Licence 3 Informatique

Finance

Histoire

Géographie

Informatique

Mathématiques

Sciences Pour l'Ingénieur

Français Langue Étrangère

Administration Économique et Sociale

Diplôme d'accès aux études universitaires

Licence mention Informatique

CTU
Centre de Télé-enseignement Universitaire
http://ctu.univ-fcomte.fr

Filière Informatique

VVI5EBD

Bases de données

Mme DAMY - SYLVIE
sylvie.damy@univ-fcomte.fr

UNIVERSITÉ DE FRANCHE-COMTÉ
UBFC
UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ

Cette brochure a été réalisée en LaTeX 2_ε.

Préface

Le cours de base de données est composé de quatre parties subdivisées en six chapitres, correspondant à l'introduction de différentes notions très générales comme dans la partie 1 (chapitre 1), ou beaucoup plus précises telles que dans la partie 3 (chapitre 5) qui présente le langage SQL (DML).
Seuls les chapitres 2, 5, et 6 donneront lieu à la présentation d'exercices. Les énoncés et les corrigés de ces TDs sont disponibles dans le document *TDs Base de Données*, respectivement aux chapitres 1 (modélisation relationnelle), 2 (SQL) et 3 (normalisation). Ce document propose dans une première partie les énoncés des exercices et dans une deuxième les corrigés précédés chaque fois de quelques indications générales.

Cette introduction aux bases de données est composée de quatre parties:

- La **partie 1** présente ce qu'est une base de données ainsi qu'un Système de Gestion de Bases de Données. Nous insisterons sur la structure des Systèmes de Gestion de Bases de Données. (Chapitre 1)
- La **partie 2** est consacrée à l'approche relationnelle. Nous présentons le modèle mathématique proposé par Codd. Nous y présentons en particulier la notion de relation (Chapitre 2), élément fondamental pour la représentation des données, ainsi que les opérateurs de l'algèbre relationnelle (Chapitre 4) qui permettent de définir les opérations de consultation des informations.
- La **partie 3** s'intéresse au langage standard des Systèmes de Gestion de Bases de Données relationnels, à savoir SQL. (Chapitres 5, 6 et 7). Un chapitre sur le langage de contrôle des données et des transactions a été mis en annexe. Les étudiants souhaitant aller plus loin dans ce cours pourront lire cette partie.
- La **partie 4** enfin, aborde la normalisation. Cette partie concerne l'étude de propriétés particulières des modèles relationnels, telles que les dépendances fonctionnelles. De telles propriétés permettent à l'aide du théorème de décomposition, en particulier, une conception rationnelle d'une base de données. (Chapitres 8 et 9).

Le cours est illustré par un exemple fil rouge : **les Cinémas**. Cette base de données simple, permet de gérer des cinémas ainsi que les films qui y sont à l'affiche. Elle est présentée au début du document : *Corrigés des exercices de cours.*

- Il est important de lire les chapitres d'une partie dans l'ordre.
- Il est possible de traiter la partie 4 avant ou en même temps que la partie 3.
- La réalisation des exemples et des exercices de TDs est importante pour la compréhension de l'ensemble (en particulier pour la partie 4).
- Les dates importantes de ce cours sont dans le document : *"Planning : parcours pédagogique"*

Parcours pédagogique

Les différentes parties du cours de bases de données sont à voir séquentiellement. Une durée à passer sur chaque partie est proposée dans le calendrier présenté ci-dessus.

Sem.	Date	Chapitre	Paragraphe(s)	Exercice(s)	Devoir	Correction
1	09 Oct	Chap 1				
2	16 Oct	Chap 2				
3	23 Oct	Chap 3 et 4	Para. 1, 2	Exos 1, 3		
	30 Oct	*Vacances*				
4	06 Nov	Chap 4	Para. 3	Exos 4, 5		
5	13 Nov	Chap 5, 6			Devoir 1	
6	20 Nov	Chap 7	Para. 1	Exo 1		
7	27 Nov	Chap 7	Para. 2, 3	Exo 2		
8	04 Dec	Chap 7	Para. 4 à 6	Exo 3, 5		Devoir 1
9	11 Déc	Chap 8		Exos 1,3	Devoir 2	
10	18 Déc	Chap 9	Para. 1 à 5	Exos 4, 6		
	25 Déc	*Vacances*				
	01 Janv	*Vacances*				
11	08 Janv	Chap 9	Para. 6	Exos 7,8	Devoir 3	Devoir 2
12	15 Janv	Révisions				Devoir 3
	22 Janv		Examen /23			

Remarque : Les dates données dans la colonne *"correction"* correspondent à la date de retour de vos devoirs notés (votre devoir avec des remarques et votre note sont déposés sur moodle).

