ECOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE

Fouad DAHAK

Bases De Données

Le Support de Cours

Enseignant-Chercheur

http://dahak.esi.dz

f_dahak@esi.dz

Version 1.0

ESI, Mars 2017

Ecole Supérieure d'Informatique

Préambule

Après onze années en tant que chargé de cours du module « Bases De Données » à l'Ecole Supérieure d'Informatique, je me rappelle toujours du coup de fil de Mr Medjaoui (Directeur des études de l'époque) me demandant d'assurer ce cours pour les étudiants de 3ème année option « Systèmes d'Information ». A peine recruté, je me retrouve comme chargé de cours devant 150 étudiants, tous assoiffés de savoir et qui en demandaient toujours d'avantage.

N'ayant pas une grande expérience en pédagogie, ce fut uniquement mon expérience professionnelle en tant que développeur et administrateur de bases de données qui a poussé le DE à me désigner pour cette tâche. J'ai commencé alors à chercher des supports de cours pour m'imprégner du module et préparer mes leçons, mais, hélas, je n'ai pas trouvé ne serait-ce qu'un brouillon. Je me suis alors réfugié à la bibliothèque de l'école où j'ai pratiquement dévoré tous les livres qui parlaient du domaine. J'en ai lu ainsi une dizaine et je me suis rendu compte qu'enseigner une notion comme les bases de données n'avait absolument rien à voir avec la pratique et le milieu professionnel. On peut être le meilleur DBA au monde mais cela ne signifie pas qu'on puisse transmettre les concepts théoriques aux autres.

Ça n'a pas été facile de construire un cours de bout en bout. Il fallait préparer les diapos, les séries d'exercices, les travaux pratiques et les examens. Mais ce qui était le plus difficile, c'était la pédagogie et surtout la didactique. Deux choses qu'on ne nous apprend pas à l'école et que chaque enseignant doit trouver par lui-même et construire sa propre façon de faire. J'ai beaucoup appris, dans ce sens, de mes ainés qui n'ont pas hésité à partager leurs différentes expériences. Aujourd'hui je leur suis très reconnaissant.

Devant cette indisponibilité d'un cours qui devait normalement être à la portée de tous, je me suis fait une promesse : celle de construire un support pédagogique et de faire en sorte qu'il soit le plus complet possible. Un tel support serait bénéfique à la fois aux étudiants et aux enseignants. J'ai alors fait des diapos pour chaque chapitre, un support de cours au format html et PDF que j'ai diffusé sur mon site personnel et des vidéos illustratives que j'ai mis sur YouTube.

J'ai également développé des outils pédagogiques pour aider les étudiants dans la compréhension des différents concepts du module avec des études de cas assez riches. Ce modeste support pédagogique, représente l'aboutissement de ces dix dernières années d'enseignement du module.

En lisant le titre, certains diraient que des supports pareils, il en existe des milliers. Cependant, ce qui est particulier avec ce présent support c'est la complétude de la partie concernant la conception des bases de données avec l'Entité Association. J'ai pratiquement fait le tour des livres et des supports de cours de bases de données et je peux certifier qu'aucun ne traite cette partie comme je le fais dans ce support. Car, en plus des concepts théoriques connu, j'y apporte une touche particulière en expliquant dans les détails les différents concepts et en y intégrant des exemples rapprochant la réalité de la théorie.

Pour finir, je pense avoir synthétisé ma modeste expérience dans ces quelques pages que je présente à nos étudiants, à nos enseignants et à toute personne désireuse d'apprendre le domaine des bases de données qui est des plus intéressants.

Fouad DAHAK

Table des matières

Liste des Figures

Présentation du cours

Ce cours s'adresse aux étudiants de première année cycle supérieur de la formation d'ingénieur en informatique de l'Ecole nationale Supérieure d'Informatique (ESI) d'Alger.

Les bases de données étant un concept très utilisé de nos jours est indispensable au développement de tout système informatique, il est ainsi nécessaire de maitriser les concepts relatifs à ce domaine afin de pouvoir concevoir correctement des bases de données optimales et être capable de les réaliser et exploiter leurs données.

Le cours de bases de données traite les trois paliers de modélisation des données, à savoir : le niveau conceptuel, le niveau logique et le niveau physique. Des notions théoriques très solides sont enseignées ainsi qu'un volet considérable d'exemples et de travaux pratiques. Au niveau conceptuel, on utilise les notions du modèle entité/association pour modéliser une situation réelle. Ce modèle est traduit vers le modèle relationnel au niveau logique, dans lequel les techniques de normalisation des données sont appliquées. Le schéma relationnel est manipulé avec l'algèbre relationnelle. En fin le modèle est traduit vers un script SQL de création d'une base de données.

A la fin de ce cours, l'étudiant sera en mesure de :

1. *Concevoir une base de données partant d'une réalité perçue avec le modèle entité/association,*
2. *Traduire un modèle entité/association vers un schéma relationnel, le normaliser et le manipuler avec l'algèbre relationnelle.*

3. *Créer la base de données correspondante au schéma relationnel en construisant un script SQL, manipuler la structure de la base avec le DDL et manipuler les données avec le DML.*

Afin de comprendre les concepts présentés dans ce présent cours, un certain nombre de prérequis sont requis :

1. *Bases de la programmation (Structure de données : Structures et fichiers),*
2. *Théorie des ensembles,*
3. *Logique du premier ordre.*

Ce support de cours est organisé comme suit : le premier chapitre est consacré à la présentation et la définition d'une base de données. Il permet ainsi de définir ce que c'est une donnée, une information et une base de données.

Le deuxième chapitre aborde le niveau conceptuel en présentant les éléments du modèle Entité/Association ainsi que la démarche de conception d'une base de données.

Le chapitre 3 est consacré au modèle relationnel. Les fondements du modèle sont ainsi présentés ainsi que les dépendances fonctionnelles et la théorie de la normalisation.

Au niveau du chapitre 4, l'algèbre relationnelle est traitée en présentant en détail les différentes opérations algébriques ainsi que les techniques d'optimisation algébrique des requêtes.

Finalement, le chapitre 5 aborde le langage SQL en présentant en détail le DDL et le DML avec des exercices et des exemples illustrés sous le SGBD MySQL et en respectant la norme 92.

A la fin de ce support de cours, deux annexes sont jointes : La première résume les fonctions utilisées dans la norme SQL et la seconde présente les deux outils pédagogiques (FDE et FRAI) développés par Fouad DAHAK et utilisés à l'ESI dans les TD/TP de BDD.

Chapitre 1 : Introduction Aux Bases De Données

Objectifs

L'objectif principal de ce chapitre est l'introduction du concept de base de données à travers son utilité. Il sert également à définir les concepts de base tels que la donnée et l'information ainsi que la différence entre les deux.

A la fin de chapitre l'étudiant sera en mesure de comprendre :

1. *La signification d'une base de données*
2. *L'utilité d'une base de données*
3. *Les problèmes que résout une base de données*

Résumé

Ce chapitre introduit le concept de bases de données et présente son utilité et les problèmes que peut résoudre son utilisation. Nous définissons ainsi les systèmes de gestion des bases de données et donnons un aperçu des méthodes de conception de ces dernières.

1. Introduction et historique

De nos jours, tout le monde utilise une base de données. Que ce soit directement ou indirectement, nous sommes tous confrontés à l'utilisation d'une base de données. Quand on va à la mairie pour demander notre extrait de naissance par exemple, on accède à une base de données. Quand on consulte notre mail, quand on consulte nos notes ainsi de suite. Les base de données existent partout et c'est le moyen le plus utilisé pour le stockage de données numériques.

Les premiers systèmes de gestion des bases de données (SGBD) sont apparus dans les années 1960. Avant cette date les données étaient stockées dans des fichiers mais on se rendit compte que cela nécessitait beaucoup de traitement et d'effort de développement et peu d'efficacité. L'idée principale fut alors d'introduire, entre le système d'exploitation et les applications, une couche de logiciel spécialisée dans la gestion de données structurées. L'un des premiers SGBD fut le système IMS basé sur un modèle hiérarchique. Parallèlement, le Database Task Group (fondé par Charles Bachman) définit la norme CODASYL, qui s'appuie sur une structuration des informations en réseaux. Les modèles hiérarchie et réseau nécessitaient une connaissance approfondie de la structuration physique des données pour leur exploration.

En 1970, Ted Codd (A. M. Turing Award 1981), proposa le modèle relationnel pour la représentation des données en se basant sur la théorie des ensembles. Les données sont ainsi organisées en plusieurs tables ou relations homogènes qui peuvent être interrogées et combinées grâce à des opérateurs ensemblistes. La simplicité de ce modèle lui a valu un succès incomparable. La preuve est que pratiquement tous les SGBD actuels se basent sur ce modèle.

Dans les années 1990, un nouveau type de modèles apparait, il s'agit des modèles semi-structurés. Ces derniers sont mieux adaptés à la gestion et à l'intégration de documents hétérogènes tels qu'ils sont publiés sur le Web. Le standard XML est un représentant de ce type de modèles.

Les bases de données ont donc toujours existé dans les systèmes informatiques. Les modèles ont évolué mais la finalité reste la même : stocker les données pour permettre une exploitation optimale et efficace.

2. Donnée et Information

Avant de parler de bases de données, il est important de définir ce que c'est qu'une donnée et de présenter ces caractéristiques. La plupart des gens confondent entre le concept de donnée et d'information. Or une donnée n'est pas une information.

Définition
C'est un ensemble de faits objectifs concernant un objet ou un événement. Elles n'ont d'utilité que quand elles sont converties en informations. Elles sont obtenues à partir d'une observation, une expérimentation ou un calcul (Déduction).

Une donnée est donc la plus petite unité composant l'information et ne porte aucune information prise séparément de son contexte. Une combinaison de données dans un contexte bien définit forme une information.

La donnée est obtenue de la réalité avec le processus d'observation ou bien produite grâce à un calcul ou une déduction. En observant par exemple, la température de la pièce on capte une donnée qui est le degré de température (25°). Cette donnée est élémentaire et n'a aucun sens quand elle est prise toute seule.

Par contre, si je la mets dans un contexte elle prend tout son sens et devient ainsi une information. Par exemple : Le temps et bon, le thermomètre indique 25°.

Si la donnée n'est pas l'information, alors c'est quoi au juste une information ?

Définition
Tout peut être information, mais c'est uniquement le regard porté sur un objet qui le rend porteur d'information.

La définition de l'information est beaucoup moins formelle que celle de la donnée. Tout simplement, parce que son interprétation est subjective et dépend de la personne qui la manipule.

La même information peut être perçue différemment par deux personnes différentes. Comme elle peut perdre son importance et devient sans effet avec le temps. Chose qui ne s'applique à une donnée qui reste tout le temps valable car elle représente un constat d'une situation donnée.

C'est pour toutes ces raisons qu'il existe des bases de données et non pas des bases d'informations. Connaissant ce que c'est qu'une donnée, peut-on maintenant définir ce qu'est une base de données ?

3. Base de données

Il est difficile de donner une définition exacte de la notion de base de données. On retrouve plusieurs définitions dans la littérature mais toutes se rejoignent quand il s'agit de l'utilité d'une base de données.

Définition

Une base de données est un ensemble structuré de données modélisant les objets d'une partie du monde réel, enregistrées dans un ordinateur et accessibles de façon sélective par plusieurs utilisateurs.

Plus précisément, on appelle base de données un ensemble structuré et organisé permettant le stockage de grandes quantités d'informations afin d'en faciliter l'exploitation. Une base de données permet ainsi l'ajout, la mise à jour et la recherche de données.

Une base de données permet de mettre des données à la disposition de plusieurs utilisateurs pour une consultation, une saisie ou bien une mise à jour, tout en s'assurant des droits accordés à ces derniers.

On peut également définir une base de données en fonction de son utilité, i.e. quand utilise-t-on une base de données ?

Les critères essentiels qui nous poussent à utiliser une base de données sont les suivants :

1. La taille des données : la masse importante des données manipulées nous oblige à les organiser et les structurer afin de les exploiter plus efficacement et plus rapidement.
2. La sécurité des données : l'utilisation d'un simple fichier pour sauvegarder des données ne garantit pas la sécurité de ces dernières. Une base de données quant à elle permet une sécurité des accès et des données.
3. L'intégrité : quand plusieurs utilisateurs accèdent à la même donnée, il est important de garantir son intégrité et d'assurer que la donnée n'est pas altérée.
4. La recherche : permettre des recherches multicritères sur les données afin d'accéder uniquement à l'information désirée dans les meilleurs délais.
5. Multiutilisateurs : permettre à plusieurs personnes d'accéder à la même donnée tout en garantissant la sécurité et l'intégrité de cette dernière.

4. Système de 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) ou en anglais DBMS (Database management system). Le SGBD est un ensemble de services (applications logicielles) permettant de gérer les bases de données. Il permet ainsi l'accès aux données de façon simple, un accès aux informations à de multiples utilisateurs et la manipulation des données présentes dans la base de données (insertion, suppression, modification).

Définition

Un Système de Gestion de Base de Données (SGBD) est un ensemble de logiciels systèmes permettant aux utilisateurs d'insérer, de modifier, et de rechercher efficacement des données spécifiques dans une grande masse d'informations (pouvant atteindre plusieurs milliards d'octets) partagée par de multiples utilisateurs. Un SGBD est caractérisé par un modèle de description des données.

Le SGBD se décompose généralement en trois sous-systèmes :

1. Le système de gestion de fichiers qui permet le stockage des informations sur un support physique,
2. Le SGBD interne qui gère l'ordonnancement des informations,
3. Le SGBD externe qui représente l'interface avec l'utilisateur.

Les objectifs d'un SGBD sont les suivants :

1. Indépendance physique : la façon dont les données sont définies doit être indépendante des structures de stockage utilisées
2. Indépendance logique : un même ensemble de données peut être vu différemment par des utilisateurs différents. Toutes ces visions personnelles des données doivent être intégrées dans une vision globale.
3. Non redondance d'information : afin d'éviter les problèmes lors des mises à jour, chaque donnée ne doit être présente qu'une seule fois dans la base
4. Partage des données : interrogations et modifications « en même temps » dans un contexte multi-utilisateur

5. Méthodes et Modèles

La création d'une base de données passe généralement par trois étapes essentielles : la modélisation conceptuelle, la modélisation logique et la réalisation physique de la base. Un modèle spécifique est utilisé à chaque niveau et un passage est assuré entre les différents niveaux. Nous déroulons tout au long de ce cours les différents modèles utilisées à chaque niveau.

Chapitre 2 : Conception Des Bases De Données Avec Le Modèle Entité/Association

Objectifs

L'objectif principal de ce chapitre est d'apprendre à l'étudiant comment modéliser une situation réelle en utilisant le modèle entité association. La source d'information est généralement des descriptifs textuels des cas à étudier. On apprend à comprendre ces textes et à déceler les différents objets formant le système ainsi que les interactions entre ces derniers.

A la fin de ce chapitre l'étudiant maîtrisera les différents concepts du modèle, acquerra une démarche de modélisation et sera en mesure de traduire une réalité perçue en un modèle entité association.

Résumé

Tout au long de ce chapitre, nous introduisons les différents concepts de conception des bases de données en nous appuyant sur les principes du modèle entité association de Chen.

Nous présentons en premier lieu les concepts de bases qui représentent les fondements du modèle EA et nous nous approfondissons par la suite en présentant les concepts supplémentaires et la modélisation des contraintes.

1. Introduction

Une base de données représente une image de la réalité perçue. Elle organise les données manipulées dans le domaine étudié afin de permettre leur exploitation et leur manipulation.

Avant d'aboutir à une base de données sous sa forme finale (i.e. implémentée dans le SGBD), il faut passer par une étape de conception et de modélisation. Durant laquelle on représente les objets de la réalité et les interactions entre eux de manière à ce qu'ils soient facilement manipulables.

Lors de la phase de conception, on utilise des modèles sémantiques dont l'entité-association de Chen constitue l'un des meilleurs et des plus courants.

Nous présentons dans ce chapitres les éléments du modèle entité association avec des techniques de modélisation qui vous permettront de modéliser correctement les données d'une situation réelle.

1.1. La modélisation

La réalité est souvent complexe et l'image que nous percevons d'elle n'est pas toujours fidèle et ne permet pas de cerner tous les détails. Lors de la modélisation d'une base de données, on ne peut passer directement à la base proprement dite à partir de la réalité sans passer par des représentations intermédiaires.

Définition
Un modèle est une abstraction de la réalité sur laquelle on peut opérer.

Le modèle permet de simplifier la réalité en enlevant tout ce qui n'est pas utile. Il nous fournit des moyens de représentation et de manipulation des concepts qu'on veut analyser.

Il existe plusieurs types de modèles, dans le domaine des bases de données, il en existe également plusieurs selon leur niveau d'abstraction. Le modèle de données permet de représenter les données et les liens entre elles afin de nous permettre de les manipuler facilement. L'entité association est un modèle conceptuel, il permet de modéliser le niveau sémantique des données pour permettre leur manipulation et leur transformation par la suite en une base de données.

1.2. Présentation du modèle EA

Le modèle Entité Association (EA), présenté par le Prof. Peter Pin-Shan Chen (1976), permet une description naturelle du monde réel à partir des concepts d'entité et d'association.

Figure 1 : Pr. Peter Pin-Shan Chen

Basé sur la théorie des ensembles et des relations, ce modèle se veut universel et répond à l'objectif d'indépendance données-programmes. Il est utilisé dans la phase de conception et s'inscrit notamment dans le cadre d'une méthode plus générale et très connue (MERISE).

L'idée fondamentale du modèle EA est de retenir comme concepts de base pour la représentation les mêmes concepts génériques que ceux qui guident le processus d'abstraction conduisant de l'observation d'une réalité à sa description. On suppose que la perception d'une situation observée se fait naturellement sur la base d'une identification des objets présents (qu'ils soient réels, ou abstraits), de liens entre ces objets (une personne conduit une voiture) et de propriétés observables (taille, couleur ...)

2. Concepts de base

Le modèle EA propose une représentation de chacun des trois concepts de la réalité et fait ainsi correspondre l'entité à l'objet, l'association au lien et l'attribut à la propriété. Ces trois concepts forment le modèle EA de base qui permet la modélisation des données d'une situation réelle.

2.1. Entité

Le Pr Chen est parti du principe que le monde nous entourant est composé d'objets qui interagissent entre eux. Les objets sont caractérisés pas des propriétés qui mises ensemble peuvent former une représentation de l'objet en question. Dans l'article de base de Chen, les trois concepts de base du modèle sont ainsi définis.

2.1.1. Entité

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

Dès qu'on rencontre des objets identifiables de manière unique (sans confusion ni ambiguïté) qui interagissent et qui font ou subissent des actions dans notre système on parle d'entités.

Prenons l'exemple suivant :

Ali lit le livre intitulé « Bases de données » qu'il a acheté chez Amar.

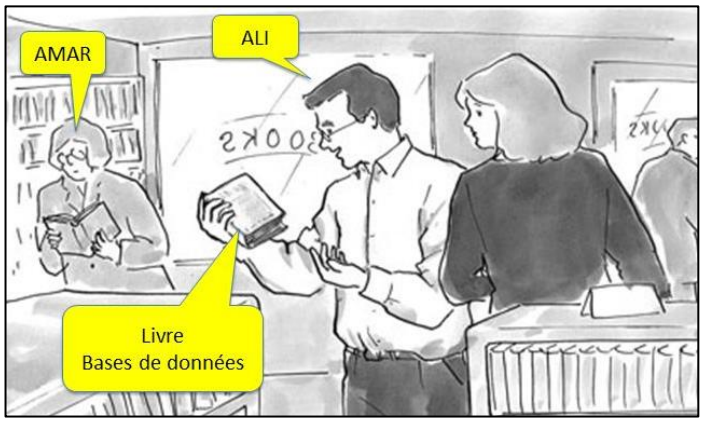


Figure 2 : Exemple d'une situation réelle

On distingue nettement trois entités : Ali, le livre « Bases de données » et Amar. Ali fait deux actions, celles de lire et d'acheter. Le livre « Bases de données » subit ces mêmes actions alors que Amar participe à l'action d'achat du livre par Ali qui est interprétée de son coté comme étant une action de vente.

Les entités concrètes sont des entités que l'on peut voir et toucher, leur présence est physique. Alors que les entités abstraites sont présentes dans notre esprit, on peut les ressentir et connaitre leur rôle dans une situation donnée mais n'ont aucune forme physique.

Quand on parle par exemple de la situation d'un enseignant qui enseigne un cours à des étudiants. Les enseignants et les étudiants sont des objets concrets et le cours est un objet abstrait.

2.1.2. Type - Entité

Lors de la modélisation d'une situation donnée, on ne peut gérer toutes les entités présentes ni les représenter dans le modèle. On est donc amené à passer à un niveau d'abstraction plus élevé. On regroupe ainsi toutes les entités de même nature afin d'en avoir

une seule représentation générale ou générique. C'est analogue au concept d'ensemble dans les mathématiques en considérant les entités comme les éléments de l'ensemble.

Cette représentation généralisée des entités est appelée type-entité et définie comme suit :

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

Dans l'exemple précédent, on a présenté une situation bien définie, celle de Ali qui lit le livre « Bases de données » qu'il a acheté chez Amar. Mais si on désire modéliser cette situation de manière générale, On fera en sorte que notre modèle puisse représenter cette situation quel que soit le lecteur, l'acheteur, le livre et le vendeur. Pour cela on ne devrait plus parler de Ali, du livre « Bases de données » et de Amar précisément mais de ce qu'ils représentent, à savoir : l'acheteur, le livre et le vendeur. Donc on pourrait dire finalement que Ali et Amar sont des entités du type-entité ***Personne*** car tous deux ont les mêmes caractéristiques et le livre « Bases de données » est une entité du type-entité ***Livre***.

Au niveau du modèle EA, ce sont les types-entités qui sont représentés et non pas les entités. Une entité est souvent nommée occurrence ou instance de son type-entité.

Par abus de langage, on utilise souvent le mot entité au lieu du mot type-entité, il faut cependant prendre garde à ne pas confondre les deux concepts.

2.2. Attribut et Valeur

2.2.1. Attribut

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

Dans l'exemple précédent, Ali et Amar sont des prénoms des deux entités qu'ils désignent et « Bases de données » est le titre du livre acheté par Ali. En fait « Ali » est une valeur de la propriété prénom, c'est donc le prénom qui est un attribut et non pas Ali, idem pour le titre du livre.

On peut dire finalement, qu'une entité est une réalisation ou concrétisation des propriétés d'un type-entité ce qui signifie qu'on donne une valeur à chaque attribut du type-entité pour obtenir une entité de ce type-entité.

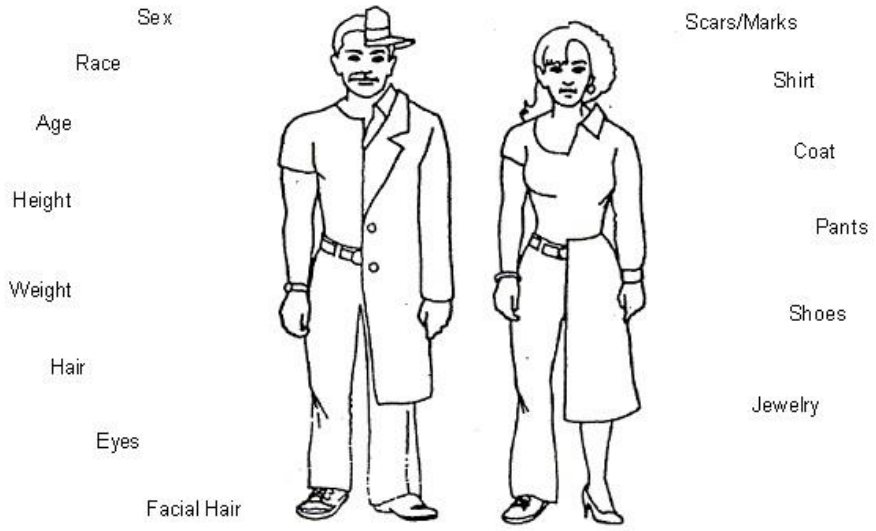


Figure 3 : Les attributs d'une personne

REGLE

Un attribut ne peut en aucun cas être partagé par plusieurs type-entités ou type-associations.

Dans certaines situations on trouve des entités différentes qui ont des propriétés portant la même appellation. Par exemple dans une situation d'enseignement, l'étudiant et l'enseignant ont tous les deux une propriété Nom. Mais au niveau du modèle EA, ce ne sont pas le même attribut car chaque attribut est défini dans le cadre de son propriétaire.

Ainsi, l'attribut Nom pris séparément ne signifie absolument rien. C'est quand il décrit une entité qu'il devient porteur de sens et ce dernier est étroitement lié à l'entité qu'il décrit. Le nom d'un enseignant n'est donc pas le même que celui d'un étudiant, même si les deux sont définis dans le même domaine de valeurs.

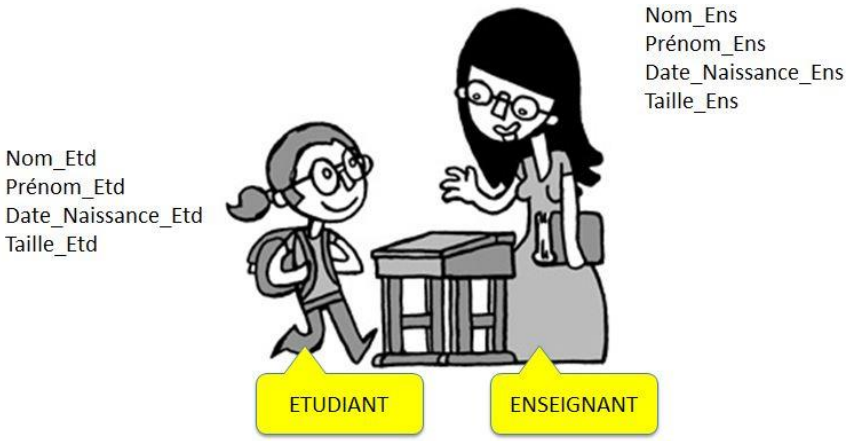


Figure 4 : Attributs partagés

Il faut faire attention à ce niveau, on parle de l'attribut et non pas des valeurs qu'il peut prendre.

Il se peut que l'étudiant s'appelle *Mohammed* et l'enseignant également, mais *Mohammed* est la valeur de l'attribut *Nom* et non pas l'attribut lui-même.

Dans des cas pareils et pour faire la distinction entre les différents attributs, il suffit de changer leurs appellations.

REGLE

Un attribut est une donnée élémentaire excluant également les données calculées ou dérivées.

Au niveau de l'entité modélisant les personnes par exemple, on ne doit pas représenter l'âge si on dispose de sa date de naissance. Car l'âge peut être calculé à partir de la date de naissance.

Aussi, dans une facture, si on a le montant hors taxes et le taux de la TVA on ne représente pas le montant TTC car ce dernier est calculé à base des deux premiers. Il faut garder en tête que chaque attribut doit contenir une information non répétitive. Ceci signifie, que l'attribut est le seul qui nous renseigne sur l'information qu'il détient et la perte de l'attribut induit la perte de cette information.

REGLE

Un type-entité et ses attributs doivent être cohérents entre eux (i.e. ne traiter que d'un seul sujet).

Tous les attributs d'une entité doivent décrire l'entité en question. Par exemple, il serait faux de mettre l'attribut Numéro de châssis au niveau de l'entité Personne. Car ce dernier ne décrit pas une personne et l'information qu'il renseigne n'a rien avoir avec la sémantique de personne.

2.2.2. Valeur

Définition

Chaque attribut possède un domaine définissant l'ensemble de ses valeurs possibles (Entier, Chaîne de caractères, date, booléen, etc.).

Chaque attribut possède une valeur compatible avec son domaine de définition. Le domaine des valeurs d'un attribut représente un filtre aux valeurs que l'on pourrait lui attribuer.

Le domaine de définition d'un attribut représente ainsi une contrainte que les valeurs doivent satisfaire. Par exemple, l'attribut ***Date de naissance*** définit comme étant une date n'admettra que des valeurs de type date et rien d'autre.

Les domaines de valeurs des attributs sont obligatoires au niveau de la documentation des modèles entité association mais facultatifs au niveau de la représentation graphique du modèle.

2.2.3. Identifiant (clé)

Au niveau du modèle EA, chaque entité doit être définie de manière unique. Cette unicité est assurée par son identifiant qui permet de prendre des valeurs uniques (i.e. : ne se répètent pas). Ainsi, il est impossible de trouver dans notre système deux entités avec la même valeur d'identifiant. D'un autre côté, l'identifiant doit être formé d'attributs qui ont toujours une valeur. Car un identifiant n'admet pas de valeurs nulles.

Définition
L'identifiant est l'ensemble minimum d'attributs qui désignent de manière unique une entité.

Le numéro de sécurité sociale, par exemple, peut être utilisé comme identifiant d'une personne, le numéro de châssis pour une voiture et le code ISBN pour un livre.

Les attributs formant l'identifiant doivent ainsi satisfaire ces trois conditions qui sont : l'unicité des valeurs, l'existence de valeurs pour toutes les entités et l'ensemble doit être minimum. Ce qui signifie qu'on ne peut enlever aucun des attributs de l'identifiant et le reste des attributs forment également un identifiant.

REGLE

Chaque type-entité possède au moins un identifiant, éventuellement formé de plusieurs attributs.

Chaque type-entité possède au moins un attribut et quand ce dernier est le seul attribut de l'entité, il est forcément l'identifiant.

Au niveau de la définition d'un identifiant, nous avons évoqué deux caractéristiques principales : l'unicité des valeurs et le fait que l'ensemble des attributs soit minimum.

Cette notion de minimum signifie que parmi les attributs qui forment l'identifiant, on ne peut se passer d'aucun deux et quand on enlève un attribut de l'ensemble, les attributs restants perdent leur propriété d'unicité de valeur.

Supposant que pour identifier une personne on utilise son nom, son prénom et sa date de naissance. On dira que cet identifiant qui est formé de trois attributs est minimal si on ne peut pas enlever un attribut de l'ensemble sans perdre la propriété d'unicité des valeurs des attributs restants.

REGLE

Dans le cas où les attributs d'une entité ne peuvent former un identifiant ou pour des besoins de simplification et de lisibilité du modèle, il est permis d'utiliser des identifiants fictifs.

Un identifiant fictif est un attribut qui ne renseigne pas l'entité en question et ne signifie généralement pas grand-chose. On l'utilise afin de simplifier les identifiants pour mieux contrôler les valeurs ou dans le cas où l'ensemble des attributs de l'entité ne peuvent former un attribut.

Le matricule d'un étudiant, par exemple, est une propriété fictive utilisée au niveau de la scolarité pour identifier les étudiants mais elle ne représente absolument rien et ne porte aucune information relative à l'étudiant.

2.3. Association

Le modèle EA est fondé sur deux concepts essentiels : l'entité et l'association. Nous avons défini l'entité comme étant une représentation des objets de la réalités, l'association, quant à elle, est une représentation des liens sémantiques entre ces objets.

Définition
Une association (relation) est un lien sémantique entre plusieurs entités.

Dans l'exemple précédent, on distingue un lien de lecture et un lien d'achat entre Ali et le livre de bases de données ainsi qu'un lien de vente entre ce dernier et Amar.

Ces différents liens de la réalité peuvent être représentés au niveau du modèle EA via le concept d'association.

De la même manière que pour les entités, les associations ne peuvent pas être représentées en totalité dans un modèle. On parle plutôt de type-association que d'association proprement dites.

Définition
Un type-association (type-relation) désigne un ensemble de relations qui possèdent les mêmes caractéristiques.

Le type-association décrit un lien entre plusieurs type-entités. Les associations de ce type-association lient des entités de ces type-entités.

Définition
Les type-entités intervenant dans un type-association sont appelés les participants de ce type-association.

L'ensemble des participants d'un type-association est appelé la collection de ce type-association et le nombre des type-entités contenus dans cette collection est appelé la dimension, ou l'arité de l'association. La collection d'un type-association comporte au moins un type-entité, mais elle peut en contenir plusieurs, on parle alors de type-association n-aire.

Une association peut contenir des attributs qui représentent des propriétés du lien sémantique représenté par l'association en question.

REGLE

Un attribut placé dans un type-association dépend de toutes les entités liées par le type-association.

Les valeurs de l'attribut d'une association ne peuvent être définies que si toutes les entités participant à l'association sont connues. L'attribut dépend de toutes les entités participant à l'association.

Si on peut attribuer une valeur à l'attribut sans connaitre l'une des entités participant à l'association, ceci signifie que l'attribut en question ne devrait pas être modélisé au niveau de l'association.

Une association ne doit pas forcément contenir des attributs explicites comme c'est le cas pour l'entité. Cependant, elle en contient souvent des attributs implicites qui sont son identifiant.

L'identifiant d'une association est formé des identifiants de toutes les entités participant à l'association.

REGLE

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

Une association est souvent nommée occurrence ou instance de son type-association. Par abus de langage, on utilise souvent le mot association au lieu du mot type-association, il faut cependant prendre garde à ne pas confondre les deux concepts.

2.4. Représentation Graphique

On retrouve dans la littérature plusieurs représentations graphiques des entités et des associations qui sont assez similaires dans l'ensemble.

Dans la suite de ce cours nous utiliserons la convention de la méthode MERISE qui est la plus simple à implémenter et c'est celle qui est utilisée par la plupart des outils de conception EA.

Un type-entité est représenté par un rectangle et le type-association par une ellipse. Les pattes du type-association représentent le rôle que joue l'entité correspondante dans la relation. La figure ci-dessous illustre une représentation graphique d'une association binaire entre deux entités.

Par convention, l'identifiant d'une entité est représenté en gras souligné et celui de l'association est implicite. La représentation des domaines de définition des attributs est facultative.

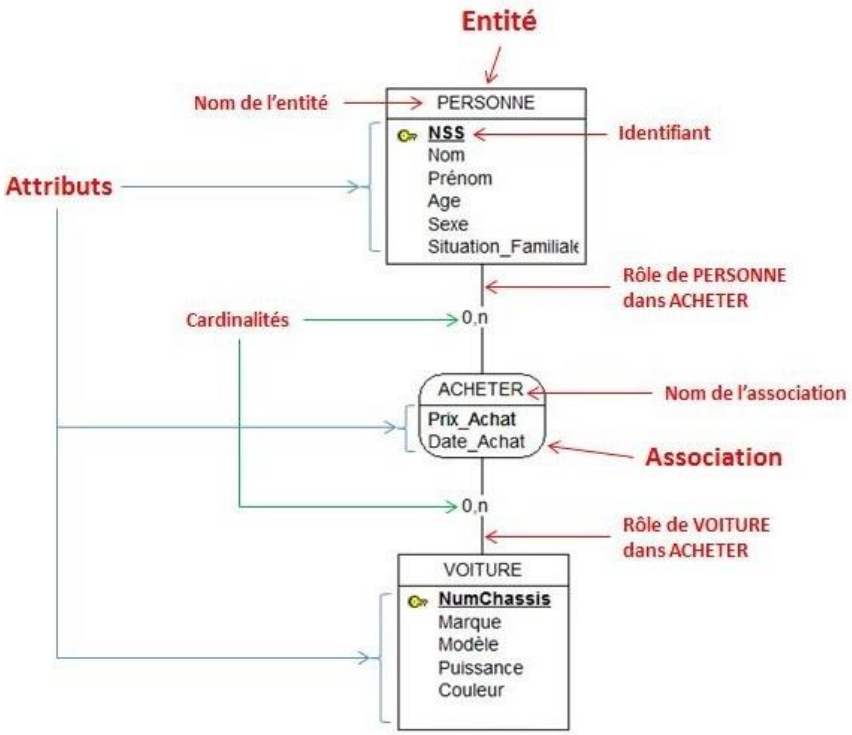


Figure 5 : Représentation graphique du modèle EA

Certains outils de modélisation rajoutent un symbole graphique à l'identifiant d'une entité telle une clé. La position de l'identifiant d'une entité n'est pas forcément en premier. Mais il est souhaitable de mettre les attributs formant l'identifiant en premier.

Le nom du rôle est facultatif. Dans la plupart des cas, le rôle n'est pas mentionné explicitement. Cependant, dans certaines situations que nous allons présenter un peu plus loin dans ce cours, la spécification du rôle est obligatoire.

3. Les Cardinalités

En modélisant une situation donnée, on met des entités liées entre elles avec des associations. Ces dernières représentent les liens sémantiques entre les objets représentés par les entités. L'exemple ci-dessous représente une personne possédant une voiture.

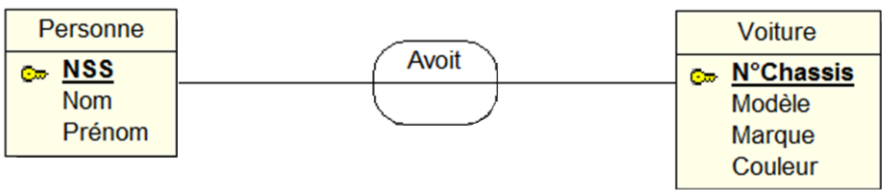


Figure 6 : Utilité des cardinalités

Ce modèle décrit très bien le fait qu'une personne possède une voiture. Cependant, si dans notre situation, une voiture ne peut être possédée que par une seule personne, le modèle devient insuffisant. Il faut que l'on ait la possibilité de représenter cette restriction sur le modèle.

Le modèle EA permet de définir des restrictions sur la participation des entités aux associations. Ces restrictions représentant des contraintes appelées cardinalités.

Définition

La cardinalité d'une patte reliant un type-association A à un type-entité E précise le nombre de fois minimal et maximal d'interventions d'une entité du type E dans une association du type A.

Les cardinalités sont notées dans le schéma EA sur le trait reliant l'entité à l'association ou juste au-dessus. La notation est la suivante : Cardinalité minimale Virgule Cardinalité maximale. Les cardinalités sont reliées au rôle de l'association et non pas à l'entité.

