REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LARECHERCHE SCIENTIFIQUE
ECOLE NATIONALE POLYTECHNIQUE

Support de cours

Introduction aux Bases de Données : De la Modélisation au SQL

2ème année cycle préparatoire

Par: Dr. Nour El-Houda BENALIA

Année Universitaire: 2019/2020

Préambule

Les bases de données ont pris une place essentielle en informatique, et particulièrement dans le domaine de la gestion. Ces systèmes jouent un rôle primordial dans le développement des technologies. L'étude des bases de données a conduit au développement de concepts, méthodes et algorithmes spécifiques, notamment pour gérer les données.

Les bases de données relationnelles constituent l'objet de ce cours. Ces bases sont conçues suivant le modèle relationnel, dont les fondations théoriques sont solides, et manipulées en utilisant l'algèbre relationnelle. Il s'agit, à ce jour, de la méthode la plus courante pour organiser et accéder à des ensembles de données. Cependant, il est difficile de modéliser un domaine directement sous une forme base de données relationnelle. Une modélisation intermédiaire est généralement indispensable, en référence à la modélisation conceptuelle de données parmi. Il existe différentes approches pour cette modélisation, comme exemple le modèle entités- associations. Le modèle entités-associations permet une description naturelle du monde réel à partir des concepts d'entité et d'association.

Dans ce cours, l'accent est mis sur les aspects fondamentaux pour la modélisation et la conception des bases de données. A cet effet, ce cours se présente comme suit : après une introduction générale abordant la problématique des bases de données, des définitions, les systèmes de gestion de bases de données et le cycle de vie d'une base de données, le chapitre 1 est dédié à la modélisation conceptuelle des données. Nous présentons le modèle entités-associations pour aborder le problème de la conception des bases de données.

Le chapitre 2 décrit le modèle relationnel, le passage du modèle conceptuel au modèle relationnel, ainsi que la normalisation du modèle. Le chapitre 3 présente l'algèbre relationnelle et ses différents opérateurs. Le dernier chapitre, le chapitre 4 , est entièrement consacré au langage SQL (Structured Query Language) qui peut être considéré comme le langage d'accès normalisé aux bases de données relationnelles. Une présentation des instructions du langage de définition de données et celles du langage de manipulation de données fera aussi l'objet ce chapitre.

Sommaire

Chapitre 1

Introduction aux bases de données

Objectifs:

- Expliquer l'importance des bases de données;
- Expliquer l'esprit des bases de données.

Dans ce cours: Vous allez voir:

- Problèmes d'utilisation des fichiers pour la manipulation des données;
- Définition des bases de données et les systèmes de gestion des bases de données;
- Histoire des bases de données;
- Différents types de bases de données;
- Gestion des bases de données;
- Types d'utilisateurs des bases de données;
- Description des données;
- Les objectifs des SGBD;
- Fonctions d'un SGBD;
- Quelques SGBD connus et utilisés;
- Cycle de vie d'une Base de données;
- Processus d'élaboration d'une base de données.

Chapitre 01

Introduction aux bases de données

1.1. Introduction

Comme c'est le cas pour de nombreuses innovations technologiques, une importante pression des besoins est à l'origine de l'émergence des Systèmes de Gestion de Bases de Données. Dans l'environnement informatique traditionnel des gros systèmes d'exploitation, le seul mode de gestion de données reste le gestionnaire de fichiers. Les données traitées par une application (gestion de la paie, des stocks, de la comptabilité, des locaux…) demeurent spécifiques à cette application.

L'organisation des données en fichiers telle qu'elle est traditionnellement conçue répond à des besoins précis: les services utilisent des informations le plus souvent distinctes pour des traitements différents. Pour cette raison, chaque équipe de l'entreprise dispose de ses propres fichiers de données où ne figurent que les informations la concernant. La forme sous laquelle cette information est stockée dépend de facteurs nombreux et variés : matériel disponible, bonne volonté et harmonisation des équipes de développement (choix du langage de programmation, choix de la structure de stockage…) ; surtout, les accès à l'information déterminent le plus souvent l'organisation choisie pour les données : le choix des informations regroupées dans un même fichier, le choix de l'organisation du fichier (séquentiel, aléatoire…).

Dans ce contexte (c-à-dire travailler directement sur un fichier), on constate l'apparition de nombreuses difficultés :

Redondance des données

La répétition en différents endroits de données identiques pose de difficiles problèmes lors des mises à jour ; chaque modification doit être répercutée sur toutes les occurrences de l'objet concerné.

Difficulté d'accès aux données

L'accès aux données nécessite l'intervention d'un informaticien (nécessité de connaître la machine où résident les informations, les structures utilisées pour stocker ces informations, les chemins d'accès disponibles…).

Problèmes de partage des données

Le problème est bien connu des concepteurs de systèmes d'exploitation : comment réagir à plusieurs ouvertures simultanées d'un même fichier. Des problèmes identiques se posent dès que plusieurs utilisateurs désirent effectuer des modifications sur les mêmes données.

Risques d'incohérence

Des liens sémantiques existent très souvent entre les données : le poste d'un employé, figurant dans le fichier du service du personnel, n'est pas sans rapport avec le salaire qui lui est versé par la comptabilité. Pourtant la modification de l'un n'entraîne pas automatiquement celle de l'autre.

Problèmes de sécurité

Qu'arrive-t-il si une panne interrompt un programme augmentant tous les salaires de 3% ? Combien de modifications ont été prises en compte ; où reprendre le travail ? La présence de la même information en plusieurs exemplaires n'est évidemment pas de nature à faciliter la tâche du SGBD et des programmes de reprise après panne. De même, la

confidentialité de l'information est plus facile à assurer avec une seule occurrence de l'information à protéger.

Manque de portabilité des applications

Que se passe-t-il lorsque l'on change de système d'exploitation ou de support de stockage ? Dans la plupart des cas, les applications développées sont si fortement liées aux possibilités propres du système et du gestionnaire de fichiers qu'il est parfois aussi difficile de les actualiser que de les réécrire entièrement.

Problèmes de maintenance

Enfin, quand la modification de la structure des données manipulées devient nécessaire, il faut assurer le passage des données depuis les anciennes vers les nouvelles structures choisies et presque toujours réécrire complètement les programmes d'application dont la plupart réalisent pourtant la même opération qu'auparavant.

Une bonne solution à ces problèmes passe par le regroupement et l'unicité des données, ainsi que par la centralisation des moyens de gestion de ces données. L'administration unique et centralisée des données garantit la cohérence de l'information et permet l'*indépendance des programmes et des données*. L'ensemble de toutes les données peut être regroupé sous l'appellation de *base de données*.

Cette base de données représente l'information de l'entreprise. Elle existe indépendamment de l'usage qu'en font les programmes d'application ou les utilisateurs finaux. L'information doit donc pouvoir être décrite indépendamment de l'usage qui en est fait. Cette description nécessite un *modèle de données*, c'est à dire un ensemble de concepts et de règles assez généraux et riches de sémantique pour pouvoir donner de façon formelle un *schéma* de la structure permanente de cette information.

Le système logiciel qui met en oeuvre le modèle de données (qui stocke les données, gère les accès, les droits des utilisateurs, et plus généralement traite au mieux l'ensemble des problèmes que nous venons d'évoquer) forme véritablement le Système de Gestion de Bases de Données(SGBD).

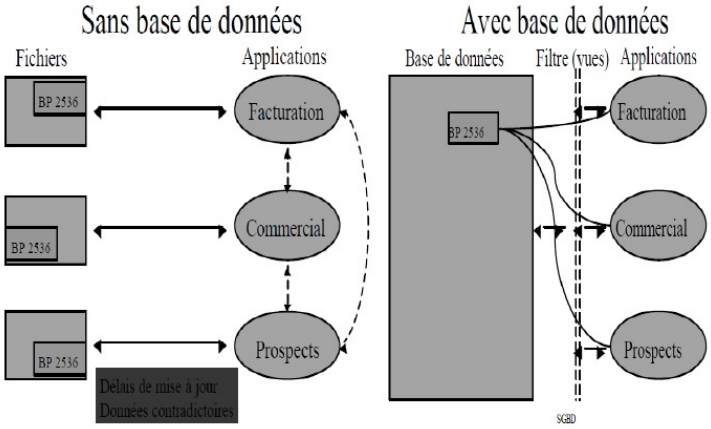


Figure 1.1 Séparation des données et des programmes

FICHIER	**BASE DE DONNEES**
Les données des fichiers sont décrites dans les programmes	décrites hors des programmes dans la base elle-même (description unique)

La multiplication des fichiers entraînait la *redondance* des données, ce qui rendait difficile les mises à jour. D'où l'idée *d'intégration* et de *partage* des données.

En utilisant une base de données :

- Une information n'est stockée qu'une seule fois
- Une seule base pour toutes les applications
- …mais chaque application ne voit que ce qu'elle doit voir (contrôle par les vues par exemple)

Exemple 1:

Stocks dans un magasin : on y trouvera les renseignements sur les articles en vente, leurs prix, leurs fournisseurs, leurs délais de livraison, etc.

Exemple 2 :

Réservations dans une agence de voyages : on y trouvera les renseignements sur les différentes compagnies aériennes, les horaires de leurs vols, les prix, les billets vendus, les billets réservés, etc.

Exemple 3 :

Service de scolarité : on y trouvera les renseignements concernant les étudiants, leur état-civil, leurs inscriptions passées et actuelles, leurs résultats, etc.…

1.2. Avantages d'une base de données

Une base de données permet de mettre des données à la disposition d'utilisateurs pour une consultation, une saisie ou bien une mise à jour, tout en s'assurant des droits accordés à ces derniers. Cela est d'autant plus utile que les données informatiques sont de plus en plus nombreuses.

Une base de données peut être locale, c'est-à-dire utilisable sur une machine par un utilisateur, ou bien répartie, c'est-à-dire que les informations sont stockées sur des machines distantes et accessibles par réseau.

L'avantage majeur de l'utilisation de bases de données est la possibilité de pouvoir être accédées par plusieurs utilisateurs **simultanément.**

1.3. Définitions

Qu'est-ce qu'une base de données ?

Une base de données (son abréviation est BD, en anglais DB, *database*) est une entité dans laquelle il est possible de stocker des données de façon structurée et avec le moins de redondances possibles. Ces données doivent pouvoir être utilisées par des programmes, par des utilisateurs différents. Ainsi, la notion de base de données est généralement couplée à celle de réseau, afin de pouvoir mettre en commun ces informations, d'où le nom de **base**. On parle généralement de système d'information pour désigner toute la structure regroupant les moyens mis en place pour pouvoir partager des données

Définition 1: une BD est un ensemble structuré de données (organisation et description de données) enregistrées sur des supports accessibles par l'ordinateur (stockage sur disque) pour satisfaire simultanément plusieurs utilisateurs (partage des données) de manière sélective (confidentielle) en un temps opportun (performance). Introduction aux bases de données

Définition 2: le logiciel qui permet d'interagir avec une BD est un Système de Gestion de Bases de Données (SGBD). Il permet à des utilisateurs de créer et maintenir une BD. Les activités supportées sont la définition d'une BD (spécification des types de données à stocker), la construction d'une BD (stockage de données proprement dites) et la manipulation des données (principalement ajouter, supprimer, retrouver des données).

Définition 3: un modèle de données est un ensemble de concepts permettant de décrire la structure d'une base de données. La plupart des modèles de données incluent des opérations permettant de mettre à jour et questionner la base. Le modèle de données le plus utilisé est le modèle relationnel.

1.4. Histoire des bases de données

L'histoire des bases de données **remonte aux années 1960**, avec l'apparition des bases de données réseau et des bases de données hiérarchiques. Dans les années 1980, ce sont les bases de données object-oriented qui ont fait leur apparition. Aujourd'hui, les bases de données prédominantes sont les SQL, NoSQL et bases de données cloud.

Il est aussi possible de classer les bases de données en fonction de leur contenu : bibliographique, textes, nombres ou images. Toutefois, en informatique, on classe généralement les bases de données en fonction de leur approche organisationnelle. Il existe de nombreux types de bases de données différentes: **relationnelle, distribuée, cloud, NoSQL**… Voici les différents types de bases de données.

1.5. Différents types de bases de données

Dans le cas d'une grande database, les multiples utilisateurs doivent être en mesure de manipuler les informations qu'elle contient rapidement et n'importe quand. De plus, les grandes entreprises ont tendance à cumuler de nombreux fichiers indépendants comprenant des fichiers liés ou même des données se superposant. Dans le cadre d'une analyse de données, il est nécessaire que les données en provenance de plusieurs fichiers puissent être liées. C'est pourquoi **différents types de bases de données ont été développé pour répondre à ces exigences** : orientée texte, hiérarchique, réseau, relationnelle, orientée objet…

1.5.1 Base de données hiérarchique

Les bases de données hiérarchiques comptent parmi **les plus anciennes bases de données**. Au sein de cette catégorie, les enregistrements sont organisés dans une structure d'arborescence. Chaque niveau d'enregistrements découle sur un ensemble de catégories plus petites.

1.5.2 Base de données réseau

Les bases de données réseau sont également parmi les plus anciennes. Plutôt que de proposer des liens uniques entre différents ensembles de données à divers niveaux, les bases de données réseaux créent **des liens multiples entre les ensembles** en plaçant des liens, ou des pointeurs, sur un ensemble d'enregistrements ou un autre. La vitesse et la polyvalence des bases de données réseau ont conduit à une adoption massive de ce type de databases au sein des entreprises ou dans le domaine du e-commerce.

1.5.3 Base de données orientée texte

Une database orientée texte, ou flat file database, se présente sous la forme d'un **fichier (une table) au format .txt ou .ini**. Un fichier plat est un fichier texte, ou un fichier combinant du texte avec un fichier binaire. En général, dans ces bases de données, chaque ligne ne comporte qu'un enregistrement. La plupart des bases de données pour PC sont des bases de données orientées texte.

1.5.4 Base de données SQL (relationnelle)

Les bases de données relationnelles ont été inventées en 1970 par E.F. Codd de IBM. Il s'agit de documents tabulaires dans laquelle **les données sont définies afin d'être accessibles** et de pouvoir être réorganisées de différentes manières.

Les bases de données relationnelles sont constituées d'un ensemble de tableaux. Au sein de ces tableaux, les **données sont classées par catégorie**. Chaque tableau comporte au moins une colonne correspondant à une catégorie. Chaque colonne comporte un certain nombre de données correspondant à cette catégorie.
L'API standard pour les bases de données relationnelles est **le Structured Query Language (SQL)**. Les bases de données relationnelles sont facilement extensibles, et de nouvelles catégories de données peuvent être ajoutées après la création de la database originale sans avoir besoin de modifier toutes les applications existantes.

1.5.5 Base de données distribuée

Une BDD distribuée est une database dont **certaines portions sont stockées à plusieurs endroits** physiques. Le traitement est réparti ou répliqué entre différents points d'un réseau.
Les bases de données distribuées **peuvent être homogènes ou hétérogènes**. Dans le cas d'un système de base de données distribuée homogène, tous les emplacements physiques fonctionnent avec le même hardware et tournent sous le même système d'exploitation et les mêmes applications de bases de données. Au contraire, dans le cas d'une database distribuée hétérogène, le hardware, les systèmes d'exploitation et les applications de bases de données peuvent varier entre les différents endroits physiques.

1.5.6 Base de données cloud

Dans ce cadre, elle est **optimisée ou directement créée pour les environnements virtualisés**. Il peut s'agir d'un cloud privé, d'un cloud public ou d'un cloud hybride.
Les **bases de données cloud offrent plusieurs avantages** comme la possibilité de payer pour la capacité de stockage et la bande passante en fonction de l'usage. Par ailleurs, il est possible de changer l'échelle sur demande. Ces bases de données offrent aussi une disponibilité plus élevée.

1.5.7 Base de données NoSQL

Les bases de données NoSQL sont utiles **pour les larges ensembles de données distribuées**. En effet, les bases de données relationnelles ne sont pas conçues pour le Big Data, et les ensembles de données trop larges peuvent poser des problèmes de performances.
Si une entreprise doit **analyser d'importantes quantités de données non structurées**, ou des données stockées sur plusieurs serveurs cloud virtuels, la database NoSQL est idéale. Avec l'essor du Big Data, les bases de données NoSQL sont de plus en plus utilisées.

1.5.8 Base de données orientée objets

Les **objets créés à l'aide de langage de programmation orientés objets** sont généralement stockés sur des bases de données relationnelles. Toutefois, en réalité, les bases de données orientées objets sont plus adaptées pour stocker ce type de contenu.

Plutôt que d'être organisée autour d'actions, les bases de données orientées objets sont **organisées autour d'objets**. De même, au lieu d'être organisées autour d'une logique, elles sont organisées autour des données. Par exemple, un enregistrement multimédia au sein d'une BDD relationnelle peut être défini comme un objet de données plutôt que comme une valeur alphanumérique.

1.5.9 Base de données orientée graphe

Une base de données orientée graphe, ou graphe, est un **type de database NoSQL utilisant la théorie des graphes** pour stocker, cartographier et effectuer des requêtes sur les relations entre les données. Les bases de données graphe sont constituées de noeuds et de bords.
Chaque noeud représente une entité, et chaque bord représente une connexion entre les noeuds. Les bases de données graphes gagnent en popularité dans le domaine des analyses d'interconnexions. Par exemple, les entreprises peuvent utiliser une BDD graphe pour **miner des données sur ses clients à partir des réseaux sociaux**.

De plus en plus souvent, des bases de données jadis séparées sont combinées électroniquement sous forme de collections plus larges **que l'on appelle les Data Warehouses**. Les entreprises et les gouvernements utilisent ensuite des logiciels de Data Mining pour analyser les différents aspects des données. Par exemple, une agence gouvernementale peut procéder ainsi pour enquêter sur une entreprise ou une personne qui ont acheté une grande quantité d'équipement, même si les achats sont disséminés dans tout le pays ou répartis entre plusieurs subsidiaires.

1.6. Gestion des bases de données

La gestion de la base de données se fait grâce à un système appelé **SGBD** (Système de Gestion de Bases de Données). Le SGBD est un ensemble de services (applications logicielles) permettant de gérer les bases de données, c'est-à-dire :

- permettre l'accès aux données de façon simple,
- autoriser un accès aux informations à de multiples utilisateurs,
- manipuler les données présentes dans la base de données (insertion, suppression, modification).