Devoir 1 : Modélisation relationnelle

Ce premier devoir a pour objectif de revoir toutes les notions vues dans le chapitre 2. Il sera composé de deux grandes parties :

- travail sur un modèle relationnel : définition des liens, compréhension du modèle proposé, utilisation des règles de mise à jour.
- extraction d'informations dans la base avec les opérateurs relationnels.

Ce devoir est à rendre pour le **dimanche 19 Novembre 2023**.

Devoir 2 : Requêtes SQL

Ce devoir a pour objectif de revoir toutes les notions vues dans le chapitre 5, à savoir le langage SQL. Il consiste essentiellement en l'écriture de requêtes SQL pour l'extraction d'informations dans la base de données.

Ce devoir est à rendre pour le **dimanche 17 Décembre 2023**.

Devoir 3 : Normalisation

Ce dernier devoir a pour objectif de revoir toutes les notions vues dans le chapitre 7. Il sera composé de différents exercices permettant de revoir :

- les notions de dépendance fonctionnelle, et de dépendance fonctionnelle totale,
- les formes normales issues de la notion de dépendance fonctionnelle : 2NF, 3NF, et BCNF
- et enfin des dépendances plus générales, avec les formes normales 4NF et 5NF.

Ce devoir est à rendre pour le **mercredi 10 Janvier 2024**.

Remarques concernant les devoirs :

- Quelques jours de retard pour un devoir peuvent être acceptables. Mais, passé un certain délai je ne peux plus vous garantir la correction de votre devoir. En particulier pour le dernier devoir, je ne dispose que de quelques jours pour assurer la correction.
- Il vaut mieux m'envoyer une partie du devoir dans les temps que rien du tout.
- Les notes obtenues à ces devoirs vous permettent d'avoir, avant l'examen, une évaluation de votre travail. Elles n'interviennent pas dans votre note finale.
- Enfin, dans votre devoir vous pouvez mettre des commentaires. N'hésitez pas à mettre des idées pour d'autres solutions, ou des explications concernant votre solution. C'est un moyen supplémentaire pour communiquer.

Contents

Préface c

Parcours pédagogique e

I Introduction aux bases de données 1

1 Bases de Données et Systèmes de Gestion de Bases de Données 3
1.1 La place des bases de données dans l'informatique . . . 3
1.1.1 Le domaine des bases de données . . . 3
1.1.2 L'utilisation des bases de données . . . 4
1.1.3 Base de données, Système de gestion de base de données, application de base de données et système d'information . . . 5
1.2 Qu'est-ce qu'une Base de Données ? . . . 6
1.2.1 Avant les bases de données : les systèmes basés sur les fichiers . . . 6
1.2.2 Les bases de données . . . 8
1.3 Qu'est-ce qu'un Système de Gestion des Bases de Données ? . . . 9
1.3.1 Définition . . . 9
1.3.2 Les trois couches d'un SGBD . . . 10
1.3.3 Les objectifs d'un SGBD . . . 11
1.3.4 L'architecture ANSI/SPARC . . . 12
1.4 Les rôles dans les bases de données . . . 14
1.4.1 Administrateurs . . . 14
1.4.2 Concepteurs . . . 14
1.4.3 Développeurs d'applications . . . 15
1.4.4 Utilisateurs . . . 15
1.5 Les différents types de bases de données . . . 15
1.5.1 Les bases hiérarchiques . . . 15
1.5.2 Les bases réseaux . . . 16
1.5.3 Les bases relationnelles . . . 16
1.5.4 Les bases objets . . . 17
1.5.5 Bases de données XML natives . . . 17
1.5.6 Bases de données décisionnelles . . . 17
1.5.7 Les bases déductives . . . 18
1.5.8 Les bases de données NoSQL . . . 19
1.6 Les principaux SGBD . . . 19

II Le modèle relationnel **23**

2 La théorie Relationnelle **25**
2.1 Définitions de base . . . 25
2.1.1 La notion de relation . . . 25
2.1.2 Autres définitions . . . 26
2.1.3 Quelques remarques sur ces définitions . . . 26
2.1.4 Exemple . . . 26
2.2 Propriétés des relations . . . 28
2.2.1 Pas de duplication de t-uplets . . . 28
2.2.2 Les t-uplets ne sont pas ordonnés . . . 29
2.2.3 Les attributs ne sont pas ordonnés . . . 29
2.2.4 Les valeurs des attributs sont dans un seul domaine . . . 30
2.2.5 Les valeurs des attributs sont atomiques . . . 30
2.2.6 Les noms des relations et des attributs sont uniques . . . 30
2.3 Notions nécessaires à la présentation de l'intégrité des données . . . 31
2.3.1 Les clés relationnelles . . . 32
2.3.2 La valeur `NULL` . . . 37
2.4 Les règles d'intégrité des données . . . 37
2.4.1 L'unicité de clé . . . 37
2.4.2 Les contraintes de domaine . . . 38
2.4.3 La gestion de `NULL` . . . 38
2.4.4 Les contraintes référentielles . . . 39

