DEFAULT.

5.2.1. Contrainte de la clé primaire

Dans une commande SQL, la ligne suivante :

... PRIMARY KEY (col1, col2,...)

indique la clé primaire d'une table. Cette clé peut être composée d'une seule colonne ou de plusieurs. Aucune des colonnes faisant partie de cette clé ne doit avoir une valeur NULL.

5.2.2. Contrainte d'unicité

La ligne de commande suivante:

... UNIQUE (col1, col2,...)

permet de définir une contrainte d'unicité pour les colonnes *col1, col2,…* d'une table. Une telle contrainte garantit que deux lignes de cette table ne peuvent en aucun cas avoir les mêmes valeurs pour ces colonnes.

5.2.3. Contrainte de la clé étrangère

Dans une commande SQL, la ligne :

… FOREIGN KEY (col1 [, col2 …])

REFERENCES tableReference (colRef1 [, colRef2…])

[ON DELETE {CASCADE | SET NULL | RESTRICT}

ON UPDATE {CASCADE | SET NULL | RESTRICT}]

permet de déclarer une contrainte d'intégrité référentielle pour les colonnes *col1, col2…* de la table en cours de définition. Les valeurs prises par ces colonnes doivent exister parmi les valeurs des colonnes de la table *tableReference* possédant une contrainte PRIMARY KEY ou UNIQUE.

La clause *FOREIGN KEY* indique que la concaténation des attributs *col1, col2,…* est une clé étrangère faisant référence à la concaténation des attributs *colRef1, colRef2,…* de la table *tableReference.*

[ON DELETE {CASCADE | SET NUL | RESTRICT}

ON UPDATE {CASCADE | SET NUL | RESTRICT}] indique au SGBD l'opération à réaliser lorsque les valeurs correspondant à la clé étrangère (tuples dans la table *tableReference*) sont supprimées ou modifiées.

5.2.4. Contrainte de vérification

La ligne de commande suivante :

... CHECK (col condition) ;

définit une contrainte de vérification qui permet de spécifier que la valeur d'une colonne particulière *col* doit satisfaire une condition booléenne.

5.2.5. Contrainte de non nullité

Dans une commande SQL, la ligne suivante :

... col type NOT NULL ;

déclare une contrainte de non nullité qui impose que la valeur d'une colonne soit renseignée (c.-à-d. ne soit pas NULL). Par exemple, dans le cas d'un *ouvrage*, il est cohérent d'imposer que le titre soit toujours renseigné. Il est à noter que par défaut (c.à.d. si on ne spécifie aucune valeur), toute colonne est facultative. Le caractère obligatoire d'une colonne se déclarera alors par la clause *NOT NULL*.

5.2.6. Contrainte de valeur par défaut

Dans la ligne de commande suivante:

... col type DEFAULT val ;

val est une valeur par défaut que le SGBD assignera automatiquement à la colonne *col* lorsque l'utilisateur omettra d'en fournir une lors de l'insertion d'un tuple.

5.3. Création/Suppression d'une table

La requête CREATE TABLE permet de créer une table dans une base de données relationnelle. La syntaxe générale de cette requête est la suivante :

CREATE TABLE nomTable (col1 type1 [Contraintes1], col2 type2 [Contraintes2] ...);

où « *nomTable* » est le nom attribué à la table créée ; *col1, col2, ... représentent* les noms des colonnes ; *type1, type2,...* représentent les types des données contenues dans les colonnes *col1, col2,...* respectivement ; *[Contraintesi]* représente la contrainte d'intégrité imposée sur la

colonne *coli* pouvant être PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL ou DEFAULT.

Exemple 5.3

Pour créer la table PERSONNES, la requête suivante est exécutée :

```
CREATE TABLE PERSONNES (
    numPers INT PRIMARY KEY,
    nomPers VARCHAR(30) NOT NULL,
    préPers VARCHAR(30),
    âge FLOAT,
    numSS VARCHAR(15),
    ville VARCHAR(30) ) ;
```

La requête DROP TABLE permet de supprimer la structure d'une table ainsi que tous les tuples qu'elle contient. Sa syntaxe générale se présente sous la forme :

DROP TABLE nomTable [{CASCADE | RESTRICT}] ;

où « *nomTable* » est le nom de la table à supprimer. L'option *CASCADE* signifie que les objets dépendant de la table supprimée (contraintes, etc.) seront automatiquement supprimés. L'option *RESTRICT* est l'option par défaut. Elle signifie que la table n'est pas supprimée si un objet en dépend.

Exemple 5.4

Pour supprimer la table PERSONNES de la base Bibliothèque, la requête suivante est exécutée :

```
DROP TABLE PERSONNES CASCADE ;
```

5.4. Modification de la structure d'une table

La commande ALTER TABLE est utilisée pour modifier la structure (colonnes, types de colonnes, contraintes, etc.) d'une table existante. Il peut s'agir de l'ajout/suppression d'une colonne, de l'ajout/suppression d'une contrainte ou de la modification du type/nom d'une colonne.