1.6.1 Avantages de l'utilisation des SGBD

• Centralisation des données => intégrité des données.
• Contrôle centralisé de l'accès aux données => sécurité accrue.
• Instructions de traitement très puissantes => grande rapidité de développement.
• Indépendance vis-à-vis de la structure physique et logique des données => maintenance facilitée.
• Pour les SGBD relationnels : langage non procédural simple => interrogation directe possible par les utilisateurs et réponses rapides à des questions non prévues par l'application

1.7. Types d'utilisateurs

Il existe plusieurs types d'utilisateurs de bases de données:

1.7.1 L'administrateur de la base

Il est chargé :
– du contrôle de la base de données, en particulier, permettre l'accès aux données aux applications ou individus qui y ont droit.
– de conserver de bonnes performances d'accès à ces données.
– des sauvegardes et des procédures de reprise après les pannes. Introduction aux bases de données.

1.7.2 Le programmeur

• écrit des applications qui utilisent la base de données
• il crée les tables et les structures associées (vues, index,...) utilisées par ses Applications.

1.7.3 L'utilisateur final

• n'a accès qu'aux données qui lui sont utiles, par l'intermédiaire d'applications en interrogeant directement les tables ou vues sur lesquelles l'administrateur lui a accordé des droits.

1.8. Description des données

Pour assurer ces objectifs, trois niveaux de description des données ont été définis.

1.8.1 Niveaux d'abstraction

On distingue généralement les 3 niveaux :

1.8.1.1 Niveau interne ou physique

Ce niveau appelé aussi niveau interne, gère le stockage et l'accès aux données c-à-dire s'occupe de la description des données et de leur organisation : en termes de fichiers, d'index, de méthodes d'accès, et. Il n'y a qu'un seul niveau physique par SGBD.

1.8.1.2 Niveau conceptuel

Il décrit les données sous une forme indépendante du matériel. Il permet la description :
• des objets : exemple OUVRAGES, ETUDIANTS
• des propriétés des objets (attributs) : exemple cote de OUVRAGES , titre de OUVRAGES, nomgre de OUVRAGES etc.
• des contraintes : le nombre d'exemplaires d'un ouvrage est supérieur à zéro.

C'est à ce niveau, également appelé le niveau logique, que l'on parle de modèle (conceptuel) de données. Ce modèle décrit l'ensemble des données de l'entreprise. C'est la description conceptuelle d'une base de données. Cette description va donner lieu à un schéma de base de données.

1.8.1.3 Niveau vue (niveau externe)

C'est le plus haut niveau d'abstraction de la base de données. Il est aussi appelé niveau externe et est propre à un utilisateur ou à un groupe d'utilisateurs et ne lui présente qu'une vue partielle de la réalité : exemple : OUVRAGES édités avant 2011. Il y a bien entendu plusieurs vues d'une même BD. L'objectif de ce niveau est la simplification et la confidentialité.

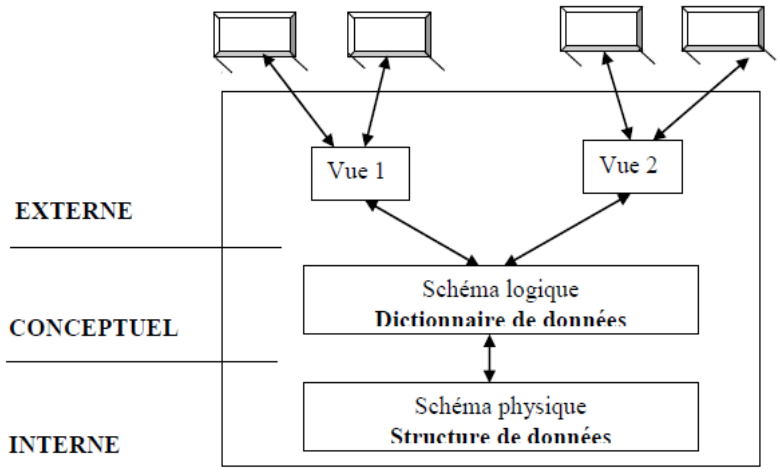


Figure 1.2 Niveaux d'abstraction

1.8.2 Avantages de la séparation des 3 niveaux

La séparation des niveaux d'abstraction présente plusieurs avantages :

• On peut limiter les modifications liées aux changements de matériel, de système d'exploitation ou des logiciels utilisés.

• La vision de chaque utilisateur est indépendante des visions des autres utilisateurs et n'est pas modifiée par les modifications du schéma conceptuel qui ne le concernent pas.

1.9. Les objectifs des SGBD

De façon plus formalisée, voici reprise et complétée la liste des objectifs des SGBD.

1 Offrir différents niveaux d'abstraction
2 Assurer l'indépendance physique des données
3 Assurer l'indépendance logique des données
4 Contrôler la redondance des données
5 Permettre à tout type d'utilisateur de manipuler des données
6 Assurer l'intégrité des données Introduction aux bases de données
7 Assurer le partage des données
8 Assurer la sécurité des données
9 Optimiser l'accès aux données

1.10. Fonctions d'un SGBD

• Description des données (grâce aux Langages de Description de Données : LDD)

• Manipulation et restitution des données (mises à jour et interrogation). Mise en oeuvre à l'aide l'un Langage de Manipulation de Données : LMD).

• Contrôle (partage, intégrité, confidentialité sécurité).

1.11. Quelques SGBD connus et utilisés

Il existe de nombreux systèmes de gestion de bases de données, en voici une liste non exhaustive :

1.11.1 Gros systèmes

PostgreSQL: http://www.postgresql.org/ – dans le domaine public ;

MySQL : http://www.mysql.org/ – dans le domaine public ;

Oracle : http://www.oracle.com/ – de Oracle Corporation ;
IBM DB2 : http://www-306.ibm.com/software/data/db2/
Microsoft SQL : http://www.microsoft.com/sql/ Introduction aux bases de données
Sybase : http://www.sybase.com/linux
Informix : http://www-306.ibm.com/software/data/informix/

1.11.2 SGBD Micro

Access, Paradox, Dbase V, …

1.12. Cycle de vie d'une Base de données

Le **cycle de vie** d'une base de donnée (BD) se décompose en quatre phases :

- Conception de la BD (schéma conceptuel),
- Implantation des données (schéma logique)
- Utilisation de la BD (interrogation, mises à jour)
- Maintenance de la BD (correction, évolution)

13. Processus d'élaboration d'une base de données

L'élaboration d'une BD suit les étapes suivantes :

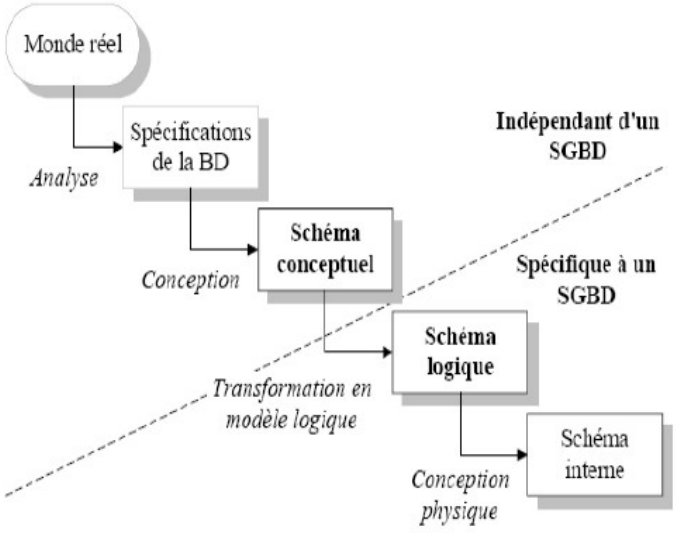


Figure 1.3 Elaboration d'une base de données

Chapitre 2

Modélisation Conceptuelle des Données

Objectifs:

- Savoir comment concevoir une base de donnée;

Dans ce cours: Vous allez voir:

- Modèle conceptuel de données (MCD);
- Modèle Entité / Association;
- Modèle entité association: Règles de bonnes pratique;
- Complément sur les associations.

Chapitre 2

Conception des bases de données : Le Modèle Entité-Association

2.1 Introduction

La modélisation d'un problème, c'est-à-dire le passage du monde réel à sa représentation informatique, se définit en plusieurs étapes pour parvenir à son intégration dans un SGBD-R et permettre la manipulation des données par le langage SQL.

Classiquement, le processus de modélisation des données passe par deux phases :

-Réalisation d'un *modèle conceptuel*
-Traduction en un *modèle relationnel*

Le premier niveau de modélisation, dit conceptuel, consiste en une phase d'analyse du problème réel. Cette phase est assez délicate et permet de définir les données à utiliser, leur mode d'évolution dans le temps et les relations entre elles. C'est le moment où l'on se pose les questions essentielles comme celle de savoir à quel usage on destine le modèle informatique que l'on est entrain de constituer. Ce travail est réalisé par des spécialistes de l'analyse. Il s'exprime dans un *formalisme* de type entité- association. Il existe d'autres types de formalisme tel que le formalisme UML.

Le second niveau de modélisation, dit relationnel, conduit à élaborer l'ensemble des objets manipulables par un SGBD-R. Ce travail est souvent réalisé par l'architecte de données, ou un administrateur de SGBD. Il peut être découpé en deux étapes :

-La conception de modèle *logique* (représentation en tables indépendantes du SGBD)
-La traduction en un modèle physique (propre à un SGBD spécifique). Tous les SGBD n'ont pas les mêmes caractéristiques du langage SQL.

2.2 Le Modèle Conceptuel de Données (MCD)

Un schéma conceptuel représente:

-Les faits et les évènements qui décrivent le monde à modéliser. Exemple : une compagnie aérienne, ses avions et ses pilotes.

-Certaines contraintes. Exemple : un pilote ne doit vole que s'il détient une licence en cours de validité et une qualification correspondant au type d'avion.

Les différentes techniques de MCD :

-Entité-Association (en anglais E-R pour *Entity Relationship*)
-Modèle binaire

-Modèle Z

Un MCD selon le modèle E/A est un diagramme avec des entités et des associations que nous voyons dans la section suivante. Pour réaliser un tel digramme, il faut s'aider d'outils graphiques informatiques spécialisés (particulièrement dans le monde de l'entreprise). Ces mêmes outils génèrent le modèle logique (le plus souvent le modèle relationnel de la base. On se sert ensuite d'un script (le plus souvent SQL) pour créer les différentes tables et contraintes.

2.3 Le modèle Entité-Association

Il est difficile de modéliser un domaine sous une forme directement utilisable par un SGBD. Une ou plusieurs modélisations intermédiaires sont donc utiles, le modèle entités-associations constitue l'une des premières et des plus courantes. Ce modèle, permet une description naturelle du monde réel à partir des concepts d'entité et d'association. Basé sur la théorie des ensembles et des relations, ce modèle se veut universel et répond à l'objectif d'indépendance données-programmes.

L'idée force du modèle entité-association est de représenter, par un schéma standardisé, les différents éléments constitutifs du système d'information, appelés *attributs* (exemple : nom, âge, ...), et les relations qui les unissent, appelées *associations*. Une manière simple de modéliser est de décrire la réalité par une phrase :

Le sujet et le complément représentent des entités, et le verbe l'association.

Exemple: un utilisateur (entité) poste (association) une news (entité).

La représentation du modèle entités-associations s'appuie sur trois concepts de base :

- l'objet ou entité ;
- l'association ;
- la propriété.

L'objet est une entité ayant une existence propre. L'association est un lien ou relation entre objets sans existence propre. La propriété est la plus petite donnée d'information décrivant un objet ou une association.

2.3.1 Entité

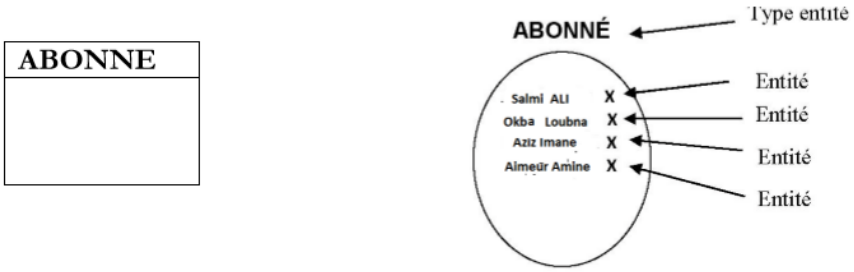
Définition 01 : Une entité est un objet, une chose concrète ou abstraite qui peut être reconnue distinctement et qui est caractérisée par son unicité.

Il faut noter que les entités ne sont pas représentées sur un modèle entité-association.

Définition 02: Un type-entité désigne un ensemble d'entités qui possèdent une sémantique et propriétés communes.

Les personnes, les livres, les voitures sont des type-entités. Dans le cas d'une personne, par exemple, les informations associées (ou propriétés) comme le nom, le prénom, ne changent pas de nature en fonction des personnes.

Dans un diagramme entité-association, un type-entité est représenté par un rectangle à cartouche. Le nom du type-entité est placé dans le cartouche : il est recommandé de choisir un nom commun décrivant le type-entité (exemple : Etudiant, Enseignant, Matière). Le nom du type-entité est écrit en majuscules dans le cartouche. Les attributs prendront place dans la partie basse.



(a) **(b)**

Figure 1.1 : (a) Représentation graphique d'un exemple de type entité sans ses propriétés associées, (b) Schéma de 4 entités du type-entité ABONNÉ

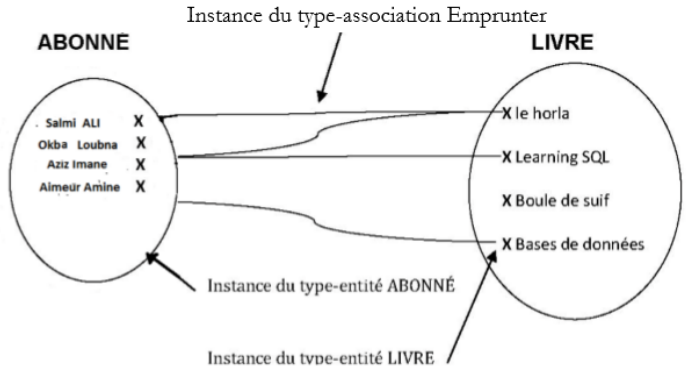
Une entité est une occurrence ou instance de son type-entité. Par abus de langage, le terme entité est régulièrement utilisé pour désigner le type-entité et ses entités. Il ne faut cependant pas confondre ces deux notions.

2.3.2 Association

Comme pour l'entité, le terme d'association est utilisé pour désigner un type-association.

Définition 03 : Une association (ou relation) est un lien entre plusieurs entités.

Exemple : L'ABONNÉ Salmi ALI Emprunte le LIVRE « Le horla ». Nous représentons ce lien de manière informelle, ci-dessous :



Et maintenant selon le modèle entité-association : Patte de l'association Emprunter

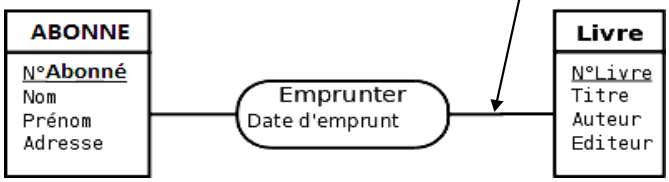


Figure 1.2 Représentation graphique d'un exemple de type association liant deux type entité.

Dans le modèle entité-association, le type-association est représenté par une ellipse à cartouche. Le nom du type-association est placé dans le cartouche. Ce nom doit être un verbe à l'infinitif, à la forme passive ou bien accompagné d'un adverbe. Exemple : Enseigner, AvoirLieuDans. Notez que l'initiale du nom de l'association est en majuscule.

Les attributs de l'association sont placés dans la partie inférieure au cartouche.

Définition 04 : Les type-entités intervenant dans un type-association sont appelés participants.

Définition 05 : L'ensemble de participants d'un type-association est appelé la collection de type- association. La collection d'un type-association peut ne comporter d'un seul type-entité.

Définition 06 : La dimension ou l'arité d'un type-association est le nombre de type-entités contenu dans la collection.

Un type-association est dit:

- N-aire : dans le cas général,
- Binaire : dans le cas où n=2 (la collection contient deux type-entités),
-Ternaire : dans le cas où n=3. Les type-associations avec n>2, bien qu'existants, sont rares et problématiques.

2.3.3 Attribut et valeur

Définition 07 : Un attribut (ou propriété) est une caractéristique associée à un type-entité ou à un type association.

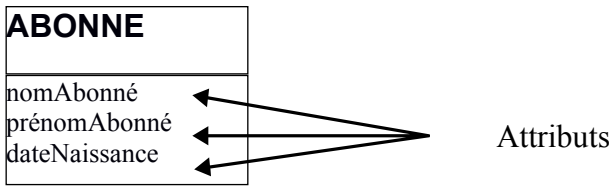


Figure 1.3 : Représentation graphique d'un exemple de type entité comportant trois attributs

Le nom des attributs est composé de deux morphèmes accolés : le premier tout en minuscule et la première lettre du second en majuscule.

Définition 08 : Une valeur au niveau du type-entité ou de type-association, chaque attribut possède un domaine qui définit l'ensemble des valeurs possibles qui peuvent être choisies pour lui (entier, chaîne de caractères, booléen…). Au niveau de l'entité, chaque attribut possède une valeur compatible avec son domaine.

1. **Règles :**
   - Dans le modèle entité-association, chaque attribut est destiné à recevoir une valeur.
   - Un attribut ne peut être partagé par plusieurs types-entité ou type-associations.
   - Un attribut est une donnée élémentaire, ce qui exclut les données calculées ou dérivées.
   - Un attribut peut être placé dans un type-association uniquement lorsqu'il dépend de toutes les entités liées par le type-association.
   -

Un type-association peut ne pas posséder d''attribut explicite et cela est relativement fréquent, mais il peut posséder au moins des attributs implicites.

2.3.4 Identifiant

Définition 09 : Un identifiant (ou clé) d'un type-entité ou d'un type-association, est un ensemble minimal d'attributs qui permet d'identifier chaque entité ou association de manière unique.

Par exemple dans le cas du type-entité ABONNÉ, si nous choisissons nomAbonné et prénomAbonné comme identifiants(ou clés) et que l'on a des homonymes dans la liste des abonnés, il y aura doublons dans les clés choisies ce qui est interdit. Par contre si nous prenons comme clé une entité numéroSecuritéSociale, nous aurons la garantie de ne pas avoir de doublons. Deux abonnés peuvent avoir le même nom et prénom mais pas le même numéro de sécurité sociale.