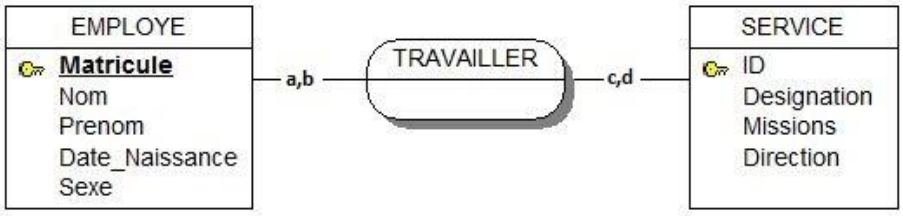


Figure 7 : Illustration des cardinalités

Dans l'exemple ci-dessus : les cardinalités a, b représentent respectivement : la cardinalité minimale et maximale de participation d'une occurrence du type entité *EMPLOYÉ* à une occurrence du type association *TRAVAILLER*. Ce qui se traduit par l'interprétation suivante du rôle du côté de *EMPLOYE* : *Un employé travaille dans au minimum* ***a*** *service(s) et au maximum* ***b.***

REGLES

La cardinalité minimale doit être inférieure ou égale à la cardinalité maximale.

La cardinalité minimale ne peut être supérieure à la cardinalité maximale, ce qui est logique vu que les deux cardinalités représentent un intervalle de valeurs représentant le nombre de participations d'une entité à une association.

REGLES

L'expression de la cardinalité est obligatoire pour chaque patte d'un type-association.

L'expression des cardinalités est obligatoire dans le modèle EA. Quand celles-ci sont omises, cela est considéré comme une erreur syntaxique.

REGLES

Une cardinalité minimale est toujours 0 ou 1 et une cardinalité maximale est toujours 1 ou n.

Le modèle EA définit deux valeurs possibles pour la cardinalité minimale et deux valeurs pour la cardinalité maximale. Ainsi, les cardinalités admises pour un rôle d'une association sont : 0,1 0,n 1,1 1,n

Chacune de ces cardinalités implique des contraintes sur la participation de l'entité à une association donnée. Les cardinalités permettent donc de réduire le nombre de participations d'une entité à une association.

3.1. La Cardinalité Minimale 0

Prenant l'exemple suivant, modélisant la relation d'adhésion d'un étudiant à un club. Les étudiants sont représentés par le type entité *ETUDIANT* et les clubs par le type entité *CLUB*. Les associations d'adhésion sont représentées par le type association *ADHERER*.

Les cardinalités du côté de *ETUDIANT* signifient le nombre minimum et maximum d'adhésion d'un étudiant à des clubs et celles du côté du club signifient le nombre minimum et maximum d'étudiants adhérents du club en question.

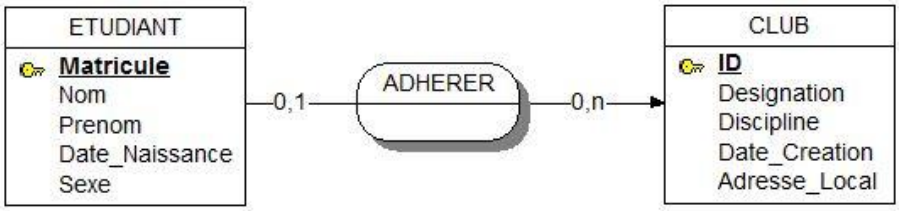


Figure 8 : Cardinalité minimale 0

Définition

La cardinalité minimale 0 signifie qu'une occurrence du type-entité peut exister tout en n'étant impliquée dans aucune occurrence du type-association.

En d'autres termes, On peut instancier (Créer une occurrence) le type-entité indépendamment de sa relation avec le type-association.

Si on applique cette définition à l'exemple précédent on dira qu'on peut avoir des occurrences du type entité *ETUDIANT* (des étudiants) qui ne participent à aucune occurrence du type association *ADHÉRER*. Autrement dit, nous pouvons avoir des étudiants dans notre système qui ne sont des adhérents d'aucun club.

Selon la théorie des ensembles, l'ensemble Etudiant est lié à l'ensemble Club via la relation Adhérer. La cardinalité minimale 0 signifie l'existence d'éléments de l'ensemble Etudiant ne participant à aucune relation Adhérer. Ceci peut être illustré comme suit :

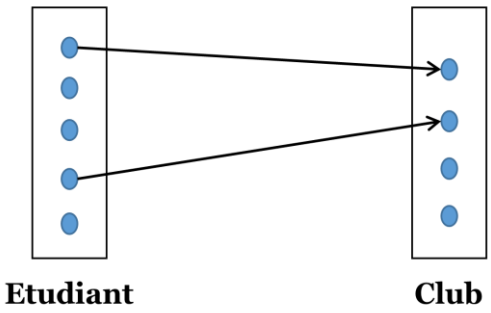


Figure 9 : Représentation ensembliste de la cardinalité minimale 0

Chaque occurrence de *ETUDIANT* (chaque étudiant) peut adhérer ou ne pas adhérer à un club.

Selon cette définition, la cardinalité 0 lève toute restriction sur la participation d'une entité à une association. Elle représente ainsi l'inexistence de contraintes sur le nombre de participations minimales.

3.2. La Cardinalité Minimale 1

La cardinalité minimale 1 est plus restrictive que la cardinalité minimale 0. Sa présence met des restrictions sur l'instanciation du type-entité et la rend dépendante de la participation de l'entité à la relation.

Définition

La cardinalité minimale 1 signifie qu'une occurrence du type-entité ne peut exister que si elle participe au moins à une association du type association en question.

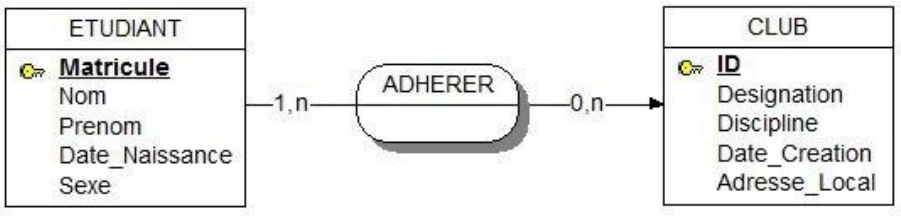


Figure 10 : Cardinalité minimale 1

Dans le cas ci-dessus, Le fait de mettre une cardinalité minimale 1 du côté de l'étudiant signifie qu'à chaque fois que l'on souhaite instancier un étudiant le modèle exigera que ce dernier soit participant d'au moins une association de type *ADHERER*.

En d'autres termes, la cardinalité minimale 1 signifie que tous les éléments de l'ensemble Etudiant participent à au moins une relation Adhérer. Cette contrainte de participation obligatoire à la relation Adhérer devient ainsi une caractéristique des éléments de l'ensemble Etudiant.

Ceci peut être illustré comme suit :

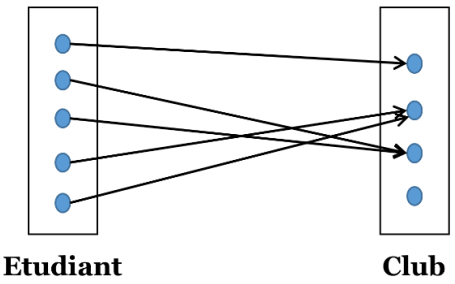


Figure 11 : Représentation ensembliste de la cardinalité minimale 1

Chaque élément de l'ensemble Etudiant est lié à un élément de l'ensemble Club via une relation Adhérer.

3.3. La Cardinalité Maximale 1

Les cardinalités maximales définissent le nombre maximum de participations d'une entité à une association. Ce qui peut être vu comme étant un plafond à ne pas dépasser. Ceci réduit le champ d'action de l'entité par rapport à l'association.

Définition

La cardinalité maximale 1 signifie qu'une occurrence du type-entité ne peut participer à une occurrence du type-association qu'une seule fois.

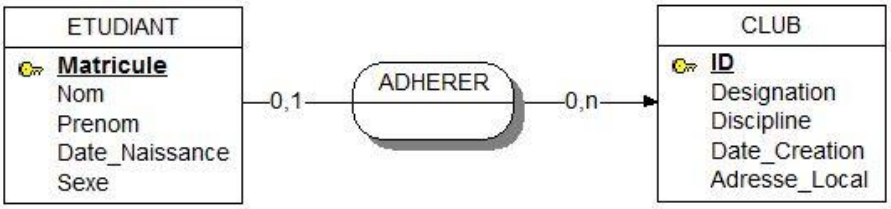


Figure 12 : Cardinalité maximale 1

Dans l'exemple ci-dessus, un étudiant n'a le droit d'adhérer qu'à un seul club à la fois. Ce qui signifie qu'à n'importe quel moment de la durée de vie du modèle, on ne pourra trouver une instance de *ETUDIANT* participant à plus d'une association *ADHERER*.

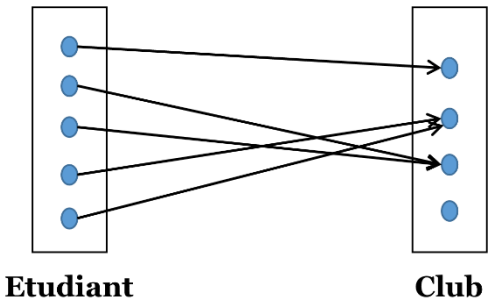


Figure 13 : Représentation ensembliste de la cardinalité maximale 1

Chaque élément de l'ensemble Etudiant n'a au maximum qu'une seule relation Adhérer. Autrement dit, il ne peut être en relation qu'avec un seul élément de l'ensemble Club.

3.4. La Cardinalité Maximale n

Quand on n'a aucune limitation dans le nombre de participations d'une entité à une association on met la cardinalité maximale n. Celle-ci élimine donc la restriction sur le nombre maximum de participations.

Définition

La cardinalité maximale n signifie qu'une occurrence du type-entité peut participer un nombre illimité de fois à des associations du type association.

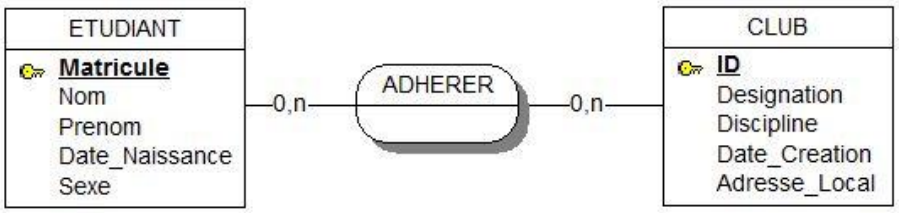


Figure 14 : Cardinalité maximale n

Dans l'exemple ci-dessus, un étudiant peut adhérer un nombre illimité de fois à des clubs. Et la représentation ensembliste correspondante est la suivante :

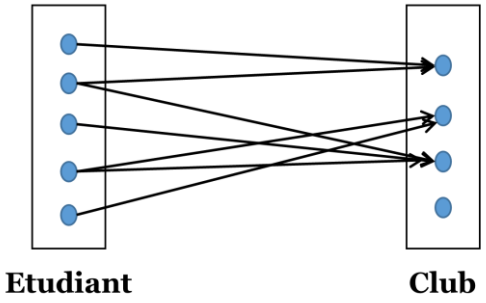


Figure 15 : Représentation ensembliste de la cardinalité maximale N

3.5. Interprétation Des Associations

Un modèle est utilisé afin de représenter une réalité complexe sous une forme compréhensible et facilement communicable. Il est donc important de savoir lire et interpréter tous les éléments du modèle.

Au niveau de la modélisation Entité/Association, l'interprétation des entités et des attributs est une chose simple et intuitive, ce qui n'est pas le cas pour les associations surtout quand l'arité de ces dernières est supérieure à 2. Nous allons expliquer dans cette section la méthode de lecture des liens d'une association et la manière de déterminer les cardinalités adéquates.

3.5.1. Association binaire

La collection d'une association binaire est composée de deux entités. Le lien entre chaque entité avec l'association représente le rôle que joue l'entité dans cette association. Ce rôle est généralement implicite, on peut ne pas le mentionner au niveau du modèle sauf dans certains cas particuliers où il devient obligatoire.

Soit l'exemple vu précédemment :

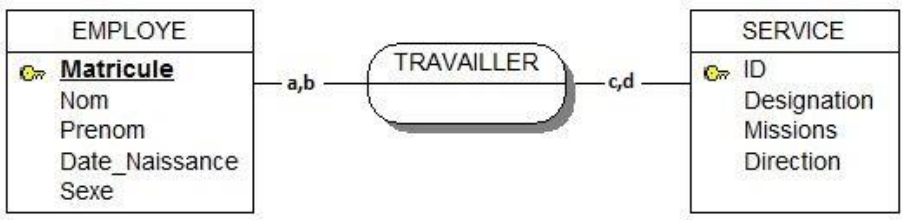


Figure 16 : Interprétation d'une association binaire

La lecture de l'association *TRAVAILLER* peut se faire dans deux sens : soit dans le sens de l'employé ou dans le sens du service. Le nom donné à l'association représente souvent le sens prédominant (le sens auquel on s'intéresse le plus) le second est déduit.

L'interprétation de l'association *TRAVAILLER* dans le premier sens avec intégration des cardinalités donne : « Un employé travaille dans a ou b services ». Le second sens donne une autre lecture qui est : « Un service comporte c ou d employés ». On peut choisir un autre verbe que *COMPORTE* pour exprimer le lien entre le service et ses employés mais le sens général reste le même.

Notre souci maintenant est de déterminer les différentes cardinalités. Ce qui revient à choisir les valeurs de a, b, c et d parmi celles permises {0,1 pour la cardinalité minimale et 1, n pour la cardinalité maximale}.

Le choix de ses valeurs dépend de la situation étudiée et de ce qui est attendu du modèle. Pour cela on doit répondre aux questions suivantes :

1. **Déterminer la cardinalité minimale :**

La question générique qui permet de déterminer la cardinalité minimale est la suivante :

« Peut-on avoir des entités du type-entité E1 qui ne sont associées à aucune entité du type-entité E2 via une association du type-association A ?»

Pour déterminer la valeur de la cardinalité minimale ***a*** dans l'exemple précédent, on va adapter cette question à notre situation comme suit : « Peut-on avoir des employés qui ne travaillent dans aucun service ?»

SI la réponse est OUI Alors a = 0 SINON a=1

Pour déterminer la valeur de la cardinalité minimale c on instancie la question générique pour le Type-Entité *SERVICE* comme suit : « Peut-on avoir des services dans lesquels ne travaillent aucun employé ?»

SI la réponse est OUI Alors c = 0 SINON c=1

2. **Déterminer la cardinalité maximale :**

La question générique qui permet de déterminer la cardinalité maximale est la suivante :

« A Combien d'entités du type-entité E2, une entité du type-entité E1 peut être associée via une association du type-association A ?»

Pour déterminer la valeur de la cardinalité maximale b dans l'exemple précédent on instancie cette question pour le Type-Entité Employé comme suit : « Dans combien de services un employé peut-il travailler ?»

SI la réponse est UN SEUL Alors b = 1 SINON (>1 ou Plusieurs) b=n

Pour déterminer la valeur de la cardinalité maximale d on instancie la question générique pour le Type-Entité SERVICE comme suit : « Combien d'employés peuvent travailler dans un service ?»

SI la réponse est UN SEUL Alors d = 1 SINON (>1 ou Plusieurs) d=n

3.5.2. Association ternaire

Soit l'exemple suivant représentant la situation d'enseignement. Des enseignants enseignent des modules par année universitaire.

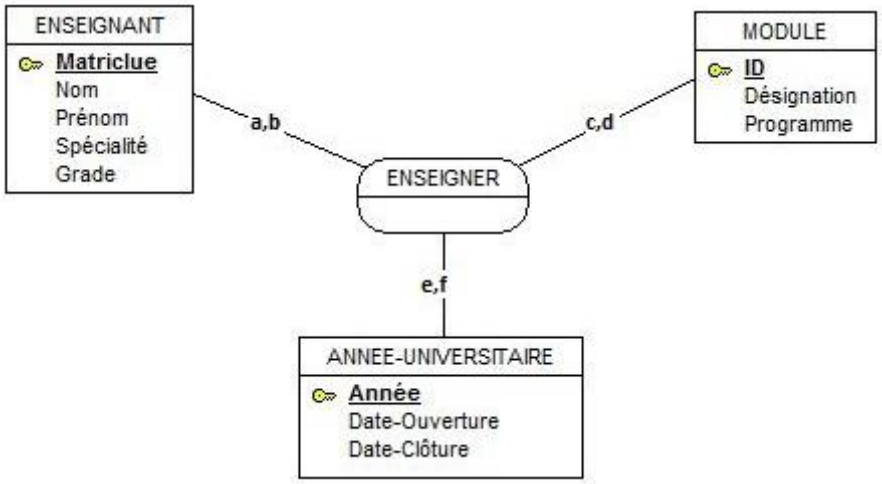


Figure 17 : Interprétation d'une association ternaire

Une association ternaire représente un lien entre trois entités. Généralement il existe un lien fort entre deux entités et la troisième n'est là que pour jouer le rôle d'un arbitre. Le rôle de la troisième entité est un rôle de témoin permettant à l'autre couple de se réunir plusieurs fois.

L'interprétation des associations ternaires n'est plus la même que pour les binaires. A ce niveau on interprète le rôle d'une entité par rapport au couple correspondant formé des deux autres entités participantes à l'association.

La lecture de l'association de l'exemple ci-dessus partant du côté de l'entité ENSEIGNANT est la suivante : Un enseignant enseigne des modules à des années universitaires données.

Du point de vue du MODULE : Un module est enseigné par des enseignants à des années universitaires données. Et du point de vue de l'ANNEE UNIVERSITAIRE : Durant une année universitaire, des enseignants enseignent des modules.

On remarque dans cet exemple que le lien d'enseignement est à l'origine un lien entre enseignant et module, l'année universitaire est un arbitre dans l'association.

Pour trouver les différentes cardinalités, on utilise la même approche que pour l'association binaire sauf qu'au lieu de chercher la participation d'une entité avec une autre entité via l'association on cherchera la participation d'une entité avec un couple d'entités via cette association.

Par exemple, pour trouver la cardinalité minimale ***a*** dans l'exemple précédent on pose la question suivante : « Peut-on avoir des enseignants qui n'enseignent aucun module à aucune année universitaire ?». Et pour trouver la cardinalité maximale b on pose la question : « A combien de couples (module, année universitaire) un enseignant peut-il être associé via l'association Enseigner ? ». Pour la cardinalité maximale on peut également décomposer cette question en posant deux questions et ceci en fixant la valeur d'une entité et chercher le nombre maximal de l'autre. A la fin on prend le maximum des deux réponses.

On dira donc : Combien de modules un enseignant peut-il enseigner dans une année universitaire donnée ? Et la seconde question est : Durant combien d'année universitaires, un enseignant peut-il enseigner le même module ?

ATTENTION !

La cardinalité maximale de toutes les pattes d'une association ternaire est toujours n.

Si la cardinalité maximale d'une des pattes d'une association ternaire est 1, ceci signifie qu'un seul couple des deux autres entités est permis pour association avec cette entité. Supposant que dans l'exemple précédent, la cardinalité du côté de Enseignant est 0,1 (Figure ci-dessous).

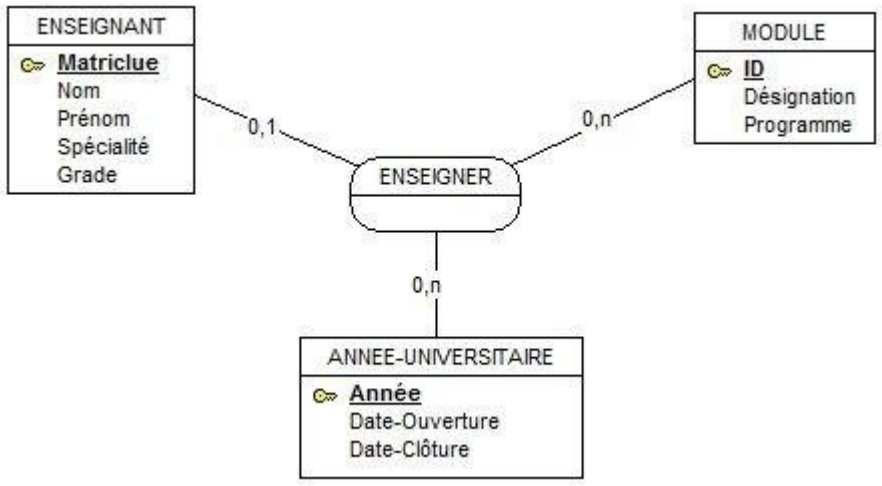


Figure 18 : Cardinalité maximale dans le cas d'une ternaire

Ceci signifie qu'un enseignant ne peut enseigner qu'un seul module quel que soit l'année universitaire et il ne peut enseigner que durant une seule année universitaire quel que soit le module.

Partant de cette interprétation, on remarque que la ternaire n'a plus aucun sens du moment que le lien entre le module et l'année universitaire a été cassé. Dans ce cas, la ternaire est remplacée par deux associations binaires l'une entre enseignant et module et l'autre entre enseignant et année universitaire. Et la cardinalité maximale pour ces deux nouvelles associations est 1.

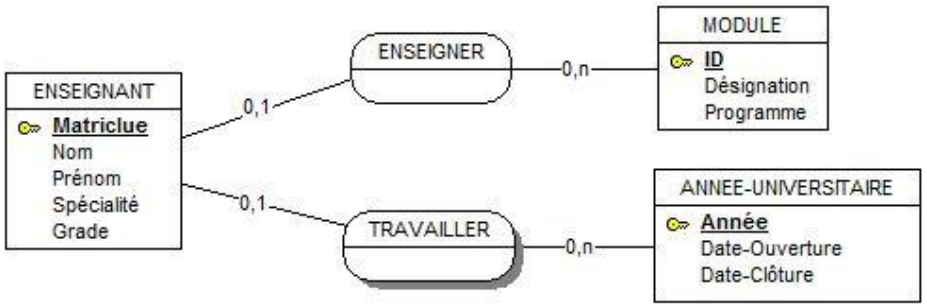


Figure 19 : Eclatement d'une ternaire avec une cardinalité maximale 1

OBSERVATIONS

Une cardinalité minimale 1 doit se justifier par le fait que les entités du type-entité en question ont besoin de l'association pour exister. Dans tous les autres cas, la cardinalité minimale est 0.

Une cardinalité minimale 0 signifie qu'une entité du type-entité correspondant peut exister tout en n'étant impliquée dans aucune association.

3.6. Cas Particuliers

3.6.1. La Cardinalité Minimale 1

L'utilisation de la cardinalité minimale 1 peut causer certaines incohérences dans le modèle global pour les entités qui participent à plusieurs associations.

Soit le cas de figure suivant représentant deux associations : PROPOSER qui modélise la proposition de produits par des fournisseurs à des dates données et Ranger qui modélise le rangement des produits dans des rayons à des dates données.

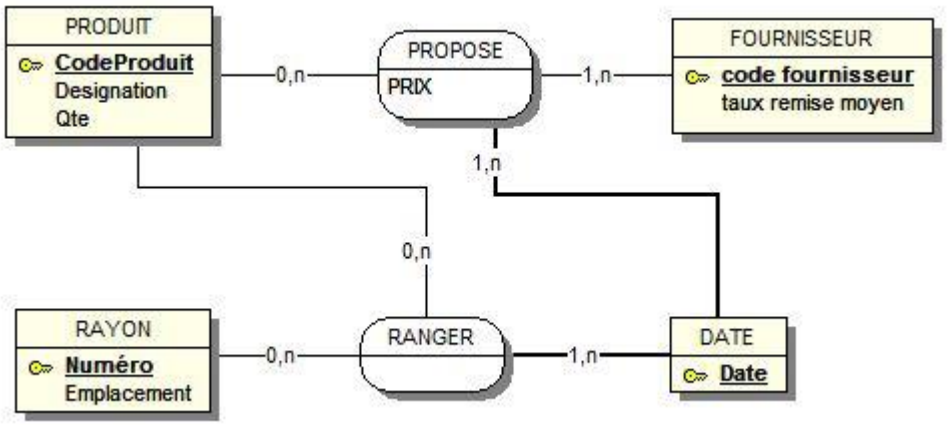


Figure 20 : Problème de la cardinalité minimale 1

Le fait de mettre 1 comme cardinalité minimale impose la participation de l'entité concernée à l'association pour exister. Dans l'exemple ci-dessus, la date ne peut exister que si elle participe à la fois à l'association « Proposer » et « Ranger ». Or la date de proposition d'un produit par un fournisseur n'existe pas forcément comme date de rangement et vis-versa. Ce qui fait que dans un cas pareil, la cardinalité minimale doit impérativement être 0 vu que les deux associations « Proposer » et « Ranger » sont sémantiquement indépendantes.

REGLES

Pour trouver les cardinalités d'une patte d'une association il faut :

1. *Prendre en considération toutes les pattes de l'association,*

2. Pour chaque participant, prendre en considération toutes ses participations.

Cette règle est surtout valable dans le cas de présence de la cardinalité minimale 1.

3.6.2. L'Association 1.1 - 0.n ou 1.1 - 1.n

Les attributs d'une association doivent dépendre de tous les participants à cette association. Ce qui signifie que la valeur d'un tel attribut est déterminée par tous les participants sans exception.

Soit l'exemple ci-dessous représentant l'action d'achat d'une voiture par une personne. Nous ne modélisons que les voitures achetées par les personnes que nous avons et que chaque voiture n'est achetée que par une seule personne.

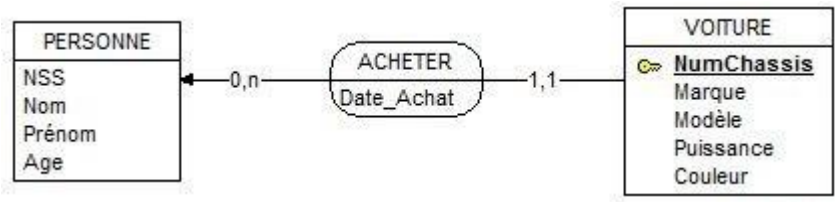


Figure 21 : Association 1.1, 1.n avec attributs

La propriété Date Achat est modélisée au niveau de l'association « ACHETER », ce qui signifie qu'elle dépend à la fois de Voiture et de Personne. Or, le fait que la cardinalité du coté de Voiture est 1,1 signifie qu'une voiture ne peut exister que si elle participe Une et Une fois à l'association ACHETER, ce qui fait, que finalement, la date achat dépend uniquement de Voiture et non pas des deux entités ensemble. C'est pourquoi elle devrait être modélisée au niveau de Voiture plutôt que dans l'association ACHETER.

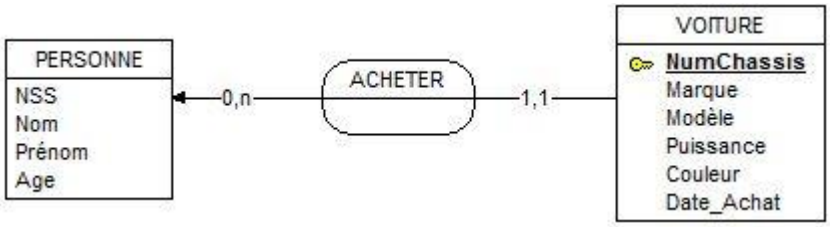


Figure 22 : Association 1.1 ne doit pas avoir d'attributs

Alors que dans le cas d'une association avec une cardinalité 0,1, la date achat dépend des deux entités. Car si on la met au niveau de voiture et sachant qu'une voiture peut exister sans participer à l'association, cette date pourrait ne pas avoir de valeur, ce qui n'était pas le cas dans la cardinalité 1,1. Donc, dans le cas de la cardinalité 0,1, la date achat doit être placée au niveau de l'association.

REGLE

L'association 1,1 - 1,N ne doit pas posséder d'attributs.

4. Concepts Avancés

4.1. Entité Faible

Jusqu'à présent nous avons considéré le cas d'entités indépendantes les unes des autres. Chaque entité, disposant de son propre identifiant, pout être considérée isolément.

Il existe des cas où une entité ne peut exister qu'en étroite association avec une autre entité, et son identifiant est relatif à celui de cette dernière. On parle alors d'entité faible.

Prenons l'exemple suivant modélisant les chambres d'hôtel. Les chambres de nos hôtels sont toutes identifiées par un numéro séquentiel partant de 1 jusqu'à la dernière chambre.

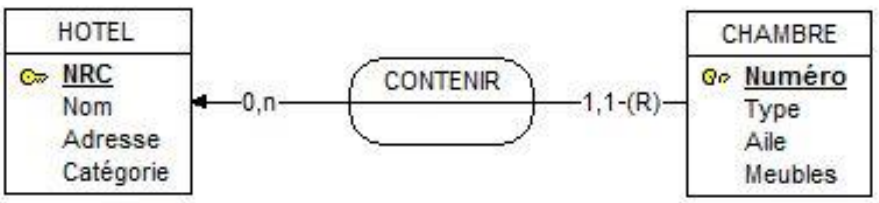


Figure 23 : Entité faible

Définition

Une entité faible est une entité possédant un identifiant insuffisant de par lui-même pour identifier de manière unique chacune de ses occurrences. Sa caractéristique d'identifiant n'est valable qu'à l'intérieur du contexte spécifique de l'occurrence d'une entité principale.

Dans le cas de l'exemple précédent, l'identifiant d'une chambre est constitué de deux parties : l'identifiant de l'hôtel et le numéro de la chambre.

OBSERVATION

L'introduction d'entités faibles n'est pas une nécessité absolue puisqu'on peut très bien utiliser une association classique. La principale différence est que, dans le cas d'une entité faible, on obtient une identification composée qui est souvent plus pratique à gérer, et peut également rendre plus faciles certaines requêtes. Alors que dans le second cas, on doit créer nous même un identifiant fictif qu'il faudra gérer par la suite.

La cardinalité du côté de l'entité faible est toujours 1,1. La cardinalité maximale 1 se justifie par le fait d'identification composée (Chaque entité du type-entité faible n'a qu'une seule entité de l'autre type-entité par rapport à laquelle elle s'identifie) et la cardinalité minimale ne peut pas être 0 car cela signifierai qu'une entité du type-entité faible pourrait exister sans participer à l'association ce qui est contradictoire avec la définition de l'entité faible

Peut-on définir des entités faibles en cascade ?

Il est possible d'avoir des entités faibles en cascade, l'identifiant des entités est obtenu dans ce cas en effectuant la migration des identifiants des pères vers les fils, en cascade également.

Considérons l'exemple suivant :

Dans notre école les niveaux d'études sont identifiés par rapport au cycle : {Cycle Commun, Cycle Supérieur}, nous avons 2 niveaux du premier cycle et trois niveaux du second cycle. Dans chaque cycle on peut trouver plusieurs sections qui sont identifiées par des lettres de l'alphabet {Section A, Section B etc.}. Les groupes sont identifiés par un numéro séquentiel dans le niveau {Groupe 1, Groupe 2 etc.}

On peut donc trouver plusieurs groupes avec le numéro 1 mais dans des niveaux différents, et plusieurs sections A mais également dans des niveaux différents. Idem pour les niveaux, on a le niveau 1 du cycle CC et le niveau 1 du cycle CS.

Pour modéliser cela, on peut utiliser des associations classiques avec des cardinalités 1,1 et gérer des identifiants avec des valeurs composées. Par exemple, au lieu de dire que l'identifiant du groupe est 1 on dira que c'est 11CS ce qui correspond au groupe 1 du niveau 1 du cycle supérieur. Mais cette solution nous posera quelques difficultés si on souhaite faire une requête sur tous les groupes 1 quel que soit leur niveau. Il faudra alors traiter des chaînes de caractères afin d'extraire le groupe.

La modélisation de cette situation avec des entités faibles en cascade donne le schéma suivant :

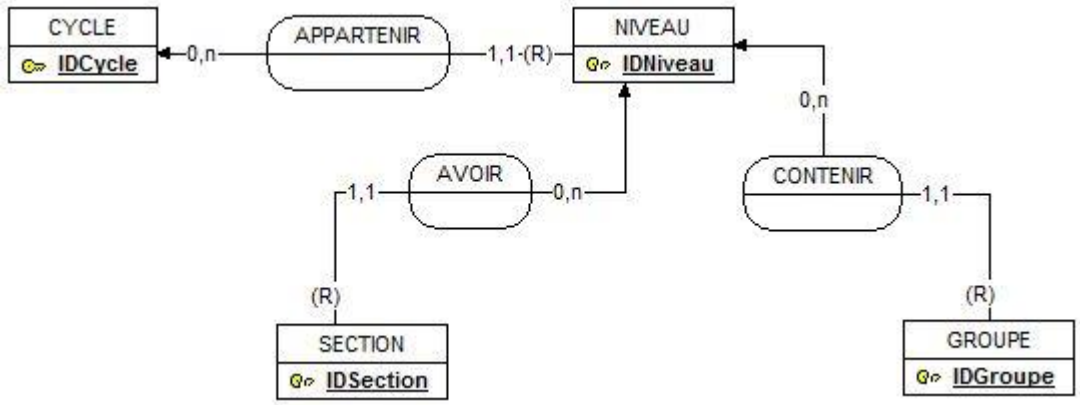


Figure 24 : Entités faibles en cascade

Dans ce cas, les identifiants sont générés en cascade en partant de l'entité mère (CYCLE) jusqu'à l'entité faible. Les identifiants des entités faibles du modèles sont :

- NIVEAU = {IDCycle, IDNiveau}
- SECTION = {IDCycle, IDNiveau, IDSection}
- GROUPE = {IDCycle, IDNiveau, IDGroupe}

4.2. Structures Hiérarchiques

On rencontre souvent lors de la modélisation des situations réelles une suite d'associations entres des entités avec des cardinalités hiérarchiques (1,1) représentant des liens Père/Fils successifs. On appelle ce type d'associations des structures hiérarchiques.

Définition

Une structure hiérarchique représente une décomposition de concepts allant du général au particulier. Il s'agit d'une structure où un parent peut avoir plusieurs enfants, mais où chaque enfant ne peut avoir qu'un seul parent.

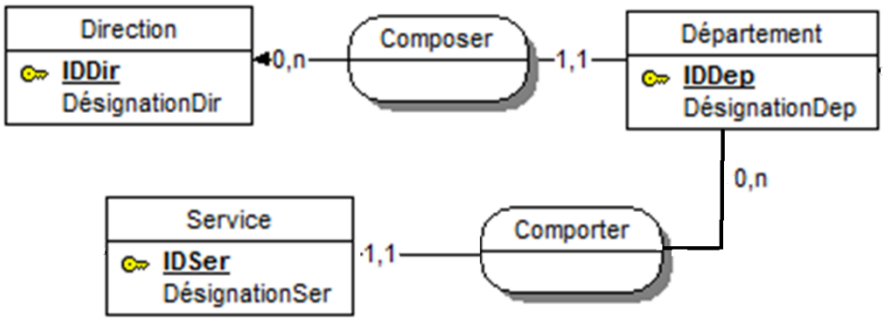


Figure 25 : Les Structures Hiérarchiques

Dans l'exemple ci-dessus, nous modélisons des structures hiérarchiques représentant les structures administratives d'une entreprise. Une entreprise comporte des directions dont

chacune est composées de départements et chaque département est composé à son tour de services. Dans ce cas, on ne peut avoir un service que si on a son département, et le département dépend de la direction.

Les associations directes entre les entités sont interdites dans des structures hiérarchiques. Par exemple, il nous est interdit de mettre une association entre le service et la direction, même si sémantiquement parlant un service appartient bel et bien à une direction. Cependant, quand on a deux chemins qui lient deux entités et dont la sémantique est la même on élimine le plus court des deux. Le schéma ci-dessous est faux.

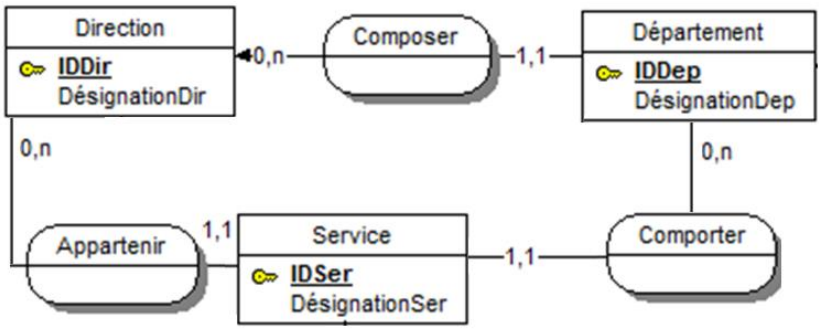


Figure 26 : Structures hiérarchiques avec plusieurs chemins

Il existe deux chemins sémantiquement identiques entre direction et service. La relation Appartenir doit être supprimée car le lien qu'elle modélise peut-être déduit des deux autres associations.

4.3. Associations Plurielles

Les associations plurielles sont des associations qui ont la même collection (i.e. : Les mêmes participants).

> ***Définition***
> *Les associations plurielles expriment le fait que deux objets puissent avoir plusieurs liens sémantiquement distincts.*

La situation décrite dans le modèle ci-dessous représente des livres, leurs auteurs ainsi que les critiques sur ces livres. Les gens qui font la critique sont également des auteurs.

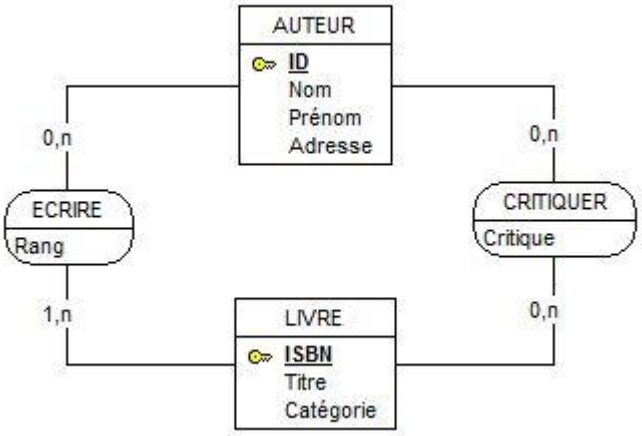


Figure 27 : Associations plurielles

Beaucoup d'étudiants modéliseraient une situation pareille avec une seule association et utiliseraient des attributs (Souvent booléen) pour représenter l'action d'écrire et celle de critiquer. Cette démarche serait complètement erronée, car on ne peut représenter un lien par des attributs encore moins quand ceux-là sont insignifiants. Il faut donc mettre autant d'associations qu'il y a de liens sémantiques entre les entités.