5.4.1. Ajout/Suppression d'une colonne

La syntaxe de la commande permettant l'ajout d'une colonne à une table est la suivante :

ALTER TABLE nomTable ADD [COLUMN] col type [Contraintes]

où *Col* et *type* représentent, respectivement, le nom et le type de la colonne à ajouter.

Exemple 5.5

Ajouter une colonne « numTéléphone » à la table « PERSONNES ».

```
ALTER TABLE PERSONNES ADD numTéléphone  VARCHAR(15) ;
```

La syntaxe de la commande permettant de supprimer une colonne dans une table est :

ALTER TABLE nomTable DROP [COLUMN] col ;

Exemple 5.6

Supprimer la colonne « numTéléphone » de la table « PERSONNES ».

ALTER TABLE PERSONNES DROP numTéléphone ;

5.4.2. Ajout/Suppression d'une contrainte

La syntaxe de la commande permettant d'ajouter une contrainte imposée à la colonne *col* est :

ALTER TABLE nomTable ADD [CONSTRAINT nomConst] Contrainte col [autresOptions] ;

où *nomConst* représente le nom attribué explicitement à la contrainte par la clause *CONSTRAINT*.

Contrainte est le type de la contrainte (PRIMARY KEY, FOREIGN KEY, UNIQUE, etc.) définie sur la colonne *col*.

autresOptions concerne les contrainte d'intégrité référentielle.

Exemple 5.7

Écrire la requête permettant de définir *codeFilière* comme étant la clé étrangère de la table OUVRAGES. On suppose que la table FILIÈRES a été déjà créée.

ALTER TABLE OUVRAGES ADD CONSTRAINT fkOuvrages FOREIGN KEY(codeFilière) REFERENCES FILIÈRES Personnes(codeFilière) ;

La commande suivante permettra de supprimer une contrainte nommée sur une colonne :

ALTER TABLE nomTable DROP CONSTRAINT nomConst ;

où *nomConst* est le nom attribué à la contrainte lors de sa création et *CONSTRAINT* représente le type de la contrainte à supprimer (FOREIGN KEY, UNIQUE, etc.).

Exemple 5.8

Si on veut supprimer la contrainte de clé étrangère *fkOuvrages* de la table OUVRAGES qui a été définie sur l'attribut *codeFilière*, la requête suivante est exécutée :

```
ALTER TABLE OUVRAGES DROP FOREIGN KEY fkOuvrages ;
```

5.4.3. Modifier les caractéristiques d'une colonne

La commande suivante permet de modifier le type d'une colonne d'une table:

ALTER TABLE nomTable MODIFY [COLUMN] col nouveauType [Contraintes] ;

Pour modifier le nom d'une colonne au sein d'une table, la commande suivante est utilisée :

ALTER TABLE nomTable CHANGE ancienNomCol nouveauNomCol type [Contraintes] ;

5.4.4. Renommer une table

Pour modifier le nom d'une table, la commande suivante est utilisée :

ALTER TABLE nomTable RENAME TO nouveauNomTable ;

où *nomTable* et *nouveauNomTable* représentent, respectivement, le nom actuel et le nouveau nom de la table *nonTable*.

6. Langage de manipulation de données

6.1. Les mises à jour de données

Trois commandes sont utilisées pour la mise à jour des données contenues dans une base : INSERT pour l'insertion, UPDATE pour la modification et DELETE/TRUNCATE pour la suppression. Nous détaillons ces commandes dans la présente section à travers des exemples.

6.1.1. Remplir une table

Après la création d'une table, il est possible d'y ajouter des tuples par la commande INSERT INTO en spécifiant les valeurs à insérer. La syntaxe générale de cette commande est :

INSERT INTO *nomTable [(col$_1$, col$_2$, …, col$_n$)]*

VALUES *(val$_1$, val$_2$, …, val$_n$) ;*

La liste des valeurs *« val$_1$, val$_2$, …, val$_n$ »* à insérer doit être en correspondance avec la liste des colonnes *« col$_1$, col$_2$, …, col$_n$ »*.

Dans le cas où la liste des noms de colonnes *col1, col2,…* est omise, l'ordre utilisé pour l'insertion est celui spécifié lors de la création de la table. En revanche, dans le cas où une liste est spécifiée, les colonnes n'y figurant pas recevront la valeur NULL ou, le cas échéant, la valeur par défaut.

Exemple 5.9

Insérer la personne Laurent AUDIBERT de numéro 10, âgé de 42 ans, habitant Paris et ayant 121A310 comme numéro de sécurité sociale.