3 L'exemple des cinémas **43**
3.1 Les types de données dans les SGBD . . . 43
3.1.1 Les chaines de caractères . . . 44
3.1.2 Les données numériques . . . 44
3.1.3 Les dates . . . 45
3.1.4 Les autres types de données . . . 45
3.2 Les relations de `CINEMAS` . . . 46
3.2.1 La relation CINEMA . . . 47
3.2.2 La relation `FILM` . . . 47
3.2.3 La relation `AFFICHE` . . . 48
3.2.4 La relation `DISTRIBUTION` . . . 48
3.3 Les clés candidates et les clés primaires . . . 49
3.3.1 La relation `CINEMA` . . . 49
3.3.2 La relation `FILM` . . . 49
3.3.3 La relation `AFFICHE` . . . 50
3.3.4 La relation `DISTRIBUTION` . . . 50
3.4 Les liens . . . 50
3.4.1 Le modèle relationnel de la base `CINEMAS` . . . 50
3.4.2 Les clés étrangères . . . 50

4 Les opérateurs relationnels **53**
4.1 Les opérateurs ensemblistes classiques . . . 53
4.1.1 L'Union . . . 53
4.1.2 L'intersection . . . 54
4.1.3 La différence . . . 55
4.1.4 Le produit cartésien . . . 56

4.2 Les opérateurs relationnels spécifiques . . . 57
4.2.1 La sélection . . . 57
4.2.2 La projection . . . 59
4.2.3 La jointure . . . 61
4.3 Les opérateurs relationnels dérivés . . . 64
4.3.1 L'intersection . . . 64
4.3.2 La division . . . 65
4.3.3 La jointure externe . . . 65

III Le langage SQL 69

5 Les Langages des SGBD 73
5.1 Le langage QBE . . . 73
5.1.1 Pourquoi QBE ? . . . 73
5.1.2 La construction de requêtes en QBE . . . 73
5.2 Le langage SQL . . . 76
5.2.1 SQL : Historique . . . 76
5.2.2 Les sous-langages de SQL . . . 76

6 Le langage de définition des données - DDL 77
6.1 Règles de nommage . . . 77
6.2 Les tables . . . 77
6.2.1 La définition des attributs de la table . . . 78
6.2.2 L'expression des contraintes d'intégrité . . . 79
6.2.3 Modification d'une table . . . 82
6.2.4 Suppression d'une table . . . 83
6.3 Les index . . . 83
6.3.1 Qu'est ce qu'un index ? . . . 83
6.3.2 Création d'un index . . . 84
6.3.3 Suppression d'un index . . . 84
6.4 Les vues . . . 85
6.4.1 Qu'est ce qu'une vue? . . . 85
6.4.2 Création d'une vue . . . 85
6.4.3 Les contraintes des vues . . . 86
6.4.4 Suppression d'une vue . . . 86

7 Le langage de manipulation des données - DML 87
7.1 Les requêtes de sélection simples . . . 87
7.1.1 La requête de sélection de données . . . 87
7.1.2 La clause SELECT . . . 88
7.1.3 La clause FROM . . . 90
7.1.4 La clause WHERE . . . 91
7.1.5 La Jointure en SQL . . . 92
7.1.6 Les fonctions de groupe . . . 95
7.1.7 La clause GROUP BY . . . 97
7.1.8 La clause HAVING . . . 98
7.1.9 La clause ORDER . . . 99
7.2 Les sous-requêtes . . . 100
7.2.1 Les sous-requêtes indépendantes . . . 101

7.2.2 Les sous-requêtes synchronisées . . . 103
7.3 Les opérateurs ensemblistes en SQL . . . 104
7.3.1 L'Union : UNION . . . 105
7.3.2 L'Intersection : INTERSECT . . . 105
7.3.3 La différence : MINUS . . . 105
7.4 La division en SQL . . . 106
7.4.1 La division avec la fonction d'agrégation COUNT . . . 106
7.4.2 La division avec la clause EXISTS . . . 107
7.5 Les expressions et fonctions en SQL . . . 107
7.5.1 Les expressions . . . 108
7.5.2 Les différents types de fonctions . . . 110
7.6 Les requêtes de mise à jour des données . . . 111
7.6.1 La suppression de données . . . 111
7.6.2 La modification de données . . . 112
7.6.3 L'ajout de données . . . 113