4.4. Association Réflexive

Comme on l'a mentionné précédemment, une association exprime un lien entre des objets de la réalité. Il se trouve, dans certains cas que ces objets-là soient de même nature. Ce qui fait que leur modélisation soit faite par un seul type-entité. Ceci nous emmène à modéliser les liens entre ces objets par des associations entre un type-entité et lui-même.

Définition
Un type-association est qualifié de réflexif quand il matérialise une relation entre un type-entité et lui-même.

Soit l'exemple ci-dessous modélisant l'Etat Civil des personnes. On retrouve ainsi deux liens (associations) : un lien de mariage ÉPOUSER et un lien de parenté AVOIR. Une personne épouse une autre personne et peut être parente d'autres personnes.

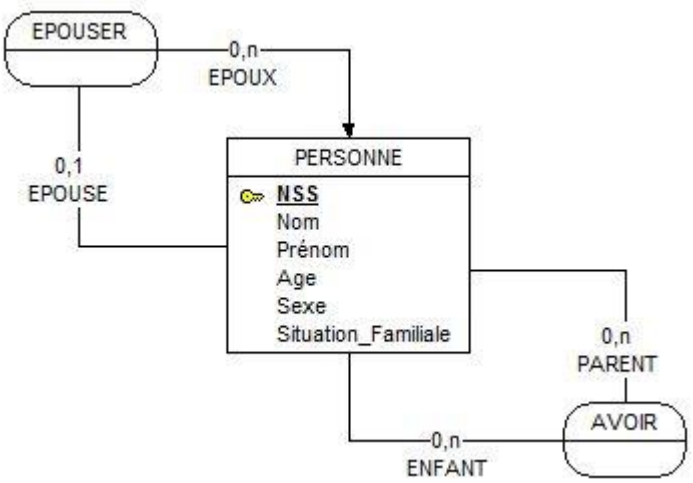


Figure 28 : Association réflexive

NOTE

Dans le cas d'une association réflexive, les rôles doivent être mentionnés clairement sur le diagramme.

Dans une association réflexive il est primordial de spécifier les rôles et les identifier avec leurs noms. Ceci permet de distinguer les rôles que jouent deux entités du même type-entité dans l'association réflexive.

L'utilisation des associations réflexives induit beaucoup de cas indésirables qu'il faudra gérer par la suite en utilisant les contraintes. Par exemple, dans le cas présenté ci-dessus, on devrait s'assurer que ne soit pas la même entité qui participe en tant que époux et épouse à la fois.

4.5. Les Domaines De Valeur

Certaines informations ne peuvent accepter qu'un ensemble déterminé et limité de valeurs. Elles sont donc restreintes à un domaine de valeurs comme pour le sexe qui ne peut être que *MASCULIN* ou *FEMININ*.

Quand on rencontre ce type d'attributs dans la réalité, deux façons de les modéliser sont possibles : une modélisation par attribut ou une modélisation par entité.

Soit l'exemple ci-dessous représentant un Employé d'une entreprise.

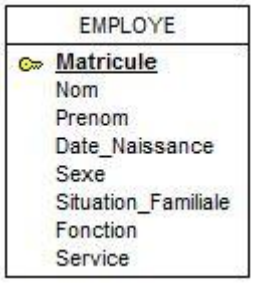


Figure 29: Domaines de valeur

Certains attributs de l'entité EMPLOYE ont des plages de valeurs restreintes et limitées. Ces attributs sont les suivants avec leurs valeurs permises respectives :

- SEXE = {Masculin, Féminin}
- SITUATION FAMILIALE = {Célibataire, Marié, Divorcé, Veuf}
- FONCTION = {Liste des fonctions de l'entreprise}
- SERVICE = {Liste des services de l'entreprise}

La modélisation de ces informations comme on la fait dans l'exemple ci-dessus n'est pas forcément correcte. C'est la nature et l'utilisation ultérieure de ces informations qui déterminent le type de modélisation.

4.5.1. Modélisation par attribut

Ajouter un attribut au sein de l'entité concernée et indiquer, dans la documentation du modèle, les valeurs permises. Cette méthode est bien adaptée aux cas où l'utilisateur n'a aucun contrôle sur le domaine de valeurs en question (on ne peut ni ajouter ni retirer des valeurs permises).

Ce type de modélisation est très bien adaptée au sexe et à la situation familiale car on n'a aucun contrôle sur les valeurs du sexe ni sur celles de la situation familiale. On ne peut pas ajouter de valeurs à celles permises pour le sexe ni à celles permises pour la situation familiale et on ne peut pas en enlever également.

4.5.2. Modélisation par entité

Dans ce type de modélisation, on ajoute une entité au modèle pour représenter le domaine de valeurs et on relie l'ancienne entité à la nouvelle entité représentant le domaine de valeur par une association hiérarchique.

Cette méthode doit être utilisée lorsque l'utilisateur a plein contrôle sur le domaine de valeurs. Ce qui signifie qu'on peut ajouter, modifier ou supprimer des valeurs.

Ce type de modélisation est bien adaptée à la fonction et au service car on peut toujours ajouter une nouvelle fonction au sein de l'entreprise et la même chose pour les services.

La modélisation la plus proche de la réalité de l'exemple précédent est donc représentée par le schéma suivant :

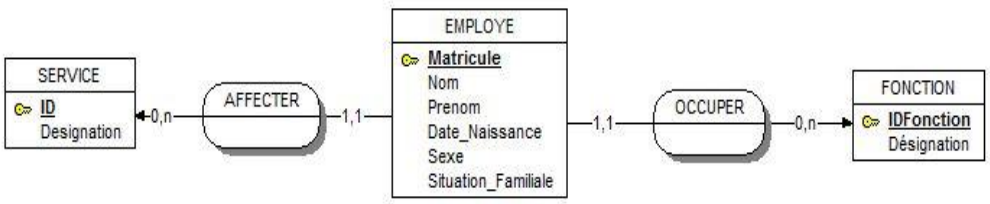


Figure 30 : Domaines de valeur

4.6. Conservation De L'historique

Considérons la situation suivante où un employé d'une entreprise occupe une fonction à partir d'une date donnée.

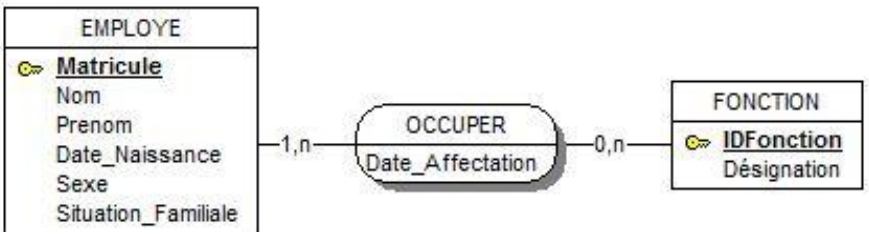


Figure 31 : Conservation de l'historique

L'association OCCUPER modélise un lien entre EMPLOYE et FONCTION, son identifiant est composé des identifiants de ces deux entités. Supposons maintenant que nous voulions modéliser le fait que l'employé N° 203 occupe la fonction de superviseur le 01/03/2013 et celle de directeur technique le 01/04/2013. Le diagramme d'occurrences suivant représente ce cas :

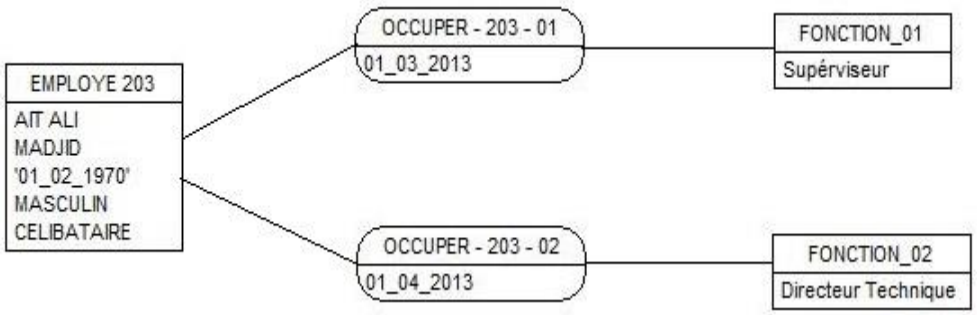


Figure 32 : Diagramme d'occurrences

Supposons maintenant qu'à la date du 20/04/2013 ce même employé 203 réoccupe la même fonction de superviseur qu'il a occupé précédemment. Dans ce cas un conflit se présente : Nous avons déjà une association entre l'employé 203 et la fonction 01, si on prend la nouvelle association on devra écrase l'ancienne et perdre ainsi l'information que ce dernier ait déjà occupé cette fonction auparavant.

Dans ce cas si on veut garder les deux associations il nous faut introduire un arbitre qui sera témoin des rencontres du couple EMPLOYE, FONCTION et la date va jouer ce rôle.

REGLE

Pour garder l'historique d'une association on ajoute l'entité DATE à sa collection.

Le fait d'ajouter la date à la collection de l'association signifie que cette dernière va rentrer dans la composition de l'identifiant de cette association.

Le modèle précédent va devenir comme suit : la date d'affectation sort de l'association pour en devenir une entité.

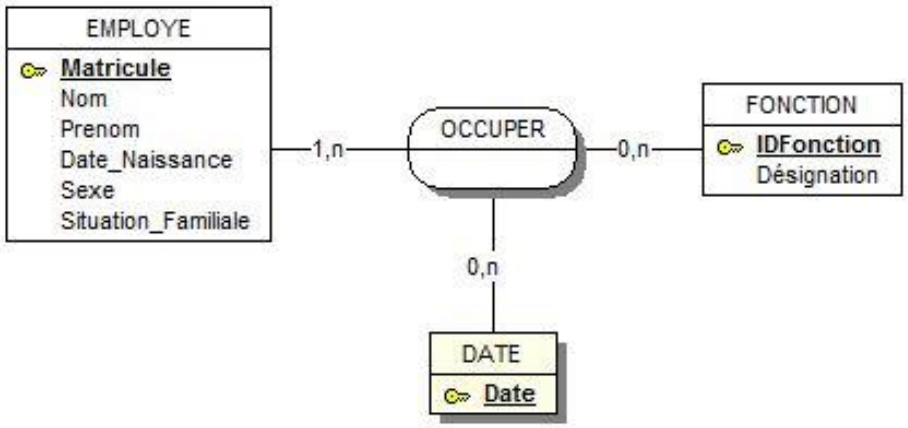


Figure 33 : Garder l'historique

Avec cette nouvelle modélisation, les deux cas précédents peuvent être représentés tous les deux en même temps et sans aucun problème car les deux associations sont identifiées non seulement avec l'identifiant de l'employé et celui de la fonction mais on y ajoute la date.

4.7. L'Agrégation

Les concepts de base du modèle Entité/Association que nous avons vu jusqu'ici permettent de modéliser pratiquement toutes les situations envisageables. Cependant, il existe des situations où la modélisation avec ces concepts s'avère difficile et pas du tout évidente.

Soit la situation suivante : Une personne travaille dans des entreprises et peut occuper différentes fonctions dans la même entreprise. On peut modéliser cette situation avec une association ternaire entre la personne, l'entreprise et la fonction comme suit :

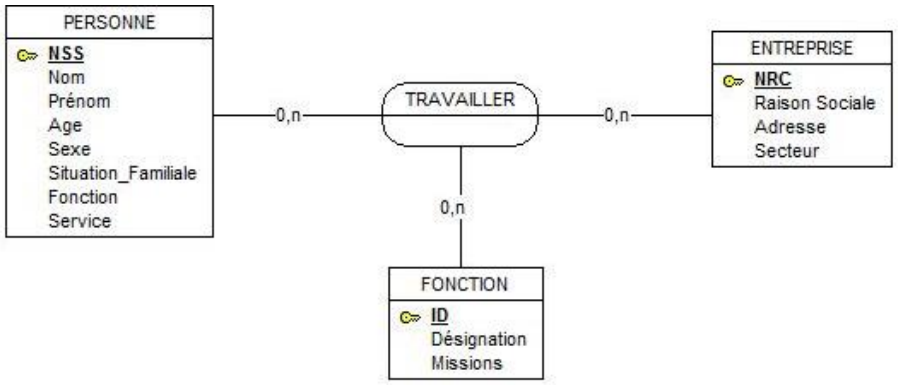


Figure 34 : Agrégation : Solution N° 1 = Association ternaire

Cette modélisation est sémantiquement incorrecte vu que nous avons jumelé deux liens sémantiques (Travailler et Occuper une fonction) dans une seule association. D'un autre côté, si la même personne occupe plusieurs fonctions dans la même entreprise, cela nous

oblige à dupliquer l'association TRAVAILLER pour toutes ces fonctions. Cette modélisation est donc incorrecte.

On peut envisager deux associations distinctes, une pour modéliser le fait qu'un employé travaille dans une entreprise et la seconde pour modéliser l'occupation de l'employé des fonctions au seins de l'entreprise.

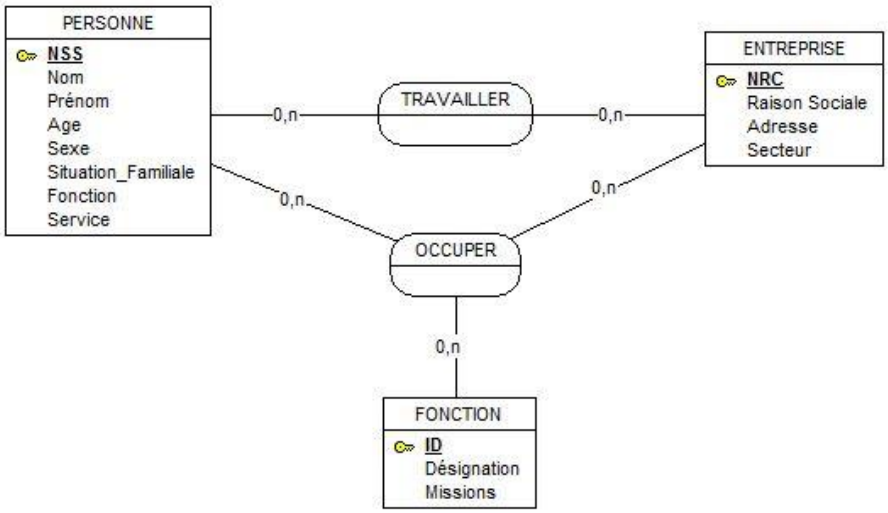


Figure 35 : Agrégation : Solution N° 2 = Deux associations distinctes

A première vue cette solution tient la route. Cependant, on peut avoir le cas d'un employé qui occupe des fonctions dans une entreprise sans pour cela qu'il y travaille. Les deux associations ne sont donc pas aussi indépendantes qu'on pourrait le penser. Mais l'occupation d'une fonction par un employé dans une entreprise ne peut avoir lieu que si ce dernier travaille dans l'entreprise en question.

4.7.1. L'agrégation

Un nouveau concept introduit dans le modèle entité association de base permet une modélisation adéquate de cette situation. C'est l'agrégation.

> ***Définition***
> *Agréger c'est réunir de manière à constituer un tout.*

L'agrégation est un concept qui est rajouté au modèle entité association pour permettre d'exprimer certaines situations où nous avons un lien entre un objet et un ensemble d'objets liés avec une association.

> ***Définition***
> *L'agrégation dans la modélisation des données est le fait de représenter un ensemble d'entités par l'association qui les lie et la considérer comme étant une entité capable de faire partie des collections d'autres associations.*

Selon cette définition et en appliquant le concept d'agrégation, la solution à notre exemple est modélisée ci-dessous.

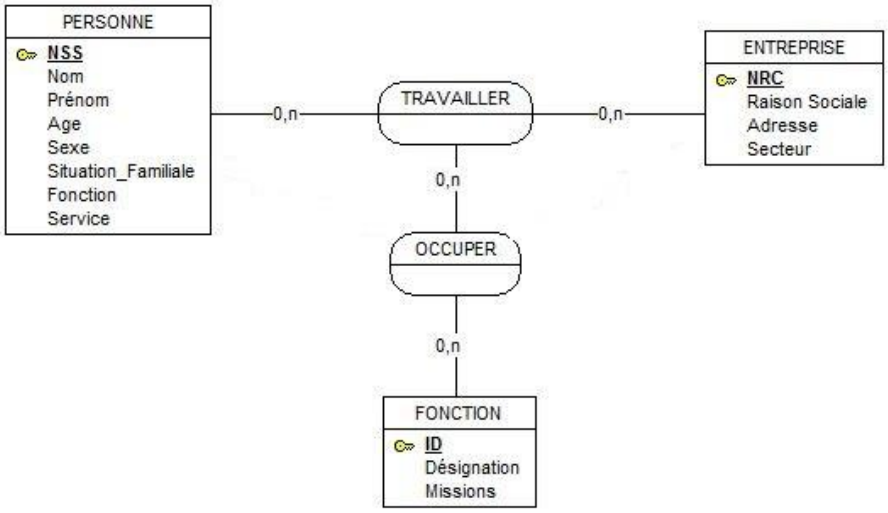


Figure 36 : Solution N° 3 = Agrégation

Dans le cas de la solution N°2, l'identifiant de l'association OCCUPER est composé de l'identifiant de PERSONNE, celui de *ENTREPRISE* et celui de *FONCTION*. Pour pouvoir instancier cette association il faut qu'on ait une instance de personne, une instance de *ENTREPRISE* et une instance de fonction.

Dans le cas de la solution N° 3, l'identifiant de l'association OCCUPER est composé de celui de FONCTION et celui de TRAVAILLER qui, à son tour, est composé de celui de PERSONNE et celui de *ENTREPRISE*. Pour pouvoir instancier l'association OCCUPER dans ce cas, il faut une instance de fonction et une instance de TRAVAILLER. Ce qui fait qu'on est toujours sûr et certain que si on a une instance de l'association OCCUPER c'est que la personne correspondante travaille bel et bien dans l'entreprise en question.

4.7.2. Cas Particuliers

Quand on a une association avec une cardinalité 1,1 d'un côté, l'agrégation de cette association peut être facilement évitée.

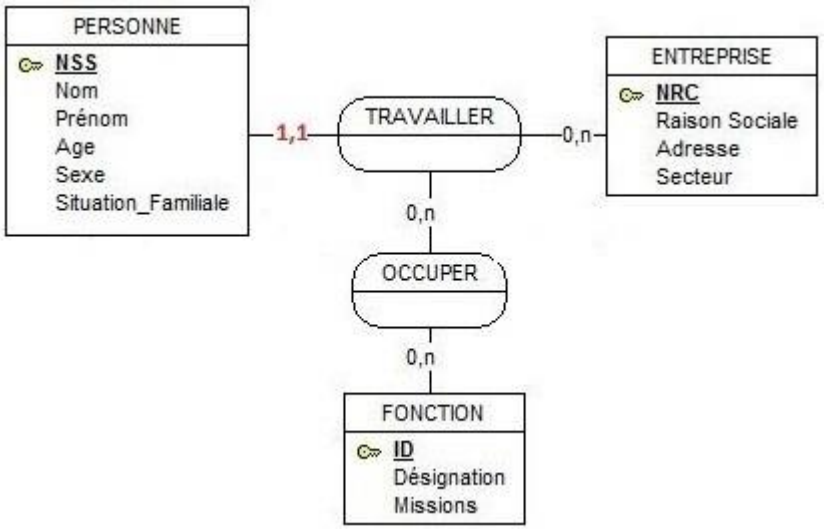


Figure 37 : Agrégation : cas particulier (1.1)

Comme l'existence du type-entité fils (du coté 1,1) est conditionné par l'existence du type-entité père, l'association entre les deux peut être remplacée par ce type-entité dans l'agrégation.

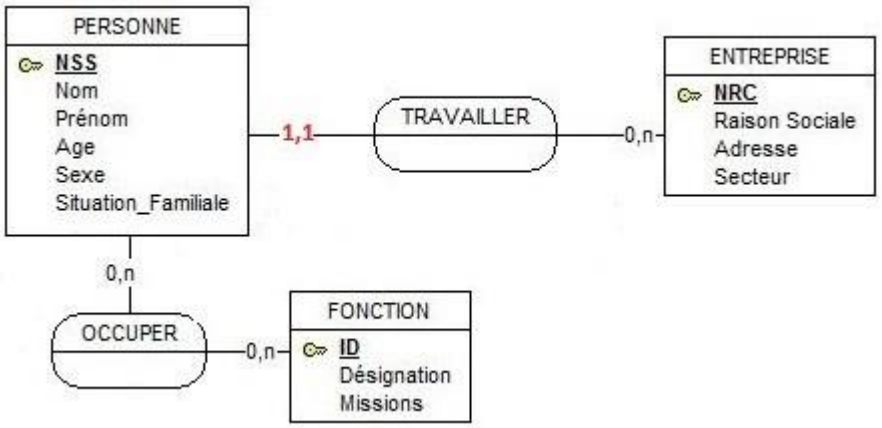


Figure 38 : Correction du cas particulier 1.1

REGLE

Quand l'une des pattes de l'association qu'on veut agréger à la cardinalité 1.1, on enlève l'agrégation et toutes les associations auxquelles elle participe seront liées à l'entité se trouvant du côté de la cardinalité 1.1 (le fils).

Quand la cardinalité minimale dans une association avec une agrégation du côté de l'association agrégée est un 1 signifiant que l'association agrégée ne peut exister que si l'association à laquelle participe l'agrégation existe. L'agrégation n'a pas lieu d'être car ceci signifie que l'entité liée à l'agrégation n'est en réalité qu'un participant de l'association agrégée et que les deux associations (agrégée et externe) ne forment qu'une seule association.

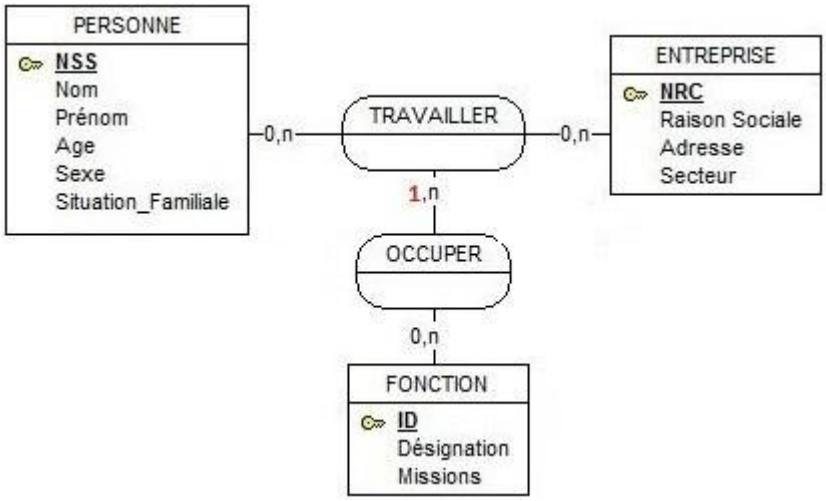


Figure 39 : Agrégation, cas particulier 2

On enlève donc l'agrégation et on modélise avec une simple association ternaire.

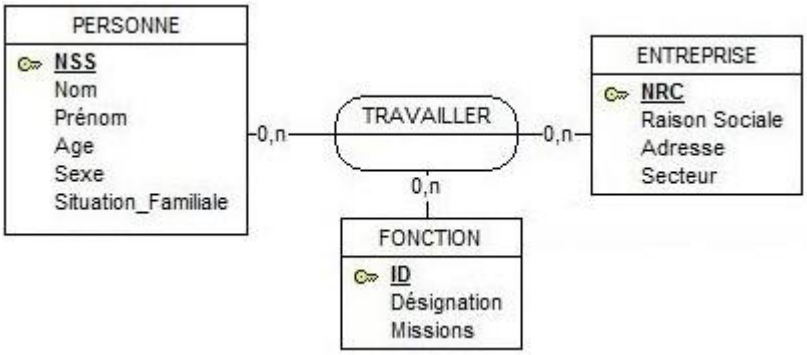


Figure 40 : Agrégation, Solution au cas particulier 2

4.8. L'Héritage

4.8.1. Présentation

En prenant comme exemple la gestion des ressources humaines de notre école, on retrouve des enseignants, des étudiants et des employés. A première vue, les trois entités sont toutes des personnes partageant plusieurs propriétés. Quand on modélise cette situation, on a le choix entre deux possibilités : - L'utilisation d'une seule entité personne qui va regrouper les enseignants, les étudiants et les employés et pour faire la distinction on rajoute un attribut fictif dans ce sens. – l'utilisation de trois entités distinctes.

La première approche n'est pas très proche de la réalité car on a jumelé trois entités en une seule entité. Ceci va cacher un niveau de la réalité d'un côté et d'un autre côté peut créer des confusions. Si par exemple, seuls les enseignants peuvent enseigner, avec cette modélisation toute entité de personne pourra enseigner, ce qui n'est ne représente pas la réalité.

La seconde approche est plus proche de la réalité du point de vue sémantique. Cependant, on va risquer beaucoup de redondances. Premièrement dans les propriétés communes qui vont se répéter pour les trois entités et deuxièmement au niveau des associations. Les trois entités par exemple peuvent effectuer des missions et des déplacements dans le cadre de leurs activités, pour cela l'association déplacement sera dupliquée pour les trois entités également.

Un des concepts supplémentaires de l'entité association qui permet la modélisation d'une telle situation est l'héritage (Généralisation/Spécialisation). Et la figure ci-dessous l'illustre.

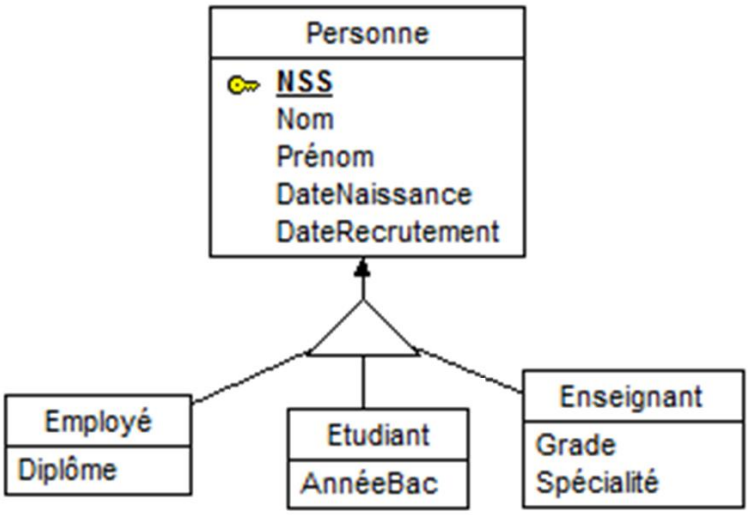


Figure 41 : Héritage

L'entité Personne est dite entité générique. Employé, Etudiant et Enseignant sont des entités générées. Les entités générées héritent de toutes les propriétés de l'entité générique y compris l'identifiant. C'est pour cela que l'on ne représente au niveau des entités générées que les attributs spécifiques.

La généralisation est le fait de regrouper différents types-entité en faisant abstraction de leurs différences par mise en facteur leurs attributs communs, générant ainsi un type entité générique.

> ***Définition***
> *Le processus de généralisation est un processus d'abstraction consistant à généraliser les entités, et les ensembles d'entités, en un seul ensemble ascendant.*

La spécialisation pour un type entité donné c'est la définition de sous-types en mettant en évidence leurs particularités que ce soit en termes d'attributs où d'associations.

> ***Définition***
> *Les sous-ensembles d'entités dans une hiérarchie de généralisation résultent du processus de spécialisation.*

4.8.2. Portée des associations

Quand on utilise l'héritage, les associations qui sont liées à l'entité générique n'ont pas la même signification ni la même interprétation que celles liées aux entités générées. Prenant l'exemple ci-dessous.

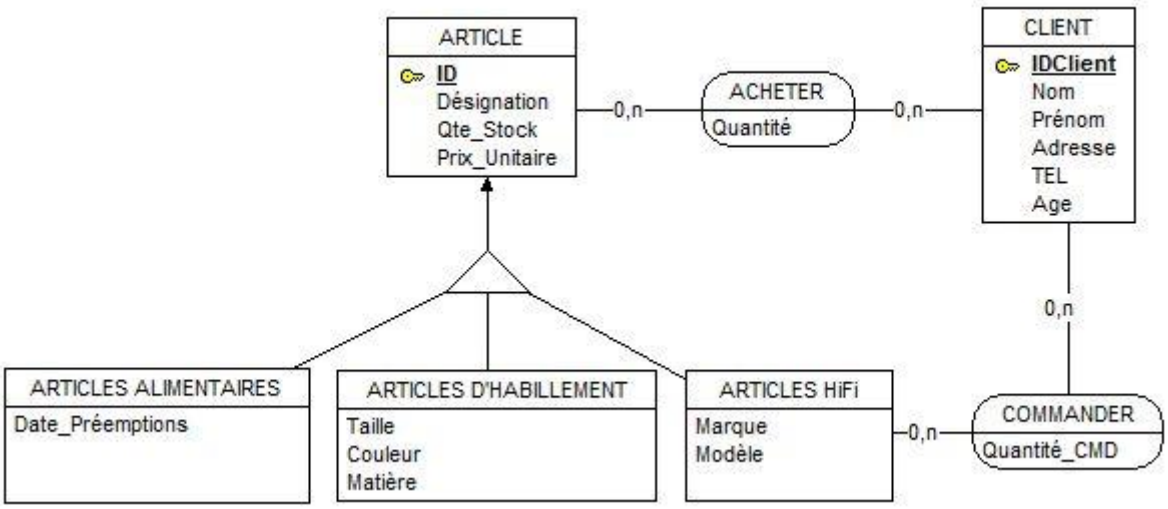


Figure 42 : Portée des associations dans le cadre de l'héritage

Dans ce cas, l'association ACHETER qui est liée à l'entité générique ARTICLE s'applique à toutes les entités générées. Un client achète donc tout type d'articles, qu'ils soient HiFI, d'habillement ou alimentaires. Alors que l'association COMMANDER qui est liée à l'entité ARTICLES HiFI spécifie uniquement ces articles. Un client ne peut donc commander des articles d'habillement ni alimentaires.

Toutes les actions qui sont modélisées au niveau de l'entité générique sont automatiquement répercutées sur les entités générées et l'inverse n'est pas vrai. Les associations qui concernent une entité générée ne peuvent être généralisées à l'entité générique.

REGLE

L'identifiant des entités générées est le même que celui de l'entité générique. Il est donc interdit de le représenter au niveau du modèle ni de définir un autre identifiant pour une entité générée.

5. Les Contraintes d'intégrité

5.1. Concept de contrainte d'intégrité

Les concepts du modèle entité association vus jusqu'ici sont suffisants pour représenter n'importe quelle situation de la réalité. Cependant, certaines contraintes liées aux occurrences des entités et des associations ne peuvent être exprimées directement sur le modèle.

Les contraintes sont appliquées pour restreindre le nombre d'occurrences, elles sont donc considérées comme des filtres qui traduisent des règles de la réalité modélisée.

Définition
Une contrainte d'intégrité (C.I.) est une propriété non représentée par les concepts de base du modèle E.A que doivent satisfaire les données appartenant à la base de données.

Une base de données est cohérente si toutes les contraintes d'intégrité définies sont respectées par les valeurs de la base.

On définit les contraintes d'intégrité pour spécifier des propriétés sémantiques du réel perçu qui ne sont pas exprimables avec le modèle E.A. Leur application permet de limiter les occurrences possibles des structures d'information.

5.2. Types de contraintes d'intégrité

On distingue deux types de contraintes selon le moment où elles doivent être vérifiées : Les contraintes statiques et les contraintes dynamiques.

5.2.1. Contraintes statiques

Les contraintes dites statiques sont des contraintes qui doivent être vérifiées à tout moment (Au moment de l'instanciation, de la mise à jour et de la suppression). Ce sont généralement des contraintes définies sur les valeurs des attributs ou des cardinalités.

Définition
Une contrainte statique est une propriété qui doit être vérifiée à tout moment

On peut définir une contrainte statique sur l'âge d'une personne en disant par exemple que l'âge doit être supérieur à 18 ans. Ce qui signifie qu'à n'importe quel moment on ne peut avoir au niveau de notre modèle une instance de l'entité Personne qui ne respecte pas cette contrainte. Elle est vérifiée au moment de l'instanciation de l'entité et au moment de la modification de l'attribut âge ou date de naissance.

5.2.2. Contraintes dynamiques

Les contraintes dynamiques interviennent lors du changement dans l'état du modèle. Ce qui signifie, au moment de la modification des valeurs des attributs des entités ou des associations mais ne sont pas vérifiées au moment de l'instanciation.

Définition
Une contrainte dynamique est une propriété que doit respecter tout changement d'état de la BDD.

On peut, par exemple, définir une contrainte dynamique sur la situation familiale d'une personne en disant qu'une personne mariée ne peut devenir que divorcée ou veuve mais jamais célibataire. Cette contrainte n'est pas vérifiée au moment de la création de l'entité personne mais uniquement au moment de la modification de son attribut Situation Familiale.

5.3. Formulation des contraintes

Les contraintes d'intégrité doivent être décrites explicitement (avec un langage approprié tel que la logique du premier ordre) si elles ne peuvent pas être décrites avec les concepts du modèle de données.

Pour des raisons de compréhension et d'universalité du modèle, il est interdit de les exprimer en langage naturel et ceci quel que soit le niveau de détail exprimé.

5.3.1. Un formalisme inspiré de la logique du premier ordre

Les types entité et les types association sont considérés comme des ensembles, les entités et les associations sont leurs éléments respectifs. On peut appliquer la logique du premier ordre pour exprimer certaines conditions et contraintes sur les éléments de ces ensembles.

Soit l'exemple suivant décrivant une association de mariage entre deux personnes.

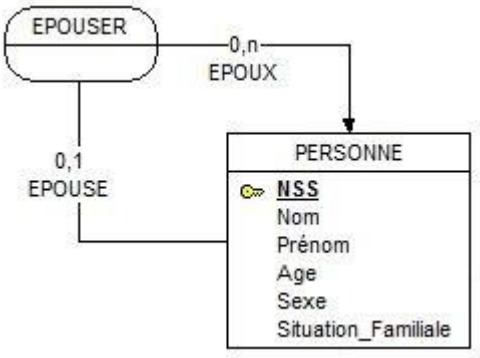


Figure 43 : Contraintes formulées en logique mathématique

Supposons que ce modèle décrit la base de données de l'Etat Civil Algérien. Tel qu'il est défini, le modèle peut admettre plusieurs situations qui pourraient contredire la réalité modélisée.

Supposons que nous ayant recensé les conditions suivantes lors de l'étude :

1. Un Mariage n'est admis qu'entre une personne de sexe masculin et une personne de sexe féminin.
2. Dans une occurrence de l'association EPOUSER, la personne participante avec le rôle EPOUX doit être de sexe masculin et l'autre de sexe Féminin.
3. Toutes personne participante à une association EPOUSER doit avoir un état civil différent de Célibataire
4. L'état civil d'une personne évolue dans le ses suivant : {Célibataire} → {Marié}. {Marié} → {Divorcé, veuf}. {Divorcé, veuf} → {Marié}

On va s'arrêter à ces quatre contraintes. On remarque qu'aucune d'elles n'est respectée par le modèle car il n'existe aucun moyen graphique qui permet de les exprimer directement

sur le modèle. On va donc associer un certain ensemble de règles écrites en logique mathématique qui vont accompagner le modèle.

1. ∀ x,y ∈ Personne, (x,y) ∈ Mariage ⇒ x.Sexe <> y.Sexe
2. ∀ x,y ∈ Personne, (x:EPOUX , y:EPOUSE)∈ Mariage ⇒ x.Sexe = MASCULIN ∧ y.Sexe = FEMININ
3. ∀ x,y ∈ Personne, (x,y) ∈ Mariage ⇒ x.état-civil <> CELIBATAIRE ∧ y.état-civil <> CELIBATAIRE
4. ∀ x ∈ Personne, ∀ t1, t2 ∈ Temps : t1< t2, x(t1).etat-civil=CELIBATAIRE ⇒ x(t2).etat-civil <> DIVORCE ∧ x(t2).état-civil <> VEUF
5. ∀ x ∈ Personne, ∀ t1, t2 ∈ Temps : t1< t2, x(t1).etat-civil=MARIE ⇒ x(t2).etat-civil <> CELIBATAIRE
6. ∀ x ∈ Personne, ∀ t1, t2 ∈ Temps : t1< t2, x(t1).etat-civil=DIVORCE ⇒ x(t2).etat-civil <> CELIBATAIRE ∧ x(t2).etat-civil <> VEUF
7. ∀ x ∈ Personne, ∀ t1, t2 ∈ Temps : t1< t2, x(t1).etat-civil=VEUF ⇒ x(t2).etat-civil <> CELIBATAIRE ∧ x(t2).etat-civil <> DIVORCE

5.3.2. Représentation graphique sur le modèle

Les contraintes d'intégrité exprimées sur les occurrences des entités et des associations ont toutes une représentation graphique directement exprimable sur le modèle.

5.4. Les Contraintes D'intégrité

Les contraintes d'intégrité peuvent être définies sur les attributs, les cardinalités, les entités et les associations. Nous allons présenter les différentes manières de les définir et de les représenter au niveau du modèle.

Certaines contraintes peuvent être représentées de manière graphique. Par contre, toutes les contraintes quelle qu'elles soient peuvent être représentées avec la logique mathématique, y compris, les cardinalités qui sont considérées comme étant le premier type de contraintes introduites par le modèle EA.

5.4.1. Contraintes d'intégrité sur les attributs

Ce type de contraintes permet de réduire les valeurs possibles à un attribut donné. Elles sont donc définies sur les valeurs des attributs et non pas les attributs eux-mêmes.

On peut définir trois types de contraintes sur les attributs :

- **Domaine de définition :** Le domaine de définition de l'attribut permet de filtrer les valeurs qu'on peut donner à un attribut. Quand on définit un entier on ne peut donner que des valeurs numériques entières, toute autre valeur sera rejetée.