INSERT INTO PERSONNES VALUES (10, 'AUDIBERT', 'Laurent', 42, '121A310', 'Paris') ;

INSERT INTO PERSONNES(numPers, nomPers, préPers, âge, numSS) VALUES (11, 'GARDARIN', 'Georges', 59, '311B870') permet d'insérer la valeur NULL dans la colonne « ville » ;

Il est possible d'insérer dans une table des tuples provenant du résultat d'une requête. La Syntaxe est : INSERT INTO nomTable [(col1, col2, …, coln)] VALUES Requête_SELECT ;

6.1.2. Modifier le contenu d'une table

La commande *UPDATE* permet de mettre à jour une ou plusieurs valeurs sur un même tuple d'une table vérifiant éventuellement une expression logique donnée (*prédicat*). La syntaxe générale de cette commande est :

UPDATE *nomTable*

SET *col1 = expr1 [, col2 = expr2…]*

*[**WHERE** predicat] ;*

où *col1, col2, …*représentent les noms de colonnes à mettre à jour et *expr1, expr2, …*sont les nouvelles valeurs à stocker dans les colonnes correspondantes. Dans le cas où la clause *WHERE* est omise, tous les tuplets de la table seront mis à jour.

Exemple 5.10

On suppose que Laurent AUDIBERT a déménagé pour s'installer à Nice. Mettre à jour cette information concernant la ville de Laurent AUDIBERT.

```
UPDATE PERSONNES
SET ville = 'Nice'
WHERE numPers = 10;
```

6.1.3. Supprimer des tuples

La commande *DELETE* permet de supprime des tuples d'une table vérifiant une expression donnée (*prédicat*) ou elle peut être utilisée pour supprimer tous les tuples d'une table donnée. La syntaxe générale de cette commande est :

```
DELETE FROM nomTable
[WHERE predicat];
```

Dans le cas où la clause *WHERE* est omise, tous les tuples seront supprimés.

Exemple 5.11

Supprimer toutes les personnes habitant Béjaia.

```
DELETE FROM PERSONNES
WHERE ville ='Béjaia';
```

6.1.4. Vider rune table

Un moyen simple et rapide de supprimer tous le contenu d'une table consiste à utiliser la commande TRUNCATE dont la syntaxe générale est :

```
TRUNCATE TABLE nonTable ;
```

Exemple 5.12

Supprimer toutes les personnes de la table PERSONNES.

```
RUNCATE TABLE PERSONNES;
```

6.2. Interrogation d'une base de données

6.2.1. Commande SELECT … FROM

La commande SELECT … FROM sert à interroger une base de données pour répondre à des questions particulières. La syntaxe générale de cette commande est la suivante :

SELECT [DISTINCT | ALL] col1, col2, ... // Colonnes à afficher

FROM nomTable [, nomTable2, ...] // Table(s) d'où sont récupérées les colonnes

[WHERE conditions] ; // Condition à satisfaire

Une commande SELECT … FROM comporte au moins deux clauses obligatoire : SELECT et FROM. La clause WHERE (et bien évidemment d'autres clauses qui seront vues ultérieurement) est facultative.

- *SELECT* permet de spécifier les colonnes à afficher dans la table résultante. Lorsque le caractère *étoile (*)* est indiqué à la place de *« col1, col2, ... »*, toutes les colonnes de la (des) table(s) indiquée(s) dans la clause FROM seront affichées.
- FROM spécifie la (les) table(s) sur laquelle (lesquelles) porte la requête. Cette clause réalise le produit cartésien si plusieurs tables y sont spécifiées.
- WHERE définie les conditions que les tuples de la table résultante doivent satisfaire. Cette clause réalise l'opération de *sélection*.

Exemple 5.13

Afficher le numéro, le nom et l'âge des personnes qui habitent la ville de Béjaia.

SELECT numPers, nomPers, âge

FROM PERSONNES

WHERE ville = 'Béjaia' ;

6.2.2. Implémentation des opérateurs relationnels

Nous considérons différentes variantes de la commande SELECT … FROM pour illustrer l'implémentation des opérateurs relationnels vus dans le chapitre 4. Le tableau suivant donne pour, chaque opérateur relationnel, l'implémentation correspondante en SQL avec un exemple pratique.