IV La normalisation 115

8 La normalisation 117
8.1 Pourquoi la normalisation ? . . . 117
8.1.1 Anomalie d'insertion . . . 118
8.1.2 Anomalie de suppression . . . 118
8.1.3 Anomalie de modification . . . 119
8.1.4 La normalisation des relations . . . 119
8.2 Les notions essentielles pour la théorie de la normalisation . . . 121
8.2.1 Les dépendances Fonctionnelles . . . 121
8.2.2 Les dépendances fonctionnelles totales ou DFT . . . 121
8.2.3 Comment définir l'ensemble des DFs? . . . 122
8.2.4 Le théorème de décomposition . . . 125

9 Les formes normales 127
9.1 Qu'est-ce qu'une forme normale ? . . . 127
9.2 La première forme normale - 1NF . . . 128
9.3 La deuxième forme normale - 2NF . . . 128
9.3.1 Transformation en 2NF . . . 129
9.3.2 Application au Cinéma . . . 129
9.4 La troisième forme normale - 3NF . . . 131
9.4.1 Transformation en 3NF . . . 131
9.4.2 Application au Cinéma . . . 132
9.5 La forme normale de Boyce-Codd- BCNF . . . 133
9.5.1 Transformation en BCNF . . . 133
9.5.2 Application au Cinéma . . . 134
9.6 Les 4e et 5e formes normales . . . 135
9.6.1 Les dépendances multivaluées . . . 135
9.6.2 La quatrième forme normale : forme 4NF . . . 138
9.6.3 Les dépendances de jointure . . . 141
9.6.4 La cinquième forme normale : forme 5NF . . . 144

V Annexes 145

Bibliography 147

Acronymes 149

Index 151

Part I

Introduction aux bases de données

Chapter 1

Bases de Données et Systèmes de Gestion de Bases de Données

Nous présentons dans ce chapitre une introduction au module de Bases de données, avec en particulier :

- La place des bases de données dans l'informatique.
- Les caractéristiques générales d'une base de données, qui la différencient d'un simple ensemble de données.
- Les systèmes de gestion de bases de données ou SGBD, en développant
  - les trois couches d'un SGBD,
  - les neuf objectifs d'un SGBD,
  - l'architecture ANSI/SPARC.
- Les rôles dans les bases de données.
- Les grands types de bases de données et les principaux Systèmes de Gestion de Bases de Données.

1.1 La place des bases de données dans l'informatique

1.1.1 Le domaine des bases de données

Pour commencer ce cours de base de données voici quelques citations qui précisent le contexte des bases de données.

> *"L'histoire de la recherche en matière de systèmes de bases de données est d'une exceptionnelle productivité et d'un impact économique surprenant. La recherche en bases de données qui, il y a à peine vingt ans, était un domaine de recherche scientifique fondamental, a généré une industrie de services de l'information estimée à quelque 10 milliards de dollars par an, pour les seuls Etats-Unis. Les accomplissements en recherche sur les bases de données ont permis des avancées fondamentales dans les systèmes de communication, dans le transport et la logistique, la gestion financière, les systèmes de bases de connaissances, l'accessibilité à la littérature scientifique, ainsi qu'une panoplie d'applications civiles et militaires. Elles forment également les fondations de progrès considérables dans les domaines scientifiques fondamentaux, de l'informatique à la biologie."*

Cette citation écrite en 1990 et reprise en 1996 dans les articles [5] et [6] montrait déjà l'importance des bases de données. Cette importance avec l'émergence d'internet notamment ne fait que s'accroître.
T. Connoly et C. Begg disent dans leur livre [2] :

"On peut considérer que le système de bases de données est le plus important développement dans le domaine du génie logiciel."

Comme ces citations le précisent le domaine des bases de données est un domaine extrêmement actif et productif de l'informatique. Ce domaine a permis de nombreux développements de l'ingénierie logicielle. Ainsi le développement d'algorithmes pour l'accès ou le stockage de données, ou l'optimisation des requêtes a eu des répercussions dans l'ensemble du génie logiciel.

1.1.2 L'utilisation des bases de données

Au quotidien nous utilisons des bases de données sans en avoir toujours conscience, comme le montrent les exemples présentés ci-dessous.

1.1.2.1 Au supermarché

Lorsque vous passez à la caisse, la caissière lit le code barre de chacun de vos articles, et cette lecture permet le calcul de votre note. L'identification du code barre permet à l'application de rechercher le prix de l'article associé dans une base de données. Cette application peut aussi mettre à jour les stocks du supermarché, ...