Un identifiant est représenté comme suit dans le modèle entité-association :

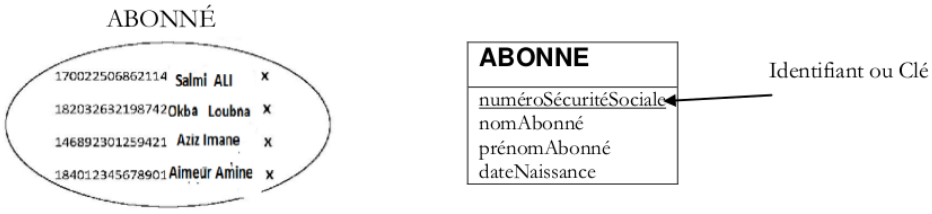


Figure 1.4 : (a) Exemple d'instance de l'entité ABONNE (b) Représentation graphique de l'exemple de type entité ABONNE avec un identifiant.

Un identifiant est représenté dans ce modèle comme un attribut à la seule différence qu'il est souligné. Ce qui permet de le distinguer des autres attributs. Les identifiants sont également placés en tête des autres attributs.

2. **Règles :**
   - Chaque type-entité possède au moins un identifiant, éventuellement formé de plusieurs attributs.
   - Tout type-entité, et type-association possède au moins un identifiant.
   - Il ne peut y avoir la même valeur d'identifiants pour deux instances d'un type-entité (ou type-association).

2.3.5 Cardinalités d'un type-association

Définition 10 : Les cardinalités d'une *patte* reliant un type-association et un type-entité précisent le nombre de fois minimal et maximal d'interventions d'une entité du type-entité dans une association de type-association. La cardinalité minimale doit être inférieure ou égale à la maximale.

Exemple:

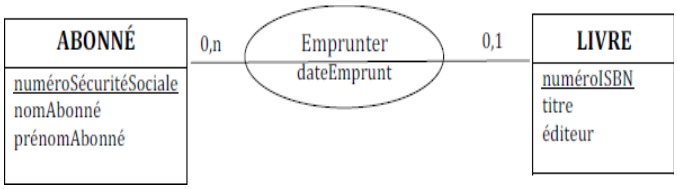


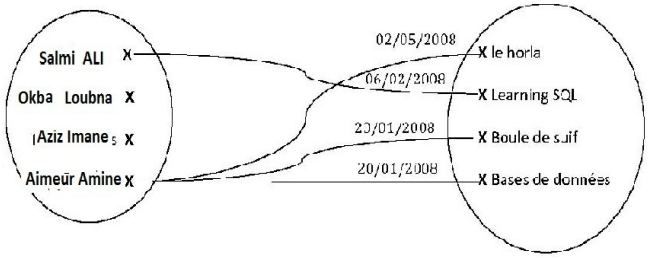
Figure 1.5 : Représentation graphique des cardinalités d'un type association.

Dans cet exemple pédagogique, on suppose qu'un livre ne peut être emprunter que par un auteur.

Il faut lire la cardinalité comme suit :

- Un abonné peut Emprunter plusieurs livres.
- Un livre ne peut être Emprunté que par un seul abonné.

Traduction en modèle informel (ou une instance possible du modèle ci-dessus):



ABONNÉ **LIVRE**

Règles:

- L'expression de la cardinalité est obligatoire pour chaque patte d'un type-association.
- Il ne peut y avoir de cardinalité maximale égale à 0, car elle rendrait le type-association inutile.
- Une cardinalité minimale est toujours 0 ou 1 et une cardinalité maximale est toujours 1 ou n.
- Si une cardinalité est connue et vaut 2 ou plus, alors nous considérons qu'elle est indéterminée et vaut n. En effet, si cette valeur est définie lors de la conception, il se peut qu'elle évolue dans le futur. Il est donc considérer n comme inconnue dès la conception.

Les seules cardinalités admises sont :

- 0,1 : une occurrence du type-entité peut exister en étant impliquée soit dans aucune association soit au maximum dans une seule.
- 0,n : une occurrence du type-entité peut exister en étant impliquée soit dans aucune association soit dans plusieurs associations (sans limite).
- 1,1 : une occurrence du type-entité ne peut exister que si elle est impliquée dans exactement une association.
- 1,n : une occurrence de type-entité ne peut exister que si elle est impliquée dans au moins une association.

2.4. Le modèle entité association : Règles de bonnes pratique

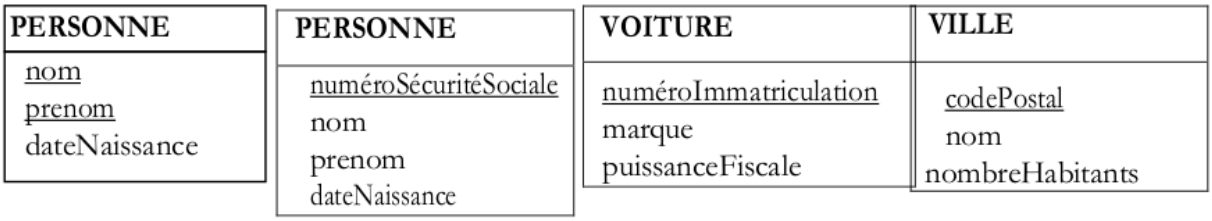
La bonne formulation d'un modèle entités-associations, permet d'éviter une grande partie des sources d'incohérence et de redondances qui pourront se manifester au niveau de la base de données. Nous présentons dans cette section les règles de bonne pratique pour concevoir un bon modèle entités-associations. Ces règles et normes sont des méthodes différentes mais complémentaires pour appliquer un même principe fondamental de conception : « **une seule place pour chaque fait** ».

2.4.1 Choix de l'identifiant

- Il faut éviter de choisir un identifiant susceptible de changer au cours du temps.
- Il faut éviter les identifiants de plusieurs attributs pour identifier un type-entité ou les identifiants dont le type est chaîne de caractères.

Exemple : Identifiants naturels posant des problèmes

Figure 1.6 : Exemples d'identifiants naturels posant des problèmes.



Les identifiants qui semblent s'imposer naturellement sont très souvent des pièges.

- Pour le premier cas du type-entité PERSONNE, le choix du couple d'identifiants (nom, prénom) ne constitue pas un bon identifiant car nous savons tous qu'il peut y avoir des homonymes. Une telle définition du type-entité PERSONNE interdit la présence d'homonymes dans votre base.

- Pour le second cas du type-entité PERSONNE, le choix de l'identifiant numéroSécuritéSociale ne constitue pas un bon identifiant pour une personne. Par exemple les enfants sont généralement enregistrés sous le numéro de sécurité sociale d'un de leurs parents. Deux personnes nées un siècle d'intervalle peuvent porter le même numéro de sécurité sociale. Aussi, un étranger qui arrive sur le sol français, par exemple, sera doté d'un numéro de sécurité sociale provisoire en attendant qu'il remplisse certaines conditions pour enfin avoir son numéro définitif. Ainsi, un même numéro de sécurité sociale peut exister plusieurs fois dans la base, et même être modifié. De même que plusieurs personnes peuvent avoir un même numéro de sécuritésociale.

- Pour le type-entité voiture, son numéro d'immatriculation ne constitue pas non plus un bon identifiant car lorsque le propriétaire change de département, il doit changer son numéro d'immatriculation en fonction du nouveau. Aussi le format de ce numéro et la réglementation a récemment changé. Il n'est donc pas judicieux d'utiliser cet attribut commeidentifiant.

- Pour l'entité VILLE aussi le codePostal ne constitue pas un identifiant valide car un même code postal peut recouvrir plusieurs communes et une commune peut avoir plusieurs code postaux.

Pour remédier à tous ces problèmes liés aux déterminants qui peuvent avoir des cas particuliers gênants pour la conception d'un modèle valide, il faut recourir à des identifiants arbitraires de type entier pour les types-entités.

Cet identifiant deviendra la clé primaire de la table correspondante du schéma relationnel. Cette clé sera incrémentée automatiquement à l'insertion d'une nouvelle instance.

2.4.2 Choix du nom

Dans le modèle Entités-Associations, le nom d'un type-entité, d'un type-association ou d'un attribut doit être unique.

Lorsque les attributs portent le même nom, c'est parfois signe d'une modélisation inachevée ou d'une redondance. Pour remédier à cela il suffit simplement d'ajouter au nom de l'attribut le nom du type-entité ou du type-association dans lequel il se trouve. Ce qui simplifiera par la suite l'écriture de requêtes sur la base si les noms des attributs sont différents.

Exemple 1:

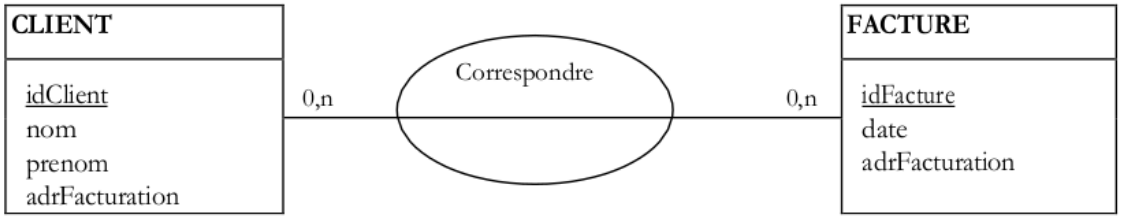


Figure 1.7 : Exemples d'attributs (Adresse de facturation) redondants situation à éviter pour risque d'incohérence.

Ces deux types-entités possèdent chacun un attribut adresseDeFacuration, c'est une redondance. Puisqu'ils décrivent la même information. Cette situation entraine un grand risque d'incohérence dans le modèle et est donc à éviter.

Exemple 2:

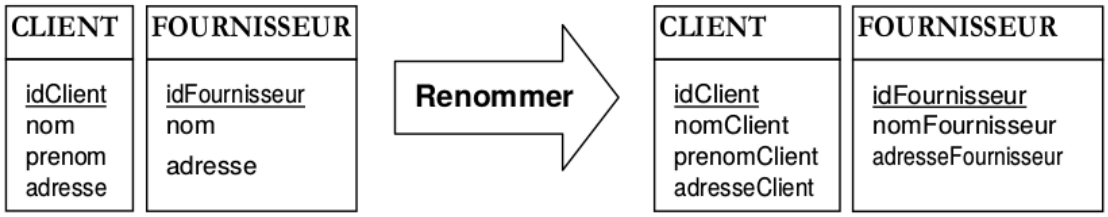


Figure 1.8 : Exemple d'attribut portant le même nom.

Dans ce deuxièmes cas, les deux attributs *adresse*, *nom* doivent être renommés car ils ne contiendront pas les mêmes informations. Les premières concernent le CLIENT et les secondes le FOURNISSEUR.

4.4.3 Normalisation des attributs

- Il faut remplacer un attribut multiple ou composite par un type-association et un type- entité supplémentaires.
- Un attribut est une donnée élémentaire, ce qui exclut des données calculées ou dérivées.
- Un attribut peut-être placé dans un type-association uniquement lorsqu'il dépend de toutes les entités liées par le type-association.
- Une conséquence immédiate de cette règle est: un type-association dont la cardinalité maximale de l'une des pattes est 1, ne peut pas posséder d'attributs puisque l'identifiant d'un tel type-association est simplement composé de

l'identifiant du participant situé à côté de la cardinalité maximale 1.

- Un attribut correspondant à un type énuméré est généralement avantageusement remplacé par un type-entité.

Exemple 1: attribut multiple

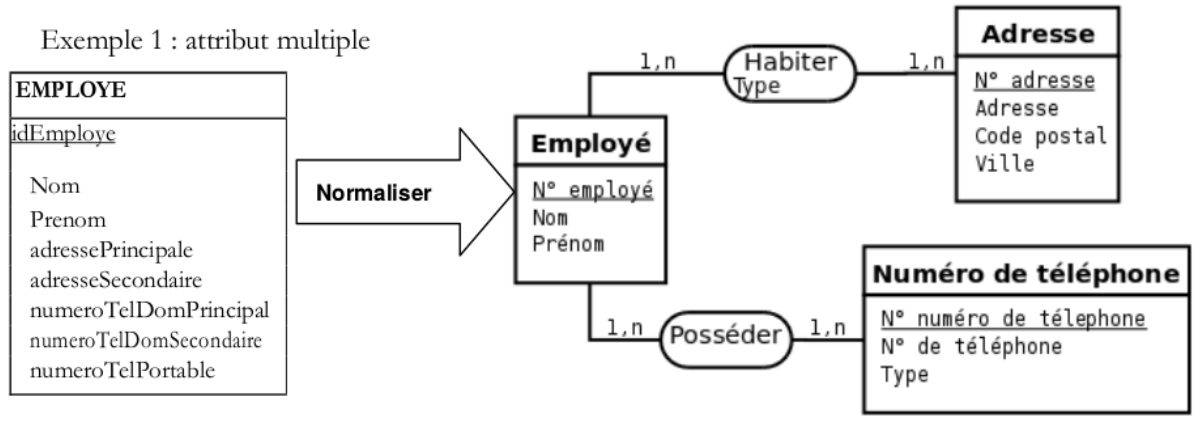


Figure 1.9 : Remplacement des attributs multiples en un type association et un type entité et décomposition des attributs composites.

Dans le type-entité de gauche ici, comment faire si un EMPLOYE possède deux numéros de portables ou de fixe. Ou bien s'il possède deux résidences secondaires ? D'autre part, Si un employé ne possède pas de numéro de portable ou aucune résidence secondaire, alors certains attributs deviennent sans objet. Ce qui est contraire à la définition d'un attribut.

Pour rappel, chaque attribut est destiné à recevoir une valeur. Le seul cas toléré pour qu'un attribut ne reçoive pas de valeur est que la valeur n'est pas connue à un instant donné. La raison ne peut être le fait que l'attribut n'a aucun sens pour l'entité concernée.

Il est également intéressant de décomposer les attributs composites comme l'attribut *Adresse* par exemple. Il est en effet difficile d'écrire une requête portant sur la ville où habitent les employés si cette information est noyée dans un unique attribut *Adresse*.

Exemple 2: un attribut est une donnée élémentaire, ce qui exclut des données calculées ou dérivées.

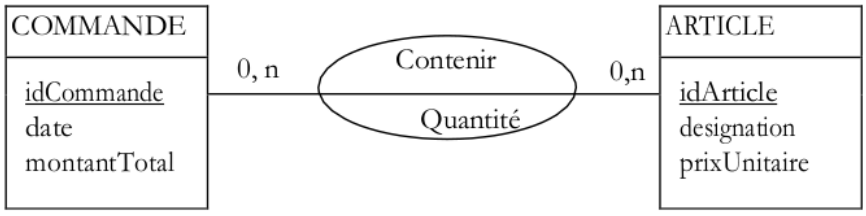


Figure 1.10 : Exemple d'attribut calculé qu'Il faut supprimer (Montant total) du type entité Commande.

Il faut supprimer l'attribut montantTotal du type-entité COMMANDE, car il peut être calculé à partir des attributs quantité et prixUnitaire.

D'autres attributs dérivés sont à éviter, un cas type est l'âge, qui peut être déduit de la date de naissance et de la date courante (d'autant plus que sa valeur change chaque année, contrairement à la date de naissance).

Exemple 3: Un attribut peut être placé dans un type association uniquement lorsqu'il dépend de toutes les entités liées par le type-association.

Par exemple l'attribut quantité du type association Contenir dépend bien à la fois de l'identifiant idCommande et de l'attribut idArticle des types-entités de la collection de Contenir.

Attention, un type-association fonctionnel possédant un attribut est une erreur de modélisation: il faut déplacer cet attribut dans le type-entité connecté à la patte portant la cardinalité maximale de 1.

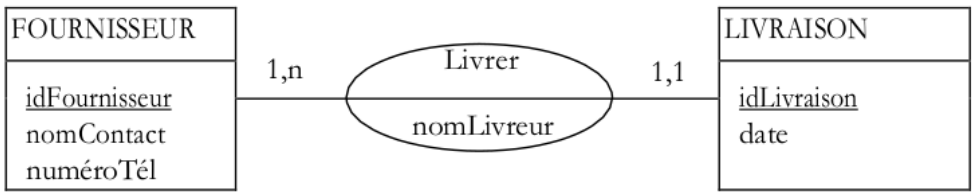


Figure 1.11 : Exemple d'attribut déplacé vers une association.

Ici la cardinalité maximale du type-association Livrer est 1 du côté du type-entité LIVRAISON, l'attribut nomLivreur de Lvrer doit donc être placé dans LIVRAISON.

Exemple 4 : Un attribut correspondant à un type énuméré est généralement remplacé par un type entité.

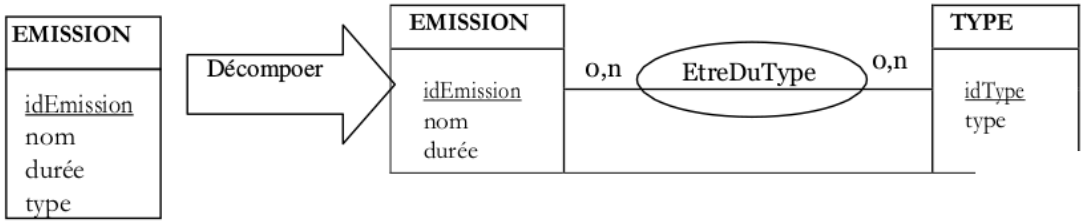


Figure 1.12 : Un attribut correspondant à un type énuméré est généralement avantageusement remplacé par un type entité.

La cardinalité pourrait différer selon la situation à modéliser. L'attribut type caractérise le type d'une émission et peut prendre des valeurs comme : actualité, culture, reportage, divertissement… Remplacer cet attribut par un type-entité permet, d'une part, d'augmenter la cohérence (éviter les variations du genre culturelle, Culturelle, culture…) et d'autre part, si

les cardinalités le permettent, de pouvoir affecter plusieurs types à une même entité (exemple : actualité et culture).

2.4.4 Fusion ou suppression de type-entité ou type- association

- Il faut fusionner (ou factoriser) les types-entités quand c'est possible. Exemple :

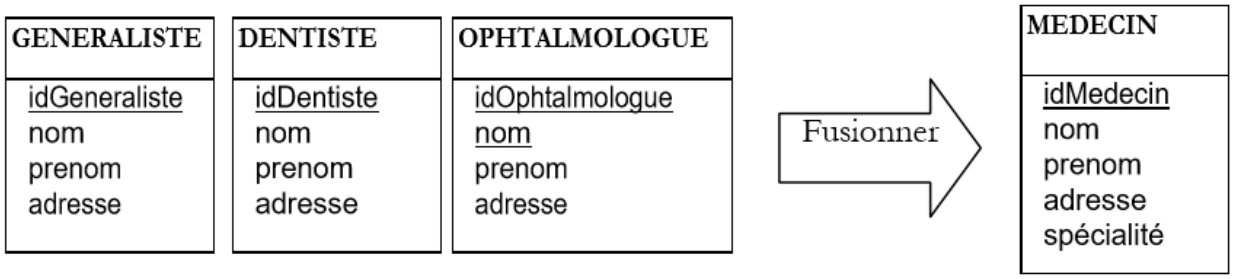


Figure 1.13 : Exemple de factorisation des types entité en introduisant un nouvel attribut.