Opérateurs	**Expression en algèbre**	**Implémentation en SQL**
Projection	$\pi_{A_1,\ldots,A_k}(\text{Relation})$	SELECT $A_1, \ldots, A_k$ FROM Relation
Exemple 5.14 Afficher le code et le titre de tous les ouvrages. SELECT codeOuv, titreOuvr FROM OUVRAGES ;		

Sélection	σ_{Exp}(Relation)	SELECT * FROM Relation WHERE Exp

Exemple 5.15

Afficher les ouvrage de la filière *INF*.

```
SELECT *
FROM OUVRAGES
WHERE codeFilière='INF'
```

Exemple 5.16

Afficher le numéro et le nom des personnes habitant la ville de Béjaia.

```
SELECT numPers, nomPers
FROM PERSONNES
WHERE ville = 'Béjaia' ;
```

Équi-jointure	Relation1 $\infty_{A_1=B_1}$ Relation2	SELECT * FROM Relation1, Relation2 WHERE Relation1.A_1= Relation2.B_1

Exemple 5.17

Trouver les ouvrages (code, titre, année) qui ont été empruntés au moins une fois.

```
SELECT codeOuv, titreOuvr, année
FROM OUVRAGES, EMPRUNTS
WHERE codeOuv=Ouvrage ;
```

Exemple 5.18

Afficher les informations (numéro, nom, prénom) de l'auteur principal des ouvrages de la filière *INF*.

```
SELECT PERSONNES.numPers, PERSONNES.nomPers, PERSONNES.prénPers
FROM PERSONNES, ÉCRIRE, OUVRAGES
WHERE numPers=Auteur AND position=1
        AND Ouvrage=codeOuv AND codeFilière='INF' ;
```

Union	R1 ∪ R2	SELECT * FROM R1 UNION SELECT * FROM R2

Exemple 5.19

Afficher les informations des personnes âgées de plus de 45 ans ou qui habitent la ville de Béjaia.

```
SELECT *
FROM PERSONNES
WHERE ville = 'Béjaia'
UNION
SELECT *
FROM PERSONNES
WHERE âge > 45 ;
```

Intersection	R1 ∩ R2	SELECT * FROM R1 INTERSECT SELECT * FROM R2

Exemple 5.20

Trouver les ouvrages (code, titre) de filière *INF* et dont le prix n'excède pas 7000 DZD.

```
SELECT codeOuv, titreOuvr
FROM OUVRAGES
WHERE codeFilière='INF' ;
INTERSACT
SELECT codeOuv, titreOuvr
FROM OUVRAGES
WHERE prix ≤ 7000 ;
```

<table>
<tr><td colspan="3">Exemple 5.21
Afficher les auteurs (numéro et nom) qui sont également lecteurs.
SELECT PERSONNES.numPers, PERSONNES.nomPers
FROM PERSONNES, ÉCRIRE
WHERE numPers=Auteur
INTERSACT
SELECT PERSONNES.numPers, PERSONNES.nomPers
FROM PERSONNES, EMPRUNTS
WHERE numPers= Lecteur ;</td></tr>
<tr><td>Différence</td><td>R1 – R2</td><td>SELECT *
FROM R1
MINUS
SELECT *
FROM R2</td></tr>
<tr><td colspan="3">Exemple 5.22
Afficher les auteurs (numéro, nom et âge) qui ne sont pas également lecteurs.
SELECT PERSONNES.numPers, PERSONNES.nomPers, PERSONNES.âge
FROM PERSONNES, ÉCRIRE
WHERE numPers=Auteur
MINUS
SELECT PERSONNES.numPers, PERSONNES.nomPers, PERSONNES.âge
FROM PERSONNES, EMPRUNTS
WHERE numPers= Lecteur ;</td></tr>
</table>

Table 5. 1 – Implémentation des opérateurs relationnels en SQL.

6.3. Colonnes dans une liste ou une plage de valeurs

Pour sélectionner des tuples d'après les valeurs d'une certaine plage ou en fonction des valeurs d'une liste déterminée, l'utilisation des opérateurs *BETWEEN* et *IN* est recommandé.

6.3.1. Opérateur BETWEEN

Cet opérateur permet d'identifier une plage de valeurs acceptables en spécifiant une borne inférieure et une borne supérieure. La syntaxe de l'utilisation de cet opérateur est la suivante :

SELECT col1, col2, …

FROM nomTable

WHERE coli BETWEEN borneInférieure AND borneSupérieure ;

Exemple 5.23

Afficher toutes les informations des âgées entre 30 et 40 ans.

```
SELECT *
FROM PERSONNES
WHERE âge BETWEEN 30 AND 40 ;
```

6.3.2. Opérateur IN

L'opérateur *IN* indique si la valeur d'une colonne est égale à l'une des valeurs se trouvant dans une liste de choix. La syntaxe d'utilisation de l'opérateur *IN* est :

SELECT col1, col2, …

FROM nomTable

*WHERE coli **IN** (val1, …, valn);*

Exemple 5.24

Trouver les numéros et noms des personnes habitant Béjaia, Alger ou Oran.

```
SELECT inPers, nomPers
FROM PERSONNES
WHERE ville IN ('Béjaia', 'Alger', 'Oran');
```

Exemple 5.25

Trouver les ouvrages qui ne sont pas parus en 2005, 2010 et 2016.

```
SELECT *
FROM OUVRAGES
WHERE année NOT IN (2005, 2010, 2016);
```

6.4. L'opérateur LIKE et les chaines de caractères

L'opérateur *LIKE* permet de comparer la valeur d'une colonne avec une chaîne spécifique. Par exemple, si on cherche les personnes dont le nom se termine par le caractère 'D' et les ouvrages dont le titre comporte la chaine de caractères 'SQL', le langage SQL offre l'opérateur *LIKE* pour vérifier la correspondance entre les valeurs de colonnes et un modèle de caractères particulier.