1.1.2.2 Votre dossier étudiant à l'Université

Lorsque vous vous inscrivez à l'université, les informations vous concernant sont enregistrées dans une base de données. Ces informations servent à différentes applications telles que :

- la consultation de votre dossier étudiant sur l'ENT,
- votre accès à la plate-forme Moodle,
- l'impression de vos résultats en fin d'année,
- le suivi de vos résultats,
- ...

1.1.2.3 Achat d'un billet de train

Lorsque vous achetez un billet de train à un guichet de gare, sur Internet, ou à distributeur automatique de billets de train, le système doit permettre :

- de lister les trains allant d'une ville à une autre,
- de réaliser la réservation,
- d'assurer au client ou à l'opérateur que personne d'autre ne peut réserver la même place.

1.1.2.4 Boutiques en ligne

Lorsque vous allez sur un site d'achat sur internet tel que Amazon.fr, vous pouvez :

- accéder à une liste de catégories de produits : livres, CD, informatique, ...
- consulter les différentes rubriques de livres, ceux d'informatique, de base de données,
- consulter la description du livre : résumé, plan, éventuellement un chapitre (cf. figure 1.1),
- commander un livre et réaliser le paiement en ligne avec ses informations bancaires,
- consulter vos précédentes commandes (dans le cas où vous êtes déjà client).

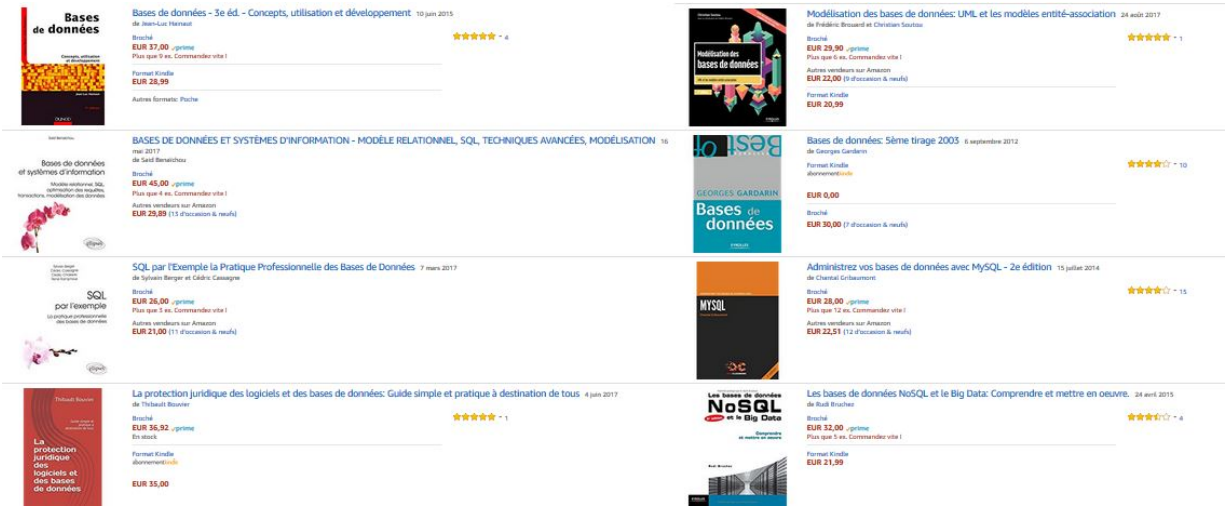


Figure 1.1 : Site Amazon : Informations sur des livres de bases de données

Il existe bien d'autres exemples de traitements qui supposent l'utilisation d'une base de données tels que l'utilisation de votre carte bancaire, la consultation d'un forum, l'emprunt d'un CD dans une vidéothèque, ... Les bases de données font vraiment partie de notre vie quotidienne et constituent un outil majeur de l'informatique.

1.1.3 Base de données, Système de gestion de base de données, application de base de données et système d'information

Avant de présenter de façon plus précise le concept de base de données, précisons quelques notions liées au domaine des bases de données.

Base de données ou BD : C'est une collection organisée de données.

Système de gestion de base de données ou SGBD : C'est un logiciel qui gère et permet l'accès aux données de la base.

Application de base de données : C'est un programme qui interagit avec la base de données à certains moments de son exécution en lui adressant une requête (en général dans le langage SQL). Il peut s'agir par exemple de programmes php, Java, Delphi. Actuellement ces programmes sont de plus en plus des applications Web.

Système d'information ou SI : Le SI est un ensemble organisé de ressources qui permet de collecter, stocker, traiter et distribuer de l'information. Il s'agit en fait de l'ensemble formé par les ressources : base de données, SGBD et applications, ... liés à cette base de données.