Dans cet exemple, le nouveau type-entité obtenu contient un attribut (spécialité) dont les valeurs possibles sont l'ensemble des noms des type-entités fusionnés (GENERALISTE, DENTISTE, OPHTALMOLOGUE).

Dans cet exemple, nous avons introduit un nouvel attribut pour aider à la fusion, mais cela n'est pas forcément nécessaire ou souhaitable. Nous présentons cela sous forme d'exemple:

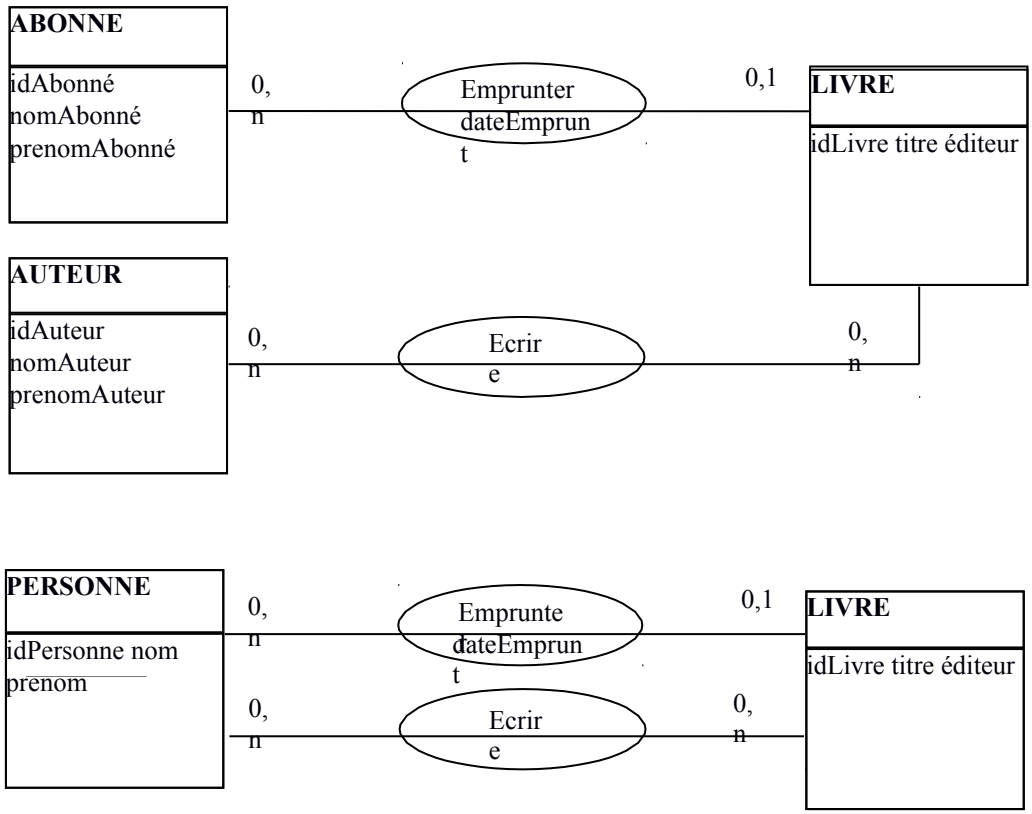


Figure 1.13 : Exemple de factorisation de type entité qu'il faut éviter.

Dans ce cas, ne pas introduire d'attribut permet de représenter une personne à la fois comme abonné et comme auteur.

- Il faut factoriser les type-associations quand c'est possible Exemple :

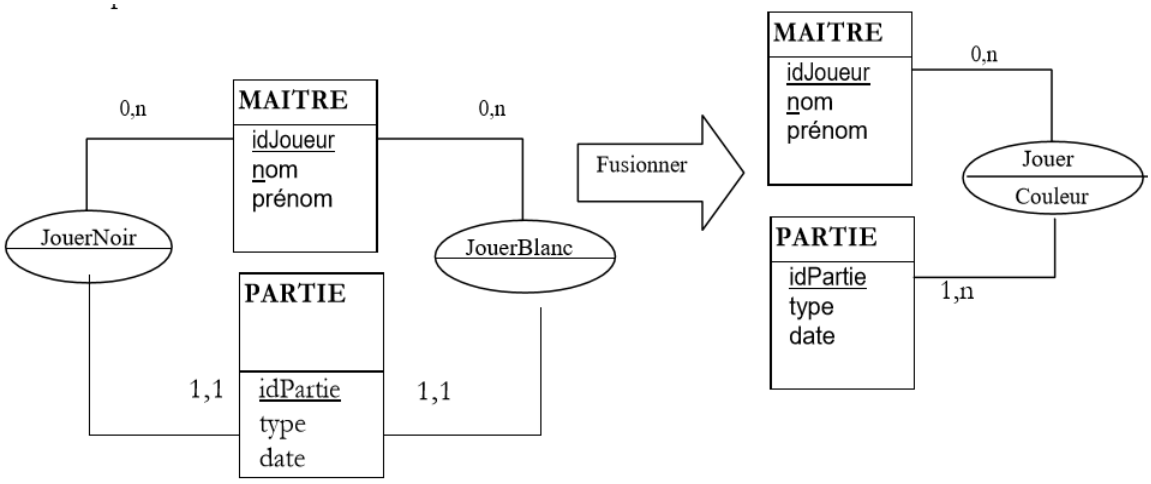


Figure 1.14 : Exemple de factorisation de type association

Un seul type association Jouer suffit pour remplacer les deux types-associations JouerNoir et JouerBlanc.

- Un type-entité remplaçable par un type-association doit être remplacé.
- Lorsque les cardinalités d'un type-association sont toutes 1,1 c'est que le type-association n'a pas lieu d'être.

Dans ce cas, le type-association doit généralement être supprimé et les type-entités correspondants fusionnés comme dans l'exemple ci-dessous.

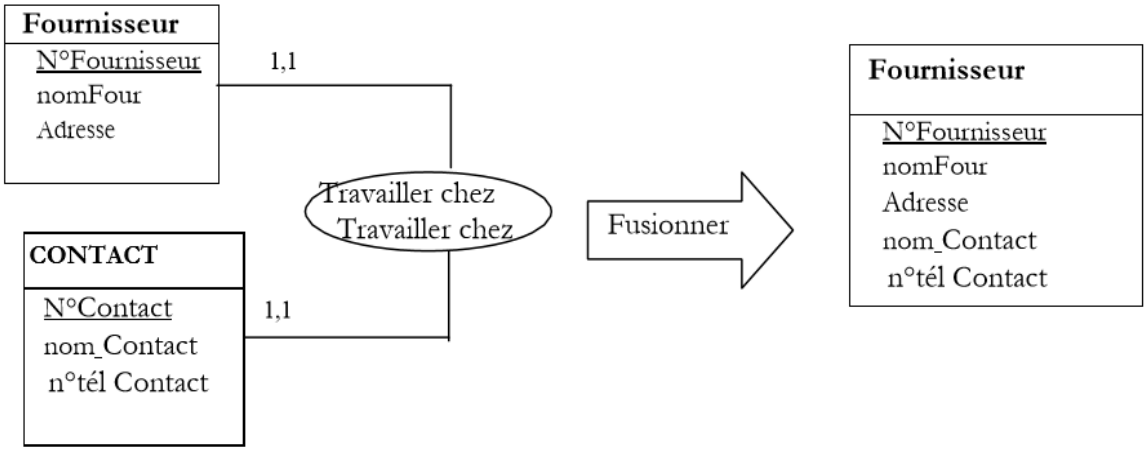


Figure 1.15 : Exemple d'un type association fontôme.

- Il faut éviter les types-associations redondants. Exemple :

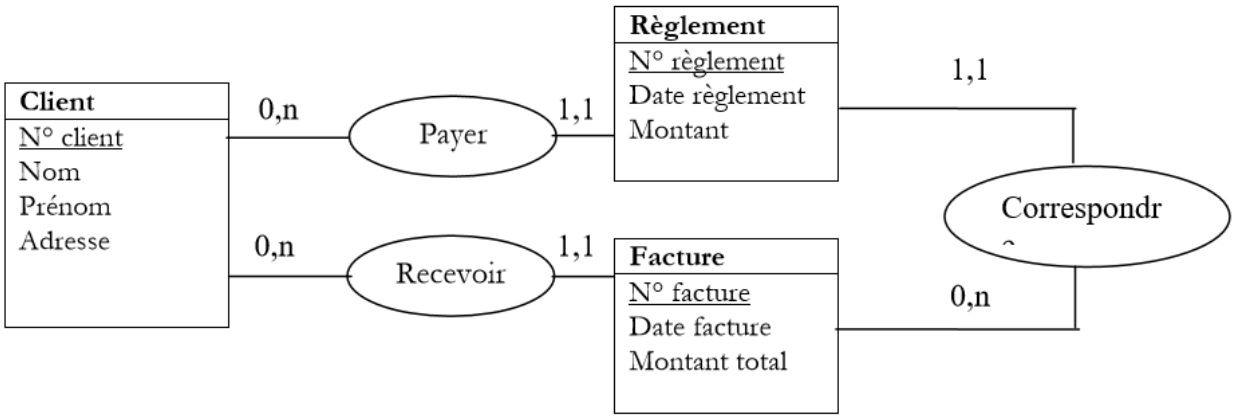


Figure 1.16 : Exemple de type association inutile (Payer).

Si un client ne peut pas régler la facture d'un autre client, alors le type-association Payer est redondant et doit être supprimé. Le client qui a effectué un règlement peut toujours être retrouvé en passant par la facture correspondante.

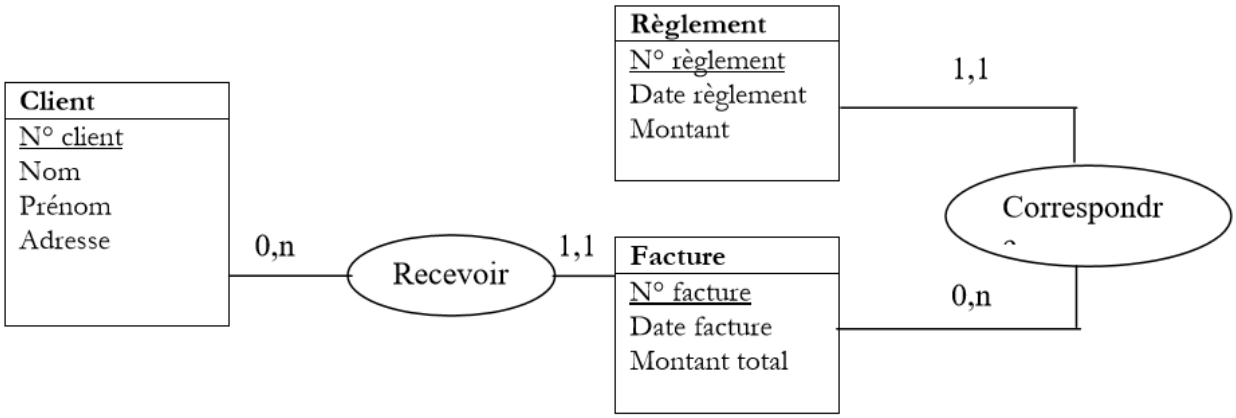


Figure 1.17 : Solution au problème de la redondance du type association de la figure 1.17.

2.5 Complément sur les associations:

2.5.1 Associations plurielles

-

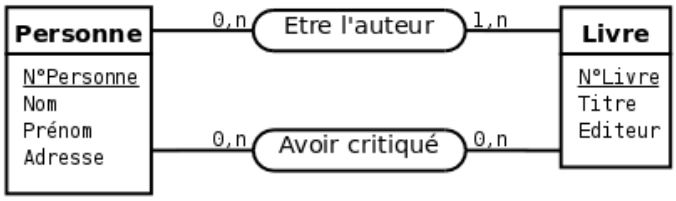


Figure 1.18 : exemple d'associations plurielles entre un type entité Personne et un type entité Livre. Sur ce schéma, un type association permet de modéliser que des personnes écrivent des livres et un autre que des personnes critiquent (au sens de critique littéraire) des livres.

Deux mêmes entités peuvent être plusieurs fois en association (c'est le cas sur la figure 1.18).

2.5.2 Associations réflexive

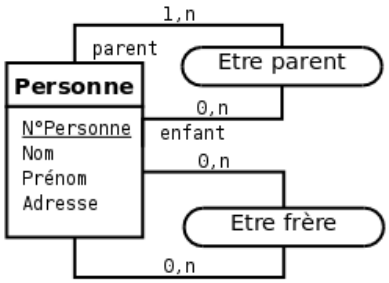


Figure 1.19: Exemple d'associations réflexives sur le type entité Personne. Le premier type association permet de modéliser la relation parent/enfant et le deuxième type association la relation de fraternité.

Les types *association* réflexifs sont présents dans la plupart des modèles.

Définition 11 -type association réflexif- Un type association est qualifié de réflexif quand il matérialise une relation entre un type entité et lui-même (cf. figure 1.19).

Une occurrence de ce type association (i.e. une association) associe généralement une occurrence du type association (i.e. une entité) à une autre entité du même type. Cette relation peut être symétrique, c'est le cas du type association Être frère sur la figure 1.19, ou ne pas l'être, comme le type association Être parent sur cette même figure. Dans le cas où la relation n'est pas symétrique, on peut préciser les rôles sur les pattes du type association comme pour la relation Être parent de la figure 1.19. L'ambiguïté posée par la non-symétrie d'un type association réflexif sera levée lors du passage au modèle relationnel.

2.5.3 Associations n-aire (n>2)

2.5.3.1 Exemple d'association n-aire inappropriée

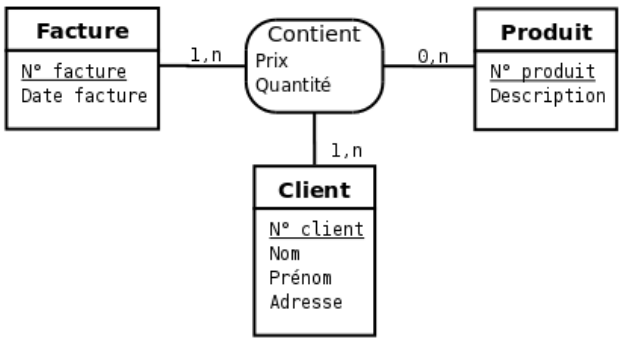


Figure 1.20: E*xemple de type association ternaire inapproprié.*

Le *type association* ternaire *Contient* associant les types *entité Facture*, *Produit* et *Client* représenté sur la figure 1.20 est inapproprié

puisqu'une facture donnée est toujours adressée au même client. En effet, cette modélisation implique pour les associations (instances du *type association*) *Contient* une répétition du numéro de client pour chaque produit d'une même facture.

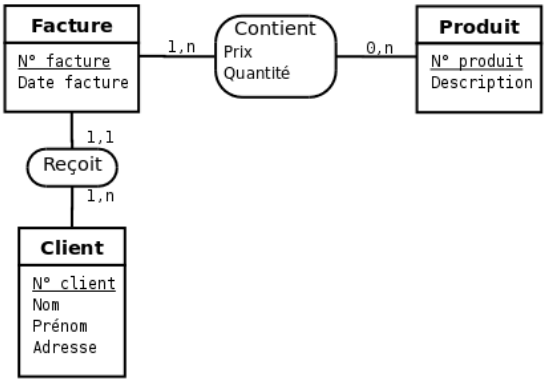


Figure 1.21: Type association ternaire de la figure 1.20 corrigé en deux type association binaires.

La solution consiste à éclater le *type association* ternaire *Contient* en deux *type association* binaires comme représenté sur la figure 1.21.

2.5.3.2 Décomposition d'une association n-aire

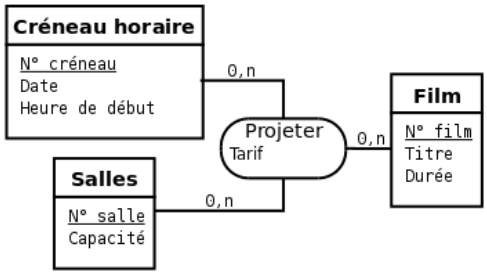


Figure 1.22: Exemple de type association ternaire entre des types entité Créneau horaire, Salle et Film.

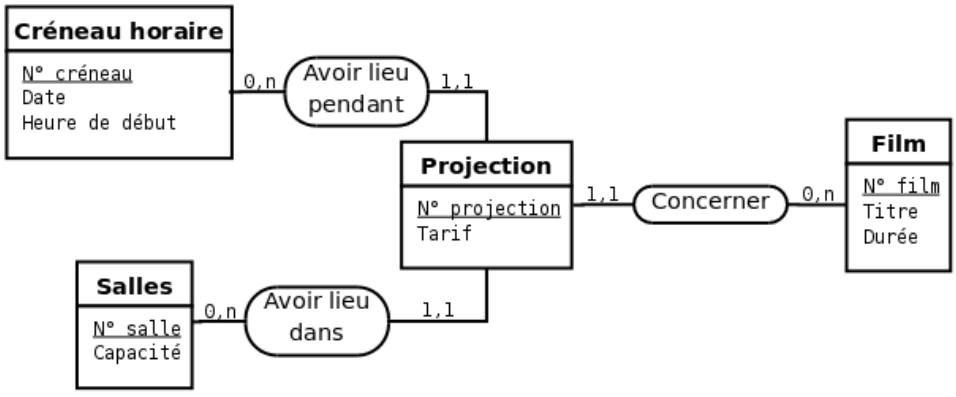


Figure 1.23: Transformation du type association ternaire de la figure 1.22 en un type entité et trois type association binaires.

La figure 1.22 nous montre un exemple de *type association* ternaire entre les types *entité Créneau horaire*, *Salle* et *Film*. Il est toujours possible de s'affranchir d'un *type association* n-aire (n>2) en se ramenant à des types *association* binaires de la manière suivante:

- On remplace le *type association* n-aire par un *type entité* et on lui attribut un identifiant.
- On crée des types *association* binaire entre le nouveau *type entité* et tous les types *entité* de la collection de l'ancien *type association* n-aire.
- La cardinalité de chacun des types *association* binaires créés est 1,1 du côté du *type entité* créé (celui qui remplace le *type association* n-aire), et 0,*n* ou 1,*n* du côté des types *entité* de la collection de l'ancien *type association* n-aire.

La figure 1.23 illustre le résultat de cette transformation sur le schéma de la figure 1.22.