Un modèle de caractères est mis entre deux apostrophes et peut comporter les caractères génériques, le symbole de pourcentage (%) et le trait de soulignement (_). La caractère % désigne une chaine formée de plusieurs caractères alors que le caractère _ désigne un seul caractère quelconque.

Exemple 5.26

Afficher les personnes dont le nom commence par « O » ou 'F'.

```
SELECT *
FROM PERSONNES
WHERE nomPers LIKE 'O%' OR nomPers LIKE 'F' ;
```

Exemple 5.27

Afficher les personnes dont le nom commence par « O » ou 'F'.

```
SELECT *
FROM PERSONNES
WHERE nomPers LIKE 'O%' OR nomPers LIKE 'F' ;
```

Exemple 5.28

Afficher les ouvrages (code et prix) dont le titre comprend le mot *SQL*.

```
SELECT codeOuv, prix
FROM OUVRAGES
WHERE titre LIKE '%SQL%' ;
```

Exemple 5.29

Afficher les filières dont le nom ne commence pas par « I ».

```
SELECT *
FROM FILIÈRES
WHERE nomFilière NOT LIKE 'I%S%' ;
```

6.5. Opérateurs ALL/ANY

Les quantificateurs ANY (i.e. SOME) et ALL permettent de comparer une valeur à celles d'un ensemble défini par une sous-requête. *ANY* signifie qu'***au moins*** un élément de l'ensemble satisfait la comparaison et *ALL* signifie que ***tous*** les éléments la satisfont.

Exemple 5.30

Trouver les numéros des personnes dont l'âge est le plus faible parmi les personnes habitant la ville de Béjaia.

```
SELECT numPers
FROM PERSONNES
WHERE âge <= ALL (SELECT âge
                  FROM PERSONNES
                  WHERE ville='Béjaia') ) ;
```

La valeur de l'*âge* est inférieure ou égale à tous les éléments de …

Exemple 5.31

Trouver les numéros des personnes dont l'âge est inférieur à celui d'une personne habitant la ville de Béjaia.

```
SELECT numPers
FROM PERSONNES
WHERE âge < ANY (SELECT âge
                 FROM PERSONNES
                 WHERE ville ='Béjaia') ) ;
```

La valeur de l'âge est inférieure à au moins un des éléments de …

6.6. Référence à une même table

Une sous-requête peut être définie sur la même table que la requête qui la contient. Il s'agit dans ce cas d'une *auto-jointure*.

Exemple 5. 32

Quels sont les personnes (numéro, nom et prénom) habitant dans la même ville que la personne numéro 10 ?

```
SELECT P1.numPers, P1.nomPers, P1.préPers
FROM PERSONNES P1 PERSONNES P2
WHERE P1.ville = P2.ville AND P2.numPers = 10 ;
```

Cette requête peut être exprimée en utilisant l'opérateur *IN*.

```
SELECT *
FROM PERSONNES
WHERE ville IN (SELECT ville
                FROM PERSONNES
                WHERE numPers = 10) ;
```

6.7. Le tri des tuples

L'ordre des tuples dans une table résultante d'une requête est toujours arbitraire. Dans une commande SELECT … FROM, la clause ORDER BY permet d'imposer un ordre au tuples du résultat d'une requête. La forme générale de cette clause est :

```
ORDER BY col1 [ASC / DESC, col2 ASC / DESC, …]
```

Les tuples seront alors tries en fonction des valeurs de *col1*, puis celles de *col2*, etc. L'ordre par défaut dans une clause ORDER BY est croissant (ASC).

Exemple 5.33

```
SELECT (codeOuv, titreOuvr, année, prix, codeFilière).
FROM OUVRAGES
ORDER BY titre, prix DESC
```

Les ouvrages sont d'abord triés par ordre croissant du *titre*, puis au sein de chaque groupe de même valeur de *titre*, les ouvrages sont triés par ordre décroissant du prix.

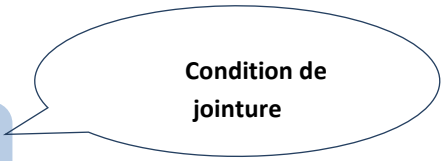
6.8. Expression de la jointure avec l'opérateur IN

La jointure peut être exprimée en utilisant l'opérateur IN. Dans ce qui suit nous illustrons ce concept à travers des exemples.