- **Valeur Obligatoire ou facultative :** Si on dit qu'un attribut est à valeur obligatoire, ceci signifie qu'on ne peut instancier l'entité ou l'association le contenant que si ce dernier possède une valeur.
- **Contrainte sur les valeurs de l'attribut :** On peut définir des contraintes sur les valeurs de l'attribut qui peuvent être par rapport à une valeur ou par rapport à un autre attribut. Par exemple, on définit cette contrainte : Âge entre 1 et 100 ; Date de naissance < date de mariage

NOTE

Les contraintes d'intégrité définies sur les attributs sont exprimées avec la logique mathématique au niveau de la documentation du modèle entité/association.

5.4.2. Contraintes d'intégrité sur les cardinalités

Les cardinalités elles-mêmes sont considérées comme des contraintes très importantes dans le modèle conceptuel. Mais il peut également exister des contraintes entre les cardinalités définies pour les différents rôles que jouent une entité dans une ou plusieurs associations distinctes, par exemple : Nombre d'enfants d'un parent = nombre d'occurrences de « est parent de » qui lient ce Parent à ces enfants.

Comme au niveau du modèle on n'a que les valeurs 1 et n pour la cardinalité maximale et les valeurs 0 et 1 pour la cardinalité minimale. Supposons que l'on soit confronté à une situation où le nombre maximum est bien défini, ce n'est pas n mais 5 par exemple. Il nous faut à ce niveau ajouter une contrainte sur les cardinalités déjà définies afin de limiter le nombre maximum de participations de l'entité, une contrainte que l'on exprime généralement avec la logique mathématique.

5.4.3. Contraintes sur les entités / association :

Des contraintes peuvent être définies pour les entités comme pour les associations. Car les deux concepts peuvent être définis comme étant des ensembles d'éléments de même nature.

Avant de définir ces différentes contraintes possibles sur les entités et les associations, on définit d'abord deux concepts très importants sur lesquels seront fondées les contraintes que nous allons présenter. Il s'agit de la couverture et la disjonction.

Définition

La couverture de plusieurs ensembles est l'ensemble recouvrant tous les éléments appartenant à ces ensembles.

Définition
Deux ensembles forment une disjonction (sont disjoints) si et seulement s'il n'existe pas d'éléments communs entre eux.

Quand on définit une contrainte sur des entités, il faut qu'elles soient de même nature. Ce qui signifie que dans le cadre des entités on ne peut définir de contraintes entre deux entités que si ces dernières ont la même définition (i.e. : le même identifiant). Et ce cas ne peut exister que dans le cadre d'une généralisation/spécialisation. Donc, les contraintes sur les entités n'existent que pour les entités générées d'une généralisation/spécialisation.

Le même principe est défini pour les associations. Il faut que les deux associations sur lesquelles portent les contraintes soient de même nature (même identifiant).

5.4.3.1. Partition

Soit le modèle ci-dessous représentant une situation modélisant les artistes d'un organisme gérant les droits d'auteurs. Dans cet exemple une spécialisation est faite sur ces artistes et les différentes entités générées à partir de l'artiste ne sont pas traitées de la même manière.

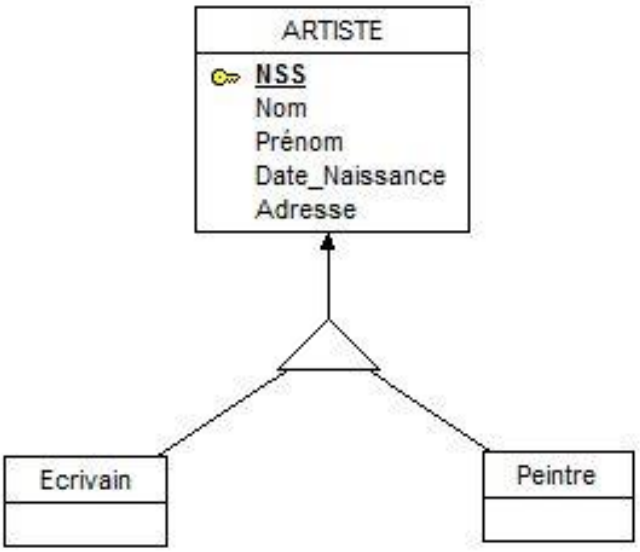


Figure 44 : La partition

Dans le cas d'une généralisation/spécialisation, les occurrences de ARTISTE peuvent être des PEINTRES, des ÉCRIVAINS ou autre type d'artistes. La contrainte de partition va nous permettre d'identifier de manière très précise les frontières entre ces ensembles.

Définition
Partition = Disjonction + Couverture

Ce qui peut être traduit par la phrase suivante : ***L'un ou l'autre mais pas les deux et pas autre chose.***

Dans l'exemple ci-dessus, la partition est interprétée de la manière suivante : L'ensemble des artistes sont soit des écrivains soit des peintres. Ils ne peuvent être autre chose et un peintre ne peut être un écrivain et vis-versa.

Une partition est représentée de manière graphique sur le modèle. Il existe plusieurs représentations, mais nous gardons celle utilisée par le logiciel Win'Design. Au niveau de la généralisation/spécialisation, les contraintes sont directement spécifiées sur l'association ISA.

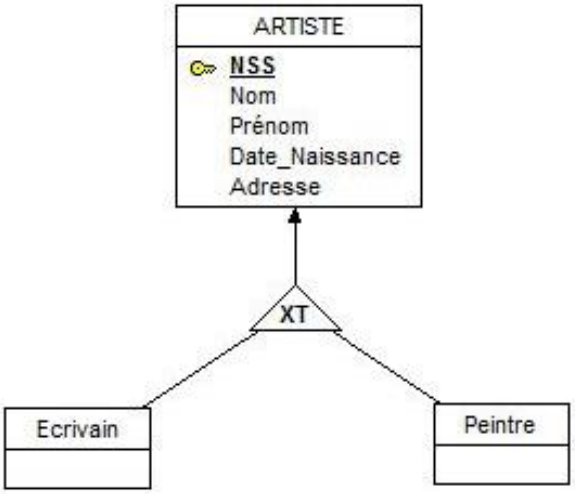


Figure 45 : Représentation graphique de la partition

5.4.3.2. Totalité

En plus de l'exemple précédent, nous considérons une relation plurielle entre l'écrivain et le Livre.

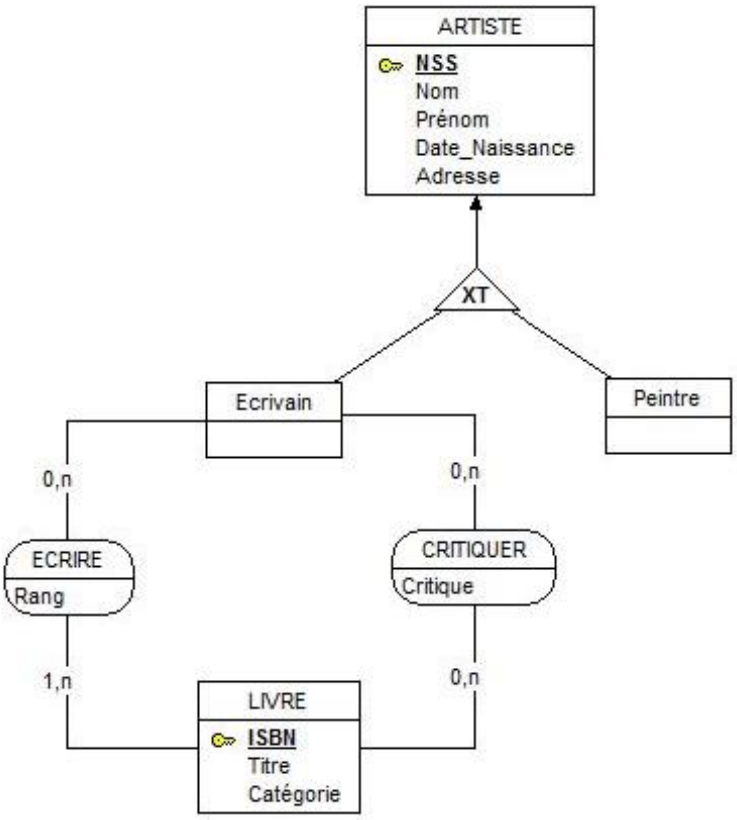


Figure 46 : Totalité

Définition
Totalité = ¬disjonction + couverture

Ce qui peut être traduit par la phrase suivante :

L'un ou l'autre ou les deux en même temps mais pas autre chose

En appliquant la totalité aux entités Peintre et Ecrivain, cela signifie qu'un artiste peut être un écrivain ou un peintre ou les deux en même temps mais pas autre chose.

En appliquant la totalité aux deux associations Ecrire et Critique (On a le droit de le faire car les deux associations ont le même identifiant). Cela signifie qu'un livre peut être écrit par un auteur ou critiqué ou les deux choses à la fois mais pas autre chose

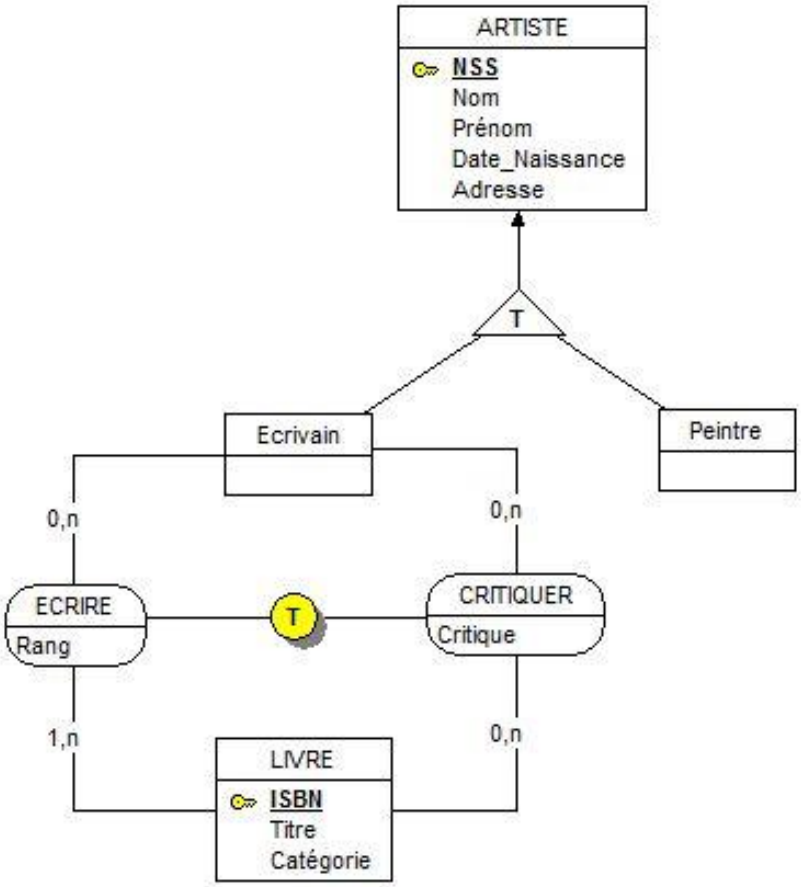


Figure 47 : Représentation graphique de la partition

Par contre si on applique la partition à ce niveau, cela signifie qu'un auteur peut soit écrire soit critiquer un livre donné.

5.4.3.3. Exclusion

Définition

Exclusion = disjonction + ¬couverture

Ce qui peut être traduit par la phrase suivante :

L'un ou l'autre ou autre chose mais pas les deux en même temps.

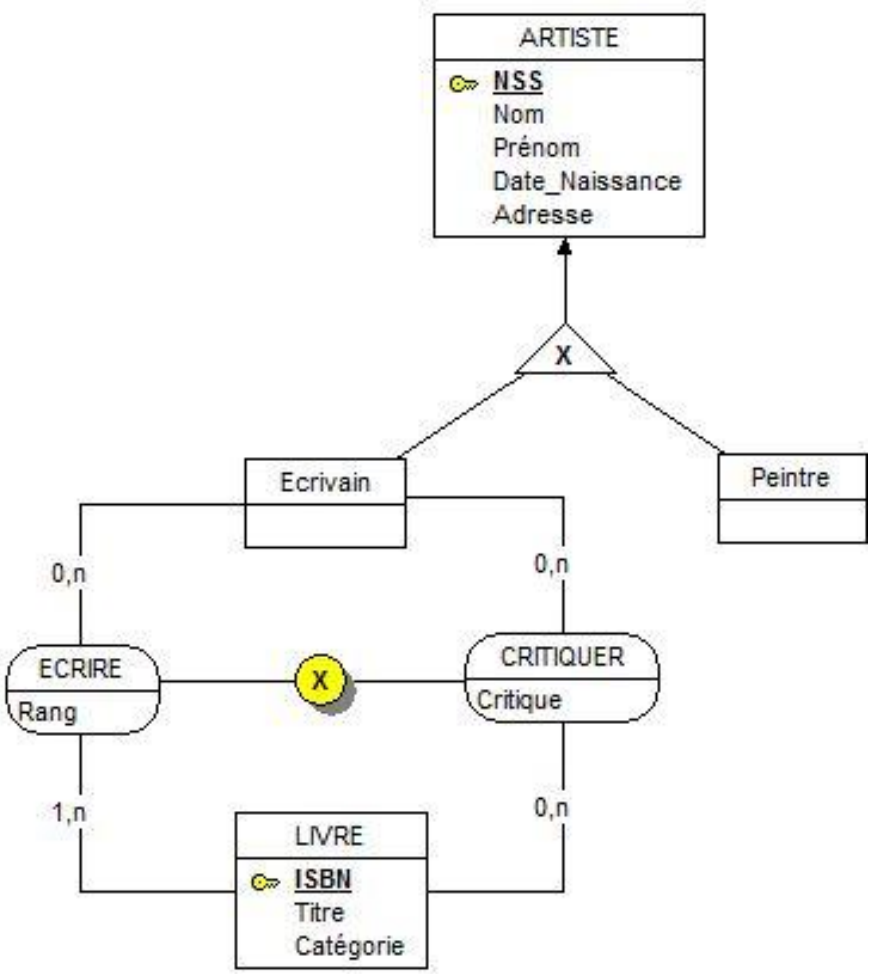


Figure 48 : Exclusion

L'exclusion appliquée aux entités Peintre et Ecrivain signifie qu'un artiste peut être un peintre ou un écrivain mais pas les deux en même temps. Cependant, il peut être autre chose que ces deux-là.

Au niveau des associations, cela signifie qu'un auteur peut être lié avec un livre via une relation d'écriture ou de critique mais pas les deux en même temps. En d'autres termes, un auteur n'a pas le droit de critiquer ses propres livres.

5.4.3.4. Inclusion

Pour l'inclusion, on va prendre le même exemple que tout à l'heure mais avec une association plurielle entre peintre et toile. Un peintre peint des toiles et les expose.

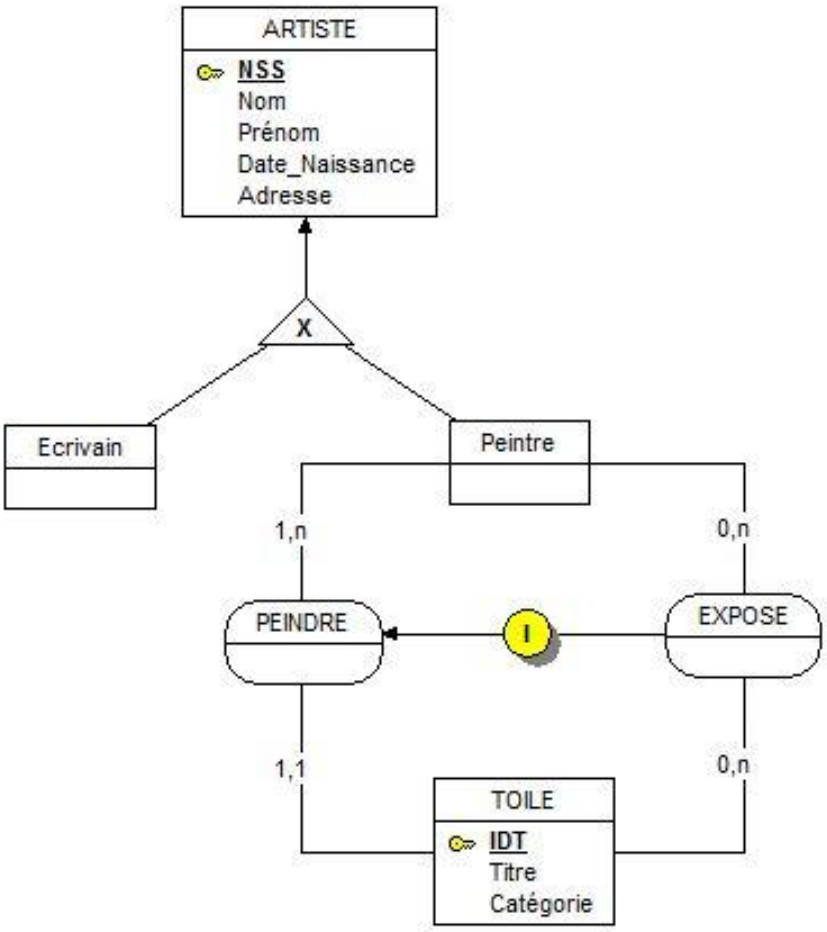


Figure 49 : Inclusion

Définition

Inclusion = Tous les éléments du premier ensemble sont inclus dans le second ensemble.

Dans l'exemple ci-dessus, l'inclusion qui est orientée de Exposer vers Peindre signifie que toutes les occurrences de EXPOSER doivent être des occurrences dans Peindre donc tous les couples (NSS, IDT) de l'association Exposer doivent être inclus dans l'ensembles des couples (NSS, IDT) de l'association Peindre. En d'autres termes, cela signifie qu'un peintre ne peut exposer que les toiles qu'il a lui-même peintes.

5.4.3.5. Unicité

Considérons l'exemple que nous avons présenté au niveau de la section : Conservation de l'historique représentant une association ternaire entre Employé, Fonction et Date.

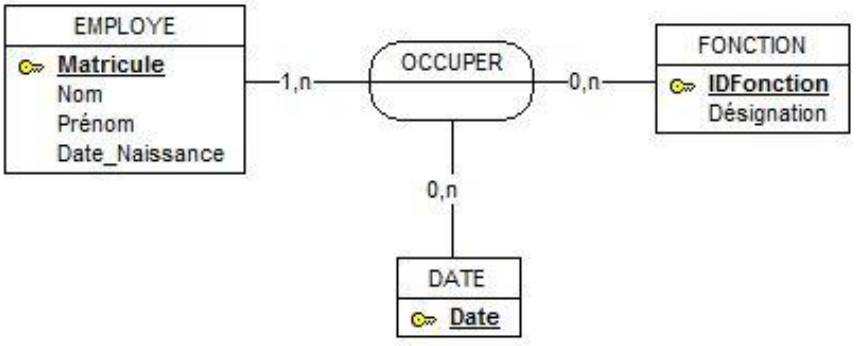


Figure 50 : Contrainte d'unicité

A ce niveau, l'identifiant de l'association Occuper est composé de Matricule, IDFonction et Date. A chaque fois qu'on change la valeur de l'un de ces trois attributs on obtient un nouveau tuple, donc une nouvelle occurrence de OCCUPER. Cependant, si on veut exprimer le fait qu'à une date donnée, un employé quelconque ne peut occuper qu'une et une seule fonction, il nous faut une contrainte sur cette association.

Définition

L'unicité appelée également dépendance fonctionnelle ou contrainte d'intégrité fonctionnelle, exprime le fait qu'en connaissant la valeur d'un objet on peut être en mesure de déduire une et une seule valeur de l'objet dépendant du premier.

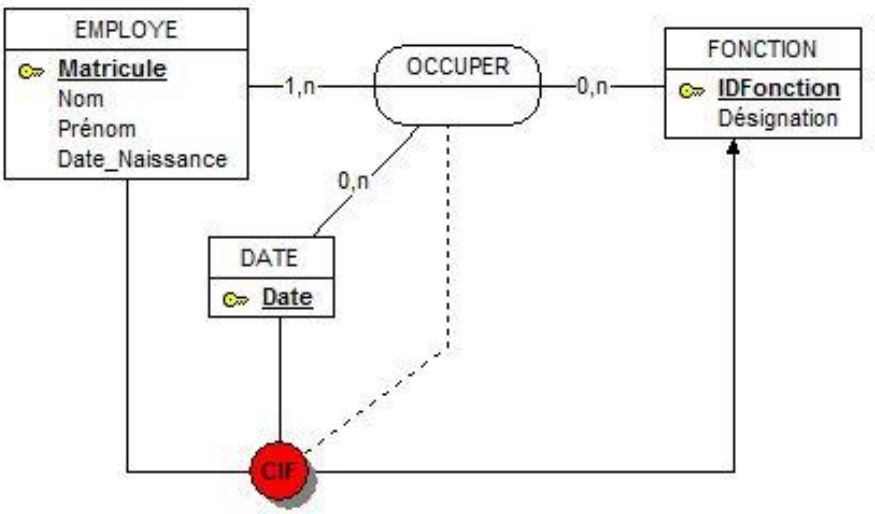


Figure 51 : Représentation graphique de l'unicité

La contrainte d'unicité dans l'exemple ci-dessus signifie que pour un employé donné à une date donnée je n'ai qu'une et une seule fonction dans le cadre de l'association OCCUPER.

5.5. Les Participants À Une Contrainte

Les contraintes entre entités ou entre associations ne sont possibles que si les deux entités ou les deux associations sons de même nature. Ce qui signifie que pour les entités, le seul cas où les contraintes sont possibles c'est dans le cas de la généralisation/spécialisation. Dans le cas des associations, il faut vérifier que les deux associations ont le même type d'occurrences. Ceci est possible dans le cas des associations plurielles. Mais quand on veut mettre une contrainte entre deux associations qui n'ont pas les mêmes types d'occurrences mais juste quelques éléments en commun, ces derniers doivent être mentionnés avec des pointillés.

L'exemple suivant montre clairement l'importance et l'influence de l'utilisation des pointillés sur le sens de la contrainte. On modélise des bûcherons chargés de couper des branches d'arbre bien identifiées. Ces bûcherons ont la possibilité de s'asseoir sur des branches pour faire leur travail.

Le modèle ci-dessous exprime cette situation

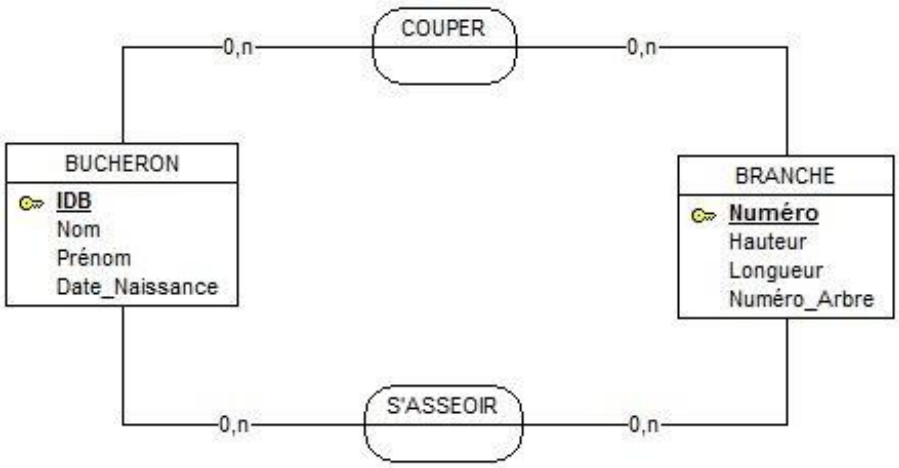


Figure 52 : Participants à une contrainte

Nous allons présenter plusieurs contraintes et pour chacune nous donnerons sa signification et son impact.

Premier Cas : On ne scie pas assis

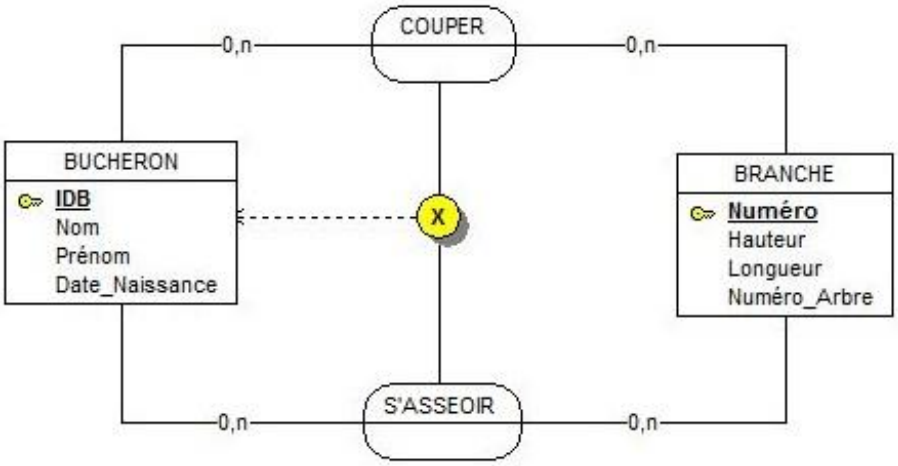


Figure 53 : Exemple 1 des participants

En mettant les pointillés du côté du Bûcheron cela signifie que seul ce dernier participe à cette contrainte et ceci quelle que soit la branche. Donc, au lieu de vérifier l'identifiant complet de l'association COUPER avec celui de l'association S'ASSEOIR, on vérifie uniquement l'attribut IDB et on applique l'exclusion sur lui.

Dans ce cas l'exclusion signifie qu'un bûcheron qui s'assoie n'a pas le droit de couper une branche et celui qui coupe ne peut pas s'asseoir.

Deuxième Cas : On ne scie pas une branche sur laquelle on est assis

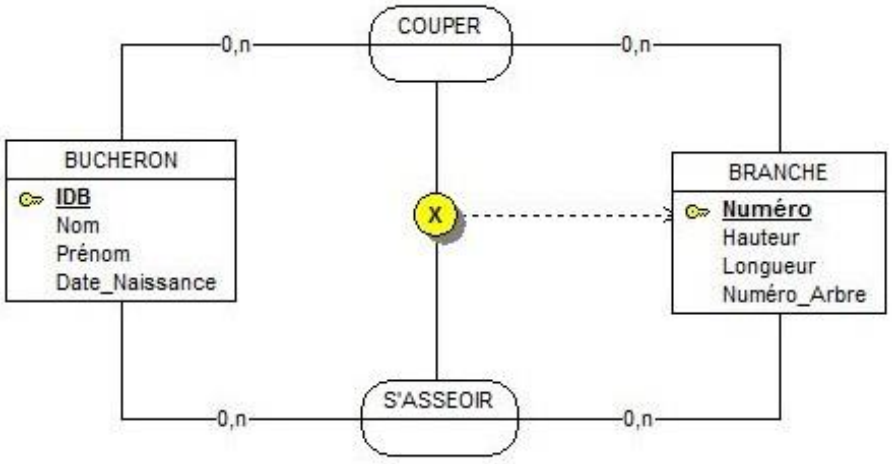


Figure 54 : Exemple 2 des participants

Une branche sur laquelle quelqu'un est assis ne peut être coupée et on ne peut s'asseoir sur une branche qu'on est en train de couper.

Troisième Cas : On ne scie pas une branche sur laquelle on est soi-même assis

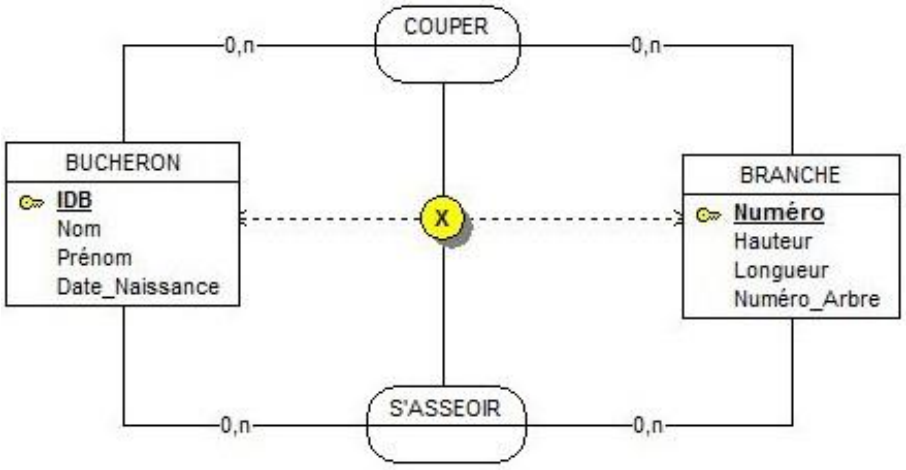


Figure 55 : Exemple 3 des participants

C'est la même chose que si on n'avait mis aucune orientation (Pointillés). Désigner les deux participants possibles ou ne rien désigner donne la même contrainte. Ce qui signifie qu'on prend tout l'identifiant des deux associations. C'est à dire qu'un bûcheron ne peut s'asseoir sur une branche et la scier en même temps.

Quatrième Cas : On ne scie pas assis et on ne scie pas une branche sur laquelle on est assis

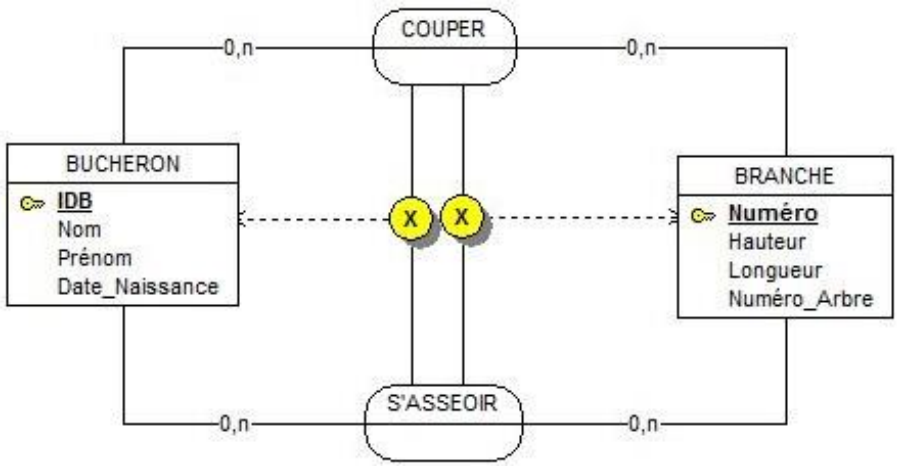


Figure 56 : Exemple 4 des participants

Ce dernier cas permet de jumeler les deux premiers cas. C'est une double exclusion prise séparément et n'a pas la même signification que celle qu'on vient de présenter.

Dans ce cas, les deux contraintes sont complètement indépendantes, la première est appliquée au bûcheron et la seconde à la branche.

Reprenons à présent, l'exemple de la cardinalité minimale 1 où on a trouvé une difficulté à séparer les dates de rangement et de proposition des articles sans pour cela permettre de garder toutes les dates possibles.

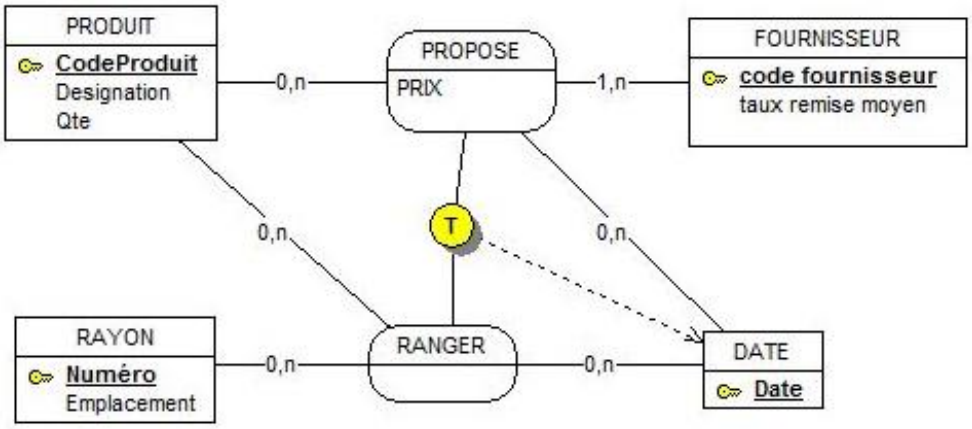


Figure 57 : Participant dans le cas d'une contrainte entre association

Nous définissons une contrainte de totalité entre les deux associations PROPOSER et RANGER. Cependant, comme ces deux associations ne sont pas de même nature, on spécifie l'entité date comme participant. Ceci signifie que lors de la vérification de la contrainte, seule la date est prise en compte.

Cette contrainte de totalité va faire en sorte qu'une occurrence de date peut être une date de proposition, de rangement, les deux en même temps mais pas autre chose. Cela nous permettra de ne garder que les dates qui nous sont utiles.

6. Le Schéma Conceptuel

Quand on fait la modélisation d'une base de données, le résultat de notre travail est généralement un rapport conceptuel assez clair et détaillé qui peut permettre aux DBA de créer facilement la base de données et aux développeurs de développer des applications dessus.

Il faut donc que notre modèle soit bien documenté pour ne pas laisser lieu à des confusions ou des manques d'information.

> ***Définition***
> *Un schéma conceptuel entité association est un ensemble de descriptions de types d'entité et de types d'association (avec leurs attributs et les liens de généralisation entre types d'entité), et des contraintes d'intégrité (CI) associées : SEA = ({TE}, {TA}, {CI})*

D'un autre côté, il faut que l'on valide notre schéma conceptuel avant sa finalisation et la rédaction du document final. La validation consiste à vérifier toutes les règles syntaxiques et sémantiques qu'on a vu tout au long de ce chapitre. Il s'agit de vérifier si le schéma que nous avons produit correspond réellement aux spécifications de départ et si ce schéma peut répondre à tous les besoins en information recensés durant la première phase de conception.

Cependant, avant de passer à la validation du modèle, on effectue d'abord une optimisation de ce dernier en suivant un ensemble de règles (Best Practices) qui vont nous

permettre d'obtenir un modèle optimal. Certaines de ces règles sont obligatoires alors que d'autres sont laissées à l'appréciation du concepteur.

6.1. Optimisation Du Modèle

Nous présentons dans ce qui suit quelques règles de conception en plus de celles que nous avons citées tout au long des différentes sections de ce chapitre.

6.1.1. Règles de conception et d'optimisation

REGLE 1

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

Il ne faut pas oublier que chaque objet de la réalité a une seule représentation au niveau du modèle. Les objets du modèle sont identifiés grâce à leur nom. Il est donc important que ce dernier soit unique.

Ce qui est à éviter dans ce cas, c'est la représentation multiple du même objet de la réalité avec plusieurs objets au niveau du modèle même-si leurs noms sont différents.

L'exemple ci-dessous est faux du point de vue sémantique.

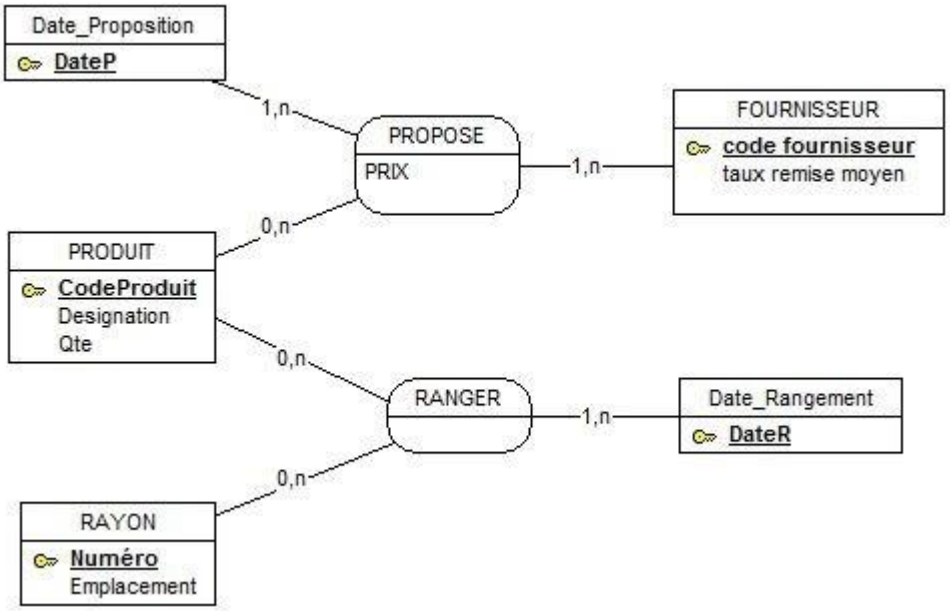


Figure 58 : Représentation du même objet réel avec plusieurs entités

L'objet date de la réalité ne peut être représenté que par une seule entité quel que soit son nom. Le fait de mettre plusieurs entités Date dans ce modèle est donc faux.

REGLE 2

Il faut remplacer un attribut multiple par un type-association et un type-entité supplémentaires.

Un attribut multiple est un attribut qui peut avoir une valeur qui est composée de plusieurs valeurs de même type. Par exemple la liste des enfants d'une personne ne doit pas être représentée par un attribut car ce dernier est multiple.

REGLE 3

Il ne faut jamais ajouter un attribut dérivé d'autres attributs qu'ils se trouvent dans le même type-entité ou non.

Un attribut dérivé est un attribut dont la valeur peut être déduite à partir des valeurs d'autres attributs. Par exemple l'âge d'une personne qu'on peut déduire à partir de sa date de naissance ou le nombre d'enfant dans une association parentale car le nombre de participations du parent à l'association représente le nombre de ses enfants.

REGLE 4

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

Un attribut dont les valeurs représente une liste de valeurs prédéfinie. Qu'elle soit exhaustive ou non, il est préférable d'utiliser une entité à la place de l'attribut. Ceci permet une meilleure visibilité et un contrôle sur les données. Cela permet également de réduire la redondance.