L'avantage du schéma de la figure 1.23 est de rendre plus intelligible la lecture des cardinalités. Il ne faut surtout pas le voir comme un aboutissement, mais comme une étape intermédiaire avant d'aboutir au schéma de la figure .23 . Ainsi, le mécanisme, que nous venons de détailler ci-dessus, de passage d'un *type association* n-aire (n>2) à un *type entité* et *n type association* binaires est tout à fait réversible à condition que :

- toutes les pattes des types *association* binaires autour du *type entité* central ont une cardinalité maximale de 1 au centre et de *n* à l'extérieur;
- les attributs du *type entité* central satisfont la règle de bonne formation des attributs de *type association*.

2.5.3.3 Détection d'une erreur de modélisation par décomposition d'une association n-aire

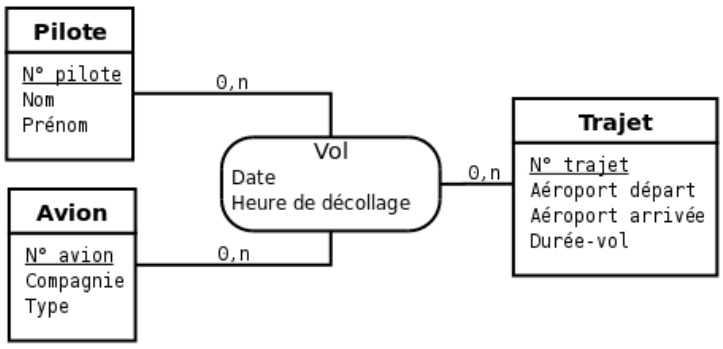


Figure 1.24 : Modèle représentant un type association ternaire Vol liant trois type entité Avion, Trajet et Pilote.

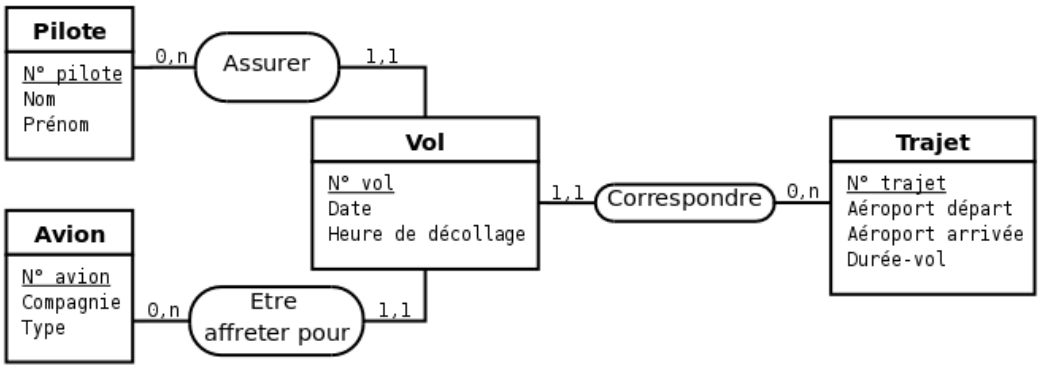


Figure 1.25: Transformation du type association ternaire de la figure 2.12 en un type entité et trois type association binaires.

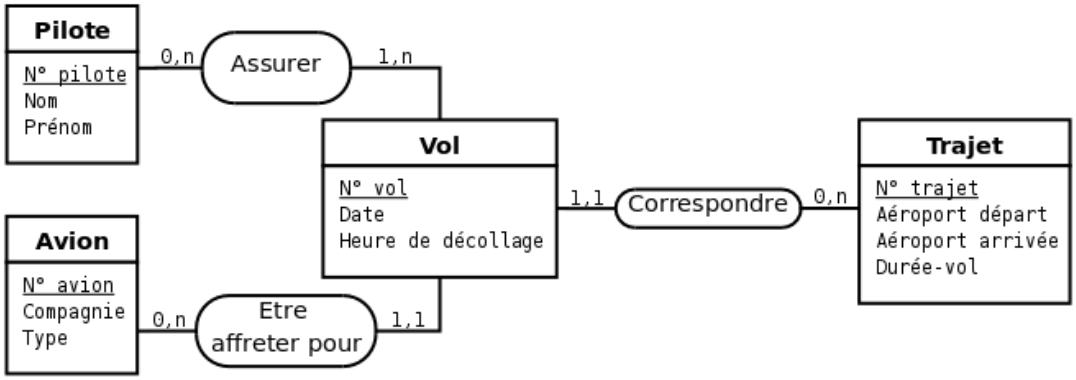


Figure 1.26: Modèle de la figure 2.13 corrigé au niveau des cardinalités.

Passer par cette étape intermédiaire ne comportant pas de *type association* n-aire (n>2) peut, dans certains cas, éviter d'introduire un *type association* n-aire inapproprié. Imaginons par exemple un *type association* ternaire *Vol* liant trois *type entitéAvion*, *Trajet* et *Pilote* comme représenté sur la figure 1.24.

La transformation consistant à supprimer le *type association* ternaire du modèle de la figure 1.25 produit le modèle de la figure 1.25. Ce modèle fait immédiatement apparaître une erreur de conception qui était jusque-là difficile à diagnostiquer: généralement, à un vol donné sont affectés plusieurs pilotes (par exemple le commandant de bord et un copilote) et non pas un seul.

Le modèle correct modélisant cette situation est celui de la figure 1.26 où le *type entité Vol* ne peut être transformé en un *type association* ternaire *Vol* comme sur la figure .24.

Chapitre 3

Le modèle relationnel de données

Objectifs:

- Présenter le modèle logique des données;
- Présenter les formes normales;
- Présenter les différentes méthodes pour normaliser une relation.

Dans ce cours: Vous allez voir:

- Introduction;
- Eléments constitutifs du modèle;
- Passage du modèle E/A au modèle relationnel;
- Dépendance Fonctionnelle (DF);
- Formes normales;

- Normalisation.

Chapitre 03

Le modèle relationnel de données

3.1. Introduction

Le modèle relationnel -aussi connu sous le nom de Modèle Logique de Données, MLD- représente la base de données comme un ensemble de tables, sans préjuger de la façon dont les informations sont stockées dans la machine. Les tables constituent donc la structure logique du modèle relationnel. Au niveau physique, le système est libre d'utiliser n'importe quelle technique de stockage (fichiers séquentiels, indexa ge, adressage dispersé, séries de pointeurs, compression…) dès lors qu'il est possible de relier ces structures à des tables au niveau logique. Les tables ne représentent donc qu'une abstraction de l'enregistrement physique des données en mémoire. De façon informelle, le modèle relationnel peut être défini de la manière suivante :

les données sont organisées sous forme de tables à deux dimensions, encore appelées relations, dont les lignes sont appelées n-uplet ou tuple en anglais ;
les données sont manipulées par des opérateurs de l'algèbre relationnelle ;
l'état cohérent de la base est défini par un ensemble de contraintes d'intégrité.

3.2. Eléments constitutifs du modèle

Définition 01 : Un **attribut** est un identifiant (un nom) décrivant une information stockée dans une base.

Exemples d'attribut : l'âge d'une personne, son nom, le numéro de sécurité sociale.

Définition 02: Le **domaine** d'un attribut est l'ensemble, fini ou infini, de ses valeurs possibles.

Exemple: l'attribut numéro de sécurité sociale a pour domaine l'ensemble des combinaisons de quinze chiffres. L'attribut nom a pour domaine l'ensemble des combinaisons de lettres (une combinaison comme cette dernière est généralement appelée chaîne de caractères ou, plus simplement, chaîne).

Définition 03: Une **relation** est un sous-ensemble du produit cartésien de n domaines d'attributs (n>0).
Une relation est représentée sous la forme d'une table à deux dimensions dans laquelle les n attributs correspondent aux titres des n colonnes.

Définition 04 : Un schéma de relation précise le nom de la relation ainsi que la liste des attributs avec leurs domaines.

Exemple de relation avec son schéma :
On note cette relation de schéma de la façon suivante :
PERSONNE (N°Sécu : Entier, Nom : Chaîne, Prénom : Chaîne)

PERSONNE		
N° Sécu	**Nom**	**Prénom**
354338532958234	Salmi	Sara
354338532958235	Bali	Amine
354338532958236	Sari	Lisa
354338532958237	Sari	Meriem

Définition 05: Le **degré** d'une relation est son nombre d'attributs.

Définition 06: Une occurrence, ou n-uplets ou tuples, est un élément de l'ensemble figuré par une relation. Autrement dit, une occurrence est une ligne de la table qui représente la relation.

Définition 07: La cardinalité d'une relation est son nombre d'occurrences.

Définition 08: Une clé candidate d'une relation est un ensemble minimal des attributs de la relation dont les valeurs identifient à coup sûr une occurrence.

La valeur d'une clé candidate est donc distincte pour tous les tuples de la relation. La notion de clé candidate est essentielle dans le modèle relationnel.

Règle : Toute relation a au moins une clé candidate et peut en avoir plusieurs.

Ainsi, il ne peut jamais y avoir deux tuples identiques au sein d'une relation. Les clés candidates d'une relation n'ont pas forcément le même nombre d'attributs. Une clé candidate peut être formée d'un attribut arbitraire qui n'a d'autre objectif que de servir de clé.

Définition 09: La clé primaire d'une relation est une de ses clés candidates. Pour signaler la clé primaire, ses attributs sont généralement soulignés.

Définition 10: Une clé étrangère dans une relation est formée d'un ou plusieurs attributs qui constituent une clé candidate dans une autre relation.

Attention, une clé étrangère != clé candidate dans une relation.

Définition 11: Un schéma relationnel est constitué par l'ensemble des schémas de relation avec mention des clés étrangères.

Définition 12: Une base de données relationnelle est constituée par l'ensemble des n-uplets des différentes relations du schéma relationnel.

3.3. Passage du modèle E/A au modèle relationnel

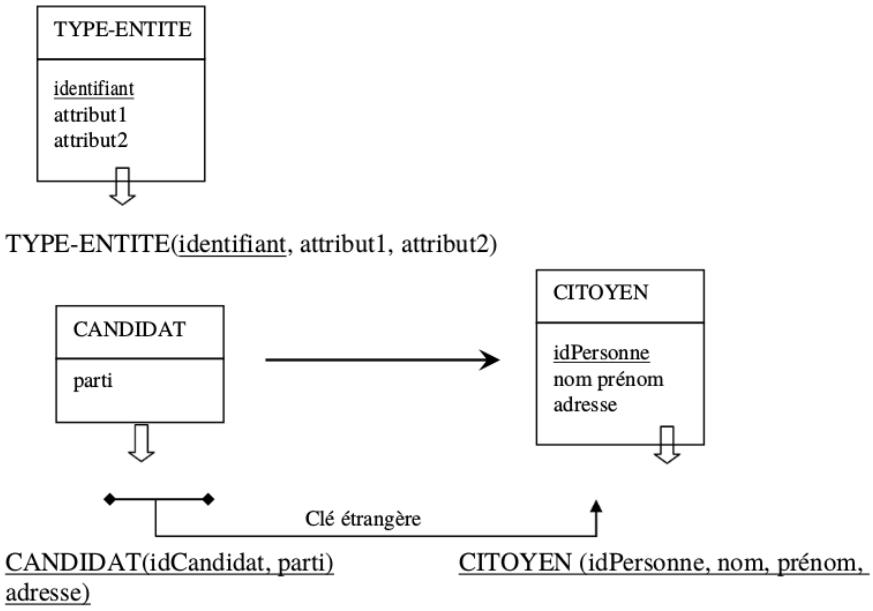
3.3.1 Règles générales

Pour traduire un modèle entités-associations vers un modèle relationnel, il faut appliquer les règles suivantes:

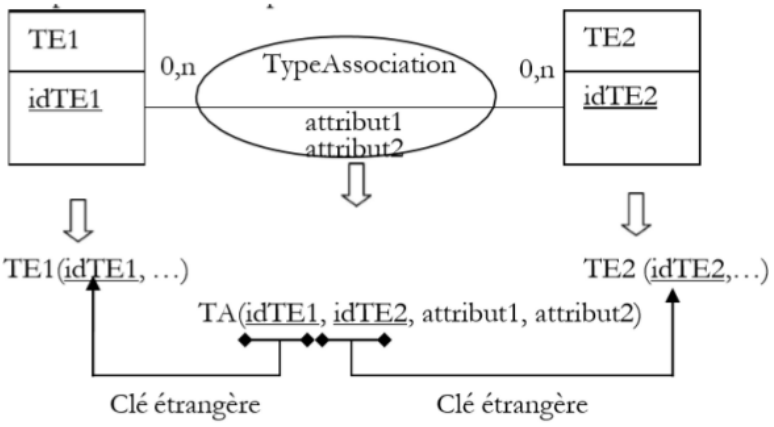
1. La normalisation devrait toujours être effectuée avant le passage au modèle relationnel. Dans les faits, elle est parfois faite a posteriori ce qui impose une surcharge de travail importante et produit un schéma relationnel non conforme au modèle entités – associations.

2. Chaque type-entité donne naissance à un schéma de relation. Chaque attribut de ce type- entité devient un attribut du schéma de relation. L'identifiant est coservé en tant que

clé du schéma de relation. Il faut faire attention aux éventuels type- entités spécifiques qui traduisent l'apparition d'au moins une clé étrangère. Cette étape est illustrée dans cette figure :



3. Chaque type-association maillé (chacune des pattes à pour cardinalité maximale n) donne naissance à un schéma de relation. Chaque attribut de ce type- association devient un attribut du schéma de relation. L'identifiant est formé par l'ensemble des identifiants des types-entités qui interviennent dans le type association. Chacun de ces identifiants devient une clé étrangère faisant référence au schéma de relation correspondant au type-entité dont l'identifiant provient. Cette étape est illustrée ci- dessous.

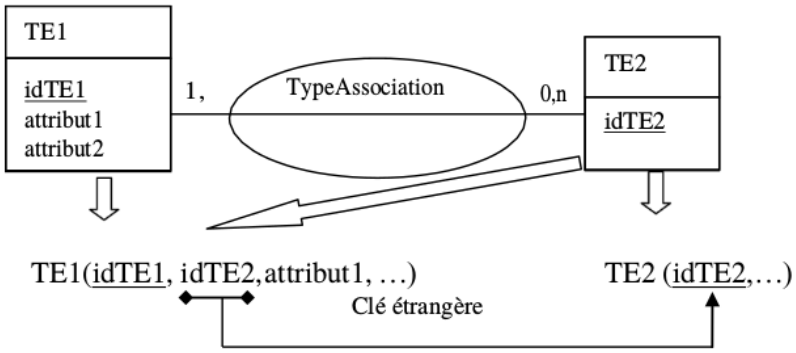


Chaque type-association maillé donne naissance à un schéma de relation dont la clé primaire est composée de clés étrangères.

4. Un type-association dont une patte a une cardinalité maximale égale à 1 (il ne doit donc pas posséder d'attribut) ne devient pas un schéma de relation. Il décrit en effet une dépendance fonctionnelle. Le schéma de relation correspondant au type-entité dont la patte vers le type-association a une cardinalité maximale valant 1, se voit ajouter comme attribut (et donc comme clé étrangère) l'identifiant de l'autre type-entité. Attention, si la

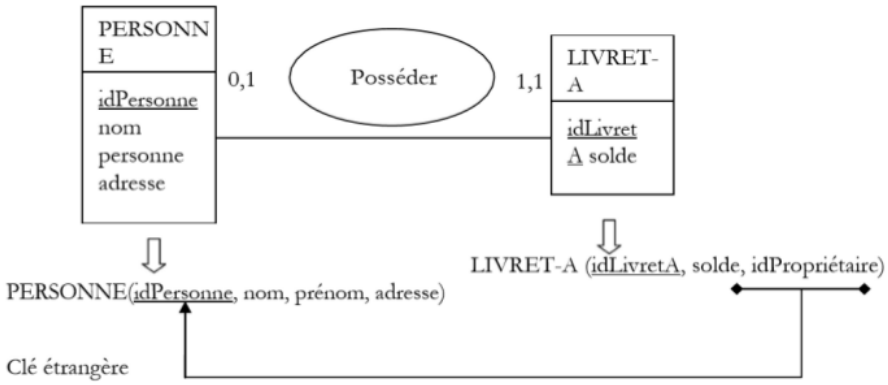
patte correspond à un lien identifiant, l'attribut ajouté doit être incorporé à la clé du schéma de relation.

Ainsi, un type-association fonctionnel ne devient pas un schéma de relation mais se traduit simplement par une clé étrangère.



3.3.2 Cas particulier d'un type-association 1 vers 1

Soit le type-association, Posséder, suivant :



Ce type-association a toutes ses cardinalités maximales à 1. L'application des règles de passage du modèle entités-associations au modèle relationnel que nous avons énoncées précédemment nous donnerait :

- PERSONNE(idPersonne, nom, prénom, adresse, idLivretA) où idLivretA est une clé étrangère qui fait référence au schéma de relation LIVRET-A.
- LIVRET-A(idLivretA, solde, idPersonne) où idPersonne est une clé étrangère qui fait référence au schéma de relation PERSONNE.

Le type-association Posséder étant du type 1 vers 1, il est entièrement matérialisé dans le schéma de relation LIVRET-A par la clé étrangère idPersonne. Il est donc inutile de la matérialiser à nouveau dans le schéma de relation PERSONNE (ou inversement). Il faut donc choisir de supprimer idLivretA de PERSONNE ou idPersonne de LIVRET-A. La cardinalité 0,1 nous indique le bon choix : une personne n'a pas forcément de livret A.
Le schéma relationnel adéquat devient donc :

- PERSONNE(idPersonne, nom, prénom, adresse)
- LIVRET-A(idLivretA, solde, idPropriétaire) où idPropriétaire , nouveau nom de idPersonne, est une clé étrangère qui fait référence au schéma de relation PERSONNE.