Important : Dans le cours présenté ici nous ne nous intéressons qu'à la création, la modification et l'utilisation d'une base de données avec un SGBD. Nous ne traiterons pas les applications de bases de données ni les systèmes d'information. Nous n'étudierons pas le fonctionnement interne du SGBD (le stockage des données, les algorithmes de recherche, l'optimisation des requêtes, ...). Nous ne présenterons pas non plus l'étape de conception d'une base de données (cette étape sera étudiée dans l'UE *Analyse et modélisation des systèmes dinformation* du second semestre). Nous travaillerons toujours avec un modèle de départ : le schéma relationnel ou au moins un ensemble de relations.

Nous verrons simplement :

- le modèle mathématique permettant de décrire une base de données relationnelle : *la théorie relationnelle*,
- le langage permettant de manipuler de telles bases : *SQL*,
- et enfin des outils permettant de vérifier qu'un modèle relationnel est correct : *la normalisation.*

De plus les bases de données ne se limitent pas au modèle relationnel. Comme nous le verrons à la fin de ce chapitre il existe différents types de bases de données : relationnelles, NoSQL, hiérarchiques, objet, ...
Nous n'étudierons dans cette introduction aux bases de données que les bases de données relationnelles.

1.2 Qu'est-ce qu'une Base de Données ?

Nous présentons dans cette section les systèmes qui ont été proposés avant les bases de données, puis à partir de cette présentation la notion de base de données.

1.2.1 Avant les bases de données : les systèmes basés sur les fichiers

Les systèmes basés sur les fichiers ont été la première proposition pour traiter des informations telles que les fiches d'une bibliothèque, les fiches de fabrication des entreprises, ...

*Un **système basé sur les fichiers** est un ensemble de programmes d'application qui définissent et gèrent un ensemble de données.*

Les informations sont stockées dans différents fichiers qui peuvent être dans des lieux différents (aspect décentralisé). Prenons l'exemple d'un centre médical dans lequel travaillent un médecin, un ophtalmologue, un kinésithérapeute, un dentiste et un laboratoire d'analyse. Un secrétariat gère l'ensemble des rendez-vous pour le centre. Cependant chaque spécialiste a sa propre façon de gérer et de stocker les informations qui concernent les clients comme le montre la figure **??**. Pour rechercher une information, il faut lire les informations les unes après les autres (séquentiellement). Dans certains cas, des systèmes d'indexation permettent d'améliorer les recherches.

Ce type de système fonctionne tant que le nombre de données reste faible, et que les recherches sont établies sur des critères simples. Si le nombre de données devient trop grand ou si l'on souhaite effectuer des recherches avec des références croisées, ce type de système ne convient plus.
Nous avons dit qu'un tel système est composé d'un ensemble de programmes d'application. Ces ensembles de programmes gèrent la saisie des données et la maintenance des fichiers. Ce sont eux qui définissent la structure et le stockage physique des données, qui peuvent donc varier d'un ensemble d'applications à un autre, ainsi les données concernant le client *"Dupont"* ne sont pas traitées et stockées de la même façon chez le dentiste et le médecin. Par ailleurs ces différents ensembles de programmes génèrent des doublons. En effet chaque ensemble gère ses propres données, ainsi une même donnée peut être stockée dans différents fichiers. Monsieur Dupont est décrit dans le système de fichiers du médecin et du dentiste.

Les limites de cette approche sont principalement :

Figure 1.2 : Centre médical

- *La séparation et l'isolement des données :* Les données sont isolées dans des fichiers et leur accès en est rendu difficile. De plus dès que l'on doit accéder à des informations placées sur plusieurs fichiers, la synchronisation des traitements devient délicate.
- *Doublons :* L'aspect décentralisé entraine une redondance non contrôlable des données. Ces doublons consomment de l'espace de stockage, et peuvent surtout conduire à une perte de la cohérence des données.
- *Dépendance programme-données :* La structure et le stockage des données dépendent des programmes d'application qui traitent ces données. Toute modification de la structure entraine la modification de tous les programmes d'application.
- *Incompatibilité des formats de fichiers :* La structure des fichiers est définie par les programmes d'application, elle dépend ainsi du langage de programmation utilisé par les programmes d'application. Il n'y a donc pas toujours de compatibilité entre les différents fichiers.
- *Ensemble figé de requêtes :* Les systèmes basés sur les fichiers ont été une première avancée qui a provoquée chez les utilisateurs une demande de nouvelles requêtes de plus en plus complexes. Ceci a eu pour effet la multiplication des programmes d'application lorsque cela était possible, et dans certains cas l'écriture des requêtes demandées s'est avérée impossible.