REGLE 5

Tout Type Entité remplaçable par un Type Association doit être remplacé

Une entité peut être représentée par une association si premièrement, son identifiant est fictif, ce qui signifie qu'il ne représente pas une propriété réelle. Et deuxièmement, il n'est lié avec d'autre entités que par des cardinalités 1,1.

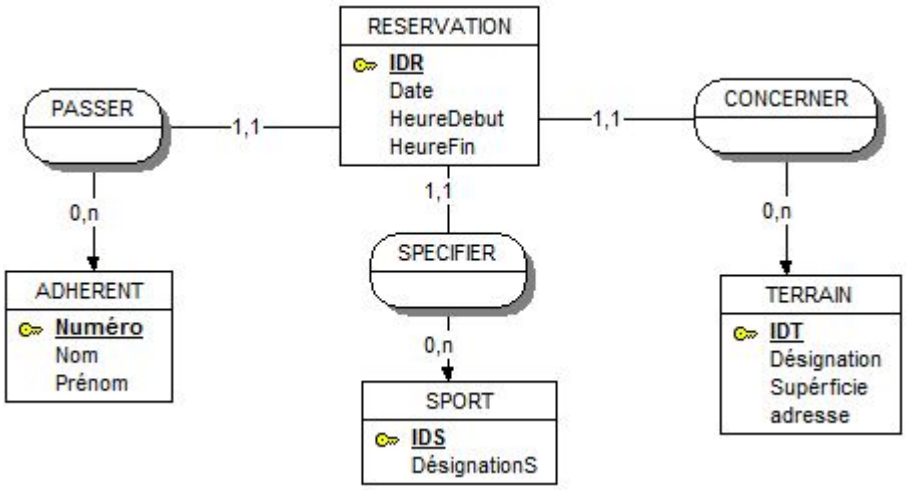


Figure 59 : Utilisation des entités à la place d'entités avec identifiant fictif

L'exemple ci-dessus modélise une réservation par une entité avec un identifiant fictif. On peut donc remplacer cette entité avec une association comme le montre la figure ci-dessous et cette solution est préférée à la première.

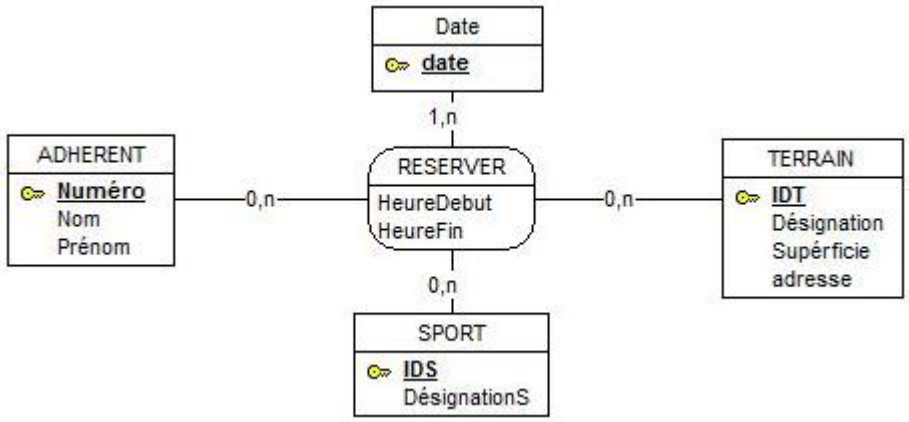


Figure 60 : Utilisation des entités à la place d'entités avec identifiant fictif

REGLE 6

Lorsque les cardinalités d'un TA sont toutes 1,1 c'est que le TA n'a pas lieu d'être.

Il est impossible de trouver un cas où deux entités sont liées via une association avec des cardinalités 1.1 de chaque côté. Car l'existence de cette cardinalité signifie que les deux entités sont complètement dépendantes l'une de l'autre et que finalement elles ne forment qu'une seule entité.

REGLE 7

Il faut éviter les TA redondants. S'il existe deux chemins pour se rendre d'un TE à un autre, il faut supprimer le chemin le plus court puisqu'il est déductible des autres chemins.

Dans l'exemple ci-dessous, l'association AVOIR est en plus car elle peut être obtenue via les deux autres associations.

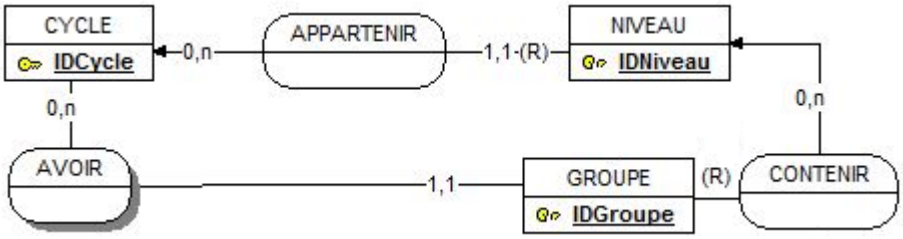


Figure 61 : Suppression du plus court chemin entre deux entités

REGLE 8

Lorsque l'une des cardinalités d'une ternaire est (1,1) ou (0,1), celle-ci est éclatée en deux autres associations à partir de cette patte.

Prenant l'exemple d'un enseignement où un enseignant enseigne un module à un niveau. Dans cet exemple nous avons mis les cardinalités du côté enseignant 0.1 ce qui signifie qu'un enseignant n'est lié qu'à un seul couple (Niveau, Module) via l'association Enseigner.

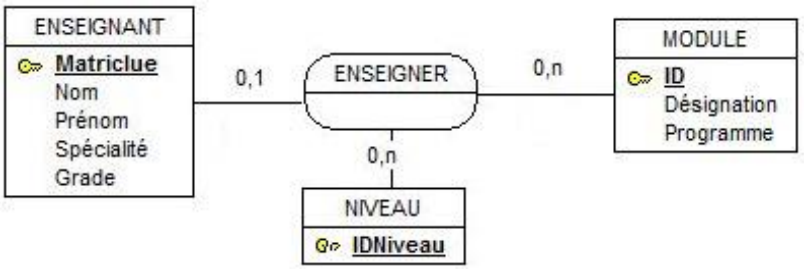


Figure 62 : Cardinalité maximale d'une ternaire

Le fait d'avoir mis cette cardinalité du côté de Enseignant brise le lien qui existait entre Module et Niveau. Car quel que soit le module, un enseignant ne peut être associé qu'à un seul module et quelques soit le module, un enseignant ne peut être associé qu'à un seul niveau.

La ternaire n'a donc aucun sens et il faut la briser de la manière suivante :

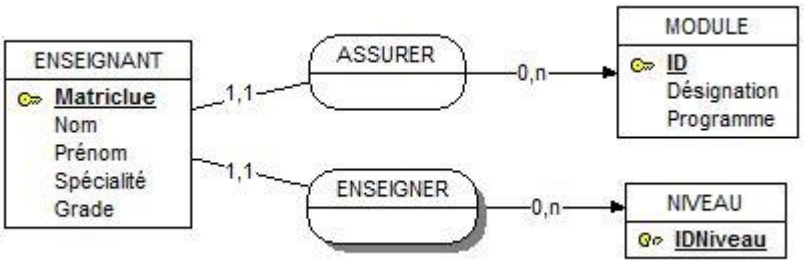


Figure 63 : Cardinalité maximale d'une ternaire

REGLE 9

Toute association n-aire peut être remplacée par une entité en lui attribuant un identifiant fictif et en créant des associations binaires entre la nouvelle entité et toutes les entités de la collection de l'ancienne association n-aire. La cardinalité de chacune des associations binaires créées est 1.1 du côté de l'entité créée et 0,n ou 1,n du côté des entités de la collection de l'ancienne association n-aire.

6.2. Validation Du Modèle

La validation d'un modèle entité/association consiste à vérifier toutes les règles syntaxiques et sémantiques qu'on a vu tout au long de ce chapitre. Il s'agit en premier lieu de

vérifier que le schéma est correct du point de vue des règles de modélisation par l'entité association et en deuxième lieu qu'il correspond réellement aux spécifications de départ.

En d'autres termes, la validation consiste à vérifier que le modèle peut répondre à tous les besoins en information recensés durant la première phase de conception.

6.2.1. Vérification Syntaxique

Il s'agit de vérifier que les règles du modèle entité association sont respectées. Par exemple, chaque entité doit avoir un nom unique et au moins un attribut qui est son identifiant ...etc.

6.2.2. Vérification Sémantique (Validation)

Il s'agit à ce niveau de vérifier que le schéma produit correspond réellement aux spécifications de départ et répond aux besoins des utilisateurs. Il existe plusieurs méthodes de vérification et de validation du modèle.

6.2.2.1. Validation par jeu d'essai

Le concepteur vérifie grâce à une mini base de données que le diagramme permet effectivement de stocker les informations nécessaires à l'entreprise.

6.2.2.2. Vérification de la complétude par rapport aux traitements

Le concepteur vérifie que le diagramme contient tous les types d'information nécessaires à l'exécution des traitements prévus.

6.2.2.3. Validation par les utilisateurs

Il s'agit de présenter le modèle aux utilisateurs, de le discuter et de le valider avec eux.

7. Conclusion

La modélisation des données avec le modèle entité association est une approche naturelle qui permet d'obtenir rapidement des modèles proches de la réalité et faciles à comprendre. Le modèle permet également de modéliser n'importe quelle situation quelle que soit sa complexité.

Cependant, le modèle E/A est un modèle de données et non de traitement. Il ne permet pas de prendre en compte l'aspect dynamique des données, les modalités de déclenchement d'une opération ainsi que la façon dont la modification est réalisée.

Le concepteur éprouve des difficultés à faire des choix parce qu'il ne discerne pas toujours les conséquences des décisions prises. Il y a souvent des ambiguïtés entre entité et

attribut, ce qui pousse parfois à devoir créer des types-entités artificiels. Enfin il manque d'un support formel pour contrôler la qualité du schéma.

Pour terminer, une quarantaine d'années sont passées depuis l'introduction du modèle entité association et il reste toujours utilisé par une large communauté de concepteurs des bases de données. Ce modèle est donc robuste et fiable et ne risque pas disparaitre d'aussitôt.

Chapitre 3 : Le Modèle Relationnel

Objectifs

L'objectif de ce chapitre est d'apprendre à l'étudiant les bases théoriques du modèle relationnel. Il apprendra ainsi les mécanismes de conception d'un schéma relationnel en utilisant les dépendances fonctionnelles ainsi que la théorie de la normalisation et son utilité.

Outils

Functional Dependencies Engine (FDE 1.0) : Un outil pédagogique développé par DAHAK Fouad à l'ESI, implémentant tous les algorithmes concernant le traitement des DF (fermeture transitive, fermeture d'un attribut, couverture minimale, détection de clé primaire...) et ceux de la normalisation des relations (décomposition et synthèse). Téléchargeable sur http://dahak.esi.dz

Résumé

Ce chapitre présente dans le détail le modèle relationnel proposé par Codd. On part ainsi des concepts de base du modèle, on présente par la suite les règles de passage de l'Entité/Association vers le relationnel et nous abordons en dernier lieux la théorie de la normalisation qui permet la conception et l'optimisation d'un schéma relationnel en utilisant les dépendances fonctionnelles.

1. Introduction

Le modèle relationnel a été défini par E.F Codd dans les années 70 et de nombreux chercheurs ont contribué à son développement. Les premiers SGBD bâtis sur ce modèle ont été SQL/DS et DB2 de IBM, d'où est né le langage de manipulation de bases relationnelles, SQL (Structured Query Language).

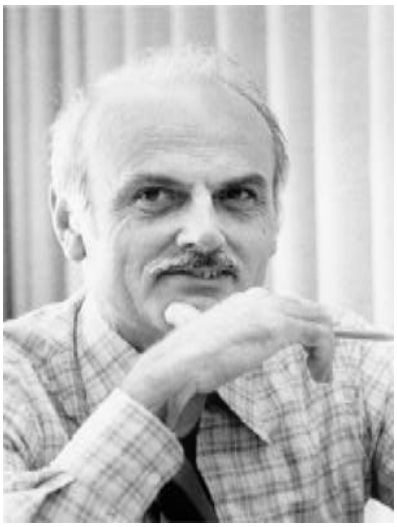


Figure 64 : E.F. Codd

Le succès du modèle relationnel est dû principalement à sa simplicité et sa puissance mathématique. Une base relationnelle est composée de tables et perçue par l'utilisateur comme un ensemble de tables et rien d'autre. Dans une table, une ligne correspond à un enregistrement et une colonne à un champ de cet enregistrement.

La structure des données est donc intuitive et simple à manipuler. D'un autre côté, les fondements mathématiques du modèle permettent une manipulation des données simple et complète. Le modèle offre ainsi un ensemble d'opérateurs permettant d'effectuer toutes les actions envisageables sur les données. Toute opération relationnelle sur une table génère une nouvelle table, c'est-à-dire fonctionne sur un ensemble de données sans que l'on ait à se préoccuper de traiter successivement chacune des données récupérées par l'opération.

Cette simplicité du modèle a fait qu'il soit toujours d'actualité et à la base de la plupart des SGBD existant actuellement.

2. Concepts de base

2.1. Domaine

Comme pour le modèle entité association, les attributs sont définis dans des domaines de valeurs. Le domaine est défini dans le modèle relationnel par :

Définition
Un domaine est un ensemble de valeurs atomiques.

Un domaine est donc un ensemble de valeurs atomiques ce qui exclut les données composées. L'ensemble des entiers est un domaine, les couleurs le sont également. Par contre, les points géographiques qui sont composés de l'altitude, longitude et de la latitude n'est pas un domaine selon le modèle relationnel.

2.2. Produit cartésien

Le produit cartésien est défini entre plusieurs domaines de définition comme suit :

Définition
Le produit cartésien $D_1 X D_2 X ... X D_n$ est l'ensemble des tuples (n-uplets) <V1,V2,....Vn> tel que : $\forall i, V_i \in D_i$.

Soit le domaine des couleurs D1 = {Bleu,Blanc,Rouge} et le domaine des marques de véhicule D2 = {Renault, Peugeot, Mercedes}.
Le produit cartésien **D1 X D2** = {(Bleu, Renault), (Bleu, Peugeot), (Bleu, Mercedes), (Blanc, Renault), (Blanc, Peugeot), (Blanc, Mercedes), (Rouge, Renault), (Rouge, Peugeot), (Rouge, Mercedes)}.

2.3. Relation

Le concept de relation représente le cœur du modèle relationnel et elle est définit par rapport au produit cartésien de plusieurs domaines de définition comme suit :

Définition
Une relation est un sous-ensemble nommé du produit cartésien d'une liste de domaines. Elle est notée R(A1:D1,…,An:Dn) où D1,…,Dn sont des domaines.

Une relation doit impérativement avoir un nom qui est unique dans le schéma relationnel. Elle représente un sous-ensemble du produit cartésien entre les domaines définissant ses attributs.

Note

On peut noter une relation sans mentionner les domaines : R(A1, A2, …, An)

Soit les deux domaines de définition présentés dans l'exemple précédent : D1 = Couleur et D2 = Marque. Soit la relation nommée Couleur_Véhicule représentant les couleurs par marque que commercialise un vendeur de véhicule par exemple. La relation est définie comme suit :

Couleur_Véhicule(Couleur : D1, Marque:D2).

La relation est représentée sous une forme tabulaire dont chaque ligne représente une extension.

Définition
L'extension d'une relation R(A1:D1,…,An:Dn) est un ensemble de n-uplets (V1,V2,...,Vn) tq $\forall i, V_i \in D_i$

Un exemple des extensions de la relation Couleur_Véhicule est présenté dans le tableau ci-dessous.

Comme la relation est un sous-ensemble du produit cartésien, on remarque que le tableau ne contient pas toutes les combinaisons possibles entre les domaines de définitions mais uniquement les enregistrements dont on a besoin (Nos données).

Couleur_Véhicule	
Couleur	**Marque**
Bleu	Renault
Blanc	Renault
Rouge	Mercedes

2.4. Attribut

Un attribut est un nom donné à une colonne d'une relation, il prend ses valeurs dans le domaine de définition de la colonne en question. Par exemple, Nom, Age et Marié sont des attributs de la relation Personne.

2.5. Clé d'une relation

Chaque extension de la relation doit être identifiée de manière unique. Pour assurer cette unicité on ne prend pas en considération tous les attributs de l'extension mais uniquement la clé de la relation.

Définition
C'est un groupe d'attributs minimum qui détermine un tuple unique dans une relation.

On dit qu'un ensemble d'attributs de la relation forme sa clé si l'ensemble est minimum et en connaissant une valeur de cette clé on peut déduire le reste des attributs. Le numéro de sécurité sociale peut être utilisé comme clé de la relation personne.

Toute relation doit posséder au moins une clé.

Il se peut donc qu'une relation possède plusieurs clés.

2.6. Schéma d'une relation

Dans le modèle relationnel, les relations sont définies avec leur schémas.

Définition
Le schéma d'une relation est représenté par le nom de la relation, la liste de ses attributs avec leurs domaines, et la liste de ses clés.

Le schéma de la relation Voiture est comme suit :
Voiture (**Châssis** : Texte, Modèle : texte, Année : Entier, Couleur : texte, Type : texte). Par convention, la clé est soulignée et représentée en gras.
de la même manière pour une relation, on peut définir le schéma d'une base de données comme suit :

Définition
Le Schéma d'une base de données est l'ensemble des schémas des relations la composant.

2.7. Clé étrangère

Au niveau du modèle Entité Association, le lien entre les entités est assuré par les associations. Dans le modèle relationnel, il existe un seul concept pour représenter les données et c'est la relation. Pour représenter les liens entre les différentes relations d'un schéma relationnel, on utilise le concept de clé étrangère.

Définition
Une clé étrangère au niveau d'une relation est un groupe d'attributs devant apparaître comme clé primaire dans une autre relation.

Les clés étrangères définissent les contraintes d'intégrité référentielles suivantes :

- Lors d'une insertion, la valeur des attributs doit exister dans la relation référencée ;
- Lors d'une suppression dans la relation référencée les tuples référençant doivent disparaître ;

Par exemple, considérons les deux relations suivantes :

PERSONNE (NSS, NOM, PRENOM)
VOITURE (MODELE, ANNEE, COULEUR, TYPE, *NSS*)

L'attribut NSS au niveau de la relation Voiture figure comme clé primaire au niveau de la relation Personne c'est donc une clé étrangère. On définit la clé étrangère de la manière suivante : VOITURE.NSS **REFERENCE** PERSONNE.NSS

3. Passage de l'E/A au Relationnel

Le modèle Entité/Association est un modèle conceptuel permettant de modéliser une réalité à l'aide de trois concepts essentiels : l'entité, l'association et les attributs. Au niveau du relationnel on ne traite que des relations. Pour passer du premier modèle au second on a besoin de règles pour faire la correspondance entre les concepts de l'un avec ceux de l'autre.

Le passage du modèle entité association se fait essentiellement par rapport à la cardinalité maximale. Les cardinalités minimales donnent naissance à une contrainte de valeur NULL sur l'attribut qui devient une clé étrangère.

3.1. Règle 1 : Entité non faible

Une entité E non faible est représentée par une relation R dont les attributs simples sont les attributs de l'entité E et la clé de R est l'identifiant de E. la relation porte le même nom que l'entité.

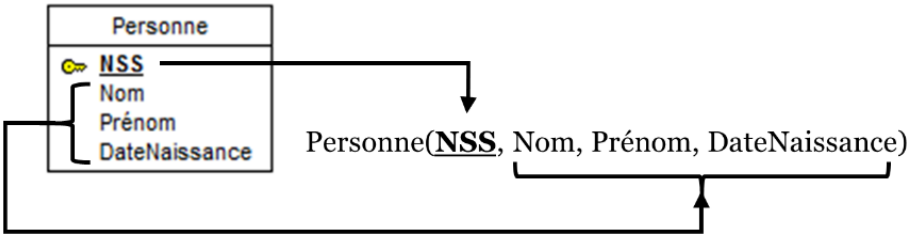


Figure 65 : Passage de l'entité non faible

3.2. Règle 2 : Relation 1 - n

Dans le cas d'une relation 1 - n (Père/fils), l'association n'est pas représentée par une relation, cependant ses attributs migrent vers la relation représentant le fils et la clé du père migre vers le fils comme clé étrangère.

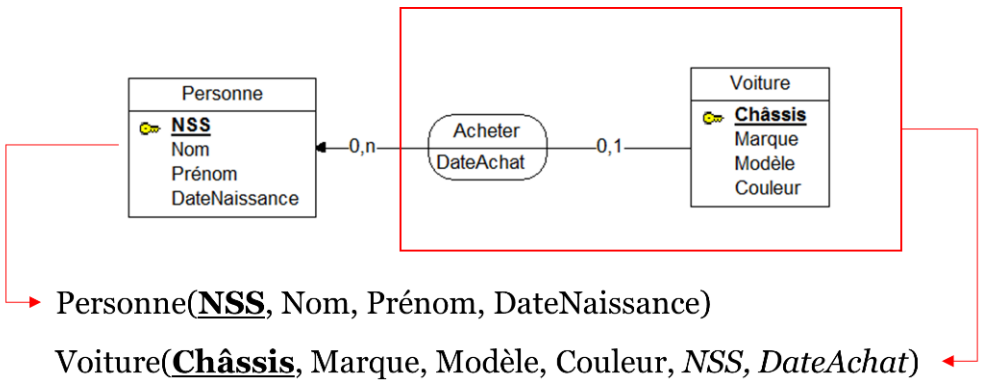


Figure 66 : Passage de l'association 1 – N

Il faut également définir NSS comme étant une clé étrangère de la manière suivante : Voiture.NSS référence Personne.NSS

3.3. Règle 3 : Relation n - m

Dans le cas d'une relation n - m, l'association A est représentée par une relation R dont les attributs sont les attributs de A et la clé est celle formée des clés des entités participant à l'association A.

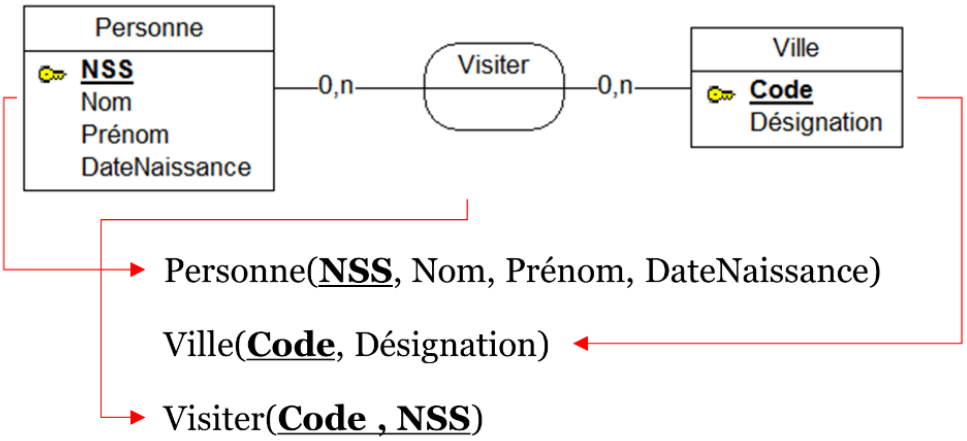


Figure 67 : Passage de l'association n- m

Il faut noter à ce niveau que les composantes de la clé de la relation Visiter sont également des clés étrangères à définir dont chacune référence la clé primaire de la relation correspondante. Pour cet exemple, on ajoute donc les définitions suivantes : Visiter.NSS référence Personne.NSS et Visiter.Code référence Ville.Code.

3.4. Règle 4 : Entité Faible

Le passage d'une entité faible à un schéma relationnel est identique à celui d'une association 1 - n classique. La seule différence est que la clé du père migre vers le fils est entre dans la constitution de la clé primaire de ce dernier.

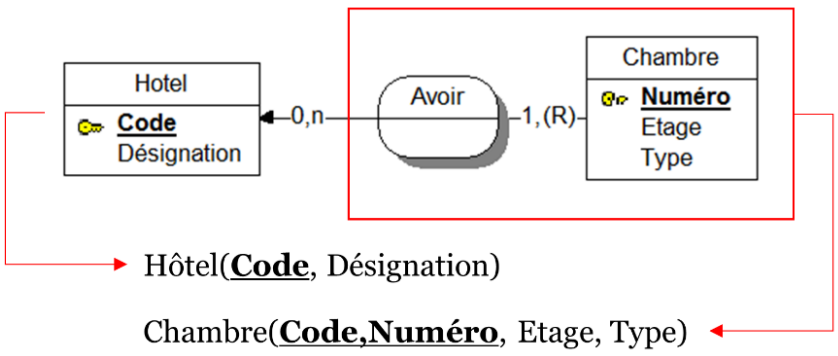


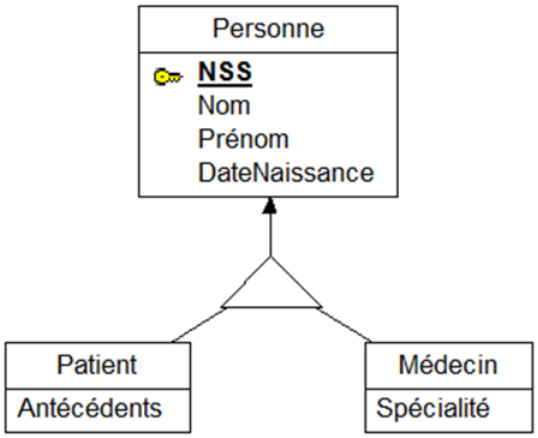
Figure 68 : Passage de l'entité faible

Il faut également définir Code comme étant une clé étrangère comme suit : Chambre.Code référence Hôtel.Code.

3.5. Règle 5 : Généralisation / Spécialisation

Dans le cas d'une généralisation / spécialisation, il existe trois manières de passer au schéma relationnel. Il n'existe pas de critère de choix entre ces manières, c'est laissé à l'appréciation du concepteur. Chacune des méthodes induit des contraintes différentes au niveau du schéma résultant.

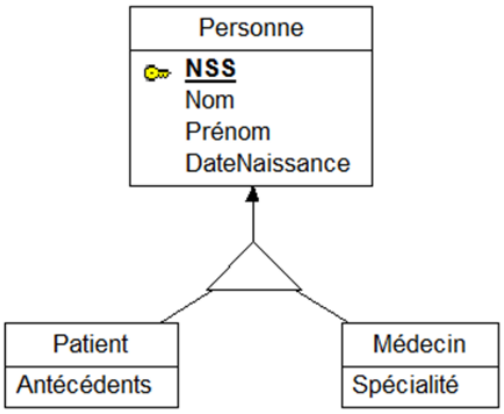
Premier Cas : Transformer chacune des entités en utilisant la règle N°1 en spécifiant comme clé des relations correspondantes aux entités générées la clé de l'entité génératrice.



Personne(**NSS**, Nom, Prénom, DateNaissance)
Patient(**NSS**, Antécédents)
Médecin(**NSS**, Spécialité)

Figure 69 : Passage Généralisation Cas N°1

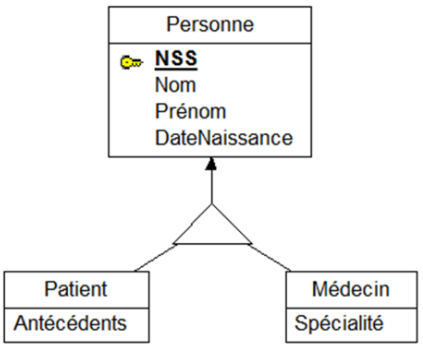
Deuxième Cas (Push down) : Dans le cas du push down, on représente les entités générées mais pas l'entité mère. Les attributs de cette dernière seront dupliqués au niveau de toutes les entités qu'elle génère.



Patient(**NSS**, Nom, Prénom, DateNaissance, Antécédents)
Médecin(**NSS**, Nom, Prénom, DateNaissance , Spécialité)

Figure 70 : Passage Généralisation Cas N°2

Troisième Cas (Push Up) : dans le cas du push up, seule l'entité mère est représentée. Les entités spécialisées ne seront pas représentées et leurs propriétés seront toutes représentées au niveau de l'entité génératrice.



Personne(**NSS**, Nom, Prénom, DateNaissance, Antécédents, Spécialité)

Figure 71 : Passage Généralisation Cas N°3

3.6. Cas particulier (Association 1 - 1)

Dans le cas d'une association 1 - 1, la migration de la clé peut se faire dans un sens comme on peut le faire dans les deux sens. Il est à noter que quand un cas pareil se présente il est presque impossible que les cardinalités minimales des deux côtés soient toutes les deux égales à **1**, l'une d'elles est toujours un **0.** Ce qui fait que le sens le plus recommandé pour la migration de la clé est celui allant du côté de la cardinalité minimale **0** vers celui de la cardinalité minimale **1**.

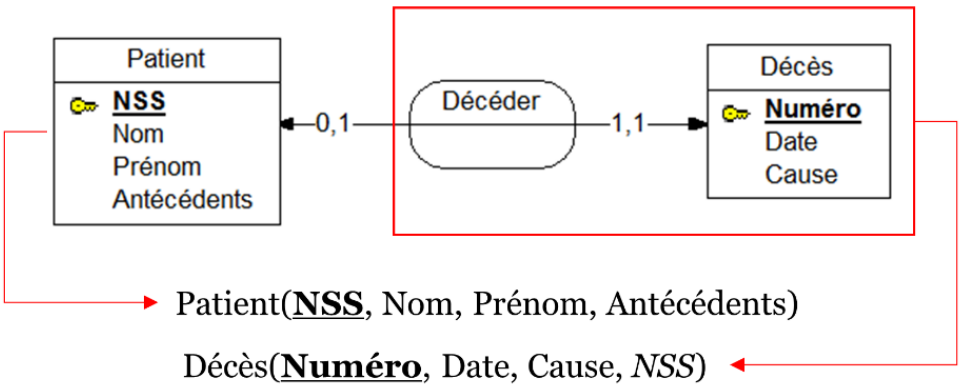
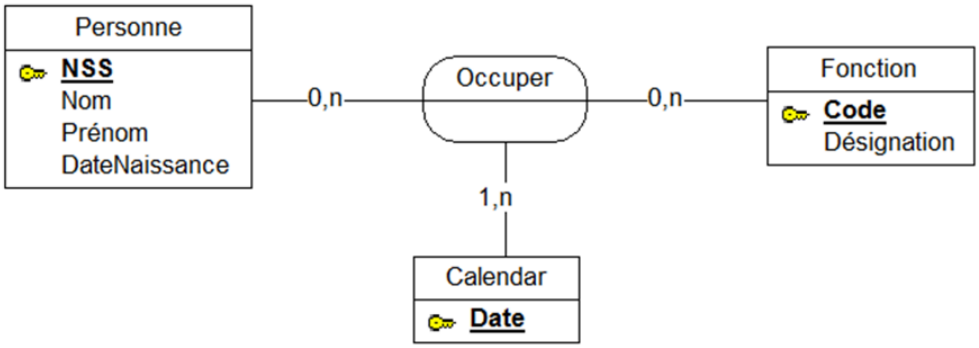


Figure 72 : Passage de l'association 1 – 1

L'identifiant de patient peut très bien être utilisé comme identifiant pour décès. Si on n'a pas besoin de numéro du décès, si ce dernier est un identifiant fictif on peut s'en passer et le remplacer par NSS. Ceci reste valable dans tous les cas similaires.

3.7. Cas particulier (Entité avec un seul attribut)

Quand on est en présence d'une entité ne possédant qu'un seul attribut qui est sont identifiant dans le modèle Entité Association et que les valeurs de cet attribut sont connues à l'avance ou ne sont pas assez importantes pour être sauvegardées. Cette entité n'est pas représentée dans le schéma relationnel correspondant comme c'est le cas pour l'entité date.



Personne(**NSS**, Nom, Prénom, DateNaissance)
Fonction(**Code**, Désignation)
Occuper(**NSS,Code,Date**)

Figure 73 : Passage Entité avec un seul attribut

5. Les Dépendances Fonctionnelles

5.1. Introduction

La notion de dépendance fonctionnelle permet d'établir des liens sémantiques entre attributs ou groupe d'attributs. On note A→B où A est dit source de la dépendance fonctionnelle et B sa destination. On interprète cela de trois manières :

- il existe une dépendance fonctionnelle d A vers B

- A détermine B

- B dépend fonctionnellement de A.

Définition
Étant donné une relation R, nous disons qu'il y a dépendance fonctionnelle (DF) X → Y avec $X \cup Y \in \Delta(R)$ *si, à une valeur de X est associée au plus une valeur de Y.*

La question à se poser pour dire qu'il existe un DF entre X et Y : Connaissant la valeur de X, puis-je connaître la valeur de Y ?

Par exemple, soit la relation voiture définie comme suit : Voiture (Châssis, Couleur, Type, Marque, Puissance), on peut définir les DF suivantes pour cette relation :
Châssis→Couleur ; Type→Marque ; Type→Puissance

5.2. DF élémentaires

Une dépendance fonctionnelle A→B est dite élémentaire, s'il n'existe pas C, inclus dans A, qui assure lui-même une dépendance fonctionnelle C→B.

Exemple

1. Ref_Article→Nom_Article
2. Num_Facture, Ref_Article→Qte_Article
3. Num_Facture, Ref_Article→Nom_Article

Les dépendances fonctionnelles 1 et 2 sont élémentaires alors que la 3 ne l'est pas. Parce qu'il existe une partie de la source de la df qui assure elle-même la dépendance fonctionnelle : c'est Ref_Article au niveau de la df 1.

5.3. DF directes

Une dépendance fonctionnelle A→B est dite directe, s'il n'existe pas un attribut C qui engendrerait une dépendance fonctionnelle transitive A→C→B.

Exemple

1. Num_Facture→Num_Représentant VRAI
2. Num_Représentant→Nom_Représentant VRAI
3. Num_Facture→Nom_Représentant FAUX

Les dépendances fonctionnelles 1 et 2 sont directes alors que la troisième ne l'est pas parce qu'elle peut être déduite par transitivité à partir des deux premières.

5.4. DF triviales

Une dépendance fonctionnelle est dite triviale s'il est impossible qu'elle ne soit pas satisfaite. Une dépendance fonctionnelle est triviale si et seulement si le membre droit (destination) est un sous-ensemble du membre gauche (source).

A1, A2, …, An→B1, B2, …, Bm

Triviale : Si B1, B2, …, Bm est un sous-ensemble de A1,A2,..,An

Non triviale : Si au moins un Bi n'appartient pas à A1,A2,..,An

Complètement non triviale : Si aucun des Bi n'appartient à A1,A2,…,An

5.5. Graphe des dépendances fonctionnelles

Un ensemble F de dépendances fonctionnelles peut être représenté avec un graphe orienté où chaque nœud représente un attribut et les arcs représentent les dépendances fonctionnelles entre les attributs.

Exemple : Soit la relation Commande(Num,Date,Client,Article,Prix,Quantité) et les DF suivante : Num→Date ; Num→Client ; Article→Prix ; Article,Num→Quantité.

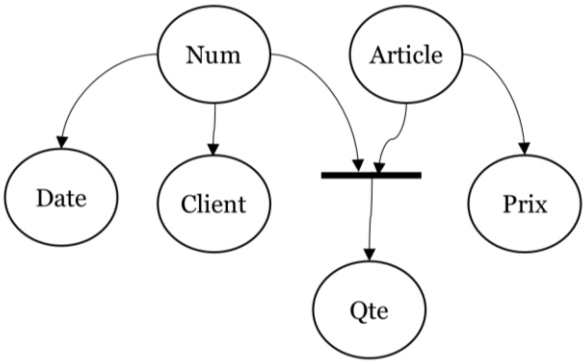


Figure 74 : Graphe des DF

5.6. Les axiomes d'Armstrong

Les Axiomes d'Armstrong sont un ensemble de règles qui permettent d'effectuer des inférences de dépendances fonctionnelles à partir d'autres dépendances fonctionnelles.

Définition

Si F est une couverture fonctionnelle de la relation R (ensemble de ses DF), On dit qu'une dépendance fonctionnelle : X⟶Y est inférée (déduite) de F si X⟶Y est valide pour tout tuple d'une extension de R.

5.6.1. Les Axiomes de base

1. Réflexivité ($Y \subseteq X \Rightarrow X \longrightarrow Y$) : Tout ensemble d'attributs détermines ses propres éléments (Attributs).
2. Augmentation ($X \longrightarrow Y \Rightarrow XZ \longrightarrow YZ$) : En ajoutant les mêmes attributs des deux côtés d'une dépendance valide, la DF résultante reste également valide.
3. Transitivité ($X \longrightarrow Y \wedge Y \longrightarrow Z \Rightarrow X \longrightarrow Z$) : C'est le principe standard de la transitivité.

Note

Il est prouvé que ces trois règles d'Armstrong sont fondées et complètes. Ce qui signifie que toute dépendance fonctionnelle déduite en utilisant la réflexivité, l'augmentation ou la transitivité est une dépendance fonctionnelle inférée et que toute dépendance fonctionnelle qui peut être inférée de F, peut l'être en utilisant seulement les axiomes mentionnés ci-dessus.

5.6.2. Les Axiomes supplémentaires

En plus des axiomes de base, il existe d'autres règles d'inférences qu'on peut utiliser lors de l'inférence.

1. Projection ($X \longrightarrow YZ \Rightarrow X \longrightarrow Y$) : On peut déduire une DF à partir d'une autre DF en projetant sur les composantes de la partie droite de la DF. La cible d'une DF peut être décomposée.
2. Union ($X \longrightarrow Y \wedge X \longrightarrow Z \Rightarrow X \longrightarrow YZ$) : On peut regrouper les parties droites des DF ayant la même partie gauche afin d'en déduire une seule DF.