3.3.3 Cas particulier d'un type-entité sans attribut autre que sa clé

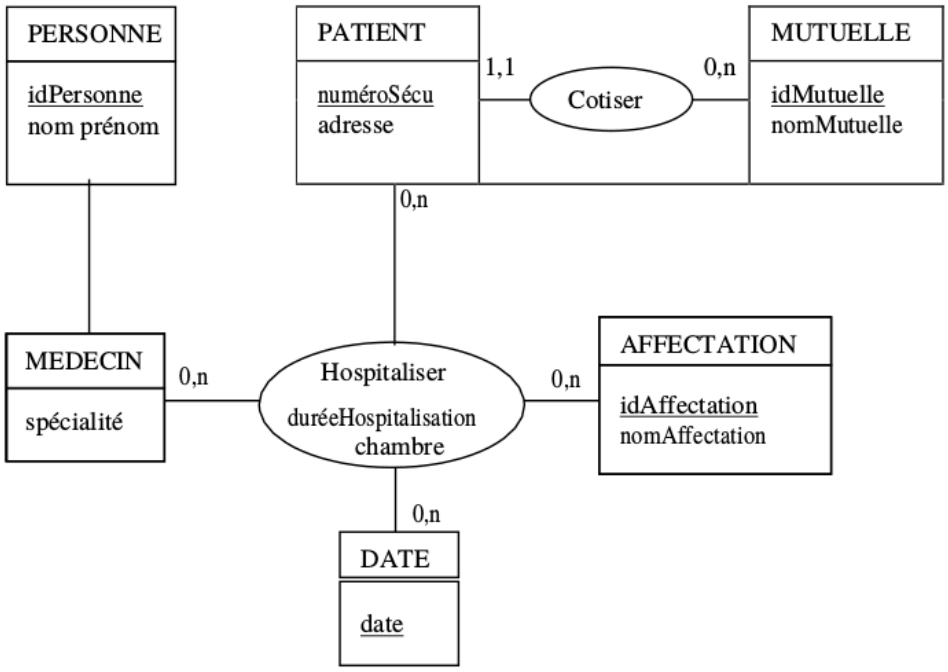
Lorsqu'un type-entité ne possède pas d'attributs en dehors de sa clé, il ne faut pas nécessairement en faire un schéma de relation.

Dans cet exemple, le type-entité DATE ne doit pas se traduire par un schéma de relation, car ce schéma ne véhiculerait pas d'information. Le schéma relationnel adéquat correspond au modèle entités-associations de cet exemple est :

PATIENT(idPatient, numéroSécu, nom, prénom)
MEDECIN(idMédecin, nom, prénom, spécialité)
CONSULTER(idPatient, idMédecin, date) où idPatient et idMédecin sont des clés étrangères qui font respectivement référence aux schémas de relation PATIENT et MEDECIN.

Par contre, si le type-entité sans attribut autre que sa clé correspond à un type énuméré (comme GENRE par exemple), il faut le matérialiser par un schéma de relation.

Exemple complet



Le schéma relationnel déduit de ce modèle entités-associations est :
- PERSONNE(idPersonne, nom, prénom)
- MEDECIN(idMedecin, spécialité) où idMedecin est une clé étrangère qui fait référence au schéma de relationPERSONNE
- PATIENT(idPatient, numéroSécu, adresse, mutuelle) où idPAtient et mutuelle sont des clés étrangères qui font respectivement référence aux schémas de relation

PERSONNE et MUTUELLE
- MUTUELLE(idMutuelle, nomMutuelle)
- AFFECTATION(idAffectation, nomAffectation)
- HOSPITALISER(idPatient, idAffectation, idMedecin, dateEntrée, chambre, durée-Hospitalisation) où idPatient, idAffectation et idMédecin sont des clés étrangères qui font respectivement référence aux schémas de relation PATIENT, AFFECTATION et MEDECIN.

3.4. La Dépendance Fonctionnelle (DF)

3.4.1 Définitions et propriétés des dépendances fonctionnelles

3.4.1.1 Définitions

Définition 13: Soit R(A1 , A2 , .., An) un schéma de relation, et X et Y des sous-ensembles de A1 , A2 , .., An . X détermine You Y dépend fonctionnellement de X si, et seulement si, des valeurs identiques de X impliquent des valeurs identiques de Y. Notation : X → Y.

Autrement dit, il existe une dépendance fonctionnelle entre un ensemble d'attributs X et un ensemble d'attributs Y, notée X →Y, si connaissant une occurrence de X nous ne pouvons lui associer qu'une seule occurrence de Y.

Définition 14 : Une dépendance fonctionnelle élémentaire est une dépendance fonctionnelle de la forme X → A, où Aest un attribut unique n'appartenant pas à X et où il n'existe pas X' inclus au sens strict dans X. (i.e. X' → X) tel que X' → A.

Autrement dit, une dépendance fonctionnelle est élémentaire si la cible est un attribut unique et si la source ne comporte pas d'attributs superflus. La question sur l'élémentarité d'une dépendance fonctionnelle ne doit donc se poser que lorsque la partie de gauche de la dépendance fonctionnelle comporte plusieurs attributs.

Définition 15: Une dépendance fonctionnelle X→ Aest une dépendance fonctionnelle directe s'il n'existe aucun ensemble d'attributs Ytel que X →Yet Y →A.
En d'autres termes, cela signifie que la dépendance entre X et A ne peuvent pas être obtenue par transitivité.

3.4.1.2 Propriétés des dépendances fonctionnelles

Les dépendances fonctionnelles possèdent des propriétés fondamentales qu'on appelle les axiomes d'Armstrong, ces propriétés sont les suivantes :

a) Réflexibilité : si Y → X alors X →Y est vérifiée, en particulier X
b) Augmentation : si X →Y est vérifiée alors XZ →YZ
c) Transitivité : si X →Y et Y →Z sont vérifiées alors X→Z

Remarque: Plusieurs autres règles peuvent être déduites de ces règles, elles sont construites à partir des axiomes pour faciliter les démonstrations.

d) Additivité : X →Y et X →Z alors X→YZ (NB : YZ= YUZ)
e) Pseudo-transitivité : X→Y et WY →Z alors WX → Z
f) Décomposition : X→Y, et Z est inclus dans Y alors X→ Z

3.4.2 Graphes de Dépendances Fonctionnelles (GDF)

Il est souvent utile de représenter les dépendances fonctionnelles par un graphe appelé graphe des dépendances fonctionnelles (GDF).:

Soit G un ensemble d'attributs et F un ensemble de DF. Le GDF relatif à G et F est un graphe où les sommets sont les attributs de G et les arcs les dépendances fonctionnelles de F.

3.4.3 Fermeture Transitive (FT)

On appelle fermeture transitive l'ensemble des DF considérées enrichi de toutes les DF élémentaires déduites par transitivité.

Exemple :

Soit F l'ensemble de DF relatif à R(N°veh, Type, Couleur, Marque, Puissance)
F= (N°vehT→ ype ; Type →Marque ; Type →Puisance ; N°veh → Couleur)
La fermeture transitive FT de F est :
FT= F U (N°veh → Marque ; N°veh →Puissance)

3.4.4 Couverture Minimale (CM)

La couverture minimale a pour objectif de trouver un ensemble minimal de DF représentant «la même information » qu'un ensemble de DF donné : F, mais sans redondances.

On appelle couverture minimale un ensemble F de DF élémentaires associé à un ensemble d'attributs vérifiant les propriétés suivantes :

a) Aucune DF dans F n'est redondante, c-à-dire pour toute DF f de F, F-f n'est pas équivalent à F ;
b) Toute DF élémentaire des attributs est dans la fermeture de F.

3.5. Les formes normales

Normaliser un schéma relationnel c'est le transformer en un schéma relationnel normalisé équivalent, c'est-à-dire, contenant les mêmes informations que le schéma de départ. La décomposition doit être sans perte de données et autant que possible sans perte de dépendance. Ceci se fait dans le but de respecter l'idée suivante : « un seul fait dans un seul lieu ».

Il existe 6 formes normales : les formes normales numérotées de 1 à 5 et la forme normale de Boyce-Codd qui en est une extension. Elles représentent un certain degré de normalisation du modèle E-A. Nous présentons ici les 3 premières formes normales et la forme de Boyce- Codd.

Il faut savoir que plus le niveau de normalisation est élevé, plus il est exempt d'anomalies.

3.5.1 Première forme normale :

Définition 16 : Une relation est en première forme normale si, et seulement si, tout attribut contient une valeur atomique (non multiple, non composée).

Prenons, par exemple, le pseudo schéma de relation suivant :

PERSONNE(idPersonne, nomPersonne, prénomPersonne, adresse, voitures)

Ce pseudo schéma de relation n'est pas 1FN, car l'attribut adresse est composite et l'attribut
voitures est multiple. Il faut le décomposer en :

- PERSONNE(idPersonne, nomPersonne, prénomPersonne, numéroEtRue, codePostal, ville)
- VOITURE(idVoiture, modèle, marque, #propriétaire)

où propriétaire est une clé étrangère qui fait référence au schéma relation PERSONNE

Remarques:
La première forme normale impose que chaque ligne d'une relation ait une seule valeur pour chaque colonne (ou attribut), ce qui est justement la définition d'une table. Donc, une table est nécessairement en première forme normale au sens du modèle relationnel.

3.5.2 Deuxième forme normale :

Définition 17 : Une relation est en deuxième forme normale si, et seulement si :

- elle est en 1FN ;
- toutes les dépendances fonctionnelles entre la clé et les autres attributs sont élémentaires.

Une relation peut être en deuxième forme normale par rapport à une de ses clés candidates et ne pas l'être par rapport à une autre. Une relation avec une clé primaire réduite à un seul attribut, ou contenant tous les attributs, est forcément en deuxième forme normale.

Soit, par exemple, le schéma de relation suivant :

AFFECTATION(idPersonne, idEtablissement, nomPersonne, prénomPersonne, nomEtablissement)

Supposons qu'une personne puisse être affectée à plusieurs établissements et qu'à un établissement sont affectées plusieurs personnes. Cette relation n'est pas en 2FN car, par exemple, nomPersonne ne dépend que de idPersonne. Pour normaliser cette relation, il faut la décomposer de la manière suivante :

- PERSONNE(idPersonne, nomPersonne, prénomPersonne)
- ETABLISSEMENT(idEtablissement, nomEtablissement)

- AFFECTATION(idPersonne, idEtablissement) où idPersonne et idEtablissement sont des clés étrangères qui font respectivement référence aux schémas de relation PERSONNE et ETABLISSEMENT.

3.5.3 Troisième forme normale

Définition 18: Une relation est en troisième forme normale si, et seulement si :

- elle est en 2FN ;
- tout attribut n'appartenant pas à la clé n'est pas en dépendance fonctionnelle directe avec un ensemble d'attributs non-clé.

Note: un ensemble d'attributs non-clé est un ensemble d'attributs qui ne constitue pas une clé candidate pour la relation.

Une relation peut être en 3FN par rapport à une de ses clés candidates et ne pas l'être par rapport à une autre. Une relation en 2FN avec au plus un attribut qui n'appartient pas à la clé primaire est forcément en 3FN.

Exemple, soit le schéma relationnel suivant :

ETUDIANT(idEtudiant, nomEtudiant, prénomEtudiant, établissement, villeEtablissement)
Cette relation n'est pas 3FN, car l'attribut villeEtablissement dépend de établissement qui n'est pas une clé candidate. Pour normaliser cette relation, il faut la décomposer de la manière suivante :

- ETUDIANT(idEtudiant, nomEtudiant, prénomEtudiant, idEtablissement) où idEtablissement est une clé étrangère qui fait référence au schéma ETABLISSEMENT.
- ETABLISSEMENT(idEtablissement, nomEtablissement, villeEtablissement).

Prenons maintenant l'exemple du schéma de relation suivant :
ETUDIANT(idEtudiant, numéroSécu, numéroINE, nomEtudiant, prénomEtudiant)
Bien que nomEtudiant dépende de numéroSécu ou de numéroINE, cette relation est bien en 3FN puisque numéroSécu et numéroINE sont des clés candidates de la relation ETUDIANT.

3.5.4 Forme normale deBoyce-Codd

Définition 19: Une relation est en forme normale de Boyce-Codd si, et seulement si, les seules dépendances fonctionnelles élémentaires sont celles dans lesquelles une clé détermine un attribut non-clé.

Cette forme normale permet de renforcer certaines lacunes à la 3ème forme normale comme le cas où un attribut non-clé détermine une partie de la clé.

Une relation en BCFN l'est pour toutes ses clés candidates. Dans la pratique, la plupart des

problèmes de conception peuvent être résolus en appliquant les concepts de 3FN et de BCFN. Un modèle en BCFN est considéré comme étant de qualité suffisante pour une implantation.
Prenons comme illustration une situation où nous voulons connaître la capacité d'accueil des lycées de France. Plusieurs lycées peuvent avoir le même nom, mais pas le même département. Une grande ville peut posséder plusieurs lycées. Nous proposons de modéliser cette situation par le schéma de relation suivant :

LYCEE(nomLycée, département, ville, capacité)

Ce schéma de relation est en 3FN. Remarquons toutefois que le couple (nomLycée, ville) est également une clé candidate, mais le schéma de relation LYCEE n'est pas en 2FN par rapport à cette clé puisque le département dépend de la ville. Bien qu'en 3FN, le schéma de relation LYCEE n'est cependant pas en BCFN puisqu'un attribut non-clé (ville) détermine une partie de la clé (département). Pour normaliser cette relation, il faut la décomposer de la manière suivante :

LYCEE(nomLycée, ville, capacité) où ville est une clé étrangère qui fait référence au schéma de relation VILLE
VILLE(ville, département)

3.6. La Normalisation

Une mauvaise répartition des données dans les relations peut occasionner de graves problèmes lors de l'évolution de la base.
La normalisation consiste en différents stades de qualité qui permette d'éviter certaines erreurs de conception qui génèrent de la redondance, la limitation ou la perte de données, l'incohérence ou l'effondrement des performances des traitements. L'idée essentielle de la normalisation est : autant que possible : 'un seul fait dans un seul lieu' c'est-à-dire une seule notion sémantique par relation. Pour obtenir cela, le processus de normalisation conduit le plus souvent à éclater des relations en plusieurs relations normalisées qui redonnent les relations d'origine par jointure.
Comment normaliser un schéma relationnel ?
- Approche par décomposition : On part d'une table contenant tous les attributs et on décompose jusqu'à ce qu'il n'y ait plus de Redondances
- Approche par synthèse : A partir de l'ensemble des attributs et des dépendances fonctionnelles on constitue les tables.

3.6.1 Décomposition sans perte d'information (SPI)

Une décomposition de R en R1,R2,....,Rn est sans perte d'informations (SPI)si : R=R1[ ] R2 [ ]... [ ] Rn

3.6.2 Préservation des DF par décomposition

On dit que la décomposition de R en R1,R2,..., Rp préserve les DF (ou sans perte de DF ,SPD) si la fermeture transitive des DF de R est la même que celle de l'union des DF de R1,R2,...,Rp.

3.6.3 Algorithme de Synthèse

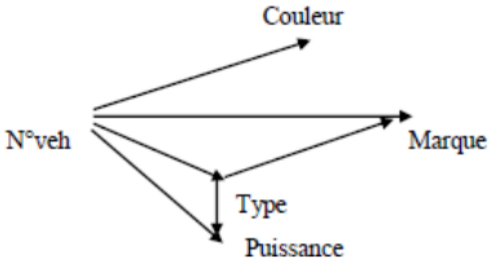
Il existe au moins une décomposition en 3FN sans perte d'information (SPI) et sans perte de DF. Une telle décomposition est générée par algorithme dit de synthèse ayant pour

entrée l'ensemble des attributs et l'ensemble des DF et pour sortie des relations en 3FN. Il est basé sur le calcul de la couverture minimale (ou irredondante) d'un ensemble de DF.

Algorithme :

1) A partir du GDF, calculer une couverture minimale C,
2) Editer l'ensemble des attributs isolés dans une même relation,
3) Rechercher le plus grand ensemble X d'attributs qui déterminent d'autres attributs,
4) Editer la relation R(X, A1, A2, ..., An),
5) Supprimer les DF (X → A1, X → A1,...., X → A) du GDF de la couverture minimale C,
6) Supprimer les attributs isolés de C,
7) Répéter l'opération de réduction du graphe C à partir de l'étape 3, jusqu'à ce que C soit vide.

Exemple : Soit le GDF suivant :



Ce graphe n'est pas une couverture minimale à cause des DF suivantes : N°veh → Marque et N°veh → puissance qui sont transitives.
On obtient une couverture minimale C du GDF en éliminant les deux DF précédentes.
En appliquant l'algorithme de décomposition sur C, on obtient le schéma de relations suivant : Véhicule (N°veh, Type, Couleur)
Type-Veh(Type, Marque, Puissance)

Chapitre 4

Algèbre relationnel

Objectifs:

- Expliquer la théorie de l'algébre relationnelle;
- Savoir comment établir des requêtes en algèbre relationnelle.

Dans ce cours: Vous allez voir:

- Opérateurs de l'algèbre relationnelle.

Chapitre 4

Algèbre relationnelle

4.1. Introduction

L'algèbre relationnelle est le support mathématique cohérent sur lequel repose le modèle relationnel. L'algèbre relationnelle propose un ensemble d'opérations élémentaires formelles sur les relations dans le but de créer de nouvelles relations. Ces opérations permettent de représenter des requêtes sur la base de données dont le résultat s'exprime sous la forme d'une relation (i.e. table). C'est ce formalisme qui est au coeur du langage de requête SQL.

Nous pouvons distinguer trois familles d'opérateurs relationnels :

Les opérateurs unaires (la sélection et la projection), qui sont les plus simples, permettent de produire une nouvelle table à partir d'une autre table.

Les opérateurs binaires ensemblistes (l'union, l'intersection et la différence) permettent de produire une nouvelle relation à partir de deux relations de même degré et de même domaine.

Les opérateurs binaires ou n-aires (le produit cartésien, la jointure et la division) permettent de produire une nouvelle table à partir de deux ou plusieurs autres tables.

La présentation des opérateurs est illustrée par deux exemples, l'un est la gestion d'une base d'invitations. Cette base de données décrit les personnes invitées et les plats qui ont été servis. Elle est composée de trois relations :

REPAS (<u>date, invité</u>) donne la liste des invités qui ont été reçus et à quelle date.
MENU (<u>date, plat</u>) donne le menu servi à chaque date.
PREFERENCE (<u>**personne**</u>, plat) donne pour chaque personne ses plats préférés.
N.B: les attributs "personne" et "invité" ont même domaine et les clés sont soulignées.
Remarque: Les notations de l'algèbre relationnelle ne sont pas standardisées.

4.2. Les opérateurs

4.2.1 Les opérateurs unaires:

4.2.1.1Affectation :

R (A1, ..., An) = expression de sélection. L'affectation permet de sauvegarder le résultat d'une expression de recherche, ou bien de renommer une relation et ses attributs.

4.2.1.2 Sélection (restriction):

R[condition-de-sélection]

Définition 01: La *sélection* génère une relation regroupant exclusivement toutes les occurrences de la relation R qui satisfont l'expression logique E. (autre notation : σ(E)R).

Il s'agit d'une opération unaire essentielle dont la signature est :

relation × expression logique relation

La sélection permet ainsi de choisir (i.e. sélectionner) certaines lignes dans une table. Le résultat de la sélection est donc une nouvelle relation qui a les mêmes attributs que R. Si R est vide (c'est- à-dire sans aucune occurrence), la relation qui résulte de la sélection est vide.