Exemple 5.34

Trouver les ouvrages (code, titre, année) qui ont été empruntés au moins une fois.

```
SELECT codeOuv, titreOuvr, année
FROM OUVRAGES
WHERE codeOuv IN
        (SELECT Ouvrage
         FROM EMPRUNTS
         WHERE codeOuv=Ouvrage ) ;
```



Cette structure emboîtée correspond à la jointure des tables « OUVRAGES» et « EMPRUNTS» sur la base des valeurs identiques des attributs *codeOuv* et *Ouvrage*. Une requête qui intervient dans la clause *WHERE* est appelée ***sous-requête***. L'exécution commence par la requête la plus interne et remonte jusqu'à atteindre la requête la plus externe. Une sous-requête peut à son tour contenir une sous-requête.

Exemple 5.35

Afficher les informations (numéro, nom, prénom) de l'auteur principal des ouvrages de la filière *INF*.

```
SELECT PERSONNES.numPers, PERSONNES .nomPers, PERSONNES.préPers
FROM PERSONNES
WHERE numPers IN
                (SELECT Auteur
                 FROM ÉCRIRE
                 WHERE position=1 AND Ouvrage IN
                        (SELECT codeOuv
                         FROM OUVRAGES
                         WHERE codeFilière='INF') );
```

7. Calcul statistique

La syntaxe d'une requête faisant éventuellement intervenir des *fonctions statistiques*, appelées aussi *fonctions d'agrégation*, une clause *GROUP BY* et une clause *HAVING* est la suivante :

```
SELECT Exp1, …, ExpN, [ FonctionsAgrégation, …]
FROM nomTable
[ WHERE Conditions ]
```

[GROUP BY Exp1, …, ExpN]

[HAVING ConditionsRegroupement]

La liste d'expressions Exp1, …, ExpN correspond généralement à une liste d'attributs. Nous nous intéressons dans cette section à *[FonctionsAgrégation]* de la clause SELECT et aux clauses *GROUP BY* et *HAVING*.

7.1. Les fonctions d'agrégation

Une fonction d'agrégation permet d'avoir une seule valeur à partir d'un ensemble de tuples en réalisant certains calculs statistiques de base. Parmi ces fonction : COUNT, SUM, AVG, MIN, et MAX.

7.1.1. La fonction COUNT et ses variantes

La fonction ***COUNT(*)*** permet de compter le nombre de tuples dans une table. Elle compte même les valeurs NULL et s'applique à tout type de colonnes.

Exemple 5.36

Trouver le nombre d'ouvrages.

```
SELECT COUNT(*) AS 'Nombre d'ouvrages'
FROM OUVRAGES ;
```

Il est à noter que *AS* est un mot clé qui est utilisé pour renommer une colonne à l'affichage.

La fonction ***COUNT(nomCol)*** permet de trouver le nombre de valeurs connues dans la colonne *nomCol* et s'applique à tout type de colonnes. Elle ignore alors les valeurs inconnues se trouvant dans la colonne *comCol*, c.-à-d., ne compte pas les tuples dont la valeur de la colonne *nomCol* est NULL.

Exemple 5.37

Trouver le nombre de personnes dont le nom de la ville est connue.

```
SELECT COUNT(ville) AS 'Nombre de villes connues'
FROM PERSONNES ;
```

La commande ***COUNT (DISTINCT nomCol)*** trouve le nombre de valeurs différentes présentes dans la colonne *nomCol* et s'applique à tout type de colonnes.

Exemple 5.38

Quel est le nombre de villes dans lesquelles habite au moins une personne âgée d'au plus 45 ans ?

```
SELECT COUNT(DISTINCT ville) AS 'Nombre de villes avec âge supérieur à 45
FROM PERSONNES
WHERE âge ≤ 45 ;
```

7.1.2. Les fonctions SUM et AVG

Les fonctions SUM(nomCol) et AVG(nomCol) permet d'avoir la somme et la moyenne, respectivement, des valeurs de la colonne *nomCol*. Ces deux fonctions ignorent les valeurs NULL se trouvant dans la colonne *nomCol*.

Exemple 5.39

Quel est le prix de tous les ouvrages ?

```
SELECT SUM(prix) AS 'Prix de tous les ouvrages'
FROM OUVRAGES ;
```

Exemple 5.40

Quel est le prix total des ouvrages de la filière *INF* ?

```
SELECT SUM(prix) AS 'Prix total des ouvrages INF'
FROM OUVRAGES
WHERE codeFilière = 'INF' ;
```

Exemple 5.41

Trouver l'âge moyen des personnes habitant la ville de Béjaia.

```
SELECT AVG(âge) AS 'Age moyen des habitants de Béjaia'
FROM PERSONNES
WHERE ville = 'Béjaia' ;
```

Exemple 5.42

Trouver le prix moyen des ouvrages parus en 2003.

```
SELECT AVG(prix) AS ‘Prix moyen des ouvrages de 2003’
FROM OUVRAGES
WHERE année BETWEEN ‘1-01-2003’ AND ’31-12-2003’;
```

7.1.3. Les fonctions MIN et MAX

La fonction ***MIN(nomCol)*** permet de trouver le minimum des valeurs de la colonne *nomCol*. Elle s’applique à tout type de colonnes.