3. Pseudo-Transitivité ($X \longrightarrow Y \wedge WY \longrightarrow Z \Rightarrow WX \longrightarrow Z$) : On applique le principe de la transitivité à un attribut de la partie gauche de la seconde DF.
4. Décomposition ($X \longrightarrow YZ \Rightarrow X \longrightarrow Y, X \longrightarrow Z$) : on applique la projection à tous les attributs formant la partie droite de la DF et on obtient plusieurs DF ayant la même partie gauche.
5. Auto-Détermination ($X \longrightarrow X$) : Tout attribut s'auto-détermine.
6. Composition ($X \longrightarrow Y, V \longrightarrow Z \Rightarrow XV \longrightarrow YZ$) : On peut composer deux DF en mettant les deux parties gauches ensemble et les deux parties droites également ensemble.
7. Augmentation à gauche ($X \longrightarrow Y \Rightarrow XZ \longrightarrow Y$) : En rajoutant des attributs à la partie gauche d'une DF, cette dernière reste toujours valide.

5.7. La fermeture transitive

Si F est une couverture fonctionnelle de la relation R, la fermeture transitive de F notée F+ est l'ensemble des dépendances fonctionnelles qui peuvent être inférées de F par transitivité.

Exemple : Soit l'ensemble des DF, F= {Châssis→Type; Type→Marque; Type→Puissance}. La fermeture transitive F+ de F est l'ensemble suivant :

F+=F UNION {Châssis→Marque; Châssis→Puissance}

Note

Deux ensembles de dépendances fonctionnelles sont équivalents s'ils ont la même fermeture transitive.

5.8. La fermeture d'un attribut

La fermeture d'un ensemble d'attributs X par rapport à F notée X+(F) est l'ensemble des attributs fonctionnellement dépendant de X sous la couverture fonctionnelle F.

Pour trouver X+ on utilise l'algorithme suivant :

```
Algorithme de calcul de X+
  Résultat = X
  Tant Que (Résultat évolue)
  Pour chaque (DF : Y→Z dans F) {
      SI (Y ⊂ Résultat) Alors Résultat = Résultat ∪ Z
  }
}
```

Algorithme détaillé

```
Résultat : Ensemble d'attributs ; // variable qui contiendra le résultat
Nouveau : Booléen // pour savoir si on a évolué ou non

Résultat = X
Nouveau = Vrai
Tant Que Nouveau {
    Nouveau = Faux
    Pour Chaque f ∈F {
        Si gauche(f) ⊂ Résultat Alors {
            Résultat=Résultat ∪ droite(f)
            Nouveau = Vrai
        }
    }
}
Retourner Résultat
```

Soit la relation suivante Employé (NE, ENom, NP, PNom, Loc, Temps) ainsi que sa couverture fonctionnelle F = { NE→ENom ; NP→{PNom, Loc} ; {NE,NP}→Temps}. En utilisant l'algorithme ci-dessus on peut déduire NE+, NP+ et {NE,NP}+ comme suit :

NE+ = {NE , ENom }
NP+ = {NP, PNom, Loc }
{NE,NP}+ = {NE,NP,ENom,PNom,Loc,Temps}

5.9. La Couverture Minimale

Définition

Une couverture minimale d'un ensemble de DF notée CM(F) est l'ensemble de dépendances fonctionnelles élémentaires associé à un ensemble d'attributs vérifiant les propriétés suivantes :

1. *Aucune dépendance dans CM(F) n'est redondante ;*
2. *Toute dépendance fonctionnelle élémentaire de CM(F) est dans F+ ;*

La couverture minimale est le sous-ensemble minimal de dépendances fonctionnelles permettant de générer toutes les autres.

On peut calculer CM(F) en utilisant un algorithme consistant à détecter les dépendances fonctionnelles redondantes et les écarter. Une dépendance est dite redondante si on peut la déduire à partir des dépendances restantes en utilisant les axiomes d'Armstrong.

```
Couverture(F) {
    CM = F
    Remplacer Chaque DF  X→(A1,…,An) par n DF  X→A1,…, X→An
    Pour chaque DF : f =X→A dans CM {
       SI (CM – {f}) Implique {f} Alors CM = CM - {f}
    }
    Retourner CM
}
```

Algorithme détaillé de CM(F) : Cet algorithme est le même que le précédent quoiqu'on détaille ici la notion de redondance de dépendances fonctionnelles.

On dit donc qu'une dépendance fonctionnelle *f* est redondante si la fermeture de sa partie gauche sur l'ensemble des dépendances restantes (F- *f*) comporte la partie droite de *f*. Ce qui signifie que pour une dépendance fonctionnelle X→Y le lien sémantique entre X et Y est gardé même en enlevant *f*.

```
CM(F:Ensemble de DF) {
    CM : Ensemble de DF
    CM = F
    Décomposer les parties droites de DF
    Pour chaque f : X→Y ∈ CM {
        Calculer X+(F-{f})
        Si Y ⊆ X+ Alors CM = CM - {f}
    }
    Retourner CM
}
```

Exemple : soit l'ensemble de DF suivant F= {M→A ; HDA→S; S→V; HDS→AMV; HDJ→S; HDM→VS; J→E }

Calcul de la couverture minimale :
Décomposition des parties droites
M→A; HDS→A; HDM→S; HDS→M; HDS→V; HDA→S;
HDJ→S; J→E ; S→V ; HDM→V

Traitement des DF
M→A : M+={M}. A ⊄ M+ donc la df n'est pas redondante
HDS→A : HDS+={H,D,S,M,A,V,E,S}. A ⊂ HDS+ donc la df est redondante
HDM→S : HDM+={H,D,M,A,V,S}. S ⊂ HDM+ donc la df est redondante
HDS→M : HDS+={H,D,S,A,V}. M ⊄ HDS+ donc la df n'est pas redondante
HDS→V : HDS+={H,D,S,A,M,V}. V ⊂ HDS+ donc la df est redondante
HDA→S: HDA+={H,D,A}. S ⊄ HDA+ donc la df n'est pas redondante
HDJ→S: HDJ+={H,D,J,E}. S ⊄ HDJ+ donc la df n'est pas redondante
J →E : J+={J}. E ⊄ J+ donc la df n'est pas redondante
S →V : S+={S}. V ⊄ S+ donc la df n'est pas redondante
HDM→V : HDM+={H,D,M,A,S,V}. V ⊂ HDM+ donc la df est redondante.
La couverture minimale est CM = {M→A; HDS→M; HDA→S; HDJ→S; J→E; S→V}

5.10. Clé candidate

Il existe plusieurs algorithmes de recherche des clés d'une relation comme ceux de :

1. FADOUS R; FORSYTH J: Finding Candidate Keys of relational Databases, ACM SIGMOD, 1975
2. LUCCHESI CL; ORSBORN SL: candidate keys of relations, Technical report, U. or Waterloo, Ontario, Canada, 1976.
3. KUNDU S, An Impoved Algorithm for finding a Key of a relation., Conf. PODS, 1975.

Nous donnons ici le principe de recherche d'une clé ainsi qu'un algorithme détaillé.

Définition

Un ensemble X d'attributs d'une relation R(⊿) est une clé si et seulement si : X+=⊿ et pour tout Y ⊆ X, Y+ n'est pas égal à ⊿.

Algorithme de recherche d'une clé

```
Soit une relation R.
SI R ne contient qu'un seul attribut {
        Cet attribut forme l'unique clé candidate de R
}
SINON {
   Soient X et Y deux ensembles non vides disjoints d'attributs de R tel que XUY = Δ.
   SI X→Y et X minimal {
      X est une clé candidate de R // il peut y en avoir plusieurs
   }
   SINON {
      //Aucun X ne peut être trouvé ainsi l'unique clé candidate de R est formée de tous
        ses attributs.
   }
}
```

Algorithme détaillé de recherche de clé

```
FindKey(R) {
  C : ensemble d'ensembles d'attributs ; // les clés candidates
  X : ensemble d'attributs
  Partie(X) //donne tous les ensembles obtenus en combinant les attributs de X
  Min(C) //Renvoi l'ensemble minimal des ensembles de C
  C={}
  Pour chaque X de Partie(Δ) {
    Si X+=Δ Alors C=C ∪ X
  }
  Retourner Min(C)
}
```

On peut proposer une amélioration de cet algorithme pour éviter de traiter toutes les parties créées à partir des combinaisons entre les attributs d'une relation. Pour cela nous définissons deux concepts : la source et le puits.

Définition

On dit qu'un attribut est une source si et seulement si cet attribut n'apparaît dans aucune partie droite des dépendances fonctionnelles de F.

Un attribut qui est une source est un attribut qui participe à la détermination d'autres attributs mais n'est déterminé par aucun attribut.

Définition

On dit qu'un attribut est un puits si et seulement si cet attribut n'est pas une source et n'apparaît dans aucune partie gauche des dépendances fonctionnelles de F ou s'il apparaît alors la partie droite de cette dépendance fonctionnelle doit être un puits.

Les puits sont des attributs qui ne participent aucunement à la détermination d'autres attributs.

Note

Toute clé de R ne contient que des sources et aucun puits.

Donc pour améliorer l'algorithme précédent, on ne considère que les attributs qui ne sont pas des puits. Et uniquement les combinaisons d'attributs contenant toutes les sources trouvées.

Observations

Dans le cas où une relation possède plusieurs clés candidates, on ne peut trouver de sources. Dans ce cas, on détecte les puits de façon récursive puis le reste des attributs font partie des clés candidates. On Exécute l'algorithme précédent pour cet ensemble d'attributs.

Soit l'ensemble d'attributs suivant, F= {M→A ; HDA→S; S→V; HDS→AMV; HDJ→S; HDM→VS; J→E} et cherchons les clés candidate de la relation R(A,M,H,D,S,V,J,E).

On applique le principe de sources et de puits. Les sources de R sont : H, D, J et ses puits sont : E, V.
On écarte ainsi les attributs E, V de la clé et on commence par vérifier que l'ensemble des sources est une clé minimale.
La clé de R est **HDJ** car HDJ+=Δ et pour tout Y de Partie(HDJ), Y+≠Δ.
En fonction de ce qu'on vient de voir, on peut redéfinir la clé d'une relation de la manière suivante :

Définition

La clé de la relation R(Δ) est un sous-ensemble $X \subseteq \Delta\ tq$

1. $\forall a \in \Delta, X \rightarrow a$
2. $\nexists\ Y \subseteq X\ tq\ Y \rightarrow \Delta$

Une clé est un ensemble minimal d'attributs qui détermine tous les autres attributs de la relation.

Un ensemble d'attributs qui inclut une clé est appelé superclé.

6. Normalisation des relations

6.1. Théorie de la normalisation

Cette théorie est basée sur les dépendances fonctionnelles qui permettent de décomposer l'ensemble des informations en diverses relations. Chaque nouvelle forme normale marque une étape dans la progression vers des relations présentant de moins en moins de redondance. On applique les formes normales successivement à la relation universelle afin d'obtenir un schéma normalisé.

Définition
La relation universelle est la relation qui regroupe toutes les informations à stocker (Tous les attributs constatés).

6.2. Pourquoi normaliser ?

En modélisant les données dans une seule relation ou plusieurs relations ne respectant pas les règles de normalisation, le schéma obtenu permettra un taux élevé de redondances. L'objectif de la normalisation est de limiter cette redondance afin de permettre de limiter les pertes de données ainsi que les incohérences entre elles. Et pour améliorer également les performances de traitement.

Soit l'exemple suivant représentant la relation Enseignements :

Matricule	**Nom**	**Module**	**Année**	**Niveau**
01234555	DAHAK	BDD	2006/2007	3SI
01234555	DAHAK	BDD	2007/2008	3SI
01234555	DAHAK	BDD	2015/2016	1CS
01234555	DAHAK	IGL	2016/2017	1CS

On remarque que pour chaque ligne de la table, le matricule et le nom de l'enseignant se répètent. D'un autre côté, en supprimant la dernière ligne on perd complètement l'information qu'on dispose d'un module appelé IGL. Finalement, si on souhaite modifier le matricule ou le nom de l'enseignant on sera obligé de le faire pour toutes les lignes sinon on cause une incohérence au niveau des données.

Ces problèmes apparaissent au niveau de cette relation parce qu'elle n'est pas normalisée. L'objectif de la normalisation est donc de pallier à ce type de problèmes au niveau des relations.

6.3. Les formes normales

6.3.1. Première forme normale (1NF)

Définition
Une relation est en première forme normale si et seulement si tous ses attributs ont des valeurs simples (non multiples, non composées)

La première forme normale est appliquée aux attributs de la relation en excluant les attributs multiples et les attributs composés. Un attribut multiple est un attribut contenant une liste de valeurs de même type comme la liste des enfants d'une personne. Et un attribut

composé est un attribut composé de plusieurs valeurs de types différents comme si on mettait la ville et le ccp dans le même champ au niveau d'une relation.

En appliquant la première forme normale on assure que tout attribut de la relation représente une donnée élémentaire du monde réel. Il ne peut ainsi désigner, ni une donnée composée d'entités de nature quelconque, ni une liste de données de même nature.

Soit l'exemple suivant :

Pers1(nom, prénom, rueEtVille, prénomEnfants)
Pers2(nom, prénom, nombreEnfants)
Les valeurs des attributs rueEtVille et prénomEnfants au niveau de la relation Pers1 ne sont pas simples. Le premier est composé et le second est multiple.

Pour normaliser une relation en première forme normale, Il existe deux solutions :

1ère Solution : Créer autant d'attributs que le nombre maximum de valeurs de l'attribut composé (stockage horizontal). On utilise cette solution quand le nombre maximum de valeurs est connu à l'avance et qu'il ne risque pas de changer plus tard.

Par exemple pour représenter les parents d'une personne, on sait qu'une personne ne peut avoir plus de deux parents donc on décompose en deux attributs un pour le père et un pour la mère.

2ème Solution : Créer une nouvelle relation comportant la clé de la relation initiale et l'attribut multiple puis éliminer l'attribut multiple de la relation initiale (stockage vertical).

Cette solution est utilisée quand on ne connait pas le nombre maximum de valeurs ou si ce dernier et trop important.

6.3.2. Deuxième forme normale (2NF)

Définition

Une relation est en deuxième forme normale si et seulement si : elle est en première forme normale et tout attribut n'appartenant pas à une clé ne dépend pas que d'une partie de cette clé mais de sa totalité.

La relation R(**A,B**,C,D,E) dont la couverture fonctionnelle est représentée par le graphe des DF ci-dessous n'est pas en deuxième forme normale.

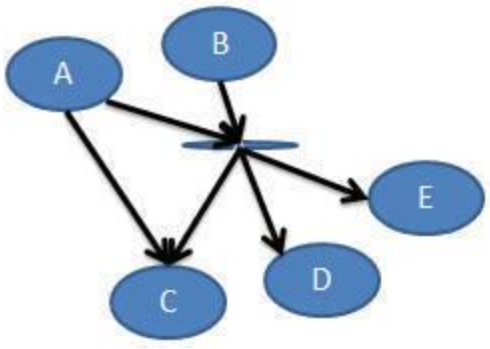


Figure 75 : Relation ne respectant pas la 2FN

L'attribut non clé C dépend d'une partie de la clé qui est A et non pas de sa totalité. La DF qui ne respecte pas la 2FN est dite DF partielle.

Soit la relation Enseignement(**NUM, CODEMATIERE**, NOM, VOLUME_HORAIRE) Avec la DF partielle NUM→Nom

Pour normaliser une relation en deuxième forme normale, il faut :

1. Isoler la DF partielle dans une nouvelle relation avec comme clé la partie gauche de la DF partielle ;
2. Enlever la cible de la DF partielle de la relation initiale ;

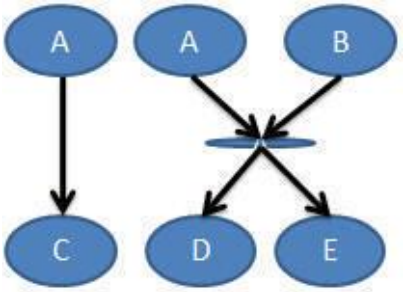


Figure 76 : Normaliser en 2FN

La normalisation de la relation Enseignement de l'exemple précédent donne le schéma relationnel suivant :

Enseignement(**NUM**,**CODEMATIERE**,VOLUME_HORAIRE)
Enseignant(**NUM**,NOM)

Note

Une relation en 1NF dont la clé est mono-attribut est forcément en 2NF.

6.3.3. Troisième forme normale (3NF)

Définition

Une relation est en troisième forme normale si et seulement si : elle est en deuxième forme normale et tout attribut n'appartenant pas à une clé ne dépend pas d'un autre attribut non clé.

La relation R(**A,B**,C,D,E) dont l'ensemble des DF est représenté par le graphe des DF ci-dessous n'est pas troisième forme normale à cause de la DF : E→D. La dépendance qui ne respecte pas la 3FN est appelée DF transitive.

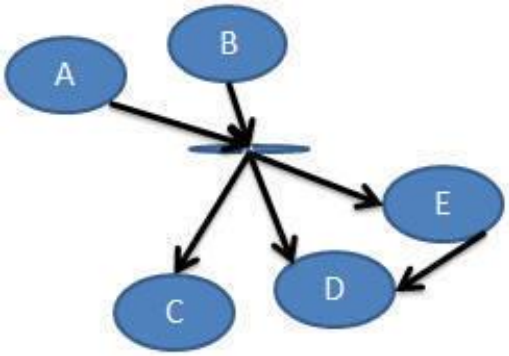


Figure 77 : Relation ne respectant pas la 3FN

Soit la relation Enseignant(**Num,** Nom, Catégorie, Classe, Salaire) avec la DF transitive : Catégorie,Classe→Salaire

Pour normaliser une relation en troisième forme normale il faut :

1. Isoler la DF transitive dans une nouvelle relation,
2. Éliminer la cible de la DF de la relation initiale.

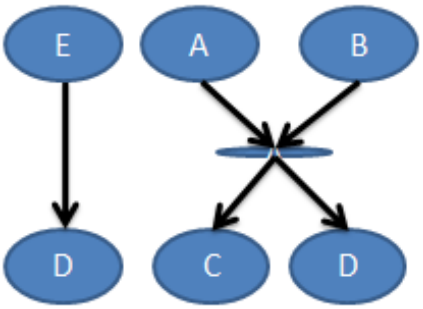


Figure 78 : Normalisation en 3FN

La normalisation en 3FN de la relation Enseignant donne le schéma relationnel suivant :

Enseignement(**Num**, Nom, Catégorie, Classe)
GrilleSalaire(**Catégorie, Classe,** Salaire)

Note

La 3FN exprime le fait que tous les attributs non clé dépendent complètement et uniquement de la clé de la relation.

6.3.4. Forme normale de Boyce Codd(BCNF)

Définition
Une relation est en Boyce-Codd forme normale si et seulement si : elle est en deuxième forme normale et toute source de dépendance fonctionnelle est une clé primaire minimale.

La relation R(**A,B**,C,D,E) dont l'ensemble des DF est représenté par le graphe des DF ci-dessous n'est pas en BCNF à cause de la DF : E→B.

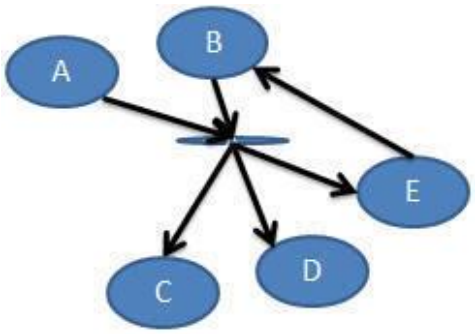


Figure 79 : Relation ne respectant pas la BCNF

Soit la relation suivante Adresse(**Rue, Commune, Bureau-Postal**, CodePostal) avec la DF problématique : CodePostal→Commune

Pour normalisation en Boyce-Codd forme normale il faut :

1. Isoler la DF problématique dans une nouvelle relation ;
2. Éliminer la cible de cette DF et la remplacer par sa source dans la relation initiale.

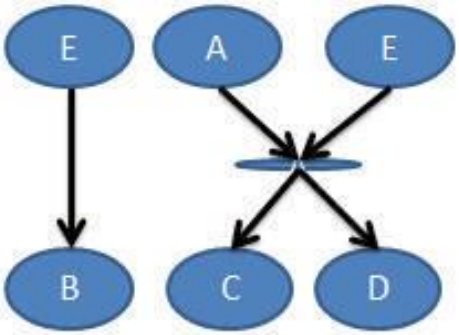


Figure 80 : Normalisation en BCNF

Remarque

Toute relation R en BCNF est forcément en 3 NF.

En disant qu'une relation R est en BCNF ceci signifie que quel que soit X tq X→U, X est une clé minimale. Supposons maintenant que R n'est pas en 3FN, ceci signifie qu'il existe une DF transitive Y→Z tel que Y n'est pas une clé. Or cela contredit la définition de la BCNF. Par contre, si R est en 3 NF elle n'est pas forcément en BCNF.

6.3.5. Synthèse

La première forme normale exprime le fait que toutes les valeurs des attributs sont atomiques (simples, non composées).

La seconde forme normale exprime le fait que tous les attributs non clé d'une relation dépendent **complètement** de la clé de cette dernière et non pas d'une seule partie de cette clé.

La troisième forme normale ajoute une autre restriction à la seconde forme normale en exprimant le fait que tous les attributs non clé dépendent complètement et **uniquement** de la clé de la relation.

La Boyce Codd forme normale pousse plus loin la restriction de la troisième forme normale en exprimant le fait que dans toute relation en deuxième forme normale, s'il existe une dépendance fonctionnelle alors sa partie gauche est forcément une clé de cette relation.

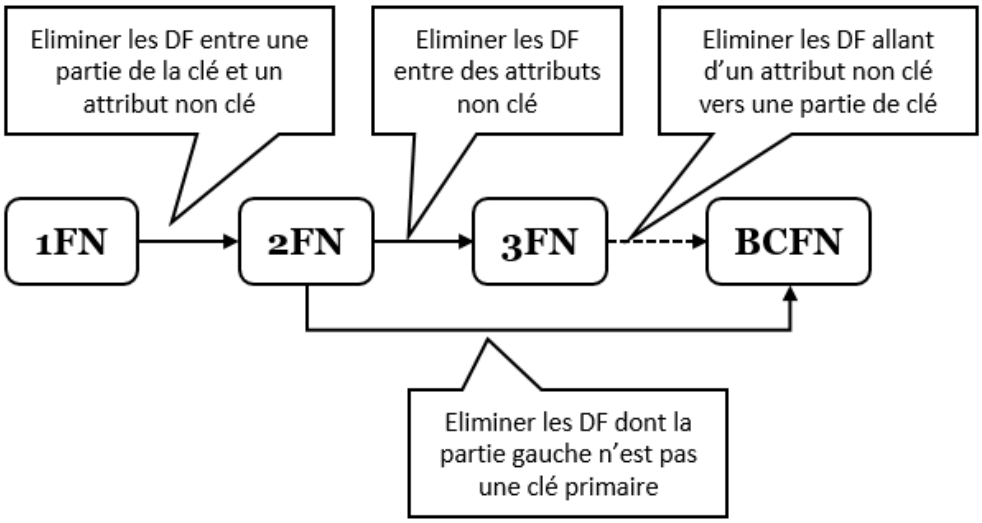


Figure 81 : Synthèse de la normalisation

7. Conception d'un schéma relationnel

Le processus de conception d'un schéma relationnel dont la qualité détermine la pertinence et l'efficacité de la base de données, doit prendre en compte un certain ensemble de critères dont les plus importants sont :

1. Normalité ;
2. Préservation des contenus ;
3. Préservation des dépendances fonctionnelles.

A partir de l'ensemble des dépendances fonctionnelles, il est possible de construire un bon schéma relationnel en exploitant des manipulations des dépendances fonctionnelles (recherche de clé) et des algorithmes de conception de schéma relationnel.

Il existe deux approches possibles pour concevoir un schéma relationnel, l'approche par synthèse et l'approche par décomposition.

7.1. Approche par synthèse

L'approche par synthèse consiste à recomposer des relations à partir d'un ensemble d'attributs indépendants et de la couverture minimale d'un ensemble de dépendances fonctionnelles.

Pour cela on utilise l'algorithme de synthèse suivant :

```
Synthèse (R:Relation universelle; F:Ensemble des DF) {
    Calculer la couverture minimale CM(F) ;
    Partitionner CM(F) en F1, F2,…, Fn Tel que toutes les DF d'un même groupe aient la
    même partie gauche.
    Construire les Ri(Δi) avec Δi constitué de l'union des attributs de Fi
    S'il reste des attributs isolés, on les regroupe dans une relation ayant comme clé
    l'ensemble de ces attributs.
}
```

Soit la relation R(A,B,C,D,E) et l'ensemble des dépendances fonctionnelles F= {A→B ; A→C ; C,D→E ; B→D}

1. La seule clé est A
2. L'ensemble des dépendances fonctionnelles fournies est une couverture minimale ;
3. F1 = {A→B; A→C} F2={B→D} F3={C,D→E}
4. R1(**A**,B,C), R2(**B**,D), R3(**C,D**,E).

Note

Les relations obtenues par l'algorithme de synthèse sont en troisième forme normale grâce au fait d'utiliser la couverture minimale ce qui a écarté toutes les dépendances partielles et transitives.

Comme l'union des F_i donne CM(F), il n'y a aucune perte de dépendances fonctionnelles.

7.2. Approche par décomposition

L'approche par décomposition consiste à remplacer la relation R(A1,A2,…,An) par une collection de relations R1,R2,…,Rp obtenues par projection de R sur des sous-ensembles d'attributs dont l'union contient tous les attributs de R. Cette approche se base sur le théorème de Heath suivant :

Toute relation R(X,Y,Z) est décomposable sans perte d'information en R1=π[X,Y]R et R2=π[X,Z]R s'il y a dans R une dépendance fonctionnelle de X vers Y (X→Y).

L'algorithme de décomposition se présente comme suit :

```
Décomposition(R:Relation universelle ; F:Esemble des DF){
    Result={R}
    SI (Il existe Ri de Result TQ Ri n'est pas en BCNF) {
        Chercher une DF non triviale X → Y dans F+ sur Ri tq X n'est pas une clé ;
        Décomposition((Result-Ri) ∪ (Ri-Y) ∪ (X,Y));
    }
   Pour tout Ri(Δi) et Rj(Δj) TQ Δi⊂ Δj dans result Faire Supprimer(Ri)
   Return Result;
}
```

Soit la relation R(A,B,C,D,E) et l'ensemble des DF F={A→B ; A→C ; C,D→E ; B→D}
La seule clé est A

Result=R;
R n'est pas en BCNF à cause de B→D on décompose
Result={R_1(**B**,D), R_2(**A**,B,C,E)}
R_2 n'est pas en BCNF à cause de B,C→E (Obtenu par pseudo transitivité entre B→D et C,D→E), On décompose
Result = {R_1(**B**,D), R_{21}(**A**,B,C), R_{22}(**B,C**,E)}
Toutes les relations R_i sont en BCNF, l'algorithme s'arrête.

Aucune relation imbriquée ;
Le résultat est {R_1(**B**,D), R_{21}(**A**,B,C), R_{22}(**B,C**,E)}

Avant d'utiliser l'algorithme de décomposition, il faut d'abord regrouper les dépendances fonctionnelles qui ont la même partie gauche pour éviter d'avoir des relations avec la même clé et des attributs différents qui forment, en réalité, une seule relation.

On refait le même exemple mais en commençant par la dépendance C,D→E
La seule clé est A
Result = R;
R n'est pas en BCNF à cause de C,D→E on décompose
Result={R1(**C,D**,E), R2(**A**,B,C,D)}
R2 n'est pas en BCNF à cause de B→D, on décompose
Result={R1(**C,D**,E), R21(**B**,D),R22(**A**,B,C)}
Toutes les relations Ri sont en BCNF, l'algorithme s'arrête.

Aucune relation imbriquée ;
Le résultat est {R1(**C,D**,E), R21(**B**,D),R22(**A**,B,C)}

Le résultat fourni par l'algorithme de décomposition dépend de l'ordre des dépendances fonctionnelles traitées.

8. Conclusion

Le modèle relationnel est le modèle de données le plus répondu actuellement. Pratiquement tous les SGBD l'implémentent et le respecte plus ou moins bien. Il doit sa puissance à sa simplicité et à ses fondements mathématiques permettant des manipulations des données de manière intuitive et très efficace.

La théorie de la normalisation permet de réduire les redondances des données et d'assurer une certaine cohérence de la base. Elle définit plusieurs niveaux de normalisation appelés formes normales dont chacun exprime des contraintes sur les données permettant de réduire ainsi les redondances des données.

Dans la pratique, la troisième forme normale est suffisante pour construire des BDD optimales et normalisées.

Chapitre 4 : Le Langage Algébrique

Objectifs

L'objectif de ce chapitre est d'apprendre à l'étudiant à maitriser les opérations algébriques afin de formuler des requêtes algébriques optimales pour répondre à n'importe quelle question. Ceci permettra une meilleure compréhension du langage SQL.

L'étudiant sera également en mesure d'optimiser une requête algébriquement en utilisant une heuristique d'optimisation basée sur huit règles d'optimisation.

Outils

Free Relational Algebra Interpretor (FRAI) : Un interpréteur graphique pour l'algèbre relationnelle supportant toutes les opérations algébriques et conçu dans un but pédagogique. Il permet aux étudiants de comprendre le fonctionnement des opérations algébriques et la construction de requêtes. Ceci via une interface intuitive et très facile à utiliser. Ce logiciel a été réalisé par Fouad DAHAK comme support pédagogique au cours Bases de données au niveau de l'Ecole Nationale Supérieure d'Informatique d'Alger.

Résumé

Ce chapitre présente trois volets principaux dans l'interrogation des données avec le langage algébrique, à savoir : les opérations algébriques, la construction d'une requête algébrique et l'optimisation algébrique d'une requête. Des vidéos des différents algorithmes des opérations algébrique sont disponible sur ma chaîne Youtube (Aller sur http://dahak.esi.dz).

1. Introduction

L'algèbre relationnelle a été proposée par E. Codd en 1970 dans le but de formaliser les opérations sur les ensembles. Elle constitue une collection d'opérations formelles qui agissent sur des relations et produisent de nouvelles relations.

Ces opérations sont regroupées, selon leurs caractéristiques, en plusieurs familles. Elles permettent d'interroger un schéma relationnel et d'effectuer des transformations sur les données.

L'algèbre relationnelle est à la base du langage d'interrogation SQL implémenté sur les SGBD actuels.

2. Les opérations ensemblistes

Les opérations ensemblistes sont appliquées à deux relations et permettent de générer une troisième relation. Elles sont donc binaires et déduites de la théorie des ensembles.

2.1. Union

> ***Définition***
> *L'union est une opération sur deux relations de même schéma R1 et R2 permettant de construire une troisième relation R3 de même schéma ayant comme tuples ceux appartenant à R1, à R2 ou aux deux mais sans doublons.*

Les tuples qui apparaissent plusieurs fois dans le résultat ne sont représentés qu'une seule fois (pas de doublons). Le doublons est considéré ici en comparant la ligne entière et non pas la clé uniquement.

Plusieurs notations sont utilisées pour les opérations algébriques, on reprend dans ce cours les notations mathématiques et celles utilisées par l'interpréteur FRAI.

On note ainsi l'union de deux relations de la manière suivante : R1 U R2 ou *UNION(R1,R2)*

L'opération est représentée graphiquement par le symbole suivant :

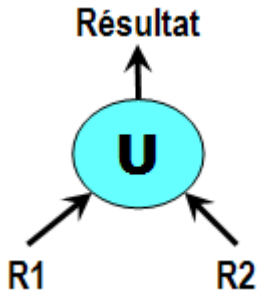


Figure 82 : Opération d'Union

Par exemple, l'union des deux relations R1 et R2 suivantes sonne la relation R3. La structure de R3 est la même que celle des deux relations R1 et R2 et les tuples représentent l'union des tuples de R1 avec ceux de R2 sans doublons.

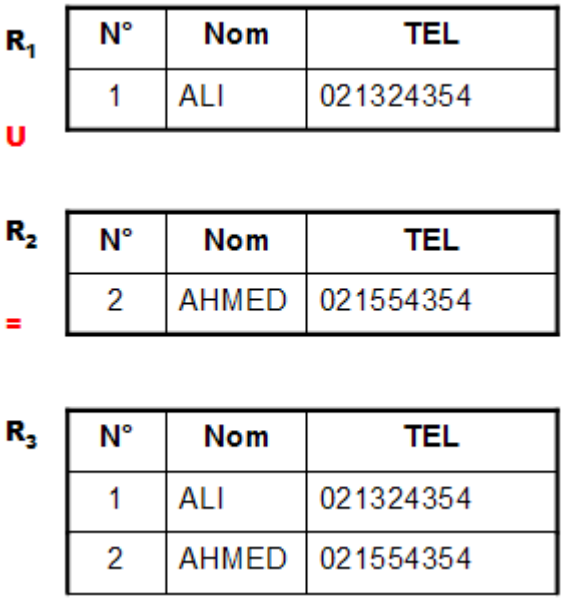


Figure 83 : Exemple d'Union

On utilise l'union, généralement, quand on souhaite afficher des données provenant de plusieurs sources différentes mais ayant la même structure.

2.2. Différence

> ***Définition***
> *La différence est une opération sur deux relations de même schéma R1 et R2 qui sert à construire une troisième relation R3 de même schéma ayant comme tuples ceux appartenant à R1 et n'appartenant pas à R2.*

La différence est utilisée quand on cherche à déduire des données à partir d'autres données. Par exemple, si on disposait des employés présents et de la liste de tous les employés on pourrait avec la différence déduire les employés absents.

Sa notation est la suivante : R1 - R2 ou *DIFFERENCE(R1,R2)*. Et la représentation graphique est représentée ci-dessous

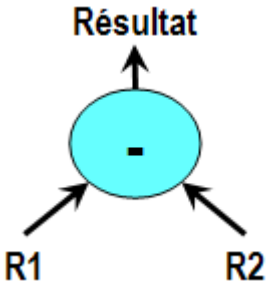


Figure 84 : Opération de différence

Soit l'exemple d'une différence entre la relation R1 et R2 ci-dessous :

R_1

N°	Nom	TEL
1	ALI	021324354
2	AHMED	021554354

-

R_2

N°	Nom	TEL
2	AHMED	021554354

=

R_3

N°	Nom	TEL
1	ALI	021324354

Figure 85 : Exemple de différence

2.3. Produit cartésien

Définition

Le produit cartésien de deux relations R1 et R2 de schéma quelconque est une relation R3 ayant pour attributs les attributs de R1 et ceux de R2 et dont les tuples sont constitués de toutes les combinaisons des tuples de R1 et de R2.

Cette opération est surtout utilisée dans d'autres opérations. Elle permet de constituer ainsi toutes les possibilités de combinaison entre deux ensembles de données afin de le manipuler dans un objectif précis.

Sa notation est la suivante : R1 x R2 ou *PRODUCT(R1,R2)*. Et sa représentation graphique est la suivante :

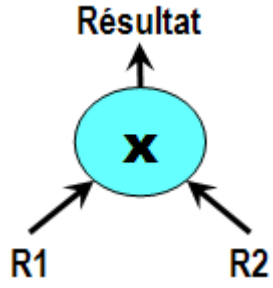


Figure 86 : Le produit cartésien

Par exemple, le produit cartésien des deux relations R1 et R2 ci-dessous donne comme résultat la relation R3. La structure de R3 est constituée des attributs de R1 et ceux de R2.

Quand les deux relations R1 et R2 possèdent des attributs identiques (de même nom), la relation R3 renomme l'un des deux attributs afin qu'il soit unique.

R_1 x R_2 = R_3

N°	Nom	TEL
1	ALI	021324354
2	AHMED	021554354

Cmd	Date
2	23/02/2007

N°	Nom	TEL	CMD	DATE
1	ALI	021324354	2	23/02/2007
2	AHMED	021554354	2	23/02/2007

Figure 87 : Exemple d'un produit cartésien

3. Les opérations spécifiques

En plus des opérations ensemblistes, un ensemble d'opérations dites spécifiques sont définies dans l'algèbre relationnelle. Certaines sont binaires et d'autres unaires et permettent également d'effectuer des transformations entre les données des relations afin d'en déduire de nouvelles relations répondant à nos besoins en information.

3.1. Projection

Définition

La projection d'une relation R(A1,A2,...,An) sur les attributs Ai,Ai+1,...,Ap (avec p<n) est une relation R2 de schéma Ai,Ai+1,...,Ap et dont les tuples sont obtenus par élimination des attributs de R n'appartenant pas à R2 et par suppression des doublons.

La projection est une opération permettant ainsi de récupérer uniquement les colonnes de la table dont on a besoin. On projette ainsi la table sur un sous-ensemble de ses colonnes et le résultat donne une relation ayant comme structure les colonnes projetées et les doublons sont éliminés.

Sa notation est la suivante : $\prod_{A1,A2,,An}(R)$ ou $PROJECT(R,A_1,A_2,..,A_p)$

Et sa représentation graphique est représentée comme suit :

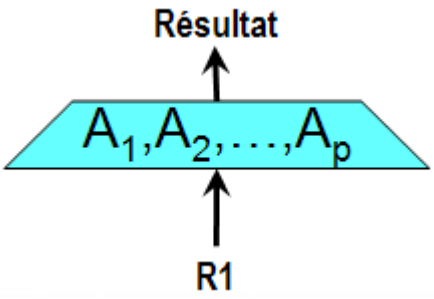


Figure 88 : La projection

En projetant les données de la relation R ci-dessous sur l'attribut Nom, on obtient une relation avec une seule colonne donnant tous les noms figurant dans R sans doublons.