Exemple 1

Soit la relation PERSONNE suivante : relation

Ci-dessous un exemple de sélection sur la PERSONNE : σ(idPersonne≥5) PERSONNE

PERSONNE		
idPersonne	**nom**	**prénom**
5	Durand	Caroline
1	Germain	Stan
12	Dupont	Lisa
3	Germain	Rose-Marie

$\sigma_{(idPersonne\geq 5)}$ PERSONNE		
idPersonne	**nom**	**prénom**
5	Durand	Caroline
12	Dupont	Lisa

Exemple2 :

Donner la liste des repas à la date 01022017. Solution en algèbre (sélection et affectation):

Res = REPAS [date=01022017]

4.2.1.3 Projection R [A1, ..., An]

Définition 02 : La *projection* consiste à supprimer les attributs autres que A1, A2,..,An d'une relation et à éliminer les n-uplets en double apparaissant dans la nouvelle version.

Autre notation: **π (A1, A2,..,An)R**

Il s'agit d'une opération unaire essentielle dont la signature

est : relation × liste d'attributs relation

En d'autres termes, la projection permet de choisir des colonnes dans une table. Si R est vide, la relation qui résulte de la projection est vide, mais pas forcément équivalente étant donné qu'elle contient généralement moins d'attributs.

Exemple 1:

$\Pi_{(nom)}$ PERSONNE
nom
Durand
Germain
Dupont

Exemple 2: Donner la liste des invités à la date 01022017.Solution en algèbre (sélection, affectation et projection):

R1 = REPAS [date=01022017]

RESULTAT = R1 [invité]

4.2.2 Les Opérateurs binaires de même schéma:

Les trois opérateurs ensemblistes opèrent sur des relations R et S de même schéma SRS.

4.2.2.1 Union

Définition 03 : l'*union* est une opération portant sur deux relations R1 et R2 ayant le même schéma et construisant une troisième relation constituée des n-uplets appartenant à l'une ou l'autre des deux relations R1 et R2 sans doublon. **Notation : R1 ∪ R2**.

Il s'agit d'une opération binaire ensembliste commutative essentielle dont la signature est :

relation × relation relation

R1 et R2 doivent avoir les mêmes attributs et si une même occurrence existe dans R1 et R2, elle n'apparaît qu'une seule fois dans le résultat de l'union. Le résultat de l'union est une nouvelle relation qui a les mêmes attributs que R1 et R2. Si R1 et R2 sont vides, la relation qui résulte de l'union est vide. Si R1 (respectivement R2) est vide, la relation qui résulte de l'union est identique à R2 (respectivement R1).

Exemple 1:

Ci-dessous un exemple d'union:

R_1	
nom	**prénom**
Durand	Caroline
Germain	Stan
Dupont	Lisa
Germain	Rose-Marie

R_2	
nom	**prénom**
Dupont	Lisa
Juny	Carole
Fourt	Lisa

$R_1 \cup R_2$	
nom	**prénom**
Durand	Caroline
Germain	Stan
Dupont	Lisa
Germain	Rose-Marie
Juny	Carole
Fourt	Lisa

Exemple 2 : Quelles sont les personnes qui ont pour plats préférés 'frites' ou 'pâtes' ? Solution en algèbre :

R1 = PREFERENCE [plat='frites']

R2 = R1 [personne]

R3 = PREFERENCE [plat='pâtes']

R4 = R3 [personne]

RES = R2 U R4

Ou bien :

R1 = PREFERENCE [plat='frites' ou plat = 'pâtes']

RES = R1 [personne]

4.2.2.2 Intersection

Définition 04 : L'*intersection* est une opération portant sur deux relations R1 et R2 ayant le même schéma et construisant une troisième relation dont les n-uplets sont constitués de ceux appartenant aux deux relations. **Notation : R1 ∩ R2**.

Il s'agit d'une opération binaire ensembliste commutative dont la signature est :

relation × relation relation

R1 et R2 doivent avoir les mêmes attributs. Le résultat de l'intersection est une nouvelle relation qui a les mêmes attributs que R1 et R2. Si R1 ou R2 ou les deux sont vides, la relation qui résulte de l'intersection est vide.

Exemple 1:

Ci-dessous un exemple d'intersection :

R_1	
nom	**prénom**
Durand	Caroline
Germain	Stan
Dupont	Lisa
Germain	Rose-Marie
Juny	Carole

R_2	
nom	**prénom**
Dupont	Lisa
Juny	Carole
Fourt	Lisa
Durand	Caroline

$R_1 \cap R_2$	
nom	**prénom**
Durand	Caroline
Dupont	Lisa
Juny	Carole

Exemple 2 : Quelles sont les personnes qui ont pour plats préférés 'frites' et 'pâtes' ? Solution en algèbre :

R1 = PREFERENCE [plat='frites']

R2 = R1 [personne]

R3 = PREFERENCE [plat='pâtes']

R4 = R3 [personne]

RES = R2 ∩ R4

4.2.2.3 Différence

Définition 05: La différence est une opération portant sur deux relations R1 et R2 ayant le même schéma et construisant une troisième relation dont les n-uplets sont constitués de ceux ne se trouvant que dans la relation R1. **Notation : R1 - R2**.

Il s'agit d'une opération binaire ensembliste non commutative essentielle dont la signature est : relation × relation relation

R1 et R2 doivent avoir les mêmes attributs. Le résultat de la différence est une nouvelle relation qui a les mêmes attributs que R1 et R2. Si R1 est vide, la relation qui résulte de la différence est vide aussi. Si R2 est vide, la relation qui résulte de la différence est identique à R1.

Exemple 1:

Exemple de différence entre deux relations.

R_1	
nom	**prénom**
Durand	Caroline
Germain	Stan
Dupont	Lisa
Germain	Rose-Marie
Juny	Carole

R_2	
nom	**prénom**
Dupont	Lisa
Juny	Carole
Fourt	Lisa
Durand	Caroline

$R_1 - R_2$	
nom	**prénom**
Germain	Stan
Germain	Rose-Marie

Exemple 2: Quelles sont les personnes qui n'ont jamais été invitées ? **Solution en algèbre** relationnelle

R1 = PREFERENCE [personne]

R2 = REPAS [invité]

RESUL= R1 – R2

4.2.3 Les opérateurs binaires de schémas différents

4.2.3.1 Produit cartésien : R X S

Définition 06: Le *produit cartésien* est une opération portant sur deux relations R1 et R2 et qui construit une troisième relation regroupant exclusivement toutes les possibilités de combinaison des occurrences des relations R1 et R2. **Notation : R1** x **R2**.

Il s'agit d'une opération binaire commutative essentielle dont la signature est: relation × relation relation

Le résultat du produit cartésien est une nouvelle relation qui a tous les attributs de R1 et tous ceux de R2. Si R1 ou R2 ou les deux sont vides, la relation qui résulte du produit cartésien est vide. Le nombre d'occurrences de la relation qui résulte du produit cartésien est le nombre d'occurrences de R1 multiplié par le nombre d'occurrences de R2.

Exemple:

PERSONNE	
nom	**prénom**
Fourt	Lisa
Juny	Carole

CADEAU	
article	**prix**
livre	45
poupée	25
montre	87

PERSONNE × CADEAU			
nom	**prénom**	**article**	**prix**
Fourt	Lisa	livre	45
Fourt	Lisa	poupée	25
Fourt	Lisa	montre	87
Juny	Carole	livre	45
Juny	Carole	poupée	25
Juny	Carole	montre	87

4.2.3.2 Jointure : R[Q] S (Jointure, thêta-jointure, jointure naturelle)

4.2.3.3.1 Jointure

Définition 07: La jointure est une opération portant sur deux relations R1 et R2 qui construit une troisième relation regroupant exclusivement toutes les possibilités de combinaison des occurrences des relations R1 et R2 qui satisfont l'expression logique E. La jointure est notée aussi: **R1 ⋈E R2**.

Il s'agit d'une opération binaire commutative dont la signature est :

relation x relation x expression logique ⟶ relation

Si R1 ou R2 ou les deux sont vides, alors la relation qui résulte de la jointure est vide. En fait, la jointure n'est rien d'autre qu'un produit cartésien suivi d'une sélection :

$$R1 \bowtie_E R2 = \sigma_E (R1 \times R2)$$

Exemple 1:

Exemple de jointure :

PERSONNE		
nom	**prénom**	**Age**
Fourt	Lisa	6
Juny	Carole	42
Fidus	Laure	16

CADEAU		
âgeC	**article**	**prix**
99	livre	30
6	poupée	60
20	baladeur	45
10	déguisement	15

		PERSONNE $\bowtie_{(\text{âge} \le \text{âgeC}) \wedge (\text{prix} \le 50)}$ CADEAU			
nom	**prénom**	**âge**	**âgeC**	**article**	**prix**
Fourt	Lisa	6	99	livre	30
Fourt	Lisa	6	20	baladeur	45
Fourt	Lisa	6	10	déguisement	15
Juny	Carole	42	99	Livre	30
			99	Livre	30
			20	baladeur	45

Cette jointure permet de générer toutes les possibilités d'association entre un cadeau et une personne en respectant l'âge maximum conseillé pour un cadeau et la somme de 50€ à ne pas dépasser.

Fidus	Laure	16
Fidus	Laure	16

Exemple 2 :

Donner la liste des plats de tous les invités. Solution en algèbre :

R1 = REPAS [invité = personne] PREFERENCE

RES = R1 [plat]

4.2.3.2.2 Thêta-jointure

Définition 08 : La thêta-jointure est une jointure dans laquelle l'expression logique E est une simple comparaison entre un attribut A1 de la relation R1 et un attribut A2 de la relation R2.

4.2.3.2.3 Équi-jointure

Définition 09 : Une équi-jointure est une thêta-jointure dans laquelle l'expression logique E est un test d'égalité entre un attribut A1 de la relation R1 et un attribut A2 de la relation R2.

4.2.3.2.4 Jointure naturelle

Définition 10 : Une jointure naturelle est une jointure dans laquelle l'expression logique E est un test d'égalité entre les attributs qui portent le même nom dans les relations R1 et R2. Dans la relation construite, ces attributs ne sont pas dupliqués, mais fusionnés en une seule colonne par couple d'attributs. Si la jointure ne doit porter que sur un sous-ensemble des attributs communs à R1 et R2 il faut préciser explicitement ces attributs.

Généralement, R1 et R2 n'ont qu'un attribut en commun. Dans ce cas, une jointure naturelle est équivalente à une équi-jointure dans laquelle l'attribut de R1 et celui de R2 sont justement les deux attributs qui portent le même nom.

Exemple 1:

PERSONNE		
nom	**prénom**	**âge**
Fourt	Lisa	6
Juny	Carole	40
Fidus	Laure	20
Choupy	Emma	6

CADEAU		
âge	**article**	**prix**
40	livre	45
6	poupée	25
20	montre	87

Ci-dessous la jointure naturelle PERSONNE et CADEAU qui peut également s'écrire PERSONNE [âge] CADEAU.

PERSONNE ⋈ CADEAU				
nom	**prénom**	**âge**	**article**	**prix**
Fourt	Lisa	6	poupée	25
Juny	Carole	40	livre	45
Fidus	Laure	20	montre	87
Choupy	Emma	6	poupée	25

4.2.3.3 La division

Définition 11: la division est un opération portant sur deux relations R1 et R2, telles que le schéma de R2 est strictement inclus dans celui de R1, qui génère une troisième relation regroupant toutes les parties d'occurrences de la relation R1 qui, associées à toutes les occurrences de la relation R2, se retrouvent dans R1. **Notation : R1÷ R2**.

Il s'agit d'une opération binaire non commutative dont la signature est :

relation x relation ⟶ relation

Autrement dit, la division de R1 par R2 (R1÷ R2) génère une relation qui regroupe tous les n- uplets qui, concaténés à chacun des n-uplets de R2, donne toujours un n-uplet de R1.

La relation R2 ne peut pas être vide. Tous les attributs de R2 doivent être présents dans R1 et R2 doit posséder au moins un attribut de plus que R2 (inclusion stricte). Le résultat de la division est une nouvelle relation qui a tous les attributs de R1 sans aucun de ceux de R2. Si R1 est vide, la relation qui résulte de la division est vide.

Exemple 1:

Ci-dessous, un exemple de division ENSEIGNEMENT ÷ ETUDIANT qui permet de dresser la table R de tous les enseignants de la relation ENSEIGNEMENT qui enseignent à tous les étudiants de la relation ETUDIANT.

ENSEIGNEMENT	
enseignant	**nom**
Germain	Dubois
Fidus	Pascal
Robert	Dubois
Germain	Pascal
Fidus	Dubois
Germain	Durand
Robert	Durand

ETUDIANT
nom
Dubois
Pascal

ENSEIGNEMENT ÷ ETUDIANT
enseignant
Germain
Fidus

Exemple 2 Quels sont les invités qui sont venus à tous les repas?

Solution en algèbre :

R1 = REPAS [date]

RES = REPAS÷ R1

Remarque: la division est également notée : R/S R division S

4.2.3.4 Opérateurs de base et opérateurs dérivés

Cinq de ces huit opérateurs forment les opérateurs de base (ce sont l'union, la différence, le produit cartésien, la selection et la projection) tandis que les trois autres, appelés opérateurs dérivés, s'obtiennent plus ou moins facilement par combinaison des opérateurs de base:

L' intersection:

$R \cap S = R - (R - S) = S - (S - R) = (R \cup S) - ((R - S) \cup (S - R))$

La jointure:

$R \mid x \mid S (Ai, Bj) = \sigma ij (R \times S)$

La division:

Soit R(X,Y) et S(Y)

$R \div S = \pi [X] R - \pi [X] (((\pi[X]R) \times S) - R)$

Les cinq opérateurs de base permettent de répondre à toutes les questions que l'on peut poser avec la logique du premier ordre (c'est à dire sans les fonctions): on dit que l'algèbre relationnelle est complète.

Chapitre 5

Le Langage de manipulation relationnel SQL

Objectifs:

- Expliquer les différentes commandes pour interroger une base de données en SQL .
- Savoir comment établir des requêtes en SQL.

Dans ce cours: Vous allez voir:

- Introduction;
- Opérateurs de l'algèbre relationnelle.

Chapitre 5

Le Langage de manipulation relationnel SQL

5.1. Introduction

Les SGBD relationnels proposent un langage de requête appelé SQL (Stuctured Query Language, en français : Langage de requêtes structuré). Il peut être considéré comme le langage d'accès normalisé aux bases de données. Il est aujourd'hui supporté par la plupart des produits commerciaux que ce soit par les systèmes de gestion de bases de données micro tel que Access ou par les produits plus professionnels tels que Oracle ou Sybase.

Le succès du langage SQL est dû essentiellement à sa simplicité et au fait qu'il s'appuie sur le conceptuel pour énoncer des requêtes en laissant le SGBD responsable de la stratégie d'exécution. Le modèle relationnel a été inventé par E.F. Codd (Directeur de recherche du centre IBM de San José) en 1970, suite à quoi de nombreux langages ont fait leur apparition :

- IBM Sequel (Structured English Query Language) en 1977
- IBM Sequel/2
- IBM System/R IBM DB2

Ce sont ces langages qui ont donné naissance au standard SQL, normalisé en 1986 par l'ANSI pour donner SQL/86. Puis en 1989 la version SQL/89 a été approuvée. La norme SQL/92 a désormais pour nom SQL 2.

Le langage SQL comporte :

- Une partie sur la définition des données :

Le langage de définition des données (**LDD**) qui permet de définir des relations (créer des tables), des vues externes et des contraintes d'intégrité ;

- Une partie sur les requêtes :

Le langage de manipulation de données (**LMD**) qui permet d'interroger une base de données, d'insérer, de modifier ou de supprimer des données dans une table d'une base de données relationnelle, sans se préoccuper de l'organisation physique des données.

- une partie sur le contrôle des données :

Le langage de contrôle de données (**LCD**) qui permet de contrôler la sécurité et les accès aux
données.

- Typologie du langage :

Il est possible d'inclure des requêtes SQL dans un programme écrit dans un autre langage (en langage C par exemple), ainsi que d'envoyer directement les requêtes SQL telles quelles au SGBD.
Les exemples dans ce qui suit s'appuient sur la base de données relative aux Clients (CLIENT), produits (PRODUIT), commandes (COMMANDE) et détails (DETAIL), décrite par le schéma suivant :

CLIENT (num-cli, nom-cli, adr-cli)
PRODUIT (num-prod, des-prod, prix-un)
COMMANDE (num-com, num-cli, date)
DETAIL (num-com, num-prod, qte-com)

5.2. Création d'un schéma
Une BD est définie par son schéma. SQL propose donc de créer ce schéma avant de définir ses composants grâce à la commande : CREATE

Exemple : create schema BDCLIENT
Cette instruction sera accompagnée de divers paramètres.

5.2.1 Domaines

Il existe des domaines prédéfinis tels que :

- char(n) : chaîne de caractères de taille fixe n
- varchar(n) : chaîne de caractères de taille variable mais inférieure à n
- int : Entier (un sous ensemble fini des entiers, dépend de la machine)
- smallint : Entier. Sous ensemble de int
- numeric(p,d) : Réel codé sur p digits et max d digits pour partie à droite de la décimale.
- real : Un réel flottant.
- date : YYYY-MM-DD (année, mois, jours)
- time : HH :MM :SS (heure, minute, seconde)

On peut créer nos propres domaines avec la commande: CREATE DOMAIN

Exemple : CREATE DOMAIN nom-client char(20)

5.2.2 Création d'une table

On utilise la clause **CREATE TABLE**

Cette opération produit une table vide(c-à-dire sans ligne).

Exemple: **create table** CLIENT (num-cli int NOT NULL,
nom-cli varchar (15),
adr-cli char (50))

5.2.2.1 Expression de l'identifiant primaire

On complétera la déclaration de la table par la clause : primary key

Exemple : **create table** CLIENT (num-cli int NOT NULL ,
nom-cli varchar (15),
adr-cli char (50),
primary key (num-cli))

5.2.2.2 Expression d'une contrainte référentielle (clé étrangère)

On déclarera la clé étrangère et la table référencée par la clause : foreign key

Exemple: create table DETAIL (num-com int NOT NULL,
num-cli int NOT NULL,
qte-com int,
primary key (num-co, num-cli),
foreign key (num-com) references COMMANDE,
foreign key (num-cli) references CLIENT)

5.2.2.3 Rajout de contraintes

Exemple: create table DETAIL (num-com in NOT NULL,
num-cli int NOT NULL,
qte-com int,
primary key (num-co, num-cli),
foreign key (num-com) references COMMANDE,
foreign key (num-cli references CLIENT,
check (num-com> 1))

Remarque : En SQL92, si un attribut est clé alors il est différent de NULL.