Exemple 5.43

Quel est le prix minimal des ouvrages de la filière *INF* ?

```
SELECT MIN(prix) AS ‘Prix le plus bas des ouvrages INF’
FROM OUVRAGES
WHERE codeFilière = ‘INF’ ;
```

Exemple 5.44

Trouver l’âge minimal des personnes habitant la ville de Béjaia.

```
SELECT MIN(âge) AS ‘Age minimal des habitants de Béjaia’
FROM PERSONNES
WHERE ville = ’Béjaia’;
```

La fonction ***MAX(nomCol)*** permet de trouver le maximum des valeurs de la colonne *nomCol*. Elle s’applique à tout type de colonnes.

Exemple 5.45

Trouver l’âge maximal parmi toutes les personnes.

```
SELECT MAX(âge) AS ‘Age maximal des personnes‘
FROM PERSONNES ;
```

Exemple 5.46

Quel est le prix maximal des ouvrages parus en 2003 ?

```
SELECT MAX(prix) AS ‘Prix maximal des ouvrages de 2003’
FROM OUVRAGES
WHERE année BETWEEN ‘1-01-2003’ AND ’31-12-2003’ ;
```

7.2. La clause GROUP BY

Un groupe représente un sous-ensemble des tuples d’une table ayant la même valeur pour un attribut [AUD 2009]. Un groupe est déterminé par la clause ***GROUP BY*** suivie du nom des attributs sur lesquels s’effectuent le regroupement.

Si plusieurs expressions sont spécifiées (Exp1, Exp2, . . .), les groupes sont définis de la façon suivante : parmi tous les tuples pour lesquels *Exp1* prend la même valeur, on regroupe celles ayant *Exp2* identique, etc.

Exemple 5.47

Trouver le nombre de personnes habitant chaque ville.

```
SELECT ville, COUNT(*) AS ‘Nombre de personnes’
FROM PERSONNES ;
GROUP BY ville ;
```

Exemple 5.48

Quel est le nombre d’ouvrages écrit par chaque auteur ?

```
SELECT Auteur, COUNT(*) AS ‘Nombre d’ouvrage’
FROM ÉCRIRE
GROUP BY Auteur;
```

Exemple 5.49

Trouver le code et le titre de chaque ouvrage ainsi que le nombre de lecteurs qui l’ont emprunté ?

```
SELECT codeOuv, titreOuvr, COUNT(*) AS ‘Nombre lecteurs’
FROM OUVRAGES, EMPRUNTS
WHERE codeOuv = Ouvrage
GROUP BY codeOuv, titreOuvr ;
```

Exemple 5.50

Quel est le prix total et le prix moyen des ouvrages de chaque filière ?

```
SELECT codeFilière, SUM(prix) AS ‘Prix total des ouvrages’ ,
        AVG(prix) AS ‘prix moyen des ouvrages’
FROM OUVRAGES
GROUPE BY codeFilière ;
```

Exemple 5.51

Trouver l’âge minimal et l’âge maximal de lecteurs de chaque ville.

```
SELECT ville, MIN(âge) AS ‘Âge minimal’, MAX(âge) AS ‘Âge maximal’
FROM PERSONNES, EMPRUNTS
WHERE numPers = Lecteur
GROUP BY ville ;
```

7.3. Expressions numériques dans GROUP BY

Le *nomCol* dans les fonctions AVG et SUM peut être remplacé par toute expression à valeur numérique.

Exemple 5.52

Quel est le prix en Euro des ouvrages de la filière *BIO* (on suppose 1 Euro = 145 DZD) ?

```
SELECT SUM(prix/145) AS ‘Prix en Euro des ouvrages BIO’
FROM OUVRAGES
WHERE codeFilière = ‘BIO’ ;
```

7.4. La clause HAVING

Tout comme la clause WHERE qui limite le nombre de tuples affichés par une instruction SELECT, la clause ***HAVING*** permet de sélectionner certains groupes dans une clause GROUP BY [MAT 2003]. La clause HAVING doit être placée derrière la clause GROUP BY.