Les systèmes basés sur des fichiers ont des inconvénients majeurs tels que :

- pas de prise en compte de la sécurité,
- pas de prise en compte de l'intégrité,
- pas de prise en compte de la récupération des données après une panne,
- pas d'accès partagé aux fichiers (un seul utilisateur peut travailler à la fois sur un fichier).
- pas de gestion de la confidentialité.

Ce premier pas dans la gestion de données s'est donc avéré rapidement insuffisant, d'où la nécessité de créer un système plus efficace : *les bases de données*. Des éléments essentiels ont été pris en compte :

- Les données et les programmes doivent être les plus indépendants possible. Cette approche correspond à l'approche objet dans le domaine de la programmation.
- L'accès aux données et leur manipulation doivent être controlés en dehors des programmes d'application.
- Une base de données est une **ressource partagée**.
- Les Système de Gestion de Bases de Données doivent permettre la factorisation des modules de contrôle des applications et une administration facilitée des données.

1.2.2 Les bases de données

La notion de base de données est antérieure à la notion de base de données informatisée, en effet de très nombreuses bases de données non informatisées existaient et existent encore.
Citons par exemple les informations gérées par une bibliothèque, les stocks dans une entreprise, les informations concernant les trains pour la SNCF, etc.
Les bases de données sont depuis toujours des éléments fondamentaux dans la connaissance, et la perception des entreprises. Les entreprises ont toujours eu des fiches clients, des fiches articles, commandes, etc, qui leur permettent de décrire leurs clients (nom, société, adresse de livraison, de facturation,...), les articles qu'elles fabriquent, leur stock, les chiffres en commande.
L'apparition de bases de données informatisées a bouleversé la culture des entreprises, et leur façon de gérer ces données. Ceci explique l'impact et l'importance des bases de données dans le monde de l'entreprise. Ces dernières décennies de nombreux outils informatiques s'appuyant sur des bases de données se sont développés au sein des entreprises, tels que les ERP ou Enterprise Resource Planning, les CRM ou Customer Relationship Management et l'informatique décisionnelle (BI pour Business Intelligence).

Une base de données peut être vue de différentes façons plus ou moins techniques :

- Une première approche est de voir une base de données comme un ensemble d'informations modélisant les objets d'une partie du monde réel et servant de support à une application informatique.
- Une base de données est au sens large une collection organisée, interrogeable et persistante de données ou d'informations, concernant un thème donné pour l'entreprise.

D'un point de vue plus informatique, une base de données est :

- Un ensemble de fichiers comprenant un certain nombre d'articles formés chacun d'un type de données, avec des liens entre ces articles.
- Un ensemble d'opérateurs permettant la recherche, le tri, la fusion ou toute autre opération de même type.
- Un ensemble de données et de métadonnées. Ces dernières sont des données qui décrivent les données, elles permettent en particulier de définir le schéma de la base.

Une base de données (ou database en anglais) est une entité dans laquelle il est possible de stocker des données de façon structurée et avec le moins de redondance possible. Ces données doivent pouvoir être utilisées par des programmes et des utilisateurs différents.
Elle représente une partie du monde réel. La figure 1.3 (cf. transparents de G. Gardarin [3]) montre les différents niveaux de modélisation utilisés dans les bases de données.