5.2.3 Suppression d'une table

Toute table peut être supprimée grâce à la commande '**drop**'. Elle est désormais inconnue et son contenu est perdu.

Exemple: drop table COMMANDE

Remarque: On peut ajouter, retirer et modifier une colonne.

2.4 Les vues

Une vue peut être considérée comme une relation quelconque lors de l'expression des requêtes.
Une vue est une relation virtuelle dans le sens où elle ne contient pas effectivement des « tuples ».

Elles permettent de définir des relations virtuelles dans le but de :

- Cacher certaines informations à des utilisateurs,
- Faciliter l'expression de certaines requêtes,
- Améliorer la présentation de certaines données.

Une vue est définie par une expression de la forme :

CREATE VIEW V AS requête

Requête est une expression quelconque de requête et V est le nom de la vue.
Exemple : Soit Emp (NumE, Salaire, Dept, Adresse) une relation, on crée la vue suivante sur cette relation :

CREATE VIEW EmpGen **AS** (
SELECT NumE, Dept, Adresse
FROM Emp)

5.3. Le langage de manipulation de données (LMD)

Le langage SQL est un langage normalisé [ISO92]. Cependant la plupart des systèmes ont introduit des spécificités ou des extensions qui ne sont pas dans la norme. Dans ce qui suit, nous rappellerons les formes les plus courantes de requêtes SQL, en ne donnant qu'une vision générale de la structure des requêtes.

La forme d'une requête d'interrogation SQL est la suivante:

SELECT [* | **DISTINCT**] att1 [, att2, att3, ...]
FROM Table1 [, Table2, Table3, ...]
[**WHERE** conditions de sélection et/ou de jointure]
[**GROUP BY** att1 [, att2, ...] [**HAVING** conditions de selection]]
[(**UNION** | **INTERSECT** | **MINUS**[**ALL**])<commande SELECT>] [**ORDER BY** att1 [**ASC** | **DESC**] [, att2 [**ASC** | **DESC**], ...];

[ ] : optionnel
| : ou

Où la clause **select** décrit la relation résultat. L'expression d'attributs désigne soit une liste d'attributs, soit une expression fonctionnelle obtenue à l'aide des fonctions **sum** (somme), **avg** (moyenne), **count** (compte), **min**, **max**. Le mot clé **distinct** permet d'éliminer les doubles après une projection.

La clause **from** désigne les relations concernées par la requête, avec éventuellement une ou
plusieurs variables synonymes pour chaque relation désignée.

La clause **where**, optionnelle, spécifie les critères de sélection. La condition de sélection est une expression logique spécifiant les prédicats de restriction ou de jointure à satisfaire par la réponse. La sous-question peut être n'importe quelle question SQL, ce qui permet de générer
une série de questions imbriquées.

Le **group by** partitionne la relation selon les valeurs de la liste des attributs spécifiée.

La clause **having** exprime une condition sur le groupe d'enregistrements associé à chaque valeur du groupage.

La clause **order by** permet de trier les résultats obtenus.

Le principe général de l'évaluation d'une requête permet de comprendre la sémantique que SQL attribue aux requêtes. Ce principe est le suivant :

- Evaluation de la clause **from**

en faisant le produit cartésien de toutes les relations qui y apparaissent
• Evaluation de la clause **where**
qui réalise les restriction et les jointures
• Evaluation de la clause **group by** qui constitue les partitions
• Evaluation de la clause **having**
qui sélectionne les partitions désirées
• Evaluation de la clause **select**
qui constitue la projection finale
• Evaluation de la clause **order by**
qui trie les tuples du résultat final

Remarque: Seules les clauses select et from sont obligatoires. Dans le cas d'une requête ne contenant que ces clauses, le résultat est le produit cartésien des relations mentionnées dans la clause from.

Exemple : 1) Donner les noms des clients.

a) résultat avec doublons : SELECT nom-client
FROM CLIENT

b) résultat sans doublons :
SELECT DISTINCT (nom-cli)
FROM CLIENT

2) Donner la liste des clients.
SELECT *
FROM
CLIENT

3) Donner le numéro des clients de nom Mohammed.
SELECT num-cli
FROM CLIENT
WHERE nom-cli=’ Mohammed’

Cette forme générale de requête peut être étendue de plusieurs façons:

1) En introduisant explicitement certains opérateurs ensemblistes tels que l'union et l'intersection. On peut ainsi formuler plusieurs sous-requêtes dont les résultats sont reliés par ces opérateurs. La syntaxe simplifiée est la suivante:

(**select** .....
from .....
where.....)
[**union** | **intersect** | **except**]
(**select** .....
from .....

where.....);

où except représente l'opération de différence ensembliste notée également par minus.

Exemple: Donnez les numéros des produits qui ne sont pas commandés.

```
Select num-prod
From PRODUIT
Minus
Select num-prod
From DETAIL
```

2) En introduisant les notions de requête imbriquée et de comparaison ensembliste. La requête externe sélectionne un ensemble de tuples qui sont comparés avec l'ensemble detuples sélectionné par la requête imbriquée à l'aide des prédicats in, not in, all, any, exists, not exists.

On peut avoir une cascade de sous-requêtes imbriquées.

a) IN: Permet de tester la présence d'une valeur particulière dans un ensemble.

Exemple: Donner num-prod et des-prod des produits commandés.

```
SELECT num-prod, des-prod
FROM PRODUIT
WHERE num-prod IN SELECT num-prod
                  FROM DETAIL
```

b) NOT IN: Permet de tester l'absence d'une valeur particulière dans un ensemble.

Exemple: Donner num-prod et des-prod des produits non commandés.

```
SELECT num-prod, des-prod
FROM PRODUIT
WHERE num-prod NOT IN SELECT num-prod
                      FROM DETAIL
```

c) ALL: Compare chacune des valeurs de l'ensemble à une valeur particulière et retourne 'VRAI' si la comparaison est évaluée pour chacun des éléments. Les comparateurs sont : <, <=, >, >=, =, !=

Exemple1: Donner le produit de prix maximal.

```
SELECT num-prod
FROM PRODUIT
WHERE prix-un >= ALL SELECT prix-un
                     FROM PRODUIT
```

d) ANY: Compare chacune des valeurs de l'ensemble à une valeur particulière et retourne 'VRAI' si la comparaison est évaluée à 'VRAI' pour au moins un des éléments.

Exemple1: Donner les produits dont le prix n'est pas minimal.

```
SELECT num-prod
FROM PRODUIT
WHERE prix-un > ANY SELECT prix-un
                    FROM PRODUIT
```

Exemple2 : Donner les numéros des clients qui ont commandé des produits en quantité > 60

```
SELECT n°cli
FROM COMMANDE
WHERE n°com= ANY SELECT n°com
                 FROM DETAIL
                 WHERE qcom >60
```

e) EXISTS : Retourne 'VRAI'si la sous-interrogation (requête imbriquée) ramène au moins une ligne.

Exemple: Donner les numéros des clients qui ont commandé au moins un produit en quantité supérieure à 100.

```
SELECT n°cli
FROM COMMANDE
WHERE EXISTS (SELECT *
              FROM DETAIL
              WHERE n°com= COMMANDE.n°com
                           And qcom > 100)
```

f) NOT EXISTS: Teste si l'ensemble est vide.

Exemple: Donner les numéros de produit qui ne sont pas commandés.

```
SELECT n°pro
FROM PRODUIT
WHERE NOT EXISTS (SELECT *
                  FROM DETAIL
                  WHERE n°pro= PRODUIT.n°pro)
```

3) En introduisant les connecteurs AND et OR entre sous-requêtes. On peut réaliser des formules très complexes.

```
select X,.....
from R1.....
where R1.X in (select X.....
               from R2.....
               where..... )
               [and |or]
               R1. X in (select X.....
                         from R3.....
                         where..... );
```

Exemple : Donnez les noms des clients qui ont commandé des produits ayant pour désignation ‘amox’ et commandés en quantité > 100

```
Select nom-cli
From CLIENT
Where num-cli in select num-cli
            From COMMANDE
            Where num-com in (select num-com
                              From DETAIL
                              Where qte-com>100 and num_prod in select num-prod
                                                                From PRODUIT
                                                                Where des-prod=’amox’)
```

4) Opération JOINTURE (équijointure)

En SQL, il est possible d'enchaîner plusieurs jointures dans la même instruction SELECT.
En SQL de base:

```
SELECT *
FROM table1, table2, table3, ...
WHERE table1.attribut1=table2.attribut1 AND table2.attribut2=table3.attribut2 AND ...;
```

Exemple :

```
SELECT *
FROM Produit, Détail_Commande
WHERE Produit.CodePrd=Détail_Commande.CodePrd ;
```

ou en utilisant des alias pour les noms des tables :

```
SELECT *
FROM Produit A, Détail_Commande B
WHERE A.CodePrd=B.CodePrd ;
```

4) Nous avons dit plus haut que SQL permettait l'utilisation de certaines fonctions de calcul. Ces fonctions sont **sum, min, max, avg et count.** Elles peuvent apparaître aussi bien dans la clause select que dans la clause **Having**. Les formes les plus générales sont les suivantes:

```
Select B, [min | max | sum | avg | count] (A)
from R1,.....
where condition
group by R1.B;
```

où les A et B sont des listes d'attributs.

La sémantique de cette requête est la suivante: après l'exécution de la clause **where (qui peut ne pas exister)**, la relation R1 est d'abord triée et partitionnée selon le critère B, puis l'une des fonctions de calcul choisie dans la clause **select** est appliquée à chaque partition.

Le résultat de la requête est un ensemble de tuples de cardinal égal à celui de l'ensemble des partitions et dont les valeurs sont les résultats de la fonction pour chaque partition. Il faut noter que les attributs B sur lesquels se fait le partitionnement des relations doivent nécessairement apparaître dans la clause **select** pour une bonne interprétation externe des résultats des fonctions de calcul.

Exemple1: Donner la somme des quantités commandées de chaque produit.

Il faut partitionner l'ensemble des tuples de DETAIL en sous-ensembles (ou groupes) par numéro de produit, on obtient donc :

```
SELECT n°pro, SUM (qcom)
FROM DETAIL
GROUP BY n°pro
```

Cette instruction génère dans les SGBD les actions suivantes:

1) Classer les tuples de DETAIL, groupe par groupe (un groupe= ensemble des tuples ayant
même n°pro).

2) Pour chaque groupe, évaluer le résultat du SELECT (faire la somme des quantités dans ce
groupe.

Exemple 2: Donnez les numéros des produits de chaque client.

```
Select num-cli, num-pod
From COMMANDE, DETAIL
Where COMMANDE.num-com= DETAIL.num-com
GROUP BY num-cli
```

On peut aussi avoir la forme suivante où on ne veut retenir que les partitions dont la fonction de calcul choisie satisfait un certain critère:

```
select B,
from R1,.....
where P
group by B
having [min | max | sum | avg | count] (A) $ valeur;
```

où $ désigne n'importe quelle relation de comparaison logique.

La clause HAVING ne s'emploie qu'avec un 'GROUP BY'. Elle exprime une condition sur le groupe d'enregistrements associé à chaque valeur du groupage.

Exemple: Donner la somme des quantités commandées des produits qui ont été commandés au moins deux fois.

```
SELECT n°pro, SUM (qcom)
```

```
FROM DETAIL
GROUP BY n°pro
HAVING COUNT (*) >= 2
```

La clause 'HAVING < condition >' permet de sélectionner les groupes qui satisfont une condition. Cette condition porte sur l'ensemble des tuples, (contrairement à la clause WHERE <condition > qui porte sur un tuple).

6) On peut ordonner les tuples du résultat en utilisant : ORDER BY.

Exemple: Donner la liste des produits par ordre décroisant des prix.

```
SELECT *
FROM PRODUIT
ORDER BY prix-un/desc
```

7) On peut faire de la recherche avec des conditions sur les ensembles

a) **CONTAINS** : Test d'inclusion

< ens1> CONTAINS < ens2> signifie que l'ensemble 1 contient (ou est égal) à l'ensemble 2

Exemple1: Quels sont les clients qui ont commandés tous les produits?

```
SELECT n°cli
FROM COMMANDE
WHERE (SELECT n°pro
        FROM DETAIL
        WHERE n°com= COMMANDE.n°com)

        CONTAINS

        (SELECT n°pro
        FROM PRODUIT)
```

Exemple2: Donner les numéros des commandes où figurent tous les produits.

```
SELECT num-com
FROM DETAIL
GROUP BY num-com HAVING SET (num-prod)
                        CONTAINS
                        SELECT num_prod
                        FROM PRODUIT
```

b) NOT CONTAINS : Test de non inclusion
(Négation de la précédente) c-à-dire elle est vraie si un élément de l'ensemble 2 n'appartient pas à l'ensemble 1.

3.1 Fonction d'agrégation

Agrégat : fonction définie en SQL qui permet de calculer un résultat atomique (un entier ou un réel) à partir d'un ensemble de valeurs. Les agrégats sont au nombre de 5 : COUNT, SUM, MIN, MAX, AVG. Ces fonctions usuelles ne peuvent être utilisées que dans une clause SELECT ou dans une clause HAVING.

a) **COUNT(expr) :** dénombre les lignes sélectionnées.
Exemple :

```
SELECT COUNT(num_prod)
FROM PRODUIT
```

b) **SUM(expr):** additionne les valeurs de type numérique.
c) **MIN (expr):** retourne la valeur minimale d'une colonne de type caractère ou numérique.
d) **MAX(expr)**: retourne la valeur maximale d'une colonne de type caractère ou numérique.
e) **AVG (expr)** : calcule la moyenne d'une colonne de numérique.

Remarques : 1) On peut préfixer expr par les mots clés

2) Ces fonctions peuvent apparaître dans la clause SELECT ou dans la clause HAVING.

Autres prédicats:

est compris entre … et …	**between … and**	…**where** numéro **between** 12 **and** 20	Les bornes de l'intervalle sont généralement incluses.
est conforme à	**Like**	…**where** nom **like** "D%"	**like** permet de vérifier qu'une chaîne de caractères est conforme à un modèle contenant _ et/ou %.
n'est pas conforme à	**not like**	…**where** nom **not like** "l%e"	**not like** est l'inverse de **like**.
est vide	**is null**	…**where** salaire **is null**	**is null** teste l'absence de valeur qui peut être assimilée à une valeur inconnue.
n'est pas vide	**is not null**	…**where** nom **is not null**	**is not null** est l'inverse de **is null**.

5.4. Lien entre algèbre relationnelle et SQL

Soit le schéma de base de données relationnel suivant :

R1 (A, B) R2 (C, D) R3 (A, E) R4 (B)

5.5. Mise à jour des relations

Opération	Expression algébrique	Expression SQL équivalente
Projection	R1[A]	SELECT A FROM R1
Sélection	R1[condition]	SELECT * FROM R1 WHERE <condition>
Produit cartésien	R1 X R2	SELECT * FROM R1, R2
Jointure	R1[A]R3	SELECT * FROM R1,R3 WHERE R1.A= R3.A
Union	R1U R2	SELECT* FROM R1 UNION SELECT * FROM R2
Intersection	R1∩ R2	SELECT * FROM R1 INTERSECT SELECT * FROM R2
Différence	R1- R2	SELECT * FROM R1 MINUS SELECT* FROM R2 OU BIEN SELECT * FROM R1 WHERE not exists (SELECT * FROM R2 WHERE R2.C = R1.A and R2.D = R1.B)

1) Insérer des tuples : INSERT INTO

- insérer un tuple : INSERT INTO nom de relation Values <tuple constant>
- insérer un ensemble de tuples : INSERT INTO nom de relation :
< requête SELECT dont le résultat a même schéma que la relation nom relation>

Exemple1: insérer le tuple (P10, desp, 100) dans PRODUIT.
INSERT INTO PRODUIT (num-prod , desig, prix-u) Values ('P10',' desp', 100)

Exemple2: rajouter le catalogue de schéma : CATALOGUE (np, nomp, pu)

INSERT INTO PRODUIT :
SELECT np, nomp, pu
FROM CATALOGUE

2) Supprimer des tuples : **DELETE**

DELETE nom relation [WHERE condition]
- Si la clause 'WHERE condition' est présente, seuls les tuples satisfaisant la condition sont
supprimés.
- Si la clause 'WHERE condition' est absente, alors tous les tuples sont supprimés ; la relation continue d'exister, mais sa population est vide.

Exemple : supprimer les produits de désignation :amox
DELETE PRODUIT
WHERE desg='amox'

3) Mettre à jour un (des) attribut(s) : UPDATE

UPDATE nom relation

SET nom attribut1=<expression1 qui définit une valeur pour l'attribut1>
.....
SET nom attributn=<expression n qui définit une valeur pour l'attribut n>
[WHERE condition]

Exemple: augmenter le prix des produits dont le prix est inférieur à 100 de 20.

UPDATE PRODUIT
SET prix-un= prix-un +20
WHERE prix-un <100

Remarque : La condition qui suit la clause WHERE peut être une requête SQL.

Bibliographie:

[1] Laurent Audibert. Base de données et langage sql. Institut Universitaire de Technologie de Villetaneuse, département informatique, 1re année, 2007.

[2] P. CRESCENZO R. GRIN Ph. LAHIRE Université Nice – Sophia Anti- polis. Sujet de td 01 bases de donnÉes. 2007.

[3] P. CRESCENZO R. GRIN Ph. LAHIRE Université Nice – Sophia Anti- polis. Sujet de td 02 bases de donnÉes : Formes normales et couverture minimale. 2007.

[4] Université Nancy 2 : Luiz Angelo Steffenel. Sgbd : Dépendance fonctionnel et normalisation d'un schéma relationnel. 2006.

[5] Licence Professionnelle CE-Stat Pierre Parrend. Base de données avan- cées b : Td3 - dépendances fonctionnelles et normalisation - correction.

[6] Licence MI2E 2007-2008 Université Paris DAUPHINE, Maude Manouvrier. Bases de données : exercices corrigés. 2007.

[7] TALEL ABDESSALEM École Nationale Supérieure des télécommunica- tions. Travaux diriges de bases de donnÉes. Septembre 2011.

[8] Licence 3 Informatique Université de Strasbourg. Bases de données 2 ,td 02 : Modèle relationnel 1 (dépendances fonctionnelles). 2011- semestre d'automne.

[9] Institut Galilée Université Paris 13. Examen base de données. 2010-2011.

[10] Sarah Cohen-Boulakia Polytech Paris Sud. Comprendre sql grâce à l'algèbre relationnelle. 2010-2011.

[11] DESS ISC Université Bordeaux I. Bases de données : Travaux dirigés. 2002-2003.

[12] UTC Formation. Interrogation de bases de données sql. 2018.