Exemple 5.53

Trouver les villes dans lesquelles habitent plus de 1500 personnes.

```
SELECT ville, COUNT(*) AS ‘Nombre de personnes’
```

```
FROM PERSONNES ;
GROUP BY ville
HAVING COUNT(*) > 1500 ;
```

Exemple 5.54

Quels sont les auteurs ayant écrit plus de 5 ouvrages de filière INF tout en étant auteur principal ?

```
SELECT Auteur, COUNT(*) AS 'Nombre d'ouvrage'
FROM ÉCRIRE, OUVRAGES
WHERE position = 1 AND Ouvrage= codeOuv AND codeFilière='INF'
GROUP BY Auteur
HAVING COUNT(*) > 5 ;
```

Exemple 5.55

Trouver les ouvrages (code et titre) ayant été empruntés moins de 200 fois ?

```
SELECT codeOuv, titreOuvr, COUNT(*) AS 'Nombre d'emprunts'
FROM OUVRAGES, EMPRUNTS
WHERE codeOuv = Ouvrage
GROUP BY codeOuv, titreOuvr ;
HAVING COUNT(*) < 200 ;
```

7.5. Expression de la division à l'aide du calcul statistique

L'opérateur de division peut être effectué en combinant les différentes fonctions d'agrégation avec les clauses GROUP BY et HAVING.

Exemple 5.56

Trouver les ouvrages (code et titre) ayant été empruntés par tous les lecteurs.

Un ouvrage est emprunté par tous les lecteurs si le nombre distinct de lecteurs ayant emprunté cet ouvrage est égal au nombre total de lecteurs.

```
SELECT codeOuv, titreOuvr, COUNT(DISTINCT Lecteur) AS 'Nombre d'emprunts'
FROM EMPRUNTS, OUVRAGES
```

```
WHERE codeOuv = Ouvrage
GROUP BY codeOuv, titreOuvr
HAVING COUNT(DISTINCT Lecteur) = (SELECT COUNT (Lecteur)
                                 FROM PERSONNES, EMPRUNTS
                                 WHERE Lecteur= numPers ) ;
```

Exemple 5.57

Trouver les lecteurs ayant emprunté tous les ouvrages de la filière *INF*.

Un lecteur a emprunté tous les ouvrages de la filière *INF* si le nombre distinct d'ouvrages de cette filière et qui sont empruntés par ce lecteur est égal au nombre total d'ouvrages de la filière *INF*.

```
SELECT Lecteur, COUNT(DISTINCT Ouvrage) AS 'Nombre d'ouvrages empruntés'
FROM EMPRUNTS, OUVRAGES
WHERE Ouvrage=codeOuv AND codeFilière= 'INF'
GROUP BY Lecteur
HAVING COUNT(DISTINCT Ouvrage) = (SELECT COUNT (codeOuv)
                                 FROM OUVRAGES
                                 WHERE codeFilière= 'INF') ;
```

Références bibliographiques

[ANS I978] ANSI/X3/SPARC Study Group on Data Base Management Systems, "Framework Report on Database Management Systems", *Information Systems*, vol. 3, no. 3, 1978.

[AUD 2009] Laurent AUDIBERT, *Bases de données – de la modélisation au SQL*, Ellipses, 2009.

[BEN 2017] Saïd BENAÏCHOU, Bases de données et systèmes d'information. Modèle relationnel, SQL, optimisation des requêtes, transactions, modélisation des données, ELLIPSES, 2017.

[BER 81] Bernstein P., Goodman N., « The Power of Natural Semijoins », Siam Journal of Computing, vol. 10, n° 4, décembre 1981, p. 751-771.

[BOU 1999] Nacer BOUDJLIDA, Bases de données et systèmes d'informations – Le Modèle relationnel : langages, systèmes et méthodes, DUNOD, 2002.

[COD 1970] E. F. Codd, A relational model of data for large shared data banks. Communications of the ACM, vol. 13, no. 6, pp. 377-387, 1970.

[COD 1972] E. F. Codd, "Further Normalization of the Data Base Relational Model," in Data Base Systems, R. Rusin ed., Prentice-Hall, 1972.

[COD 1974] E. F. Codd, « Recent Investigations in Relational Database Systems », IFIP World Congress, North Holland Ed., p. 1017-1021, 1974.

[DAT 1981] C. J. Date, "Referential Integrity", 7e Very Large Data Bases, Cannes, France, 1981, IEEE Ed.

[GAR 2003] Georges GARDARIN, *Bases de Données*, Eyrolles, 5e édition, 2003.

[HAI 2015] Jean-Luc HAINAUT, Bases de données, Concepts, utilisation et développement. DUNOD, 3e édition, 2015.

[MAT 2002] Ramon A. Mata-Toledo, Pauline K. Cushman, *Introduction aux bases de données relationnelles*, EdiScience, 2002.

[MAT 2003] Ramon A. MATA-TOLEDO and Pauline K. CUSHMAN, « Programmation SQL », Ediscience, 2003.

[TAH 2016] Z. TAHAKOURT, Cours bases de données, Université de Béjaia, Algérie, 2016.

[ULL 1988] J. D. Ullman, Principles of Database and Knowledge-base Systems, livres, volumes I & II, Computer Science Press, 1988.