R

N°	Nom	TEL
1	ALI	021324354
2	AHMED	021554354

$\prod_{Nom}(R)$

Nom
ALI
AHMED

Figure 89 : Exemple de projection

3.2. Restriction (Sélection)

> ***Définition***
> *La restriction (ou sélection) de la relation R par une condition C est une relation R2 de même schéma que R dont les tuples sont ceux de R satisfaisant la condition C.*

La restriction permet d'appliquer un filtre sur les données d'une relation. Le filtre est exprimé sous forme d'une condition de la forme :

<Attribut>Opérateur<Valeur>

Les opérateurs possibles sont {=,<,>,<=,>=,<>}

Sa notation est la suivante : $\sigma_{condition}(R)$ ou *RESTRICT(R,Condition)*

Et sa représentation graphique est la suivante :

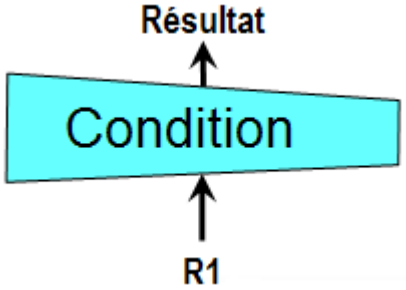


Figure 90 : La restriction

En appliquant une restriction sur la relation R ci-dessous avec la condition Nom='Ali' on obtient une relation avec la même structure que R mais uniquement les lignes contenant un nom = 'Ali'

N°	Nom	TEL
1	ALI	021324354
2	AHMED	021554354
3	ALI	034456543

$\sigma_{nom='ALI'}(R)$

N°	Nom	TEL
1	ALI	021324354
3	ALI	034456543

Figure 91 : Exemple de restriction

3.3. Thêta Jointure

> ***Définition***
> *La thêta-jointure de deux relations R1 et R2 de schéma quelconque selon une condition C'est une relation R3 dont le schéma est formé des attributs des deux relations et les tuples sont ceux du produit cartésien entre R1 et R2 respectant la condition C.*

La condition C est de la forme <Attribut>Opérateur<Attribut> et les opérateurs peuvent être arithmétiques (=,>,<,>=,<=,<>) ou logique (Et, Ou, Non)

Sa notation est la suivante : JOIN(R1,R2,Condition)

Et sa représentation graphique est comme illustré ci-dessous

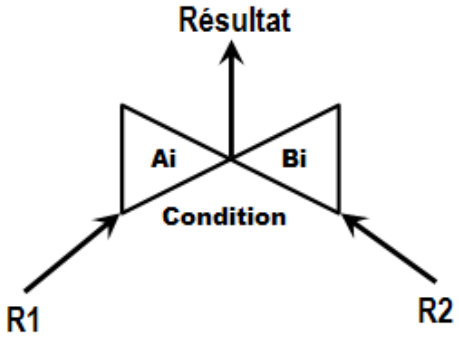


Figure 92 : Thêta-Jointure

Cette opération permet d'effectuer un lien entre les données de deux relations via les valeurs des clés étrangères. Ainsi en cherchant les véhicules d'une personne sachant que la clé de personne est une clé étrangère au niveau de véhicule, on applique une thêta-jointure avec la condition d'égalité sur la clé étrangère.

L'exemple ci-dessous donne une thêta-jointure entre deux relations R1 et R2 sur la condition N>=Num. il faut noter que la condition n'est pas forcément une égalité.

R1

N	Nom	TEL
1	ALI	021324354
2	AHMED	021554354
3	ALI	034456543

R2

Num	Nom	Adresse
0	ALI	Alger
2	AHMED	Blida
5	MALIK	Oran

R1 ⋈ R2
N>=Num

N	Nom	TEL	Num	Nom	Adresse
1	ALI	021324354	0	ALI	Alger
2	AHMED	021554354	0	ALI	Alger
2	AHMED	021554354	2	AHMED	Blida
3	ALI	034456543	0	ALI	Alger
3	ALI	034456543	2	AHMED	Blida

Figure 93 : Exemple de thêta-jointure

Si l'opérateur est « = » Alors c'est une Equijointure Sinon c'est une Inéqui-jointure

3.4. Jointure Naturelle

> ***Définition***
> *La jointure naturelle de deux relations R1 et R2 de schéma quelconque donne une troisième relation R3 dont le schéma est formé des attributs de R1 et ceux de R2 mais en ne prenants les attributs de même nom qu'une seule fois. Les tuples de R3 sont ceux de R1 et de R2 respectant une équijointure entre les attributs de même nom.*

La jointure naturelle fait la même chose que la thêta-jointure avec la possibilité de ne pas déclarer de condition car cette dernière est déduite automatiquement. D'un autre côté, les colonnes ayant le même nom sont représentées une seule fois.

La jointure naturelle est notée comme suit : JOIN(R1,R2)

Et sa représentation graphique est la suivante :

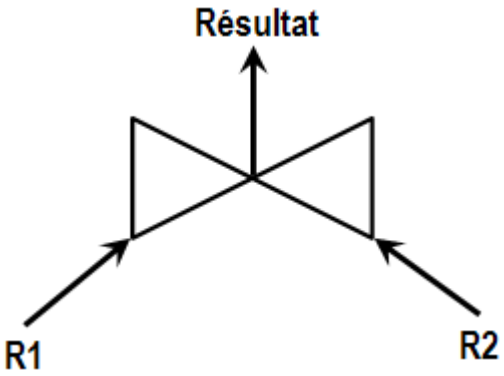


Figure 94 : Jointure naturelle

La jointure naturelle entre les deux relations R1 et R2 de l'exemple ci-dessous donne la relation R3.

R1

N	Nom	TEL
1	ALI	021324354
2	AHMED	021554354
3	ALI	034456543

R2

N	Nom	Adresse
0	ALI	Alger
2	AHMED	Blida
5	MALIK	Oran

R1 ⋈ R2

N	Nom	TEL	Adresse
2	AHMED	021554354	Blida

Figure 95 : Exemple de la jointure naturelle

Note

Une jointure naturelle entre deux relations R1 et R2 n'ayant aucun attribut en commun (de même nom) donne comme résultat le produit cartésien des deux relations.

4. Les opérations dérivées

A partir des opérations vues précédemment, on peut en déduire d'autres opération afin de les utiliser pour simplifier les opérations sur les données. Ces opérations sont dites dérivées.

4.1. Intersection

Définition

L'intersection de deux relation R1 et R2 de même schéma est une relation R3 de même schéma dont les tuples sont ceux appartenant à la fois à R1 et à R2.

L'intersection est une opération utilisée lorsqu'on cherche des données appartenant à deux ensembles en même temps. Si on cherchait par exemple les étudiants qui font à la fois du théâtre et de la musique, on chercherait ceux qui font de la musique et ceux qui font du

théâtre et l'intersection entre les deux résultats nous donne ceux qui font les deux en même temps.

L'intersection est notée comme suit : R1∩R2 ou INTERSECT(R1,R2)

Et la représentation graphique est la suivante :

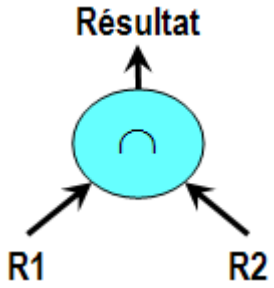


Figure 96 : L'intersection

L'intersection entre les deux relations R1 et R2 par exemple, donne la relation R3.

R_1 ∩ R_2 = R_3

N°	Nom	TEL
1	ALI	021324354
2	AHMED	021554354

N°	Nom	TEL
2	AHMED	021554354

N°	Nom	TEL
2	AHMED	021554354

Figure 97 : Exemple de l'intersection

4.2. Division

> ***Définition***
> *La division de la relation R(A1,A2,...,An) par la sous-relation R2(Ap+1,...,An) est la relation R3(A1,A2,...,Ap) formées de tous les tuples qui concaténés à chaque tuple de R2 donnent toujours un tuples de R1.*

La division est une opération qu'on utilise quand on cherche un ensemble de données avec un filtre sur un sous ensemble de ses attributs mais ce filtre ne concerne pas un seul attribut et ne contient pas une seule valeur mais plusieurs.

La notation de la division est la suivante : R1 ÷ R2 ou *DIVISION(R1,R2)*

Et sa représentation graphique est la suivante :

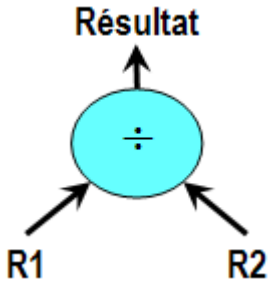


Figure 98 : La division

En divisant la relation R1 sur la relation R2 dans l'exemple ci-dessous, on cherche les noms des personnes qui sont à la fois âgées de 12 ans habitant à Alger et âgées de 23 ans, habitant également à Alger.

R_1 ÷ R_2 = R_3

Nom	Age	Ville
ALI	12	ALGER
ALI	23	ORAN
AHMED	12	ALGER
AHMED	23	ALGER
MALIK	43	ALGER

Age	Ville
12	ALGER
23	ALGER

Nom
AHMED

Figure 99 : Exemple de la division

Note

Les attributs du résultat d'une division sont ceux faisant partie de la première relation et ne sont pas dans la seconde pour que le produit cartésien du résultat avec la deuxième donnent tous les attributs de la première relation.

Pour effectuer une division entre R1 et R2 il faut que tous les attributs de R2 fassent partie de R1 et que R1 possède au moins un attribut en plus que R2.

4.3. Jointure externe

Définition

La jointure externe entre deux relations R1 et R2 de schéma quelconque est une relation R3 dont le schéma est formé des attributs de R1 et ceux de R2 en ne représentant les attributs ayant le même nom qu'une seule fois. Les tuples de R3 sont ceux obtenus avec une jointure naturelle entre R1 et R2 et ceux de R1 et de R2 ne participants pas à la jointure en représentant par des valeurs nulles ceux de l'autre relation.

La notation de la jointure externe est la suivante :EXT-JOIN(R1,R2)

Et sa représentation graphique est la suivante

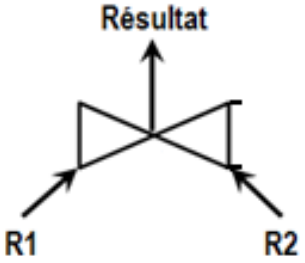


Figure 100 : Jointure Externe

Dans l'exemple suivant, la jointure externe entre les deux associations R1 et R2 donne le résultat de la jointure naturelle entre les deux relations union ceux appartenant à R1 et n'ont aucune association dans R2 et l'inverse.

R1

NA	Nom	TEL
1	ALI	021324354
2	AHMED	021554354
3	ALI	034456543

R2

NB	Nom	Adresse
0	ALI	Alger
2	MALIK	Blida
5	MALIK	Oran

R1 ⋈ R2

NA	Nom	TEL	NB	Adresse
1	ALI	021324354	0	Alger
2	AHMED	021554354	-	-
3	ALI	034456543	0	Alger
-	MALIK	-	2	Blida
-	MALIK	-	5	Oran

Figure 101 : Exemple de la jointure externe

Note

On distingue deux autres variantes de la jointure externe, la jointure externe droite et la jointure externe gauche notées respectivement REXT-JOIN et LEXT-JOIN.

La première donne tous les tuples de la relation à droite de la jointure externe et uniquement ceux de la relation gauche qui participent à la jointure. La seconde c'est l'inverse.

4.4. Semi-jointure

Définition

La semi-jointure entre deux relations R1 et R2 de schéma quelconque est une relation R3 dont le schéma est celui de R1 et les tuples sont ceux de R1 appartenant à la jointure naturelle entre R1 et R2.

La notation de la semi jointure est la suivante : SJOIN(R1,R2)

Et sa représentation graphique est comme suit :

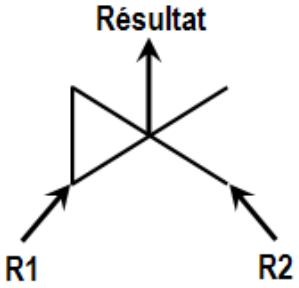


Figure 102 : Semi-Jointure

L'exemple ci-dessous donne un exemple d'une semi-jointure naturelle entre la relation R1 et la relation R2. Le résultat est une projection sur les attributs de R1 de la relation résultante de la k=jointure naturelle entre R1 et R2.

R1

NA	Nom	TEL
1	ALI	021324354
2	AHMED	021554354
3	ALI	034456543

R2

NB	Nom	Adresse
0	ALI	Alger
2	MALIK	Blida
5	MALIK	Oran

R1 ⋈ R2

NA	Nom	TEL
1	ALI	021324354
3	ALI	034456543

Figure 103 : Exemple de la semi-jointure

5. Opérations supplémentaires

Pour la construction de requêtes algébriques, on aura besoin d'autres opérations n'effectuant pas le même travail que celle déjà citées mais qui sont indispensables.

5.1. Opération de renommage

Une opération supplémentaire est utilisée au niveau de l'algèbre relationnelle permettant de renommer un attribut ou une relation afin de permettre sa réutilisation. On effectue un renommage pour les raisons suivantes :

1. Le résultat d'une expression algébrique ne possède pas de nom ;
2. On a besoin de renommer certains attributs d'une relation ou d'une expression algébrique.

Sa notation est la suivante : $\rho_{NA \leftarrow num}(Acteur)$ ou $\rho_{Acteur\_Ali}(\sigma_{nom=ali}(Acteur))$ ou RENAME(R1,1,id)

5.2. L'Affectation

L'affectation est une opération permettant d'affectant le résultat d'une opération algébrique à une variable afin de la réutiliser dans la requête. Elle suit le même principe qu'une affectation dans l'algorithmique. C'est le fait d'attribuer le résultat d'une expression algébrique à une variable temporaire qu'est dans ce cas une relation intermédiaire.

$$R1 \leftarrow \sigma_{taille=32}(VESTE)$$
$$R2 \leftarrow \sigma_{couleur=rouge}(VESTE)$$
$$R3 \leftarrow R1 \cap R2$$
$$Result = \pi_{marque}(R3)$$

Figure 104 : Affectation

6. Propriétés et Lois Algébriques

Un certain ensemble de lois algébriques peuvent être définies sur les opérations algébriques permettant ainsi une reformulation de requêtes et une transformation des expressions algébriques. Ces lois sont définies comme suit :

6.1. Restriction

Descente de restriction : Quand on applique une restriction sur le résultat d'une jointure, il vaut mieux descendre la restriction au niveau de la relation concernée par la condition de la restriction. Ainsi on réduit le nombre de tuples de la relation en question avant la jointure dont l'algorithme consomme du temps et de l'espace.

- $(\sigma c(R \blacktriangleright\blacktriangleleft S)) = (\sigma c(R) \blacktriangleright\blacktriangleleft S)$ si c porte sur R
- $(\sigma c(R \blacktriangleright\blacktriangleleft S)) = (R \blacktriangleright\blacktriangleleft \sigma c (S))$ si c porte sur S

Commutativité de la restriction : La restriction est une opération commutative.

- $\sigma c (\sigma d(R)) = \sigma d (\sigma c(R))$

6.2. Projection

Descente de la projection que par rapport à l'union : de la même manière par rapport à la restriction, la projection appliquée au résultat de l'union devrait descendre vers la relation concernée par les attributs de la projection avant l'union.

$\Pi_L (R \cup S) = \Pi_L(R) \cup \Pi_L(S)$

Descente de la projection sous une jointure :

$(\Pi_L(R \blacktriangleright\blacktriangleleft_{A=B} S)) = (\Pi_L(\Pi_M(R) \blacktriangleright\blacktriangleleft_{A=B} \Pi_N(S)))$ Où M représente les attributs de L portant sur R, suivis de A et N les attributs de L portant sur S, suivis de B .

6.3. Priorité des opérateurs

Quand une expression combine plusieurs opérateurs, à défaut de parenthésage, les règles suivantes s'appliquent.

- Les opérateurs unaires ont la plus haute priorité. Entre eux, ils sont évalués de gauche à droite.
- Les opérateurs binaires sont évalués de gauche à droite.

6.4. Ensemble minimum d'opérateurs :

L'ensemble minium d'opérateur est l'ensemble des opérateur logiques permettant de déduire le reste des opérations et qui ne sont déduits par aucune autre opération. Cet ensemble est formé des opérations suivantes : Le produit cartésien, la différence, l'union, la restriction et la projection.

Toutes les autres opérations peuvent être déduites en utilisant ces cinq opérations de la manière suivante :

- $R \cap S = R - (R - S)$
- $S \blacktriangleright\blacktriangleleft_c R = \sigma_c(S \times R)$
- $S \blacktriangleright\blacktriangleleft R = \Pi_{X,S.Z,Y}(\sigma_{S.Z=R.Z}(S \times R))$
- $R \div S = \Pi_X(\Pi_X(R) - \Pi_X(\Pi_X(R) \times \Pi_A(S) - R))$

6.5. La Valeur NULL

Au niveau du modèle relationnel, l'absence de valeur au niveau d'un attribut est représentée par une valeur spécifique qu'on nomme NULL.

Définition
NULL = « Value unknown or nonexistent »

Les opérations algébriques se comportent différemment face à une valeur NULL. Car cette dernière n'est pas considérée comme étant une valeur. Les règles suivantes sont appliquées lors de l'application des différentes opérations algébriques :

1. Toute opération arithmétique (+,-,*,/) avec une valeur NULL renvoie NULL.
2. Toute comparaison (>,<,=,>=,<=,<>) avec la valeur NULL renvoie une valeur UNKNOWN.
3. Les opérateurs booléens se comportent comme suit :
   a. **and** : (true and unknown) = unknown
      (false and unknown) = false
      (unknown and unknown) = unknown
   b. **or** : (true or unknown) = true
      (false or unknown) = unknown
      (unknown or unknown) = unknown
   c. **not** : (not unknown) = unknown

Quel est le résultat des opérations algébriques appliquées à une valeur NULL ?

Restriction : Si le résultat de la condition est true la ligne est affichée sinon (false ou unknown) la ligne n'est pas retournée dans le résultat.

Jointure : La jointure est un produit cartésien suivi d'une restriction. Le résultat est le même que pour la restriction.

Projection : La projection traite le NULL comme une valeur quelconque.

Union, Intersection, différence : Idem que la projection.

Agrégation : Idem que la projection.

7. La Requête Algébrique

Le langage algébrique est un langage d'interrogation de bases de données qui est à la base du langage SQL. Les opérations de base de l'algèbre relationnelle constituent un langage complet. En combinant entre les différentes opérations on construit des requêtes algébriques dont la dernière opération exécutée fournie le résultat souhaité.

Prenant comme exemple le schéma relationnel suivant :

```
ACTEUR(NA,NOM,PRENOM,ADRESSE,SEXE)
VESTE(NV,MARQUE,COULEUR,TAILLE)
PORTE(NA,NV,DATE,DUREE)
```

On peut poser un certain ensemble de questions sur les données de ce schéma et y répondre avec des requêtes algébriques comme suit :

Q1 : Donner les marques des vestes de taille 32 et de couleur rouge

```
R1 = RESTRICT(VESTE,TAILLE=32)
R2 = RESTRICT(VESTE,COULEUR='ROUGE')
R3 = INTERSECT(R1,R2)
RESULT=PROJECT(R3,MARQUE)
```

Q2 : Donner les noms et prénoms des acteurs qui ont mis des vestes rouges ou bleues

```
R1 = RESTRICT(VESTE, COULEUR='ROUGE')
R2 = RESTRICT(VESTE,COULEUR='BLEU')
R3 = UNION(R1,R2)
R4 = JOIN(R3,PORTE)
R5 = JOIN(R4,ACTEUR)
RESULT=PROJECT(R5,NOM,PRENOM)
```

Q3 : Donner les noms et prénoms des acteurs qui ont mis des vestes de taille 32 plus de deux heures, avec la marque de la veste.

```
R1 = RESTRICT(VESTE, TAILLE=32)
R2 = RESTRICT(PORTE,DUREE>2)
R3 = JOIN(R1,R2)
R4 = PROJECT(R3,NA,MARQUE)
R5 = JOIN(R4,ACTEUR)
RESULT=PROJECT(R5,NOM,PRENOM,MARQUE)
```

7.1. Comment construire une requête algébrique ?

Pour construire une requête algébrique répondant à une question donnée, il faut suivre les étapes suivantes :

1. Identifier les relations utiles pour exprimer la requête,
2. Recopier le schéma de ces relations, et indiquer sur ces schémas :
   a. Les attributs qui font partie du résultat de la requête
   b. Les conditions portant sur les attributs
   c. Les liens entre les relations
3. Traduire cette figure en expression algébrique
   a. Faire les sélections selon les conditions portant sur les attributs,
   b. Faire les jointures (naturelles ou thêta) selon les liens entre les relation (une jointure par lien)
   c. Projeter sur les attributs qui font partie du résultat

Note

Cette méthode est valable pour la plupart des requêtes. Cependant, certains types de requêtes nécessitent de compliquer la méthode. C'est le cas des requêtes où la même relation est utilisée plusieurs fois avec des ensembles de tuples différents.

7.2. Arbre Algébrique

Toute requête algébrique peut être représentée sous forme arborescente avec un arbre dont les nœuds représentent les opérations algébriques et les arcs les relations de base ou temporaires représentant des flots de données entre opérations.

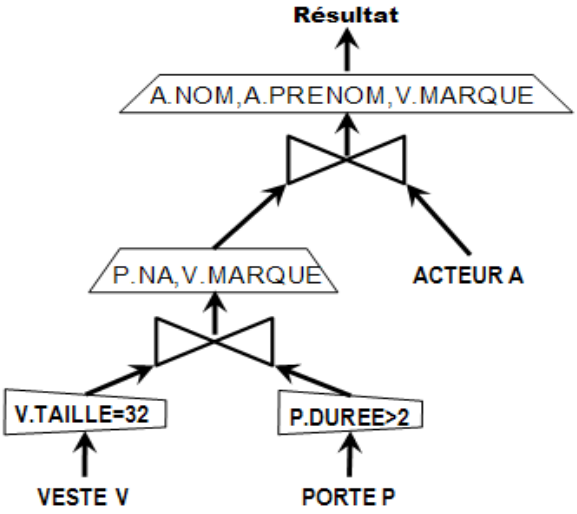


Figure 105 : Arbre Algébrique

Les arbres algébriques sont utilisés pour l'illustration des requêtes et lors de l'optimisation algébrique de ces dernières.

7.3. Fonctions et Agrégats

Au niveau des requêtes algébriques, on peut utiliser des fonctions, des expressions arithmétiques ainsi que des agrégats.

7.3.1. Fonction de calcul

On peut remplacer les attributs par des expressions d'attributs.

> ***Définition***
> *Une fonction de calcul est une expression arithmétique construite à partir d'attributs d'une relation et de constantes, par application de fonctions arithmétique successives.*

Exemple

R1 = JOIN(VESTE, PORTE,TAILLE*DUREE>DUREE/3)
R2 = RESTRICT(R1,DUREE*100/TAILLE>38)
RESULT=PROJECT(R2,NOM,TAILLE-TAILLE*DUREE/100)

7.3.2. Les Agrégats

Les agrégats sont des fonctions particulières utilisées pour effectuer des opérations de calcul sur des colonnes entières. Les fonctions les plus proposées sont :

- Somme (SUM) : Permet de calculer la somme des lignes d'une colonne donnée. Il faut que le type de la colonne permette la fonction de somme.

- Moyenne (AVG) : Permet de calculer la moyenne entre les valeurs d'une colonne d'une relation.
- Minimum (MIN) : Renvoie la valeur minimale de la colonne spécifiée.
- Maximum (Max) : Renvoie la valeur maximale de la colonne spécifiée.
- Compte (Count) : compte le nombre de ligne dans une relation.

En cherchant par exemple la moyenne des tailles des vestes du schéma relationnel précédent, on utilise la requête suivante exprimée de deux manières différentes :

Résultat=Agregat(Veste,AVG(Taille))
Résultat=AVG_{Marque}(Veste,Taille)

Les opérations d'agrégation ont la représentation graphique suivante :

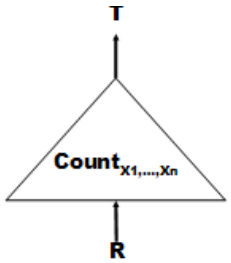


Figure 106 : Agrégat

Un agrégat peut être calculé pour toutes les lignes de la colonne spécifiée mais il peut également être regroupé par plusieurs valeurs d'un ou de plusieurs autres attributs. Par exemple, on peut calculer la moyenne des tailles des vestes regroupée par marque. L'attribut taille est appelé attribut d'agrégation et marque est l'attribut de groupage.

La colonne de la relation résultante d'une opération d'agrégation ne comporte pas de nom, il faut donc veiller à la renommer si on souhaite la réutiliser dans d'autres opérations.

8. Optimisation Algébrique

L'objectif de l'optimisation algébrique des requêtes est de créer un plan d'exécution optimal en termes de temps d'exécution et de mémoire utilisée et ceci en se basant sur les propriétés des opérateurs algébriques.

8.1. Lois et Règles Algébriques

Il existe huit lois algébriques qui découlent directement des propriétés intrinsèques des opérateurs algébriques. Ces lois sont à la base de l'algorithme d'optimisation.

8.1.1. Commutativité des Jointures

JOIN(R,S) ≡ JOIN(S,R)

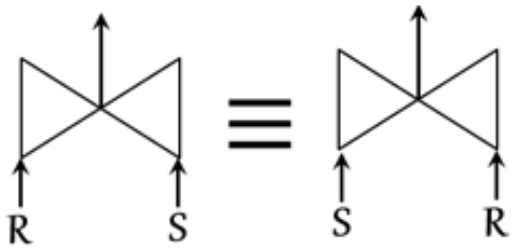


Figure 107 : Commutativité des jointures

8.1.2. Associativité des jointures

JOIN(JOIN(R,S),T) ≡ JOIN(R,JOIN(S,T))

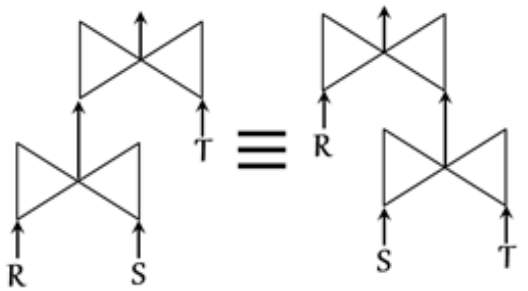


Figure 108 : Associativité des jointures

8.1.3. Groupabilité des restrictions

RESTRICT(RESTRICT(R,COND1),COND2) ≡
RESTRICT(R,COND1 AND COND2)

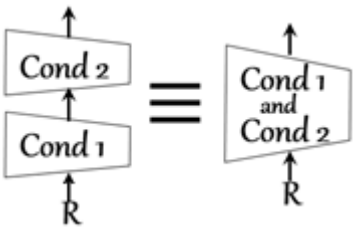


Figure 109 : Groupabilité des restrictions

8.1.4. Semi commutativité des projections et restrictions

PROJECT(RESTRICT(R,Ai=a),A1..AP) ≡
PROJECT(RESTRICT(PROJECT(R,Ai,A1…Ap),Ai=a),A1…Ap)

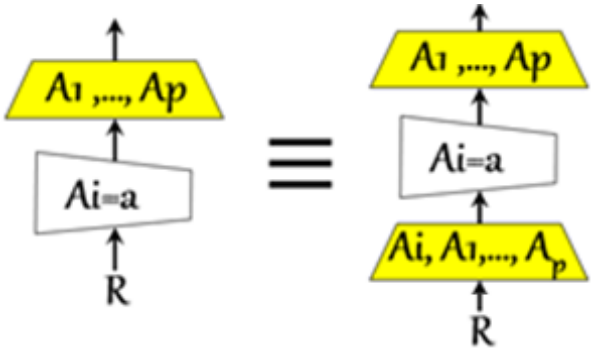


Figure 110 : Semi commutativité des projections et restrictions

8.1.5. Distributivité des restrictions sur les jointures

RESTRICT(JOIN(R,S),Ai=a) ≡ JOIN(RESTRICT(R,Ai=a),S) Si la condition porte sur R OU
RESTRICT(JOIN(R,S),Ai=a) ≡JOIN(R,RESTRICT(S,Ai=a)) Si la condition porte sur S

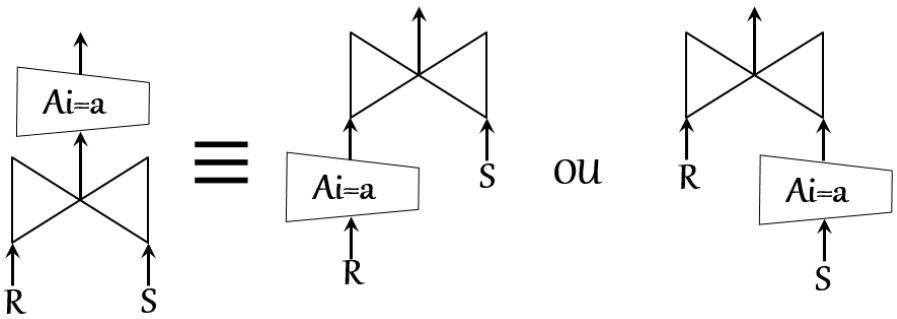


Figure 111 : Distributivité des restrictions sur les jointures

8.1.6. Semi distributivité des projections sur les jointures

PROJECT(JOIN(R,S,A=B),Ar1…Arp,As1…Asn) ≡
PROJECT(JOIN(PROJECT(RA,Ar1…Arp),PROJECT(S,B,As1…Asn),A=B))

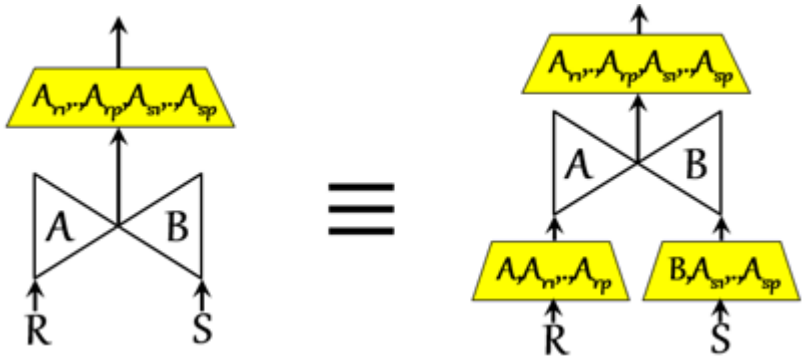


Figure 112 : Semi distributivité des projections sur les jointures

8.1.7. Distributivité des restrictions sur l'union et sur la différence

RESTRICT(UNION(R,S),Ai=a) ≡ UNION(RESTRICT(R,Ai=a),RESTRICT(S,Ai=a))

ET

RESTRICT(DIFFERENCE(R,S),Ai=a)
≡
DIFFERENCE(RESTRICT(R,Ai=a),RESTRICT(S,Ai=a))

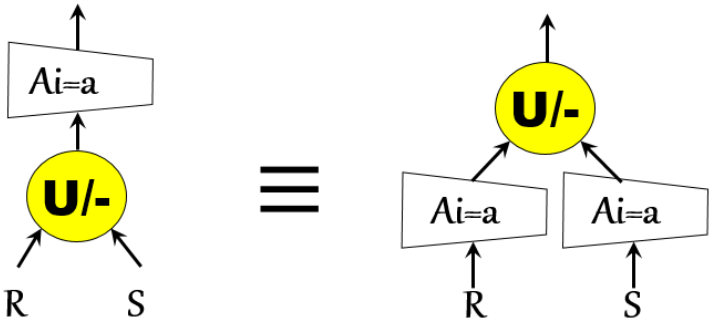


Figure 113 : Distributivité des restrictions sur l'union et sur la différence

8.1.8. Distributivité des projections sur l'union

PROJECT(UNION(R,S),A1,…,Ap)≡UNION(PROJECT(R,A1,…,Ap), PROJECT(S,A1,…,Ap))

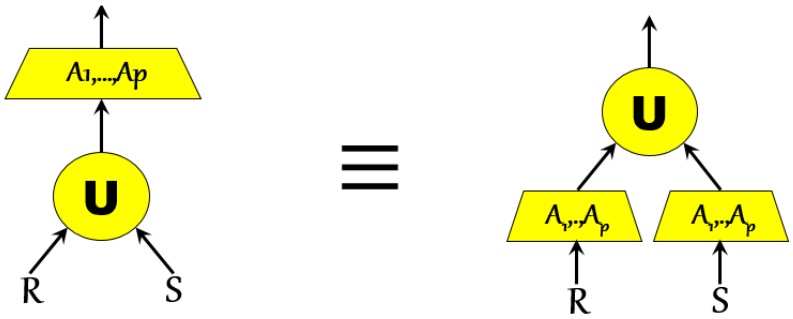


Figure 114 : Distributivité des projections sur l'union

8.2. Heuristique d'optimisation

Pour optimiser une requête algébrique, on applique les huit règles citées précédemment et en suivant l'heuristique suivante :

1. Décomposer les restrictions en restrictions Unaires (Règle 3)
2. Rapprocher les restrictions des feuilles (Règles 4,5 et 7)
3. Grouper les restrictions aux feuilles (Règle 3)
4. Rapprocher les projections des feuilles (Règles 4,6 et 8)
5. Ordonner les jointures afin de minimiser le temps (Règles 1 et 2)

Prenons l'exemple du schéma relationnel suivant :

Joueur(**NJ**,Nom,Prénom, Adr),

Jouer (**NJ,NS**,Date,Heure),

Stade(**NS**,Ville,Superficie).

Soit la requête suivante retournant les noms et prénoms des joueurs habitant à OuedSmar ayant joués le 01/01/15 à Alger.

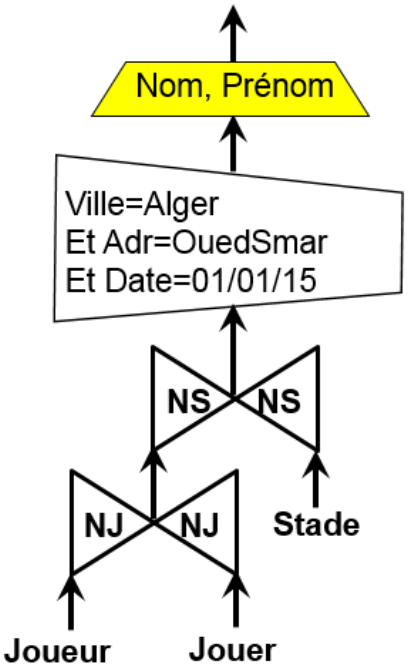


Figure 115 : Requête à optimiser

Il est évident que cette requête n'est pas optimisée. On va maintenant appliquer l'heuristique ci-dessous en appliquant les différentes règles dans l'ordre afin d'en obtenir une requête optimisée.

Le résultat du processus d'optimisation est présenté dans la requête suivante :

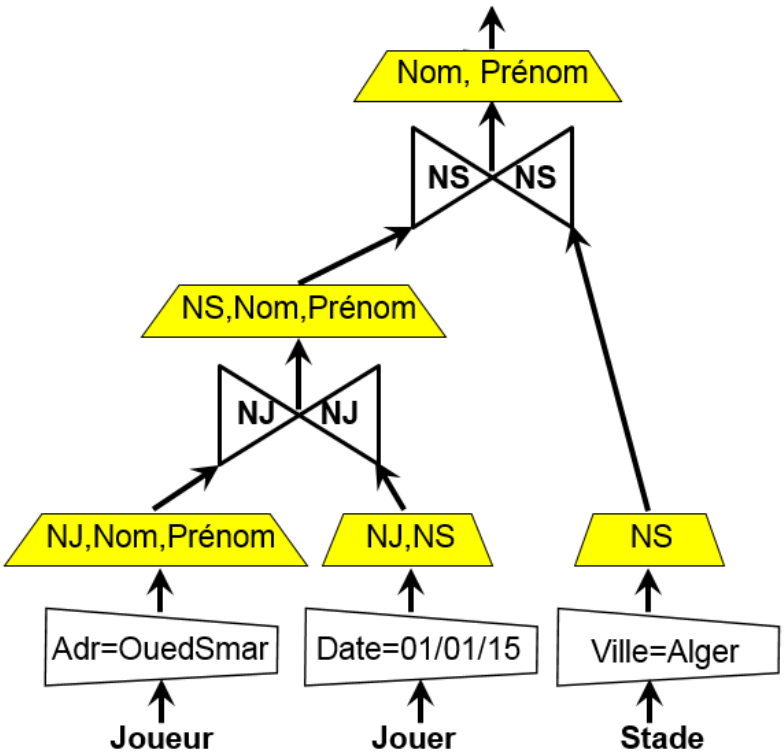


Figure 116 : Requête optimisée

9. Conclusion

L'algèbre relationnelle est un langage complet permettant la manipulation des données du modèle relationnel. Une requête algébrique constituée d'une suite d'opérations permet de répondre à n'importe quel type de question que l'on se pose sur les données.

L'algèbre relationnelle est à la base du langage SQL qui est implémenté par tous les SGBD relationnel existants. Il est certain que dans la pratique, le langage algébrique n'est pas utilisé mais son aspect pédagogique et méthodique permet de maitriser le processus de formulation de requête et surtout de comprendre le fonctionnement interne du moteur de requête ainsi que les règles d'optimisation utilisées par les SGBD afin d'optimiser les requêtes.

Chapitre 5 : SQL (Structured Query Language)

Objectifs

L'objectif de ce chapitre est d'apprendre à l'étudiant l'implémentation et l'administration d'une base de données ainsi que la manipulation des données et des utilisateurs en utilisant le langage SQL.

L'étudiant se familiarise avec les cinq parties du langage SQL et renforce ses connaissances grâce à une grande partie pratique sur le SGBD MySQL.

Résumé

Au niveau de ce chapitre nous étudions deux briques essentielles du langage SQL, à savoir : la DML et le DDL. Nous introduisons ainsi les commandes SQL de création, de modification et de suppression des objets de la base pour ensuite présenter les clauses permettant la manipulation des données. Une grande partie de ce chapitre est consacrée à l'interrogation des données avec la clause SELECT.

1. Introduction

Ce chapitre présente un résumé succinct des composantes principales du langage SQL selon la norme 92. Dans certain cas nous donneront des exemples sous le SGBD Open Source MYSQL.

Les différentes versions de SQL :

- SQL1 86: la base
- SQL1 89: l'intégrité
- SQL2 92: la nouvelle norme
- SQL3 99: les évolutions objets

SQL est dérivé de l'algèbre relationnelle et de SEQUEL, Il a été intégré à SQL/DS, DB2, puis ORACLE, INGRES, … La plupart des systèmes supportent SQL1 complet.

2. Composantes du langage SQL

Le SQL est composé de cinq grandes parties :

1. La définition des éléments d'une base de données ;
2. La manipulation des données,
3. La gestion des droits d'accès,
4. La gestion des transactions,
5. La programmation dynamique.

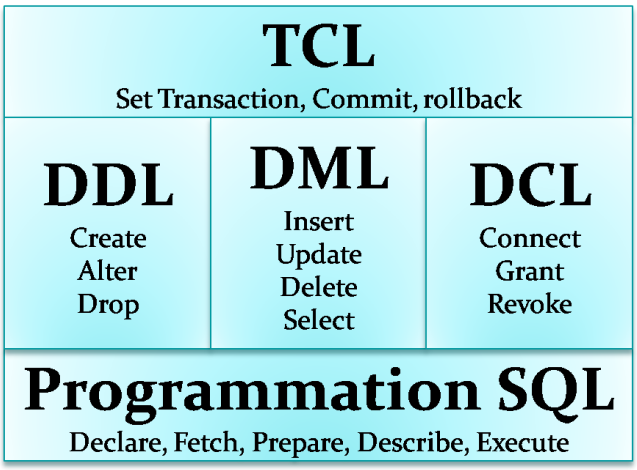


Figure 117 : Composantes du SQL

3. Data Definition Language (Langage de Définition des Données) :

Partie de SQL qui permet de créer des bases de données, des tables, des index, des contraintes, etc. Elle traite de la création des schémas de bases de données. Cette brique du SQL comporte trois clauses : CREATE, ALTER et DROP

3.1. Create Database

La requête create database est totalement dépendante du SGBD utilisé, car ça concerne la manière dont les données sont enregistrées physiquement sur le disque. C'est pourquoi, on ne retrouve pas beaucoup de détail dans la norme SQL 92.

La syntaxe sous MYSQL :

```
CREATE DATABASE [IF NOT EXISTS] db_name
[create_specification [, create_specification] ...]
create_specification:
[DEFAULT] CHARACTER SET charset_name
| [DEFAULT] COLLATE collation_name
```

Exemple

```
CREATE DATABASE IF NOT EXISTS mabase
```

La création d'une base de données induit deux actions principales :

1. Création physique des fichiers de la base de données
2. Insertion des métadonnées de description de la base dans la méta base du SQGB.

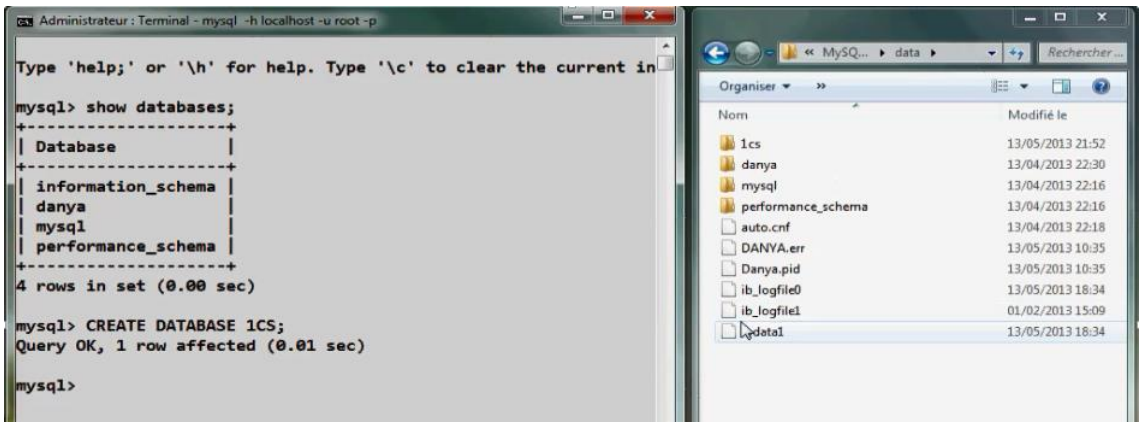


Figure 118 : Création d'une base de données sous MySQL

3.2. Create Table

Create table crée une table de nom NOM_TABLE dans la base de données courante. La syntaxe de cette instruction est respectée par tous les SGBD, la différence qu'on peut trouver est au niveau des types de colonnes utilisés.

```
CREATE TABLE nomtable(
{nomcolonne type [contrainte_colonne […]]
| contrainte _table} [,…]
)
```

Contrainte colonne peut être :

- NOT NULL
- UNIQUE
- PRIMARY KEY
- DEFAULT value

Contrainte_table peut être :

FOREIGN KEY(référence_colonne) REFERENCES référence_table(reference_colonne)

Exemple

```
CREATE TABLE avion(
num_avion  smallint primary key,
Type  varchar(10) not null,
constructeur  varchar(20) not null,
Capacite  smallint  check(capacite>0),
compagnie  varchar(30)  not null
);
```

3.3. DROP Database | Table

Drop est une instruction qui permet de supprimer une base de données ou une table. La suppression est dans la plupart des cas irréversible.

```
DROP DATABSE  mabase
DROP TABLE  matable
```

3.4. ALTER TABLE

Alter permet de modifier la structure d'une table. On peut ajouter, supprimer ou modifier des colonnes ou les index de la table.

```
ALTER TABLE nomtable {
        ADD CONSTRAINT contrainte
        DROP CONSTRAINT contrainte
        ADD COLUMN definitioncol
        DROP COLUMN colonne
        ALTER COLUMN{
                        SET DEFAULT valeur
                        DROP DEFAULT}
        }
```

Exemple

```
Alter  table  client
add column age int not null,
drop column nom_clt,
        alter column prenom drop default;
```

3.5. Les contraintes d'intégrité

Les contraintes permettent d'exprimer des conditions devant être respectées par tous les tuples d'une table. Les contraintes expriment les valeurs que peuvent prendre un attribut :

NOT NULL : l'attribut doit posséder une valeur

DEFAULT : valeur par défaut de l'attribut (quand il n'est pas défini)

UNIQUE : deux tuples ne peuvent pas avoir la même valeur pour cet attribut

CHECK : spécifie une condition devant être satisfaite par tous les tuples de la table.

PRIMARY KEY: Clé primaire.

FOREIGN KEY: Clé étrangère.

Exemple :

Le script SQL suivant permet la création de tables avec les différentes contraintes d'intégrité.

```
create table avion(
        num_avion smallint primary key,
        type varchar(10) not null,
        constructeur varchar(20) not null,
        capacite smallint check(capacite>0),
        compagnie varchar(30) not null,
        id_tav int references type_av(id_tp)
        );

create table avion(
        num_avion smallint,
        type varchar(10) not null,
        constructeur varchar(20) not null,
        capacite smallint,
        compagnie varchar(30) not null,
        id_tav int,
        Primary key(num_avion),
         check(capacite>0),
        Foreign key (id_tav) references type_av(id_tp)
        );

create table avion(
        num_avion smallint,
        type varchar(10) not null,
        constructeur varchar(20) not null,
        capacite smallint,
        compagnie varchar(30) not null,
        id_tav int,
        CONSTRAINT `01_001`Primary key
        (num_avion),
        check(capacite>0),
        CONSTRAINT `01_002`Foreign key (id_tav)
        references type_av(id_tp)
        );
```

3.6. Les index

Un index permet d'accélérer la recherche sur un champ ou un ensemble de champs.

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
CREATE INDEX part_of_name ON customer (name(10));
```

4. Data Manipulation Language (Langage de Manipulation des Données)

Partie de SQL qui traite les données (extraction, ajout, suppression, modification) dans les tables.

4.1. INSERT

INSERT permet d'insérer des lignes dans la table nom_table. On peut ne pas spécifier la définition de la table à condition que les valeurs correspondent aux champs de la table et dans l'ordre.

```
INSERT INTO nom_table(Définition de la table) VALUES (valeurs des colonnes)
Uniquement MySQL :
INSERT nom_table SET nom_col=valeur,…
```

Exemple:

```
INSERT INTO client(id_clt,nom_clt,prenom_clt) VALUES('21','MADJID','AMAR')
INSERT INTO client VALUES('21',AHMED','AMAR')
INSERT client SET(id_clt='21', nom_clt='AHMED', prenom_clt='AMAR' /* MySQL
//Insérer plusieurs lignes
INSERT INTO client(id_clt,nom_clt,prenom_clt) VALUES ('21',AHMED','AMAR'),
('21',MALIK',DJAMILA'), ('21',ILHEM',SAIDA'), ('21',ROUMI',MALIKA')
```

4.2. DELETE

DELETE permet de supprimer des lignes de la table nom_table. La suppression peut concerner toutes les lignes de la table comme on peut définir des critères de suppression dans la clause WHERE.

```
DELETE FROM nom_table WHERE condition
```

La clause **WHERE** est optionnelle, si on ne la spécifie pas on supprime toutes les liges de la table. Si non ce seront uniquement les lignes qui satisfont la clause WHERE qui seront supprimées.

Exemple :

```
Supprimer toutes les lignes
DELETE  FROM client
Supprimer uniquement les clients dont l'âge est 39
DELETE FROM client WHERE age_clt>39
Supprimer les clients dont le nom est égal au prénom
DELETE FROM client WHERE nom_clt=prenom_clt
```

4.3. UPDATE

UPDATE permet de modifier des données d'une table. La modification peut concerner toutes les lignes de la table comme on peut définir des critères de modification dans la clause WHERE.

```
UPDATE nom_table SET nom_col=valeur,… WHERE condition
```

La clause **WHERE** est optionnelle, si on ne la spécifie pas on modifie toutes les liges de la table. Si non ce seront uniquement les lignes qui satisfont la clause WHERE qui seront modifiées.

Exemple :

```
Ajouter un point à tous les étudiants
UPDATE etudiant SET note=note+1
Ajouter un point uniquement aux étudiants dont la note est inférieure à 10
UPDATE etudiant SET note=note+1 WHERE note<10
Modifier la note telle que note est la moyenne entre TP et Note quand TP est supérieur à Note
UPDATE etudiant SET note=(note+tp)/2 WHERE note<tp
```

4.4. Select

La commande **SELECT** permet d'interroger la base de données, elle existe sous plusieurs formes et pour bien illustrer son fonctionnement nous allons utiliser dans toute la suite de ce cours la base de données suivante représentée avec son script MYSQL :

```
DROP DATABASE IF EXISTS `dbexemple`;
CREATE DATABASE `dbexemple`;
USE `dbexemple`;

DROP TABLE IF EXISTS `personne`;
CREATE TABLE `personne` (
  `id` varchar(10) NOT NULL,
  `nom` varchar(30) NOT NULL,
  `pnom` varchar(30) NOT NULL,
  `daten` datetime default NULL,
  `sexe` varchar(10) default NULL,
  `id_p` varchar(10) default NULL,
  `id_m` varchar(10) default NULL,
  `nbcnj` int(11) default '0',
  PRIMARY KEY  (`id`),
  FOREIGN KEY (`id_p`) REFERENCES `personne` (`id`),
  FOREIGN KEY (`id_m`) REFERENCES `personne` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS `wilaya`;
CREATE TABLE `wilaya` (
  `id` varchar(2) NOT NULL,
  `lib` varchar(30) default NULL,
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS `mariage`;
CREATE TABLE `mariage` (
  `id_epx` varchar(10) NOT NULL default '',
  `id_eps` varchar(10) NOT NULL default '',
  `datem` datetime default NULL,
  `wilm` varchar(2) default NULL,
  PRIMARY KEY (`id_epx`,`id_eps`),
  KEY `id_eps` (`id_eps`),
  FOREIGN KEY (`id_epx`) REFERENCES `personne` (`id`),
  FOREIGN KEY (`id_eps`) REFERENCES `personne` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
DROP TABLE IF EXISTS `resider`;
CREATE TABLE `resider` (
  `id_epx` varchar(10) NOT NULL default '',
  `id_eps` varchar(10) NOT NULL default '',
  `dater` datetime NOT NULL default '0000-00-00 00:00:00',
  `wilr` varchar(2) NOT NULL default '',
  `dated` datetime default NULL,
```

```
  PRIMARY KEY  (`id_epx`,`id_eps`,`wilr`,`dater`),
  KEY `id_eps` (`id_eps`),
  KEY `wilr` (`wilr`),
  FOREIGN KEY (`id_epx`,`id_eps`) REFERENCES `mariage` (`id_epx`,`id_eps`),
  FOREIGN KEY (`wilr`) REFERENCES `wilaya` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

4.4.1. Notations :

1. Tout ce qui est entre des crochets est facultatifs […] ;
2. La barre verticale | représente un choix 'ou' ;
3. Les trois points … représentent une suite d'éléments ;

4.4.2. Recherche de base

La recherche de base permet d'effectuer des opérations de projection et de restriction et la combinaison des deux.

```
Select [ALL | DISTINCT] champ1 [AS new_name], champs2 …
From nom_table [AS Alias]
[Where condition]
```

Dans la clause select on indique les colonnes sur lesquelles se fait la projection. Chaque colonne peut être renommée avec l'opérateur ***AS.*** Les colonnes sont séparées par des virgules. Dans le cas où on veut sélectionner toutes les colonnes des tables mentionnées dans la clause From on utilise le caractère astérisque (*). Pour différencier les colonnes des tables on mentionne le nom de la colonne précédée par le nom (ou alias) de la table comme suit : nom_table.nom_colonne. Par défaut ALL est utilisé mais si on souhaite ne pas afficher les doublons on utilise DISTINCT.

La clause **From** permet de spécifier les tables dans lesquelles la recherche sera effectuée, des alias peuvent être déclarés avec l'opérateur ***AS.***

La condition de la clause Where permet de spécifier les critères de sélection. Ces critères sont extrêmement riches dans SQL mais nous nous contenterons de présenter ici uniquement les plus utilisés.

Exemples de requêtes simples avec leurs résultats sur la base de données exemple :

```
mysql> Select *
    -> From Resider;
+--------+--------+---------------------+------+-------+
| id_epx | id_eps | dater               | wilr | dated |
+--------+--------+---------------------+------+-------+
| 1445   | 1289   | 2003-01-11 00:00:00 | 16   | NULL  |
| 28     | 3474   | 2004-01-11 00:00:00 | 10   | NULL  |
| 8834   | 3435   | 2004-01-11 00:00:00 | 16   | NULL  |
+--------+--------+---------------------+------+-------+
```

Au niveau de la clause Where, les conditions suivantes sont possibles :

[NOT] condition de base
Condition between
Condition in
Condition like
Condition null
Condition AND | OR Condition

Condition de base :

Colonne = | <> | < | <= | > | >= Constante

```
mysql> Select id_epx,id_eps,dater
    -> From Resider
    -> Where (wilr='16')And(dater>'2003-01-01');
+--------+--------+---------------------+
| id_epx | id_eps | dater               |
+--------+--------+---------------------+
| 1445   | 1289   | 2003-01-11 00:00:00 |
| 8834   | 3435   | 2004-01-11 00:00:00 |
+--------+--------+---------------------+
```

Condition Between :

L'opérateur Between permet de tester l'appartenance à une plage de valeurs.

Colonne Between x and y

```
mysql> Select id_epx,id_eps,dater
    -> From Resider
    -> Where dater Between '2003-01-01' And '2004-01-01';
+--------+--------+---------------------+
| id_epx | id_eps | dater               |
+--------+--------+---------------------+
| 1445   | 1289   | 2003-01-11 00:00:00 |
+--------+--------+---------------------+
```

Condition IN:

L'opérateur IN permet de tester l'appartenance à un ensemble de valeurs. L'ensemble peut être spécifié explicitement comme il peut être le résultat d'une autre requête imbriquée.

Colonne IN (v1,v2,…,vn)

```
mysql> Select id_epx,id_eps,dater
    -> From Resider
    -> Where wilr in ('01','10','06');
+--------+--------+---------------------+
| id_epx | id_eps | dater               |
+--------+--------+---------------------+
| 28     | 3474   | 2004-01-11 00:00:00 |
+--------+--------+---------------------+
```

Condition Like :

L'opérateur like permet de comparer des chaînes de caractères.

Colonne [not] like 'modèle de chaîne'

Modèle de chaîne peut contenir n'importe quel caractère plus deux caractères spéciaux : '-' (au niveau de mysql c'est le caractère '_' qui est utilisé à la place du '-') remplace un caractère et '%' remplace plusieurs caractères.

```
mysql> select id,nom,pnom
    -> From personne
    -> Where nom like 'a%';
+------+------+---------------+
| id   | nom  | pnom          |
+------+------+---------------+
| 3435 | ABBA |               |
| 8720 | ABBA | MOHAMED RABIE |
+------+------+---------------+
mysql> select id,nom,pnom
    -> From personne
    -> Where nom like 'A__A';
+------+------+---------------+
| id   | nom  | pnom          |
+------+------+---------------+
| 3435 | ABBA |               |
| 8720 | ABBA | MOHAMED RABIE |
+------+------+---------------+
```

Condition Null :

Permet de vérifier si une colonne est null ou non.

```
Colonne IS [NOT] NULL
```

```
mysql> select *
    -> From Resider
    -> Where dated is NULL;
+--------+--------+---------------------+------+-------+
| id_epx | id_eps | dater               | wilr | dated |
+--------+--------+---------------------+------+-------+
| 1445   | 1289   | 2003-01-11 00:00:00 | 16   | NULL  |
| 28     | 3474   | 2004-01-11 00:00:00 | 10   | NULL  |
| 8834   | 3435   | 2004-01-11 00:00:00 | 16   | NULL  |
+--------+--------+---------------------+------+-------+
```

4.4.3. Recherche avec jointure

Il existe plusieurs manières d'exprimer des jointures entre plusieurs tables avec SQL. Les jointures vues dans l'algèbre relationnelles sont toutes implémentées dans pratiquement tous les SGBD avec quelques différences de syntaxe pour certains. La syntaxe globale de la jointure est la suivante :

```
reference_table, reference_table
reference_table [CROSS] JOIN reference_table
reference_table [INNER] JOIN reference_table ON condition_jointure
reference_table NATURAL JOIN reference_table
reference_table LEFT | RIGHT | FULL [OUTER] JOIN reference_table ON condition_jointure
reference_table NATURAL [LEFT | RIGHT | FULL [OUTER]] JOIN reference_table
```

Produit cartésien :

On peut exprimer le produit cartésien avec une simple requête entre deux tables sans la clause where (ligne 1 de la syntaxe) et avec la clause CROSS JOIN (Ligne 2).

```
mysql> select Mariage.*
    -> From Mariage,Resider;
+--------+--------+---------------------+------+
| id_epx | id_eps | datem               | wilm |
+--------+--------+---------------------+------+
| 1445   | 1289   | 2002-12-11 00:00:00 | 38   |
| 1445   | 1289   | 2002-12-11 00:00:00 | 38   |
| 1445   | 1289   | 2002-12-11 00:00:00 | 38   |
| 213    | 5774   | 1973-12-22 00:00:00 | 40   |
```

```
mysql> select Mariage.*
    -> From Mariage Cross Join Resider;
+--------+--------+---------------------+------+
| id_epx | id_eps | datem               | wilm |
+--------+--------+---------------------+------+
| 1445   | 1289   | 2002-12-11 00:00:00 | 38   |
| 1445   | 1289   | 2002-12-11 00:00:00 | 38   |
| 1445   | 1289   | 2002-12-11 00:00:00 | 38   |
| 213    | 5774   | 1973-12-22 00:00:00 | 40   |
```

Thêta jointure :

La thêta-jointure peut être exprimée avec une requête simple avec une condition sur les colonnes communes des deux tables dans la clause Where, ou bien en utilisant la clause INNER JOIN (ligne 3).

```
mysql> Select P.nom,P.pnom
    -> From Personne P, Resider R
    -> Where P.id=R.id_epx;
+-----------+-----------+
| nom       | pnom      |
+-----------+-----------+
| ZERIMECHE | MOAHMED   |
| REKAB     | MED YAZID |
| TEBBAKH   | EL HAIN   |
+-----------+-----------+
```

```
mysql> Select P.nom,P.pnom
    -> From Personne P Inner Join Resider R On P.id=R.id_epx;
+-----------+-----------+
| nom       | pnom      |
+-----------+-----------+
| ZERIMECHE | MOAHMED   |
| REKAB     | MED YAZID |
| TEBBAKH   | EL HAIN   |
+-----------+-----------+
```

On peut utiliser USING(liste des colonnes communes)à la place de ON dans le cas d'une équijointure.

Jointure naturelle :

La jointure naturelle peut être exprimée avec une thêta-jointure en mentionnant dans la clause USING toutes les colonnes en commun et dans la clause SELECT toutes les colonnes une seule fois (donc pas de *). Ou bien en utilisant la clause NATURAL JOIN (ligne 4)

```
mysql> Select Mariage.*
    -> From Mariage Natural Join Resider;
+--------+--------+---------------------+------+
| id_epx | id_eps | datem               | wilm |
+--------+--------+---------------------+------+
| 28     | 3474   | 2009-04-07 00:00:00 | 37   |
| 1445   | 1289   | 2002-12-11 00:00:00 | 38   |
| 8834   | 3435   | 1993-10-09 00:00:00 | 42   |
+--------+--------+---------------------+------+
```

Jointure Externe :

On distingue trois types de jointures externes : la jointure externe à gauche, la jointure externe à droite et la jointure externe globale. Elles sont obtenues avec la clause left | right | full outer join …ON condition (ligne 5). Dans la plupart des SGBD le mot clé OUTER est facultatif comme pour le mot clé INNER.

```
mysql> Select Mariage.*
    -> From Mariage Left Outer Join Resider Using(id_eps,id_epx);
+--------+--------+---------------------+------+
| id_epx | id_eps | datem               | wilm |
+--------+--------+---------------------+------+
| 1445   | 1289   | 2002-12-11 00:00:00 | 38   |
| 213    | 5774   | 1973-12-22 00:00:00 | 40   |
| 213    | 896    | 1979-04-30 00:00:00 | 35   |
| 28     | 3474   | 2009-04-07 00:00:00 | 37   |
| 3236   | 7172   | 1997-04-13 00:00:00 | 22   |
```

4.4.4. Recherche avec Tri du résultat

Pour trier le résultat d'une recherche on utilise la clause ORDER BY juste après la clause Where.

```
Order By colonne [desc]
```

Desc est spécifié dans le cas d'un tri décroissant.

```
mysql> Select id,nom,pnom
    -> From Personne
    -> Where id>20
    -> Order By nom desc;
+------+------------+----------------+
| id   | nom        | pnom           |
+------+------------+----------------+
| 7712 | ZOUFOUL    | NAAMA          |
| 28   | ZERIMECHE  | MOAHMED        |
| 8834 | TEBBAKH    | EL HAIN        |
| 1289 | RIMOUCHE   | HOUSNA         |
```

4.4.5. Les expressions SQL

Les expressions SQL permettent d'enrichir les clauses de sélection de la clause SELECT et les conditions de la clause WHERE. Nous ne donnerons pas tous les types d'expressions existants, nous présenterons uniquement quelques exemples d'utilisation de ces expressions sur notre base de données exemple.

```
mysql> Select id,nom,pnom,(datediff(now(),daten) DIV 365) As Age
    -> From Personne;
+------+------------+----------------+------+
| id   | nom        | pnom           | Age  |
+------+------------+----------------+------+
| 110  | BOUFENARA  | REBAI          |   20 |
| 1289 | RIMOUCHE   | HOUSNA         |    1 |
| 1445 | REKAB      | MED YAZID      |   37 |
| 213  | DAHRI      | KHEIRDDINE     |   24 |
```

4.4.6. Groupement de lignes

Les expressions utilisées dans la clause Select et Where font des comparaisons entre les valeurs des colonnes d'une ligne donnée. Mais des fois on a besoin d'effectuer des calculs sur des groupes de lignes comme par exemple l'âge moyen des personnes mariées par sexe…etc. Dans SQL la clause GROUP BY nous permet d'effectuer ces calculs. La clause Group by doit être utilisée quand on utilise des fonctions de calcul au niveau de la clause Select, sinon le résultat n'aurait aucune signification. La clause HAVING nous permet d'exprimer des conditions sur le groupe de lignes généré.

```
Select colonnes_de_groupement, fonction(colonne)…
From nom_table
Where condition
Groupe by colonne_de_groupement
Having condition_sur_fonction(colonne)
```

```
mysql> Select Count(*) From Personne;
+----------+
| Count(*) |
+----------+
|       28 |
+----------+

mysql> Select sexe,count(id) From Personne Group By Sexe;
+----------+-----------+
| sexe     | count(id) |
+----------+-----------+
| FEMININ  |        17 |
| MASCULIN |        11 |
+----------+-----------+
mysql> Select Sexe, Avg(datediff(now(),daten) DIV 365) As MoyenneAge
    -> From Personne
    -> Group By Sexe
    -> Having MoyenneAge>20;
+----------+------------+
| Sexe     | MoyenneAge |
+----------+------------+
| MASCULIN |    22.0000 |
+----------+------------+
```

4.4.7. Les requêtes imbriquées

Un des concepts de SQL qui contribue grandement à sa puissance et sa souplesse est la sélection imbriquée : une sélection imbriquée n'est rien d'autre qu'un bloc de qualification SFW encapsulé à l'intérieur d'un autre bloc de qualification. On peut utiliser l'opérateur **IN**.

```
mysql> Select *
    -> From Wilaya
    -> Where id in (Select wilr from resider);
+----+--------+
| id | lib    |
+----+--------+
| 10 | BOUIRA |
| 16 | ALGER  |
+----+--------+
```

Dans ce cas le SGBD commence d'abord par traiter la requête interne. Le résultat renvoyé est ensuite remplacé dans la requête externe.

Le résultat de l'évaluation de la condition IN est vrai si et seulement si la valeur de l'expression à gauche du IN est égale à au moins une des valeurs du résultat de la requête à droite de IN.

Le résultat est faux si la valeur de l'expression à gauche de IN est différente de toutes les valeurs du résultat de la droite.

Condition ALL|ANY|SOME

Dans la clause Where on utilise des comparaisons entres des colonnes de plusieurs tables et des constantes ou bien d'autres colonnes. Dans un cas général, on peut utiliser des sous requêtes retournant une seule valeur au niveau des conditions. Mais quand les sous requêtes retournent plus d'une valeur il faut utiliser les opérateurs ALL, ANY ou SOME.

ALL renvoie vrai uniquement si la comparaison est vraie pour toutes les valeurs retournées par la sous requête ou bien le résultat de la sous requête est vide.

ANY renvoie vrai quand la condition indiquée est vraie pour au moins une des valeurs retournées par la sous requête.

Exemple

```
mysql> Select Count(id)
    -> From Personne
    -> Where (datediff(now(),daten) DIV 365)>
    -> (Select Avg(datediff(now(),daten) DIV 365) From Personne);
+-----------+
| Count(id) |
+-----------+
|        15 |
+-----------+
mysql> Select *
    -> From Resider
    -> Where wilr>ANY(Select id From Wilaya Where lib like 'b%');
+--------+--------+---------------------+------+-------+
| id_epx | id_eps | dater               | wilr | dated |
+--------+--------+---------------------+------+-------+
| 1445   | 1289   | 2003-01-11 00:00:00 | 16   | NULL  |
| 28     | 3474   | 2004-01-11 00:00:00 | 10   | NULL  |
| 8834   | 3435   | 2004-01-11 00:00:00 | 16   | NULL  |
+--------+--------+---------------------+------+-------+
```

La condition EXISTS

La condition EXISTS renvoi vrai quand l'évaluation de la sous requête n'est pas vide.

```
mysql> Select Count(*)
    -> From Personne
    -> Where Exists (Select id_epx
    ->                From Mariage
    ->                Where Mariage.id_epx=Personne.id);
+----------+
| Count(*) |
+----------+
|       11 |
+----------+
```

4.5. Les vues

> ***Définition***
> *Une vue est une table virtuelle dont le contenu est défini par une requête.*

Une vue ressemble à une table réelle, avec un ensemble de colonnes nommées et de lignes de données. Toutefois, une vue n'existe pas en tant qu'ensemble de valeurs de données stocké dans une base de données. Les lignes et les colonnes de données proviennent de tables référencées dans la requête qui définit la vue et sont produites dynamiquement lorsque la vue est référencée.

Une vue fait office de filtre sur les tables sous-jacentes qui y sont référencées. La requête qui définit la vue peut émaner d'une ou de plusieurs tables ou d'autres vues de la base de données en cours ou d'une autre base de données locale ou distante. Les requêtes distribuées peuvent également être employées pour définir des vues qui utilisent des données issues de plusieurs sources hétérogènes. Cela est particulièrement utile si vous souhaitez combiner des données de structure similaire issues de différents serveurs, dont chacun héberge des données relatives à une région différente de votre organisation.

L'interrogation au moyen de vues ne fait l'objet d'aucune restriction, et la modification de données par le biais de vues, de quelques-unes seulement.

Création d'une Vue

La création d'une vue se fait avec la clause Create comme tout autre objet de la base de données. La requête SELECT est introduite juste après le mot clé CREATE, dans certain cas on peut spécifier les colonnes en sortie après le nom de la vue.

```
mysql> Create View HommeMaries
    -> As
    -> Select * From Personne
    -> Where sexe='Masculin' And
    -> id IN(Select id_epx From Mariage);

mysql> Select Id,Nom,Pnom
    -> From HommeMaries;
+------+------------+---------------+
| id   | nom        | pnom          |
+------+------------+---------------+
| 1445 | REKAB      | MED YAZID     |
| 213  | DAHRI      | KHEIRDDINE    |
| 28   | ZERIMECHE  | MOAHMED       |
| 3236 | BELHADJ    | EL KHEIR      |
```

Modification d'une vue : On ne peut modifier une vue que si :

1. Elle ne référence qu'une seule table,
2. Tous les champs requis dans la source sont dans le Select de la vue.

3. Les valeurs mentionnées respectent les conditions du ***where*** de la vue dans le cas d'utilisation de l'option ***With Check Option***.

Suppression d'une vue : Drop view nom_vue ;

5. Conclusion

Le langage SQL est un langage implémenté pas tous les SGBD relationnels et relationnels objet. La différence entre les SGBD réside au niveau de la dernière brique concernant la programmabilité SQL où chaque SGBD utilise sons propre langage. Mais, tous les SGBD implémentent les mêmes concepts comme les procédures stockées, les fonctions, les triggers etc.

Nous avons présenté dans ce chapitre les deux briques principales de SQL, à savoir le DDL et le DML. Il appartient aux étudiants de se documenter sur les autres composantes dans l'objectif de les utiliser plus tard dans la réalité tout en se spécialisant dans un SGBD particulier afin de maitriser ses particularités.

Conclusion Générale

Ce support de cours spécialement conçu pour les étudiants de l'ESI reprend les trois paliers du cycle de vie d'une base de données, à savoir : le niveau conceptuel avec la modélisation Entité/Association (Chapitre 2), le niveau logique avec le modèle relationnel et l'algèbre relationnelle (Chapitre 3 et Chapitre 4) et le niveau physique avec le langage SQL (Chapitre 5).

La partie conceptuelle a été très bien développée en détaillant les différents principes et concepts du modèle ainsi que des astuces d'application dans la réalité. Le modèle relationnel a également été traité dans sa globalité. Par contre, au niveau du langage SQL seules deux briques ont été abordées (le DDL et le DML). Les autres briques du SQL sont traitées au niveau du cours bases de données avancées en 2éme Année cycle supérieur.

Même si pas mal d'exemples sont donnés dans ce support mais il reste un support théorique. Et en tant que tel, le côté pratique reste insuffisant. Il faut donc pratiquer en traitant les différents exercices fournis dans les séries et essayer de modéliser des situations réelles en appliquant les orientations de ce cours.

Comme il existe une panoplie de livres traitant des niveaux logiques et physiques, il serait intéressant de développer la partie conceptuelle d'avantage pour en faire un ouvrage dédié avec des illustrations et des exemples plus complets et complexes. Ceci sera mon prochain projet.

Bibliographie

1. Abrams, Marshall D., Sushil G. Jajodia, and Harold J. Podell. Information security: an integrated collection of essays. IEEE Computer Society Press, 1995.
2. Adiba, M., Demolombe, R., Gardarin, G., Scholl, M., Rolland, C., & Rohmer, J. (1986). Nouvelles perspectives des bases de données.
3. Brouard, Frédéric, Rudi Bruchez, and Christian Soutou. SQL. Pearson Education France, 2012.
4. Gardarin, Georges, et al. Bases de données : les systèmes et leurs langages. Eyrolles, 1985.
5. Lewis, Philip M., Arthur J. Bernstein, and Michael Kifer. Databases and transaction processing: an application-oriented approach. Addison-wesley, 2002.
6. Navarro, Laurent. Optimisation des bases de données : mise en œuvre sous Oracle. Pearson Education France, 2010.
7. Nguyen, D. H., and Andreas Meier. Introduction pratique aux bases de données relationnelles. Springer Science & Business Media, 2006.
8. Roy, Gilles. Conception de bases de données avec UML. PUQ, 2007.
9. Silberschatz, Abraham, Henry F. Korth, and Shashank Sudarshan. Database system concepts. Vol. 4. New York: McGraw-Hill, 1997.
10. Simsion, Graeme, and Graham Witt. Data modeling essentials. Morgan Kaufmann, 2004.
11. Teorey, Toby J. "Database modeling and design: the entity-relationship approach." (1990).
12. Thalheim, Bernhard. Entity-relationship modeling: foundations of database technology. Springer Science & Business Media, 2013.

Annexe 1 : Les Fonctions dans SQL 92

Domaine	Fonction	Description
Agrégation statistique	AVG	Moyenne
	COUNT	Nombre
	MAX	Maximum
	MIN	Minimum
	SUM	Total
Fonction "système"	CURRENT_DATE	Date courante
	CURRENT_TIME	Heure courante
	CURRENT_TIMESTAMP	Date et heure courante
	CURRENT_USER	Utilisateur courant
	SESSION_USER	Utilisateur autorisé
	SYSTEM_USER	Utilisateur système
Fonctions générales	CAST	Transtypage
	COALESCE	Valeur non NULL
	NULLIF	Valeur NULL
	OCTET_LENGTH	Longueur en octet
Fonctions de chaînes de caractères	||	Concaténation
	CHAR_LENGTH	Longueur d'une chaîne
	CHARACTER_LENGTH	Longueur d'une chaîne
	COLLATE	Substitution à une séquence de caractères
	CONCATENATE	Concaténation
	CONVERT	Conversion de format de caractères
	LIKE (prédicat)	Comparaison partielle
	LOWER	Mise en minuscule
	POSITION	Position d'une chaîne dans une sous chaîne
	SUBSTRING	Extraction d'une sous chaîne
	TRANSLATE	Conversion de jeu de caractères
	TRIM	Suppression des caractères inutiles
	UPPER	Mise en majuscule
Fonctions de chaînes de bits	BIT_LENGTH	Longueur en bit
Fonctions numériques	+ - * / ( )	Opérateurs et parenthésage
Fonctions temporelles	EXTRACT	Partie de date
	INTERVAL (opérations sur)	Durée
	OVERLAPS (prédicat)	Recouvrement de période
Prédicat, opérateurs et structures diverses	CASE	Structure conditionnelle
	IS [NOT] TRUE	Vrai
	IS [NOT] FALSE	Faux
	IS [NOT] UNKNOWN	Inconnu

	IS [NOT] NULL	NULL
	INNER JOIN	Jointure interne
	LEFT, RIGHT, FULL OUTER JOIN	Jointure externe
	NATURAL JOIN	Jointure naturelle
	UNION JOIN	Jointure d'union
	LEFT, RIGHT, FULL OUTER NATURAL JOIN	Jointure naturelle externe
	INTERSECT	Intersection (ensemble)
	UNION	Union (ensemble)
	EXCEPT	Différence (ensemble)
	[NOT] IN	Liste
	[NOT] BETWEEN	Fourchette
	[NOT] EXISTS	Existence
	ALL	Comparaison à toutes les valeurs d'un ensemble
	ANY / SOME	Comparaison à au moins une valeur de l'ensemble
	UNIQUE	Existence sans doublons
	MATCH UNIQUE	Correspondance

Annexe 2 : Outils Pédagogiques

1. Functional Dependencies Engine

FDE est un outil pédagogique développé par Fouad DAHAK (http://dahak.esi.dz) permettant de gérer les dépendances fonctionnelles et la normalisation des relations du modèle relationnel.

En partant d'un ensemble de dépendances définies par l'utilisateur dans un langage simple, FDE génère les éléments suivants :

1. La fermeture transitive de l'ensemble
2. La couverture minimale de l'ensemble
3. La fermeture transitive d'un ensemble d'attributs spécifiés par l'utilisateur
4. La liste des sources et des puits
5. La relation universelle
6. Les clés candidates de la relation universelle
7. La forme normale d'une relation
8. La décomposition d'une relation en 2ème forme normale
9. La décomposition d'une relation en 3ème forme normale
10. La décomposition d'une relation en forme normale de Boyce Codd
11. FDE permet de visualiser les étapes de chaque algorithme afin d'apprendre et de comprendre les différents processus.

FDE permet également d'afficher les étapes détaillées de chacun de ces algorithmes. Il est utilisé par les étudiants pour confronter leurs solutions et leur permettre de se corriger lors de la résolution des exercices liés au modèle relationnel.

FDE est utilisé au niveau du chapitre 3 sur le modèle relationnel et la normalisation des relations avec les dépendances fonctionnelles.

2. Free Relational Algebra Interpreter

FRAI est un interpréteur graphique pour l'algèbre relationnelle supportant toutes les opérations algébriques et conçu dans un but pédagogique. Il permet aux étudiants désireux de se familiariser avec le langage algébrique de comprendre le fonctionnement des opérations algébriques et la construction de requêtes. Ceci via une interface intuitive et très facile à utiliser.

La version actuelle offre un ensemble de fonctionnalités qui simplifient la manipulation des relations.

Frai est le premier outil dans son genre...

Toutes les opérations algébriques sont supportées par FRAI : Union, intersection, division, différence, projection, restriction, renommage, produit cartésien, thêta-jointure, jointure naturelle, semi-jointure, jointure externe ainsi que l'imbrication d'opérations, les fonctions de calcul au niveau de la projection et tout type de condition au niveau de la restriction et de la thêta-jointure. Les cinq agrégats de base sont également implémentés : COUNT, SUM, AVG, MAX et MIN.

Ce logiciel a été réalisé par Fouad DAHAK (Enseignant-Chercheur à l'ESI) comme support pédagogique au cours Bases de données au niveau de l'Ecole Nationale Supérieure d'Informatique d'Alger.