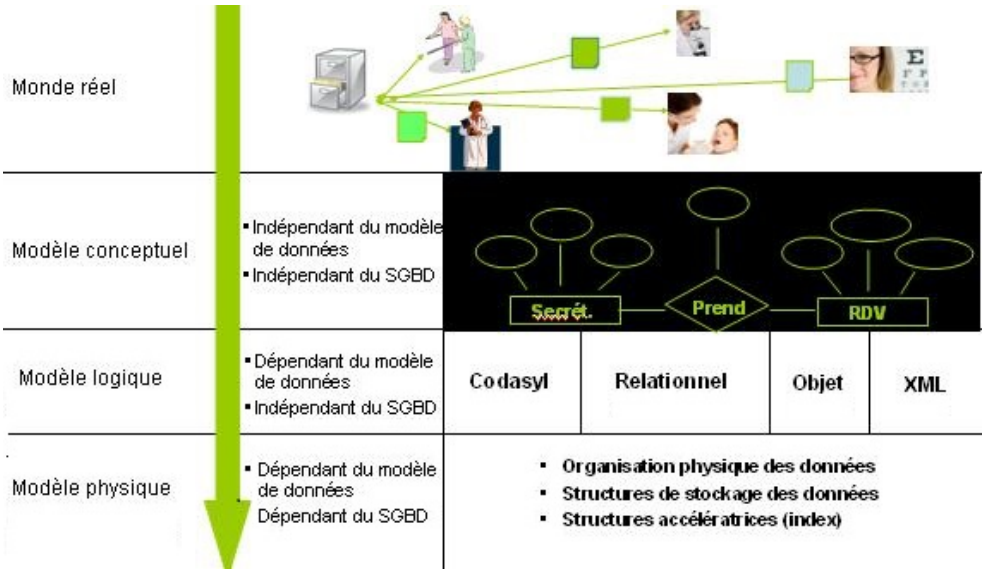


Figure 1.3 : Modélisation du monde réel

1.3 Qu'est-ce qu'un Système de Gestion des Bases de Données ?

1.3.1 Définition

Un **Système de Gestion des Bases de Données** ou **SGBD** est l'outil principal de gestion des bases de données.
C'est un ensemble de logiciels fournissant un environnement factorisant des modules de contrôle des applications pour décrire, mémoriser, manipuler, traiter des ensembles de données, en contrôler l'accès, les partager, gérer des pannes, ...

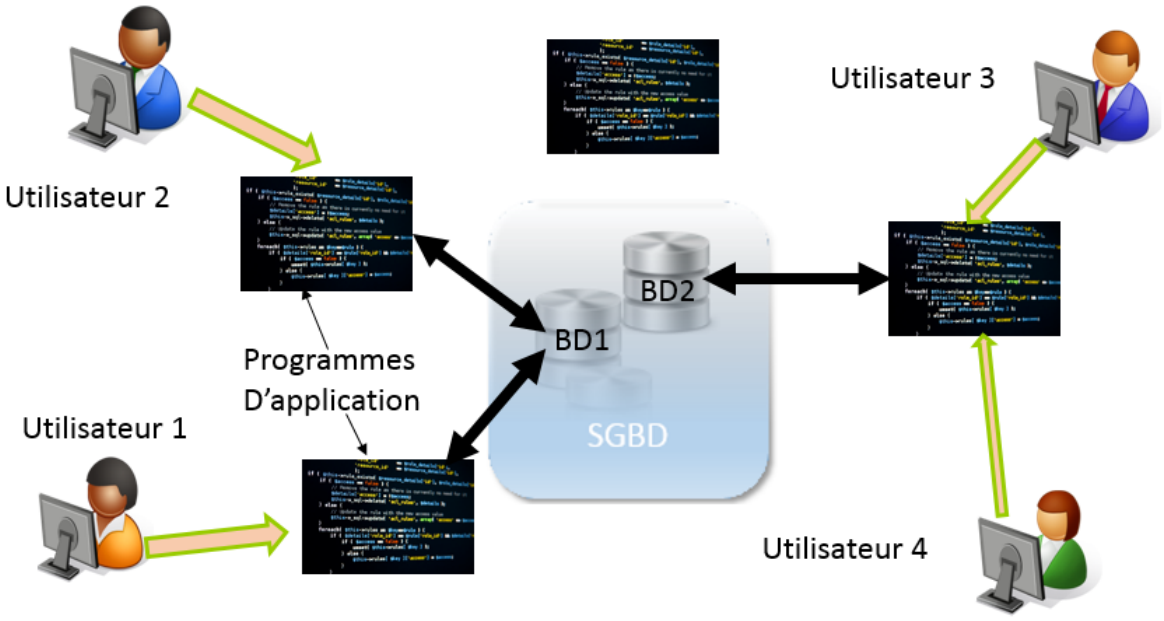


Figure 1.4 : Rôle d'un SGBD

Comme le montre la figure 1.4 , un SGBD permet aux utilisateurs via les applications de bases

de données d'accéder aux informations de la base de données :

- Toutes les demandes ou requêtes des utilisateurs sont prises en charge efficacement par le SGBD. C'est le SGBD qui par exemple s'occupe de fournir à l'utilisateur la liste des noms de cinémas. L'utilisateur ne sait pas où se trouve l'information et comment l'extraire, c'est le SGBD qui réalise ce travail.
- Il permet d'ajouter, d'effacer, de mettre à jour des données, et aussi de créer ou détruire des fichiers, et d'accéder aux données. L'utilisateur pourra ainsi grâce au SGBD créer la base de données CINÉMA et y saisir les informations concernant tous les cinémas.

Il permet de protéger l'utilisateur des détails du niveau matériel :

- L'utilisateur a une vue de la base à un niveau se situant au dessus des niveaux matériels. Le SGBD gère les demandes des utilisateurs. L'utilisateur a l'illusion que l'information est comme il le souhaite.
- Chaque utilisateur a l'impression d'être le seul à utiliser la base de données.

1.3.2 Les trois couches d'un SGBD

Les SGBD sont composés à première vue de trois couches emboîtées les unes dans les autres, depuis les mémoires secondaires jusqu'aux utilisateurs, comme le montre la figure 1.5 .

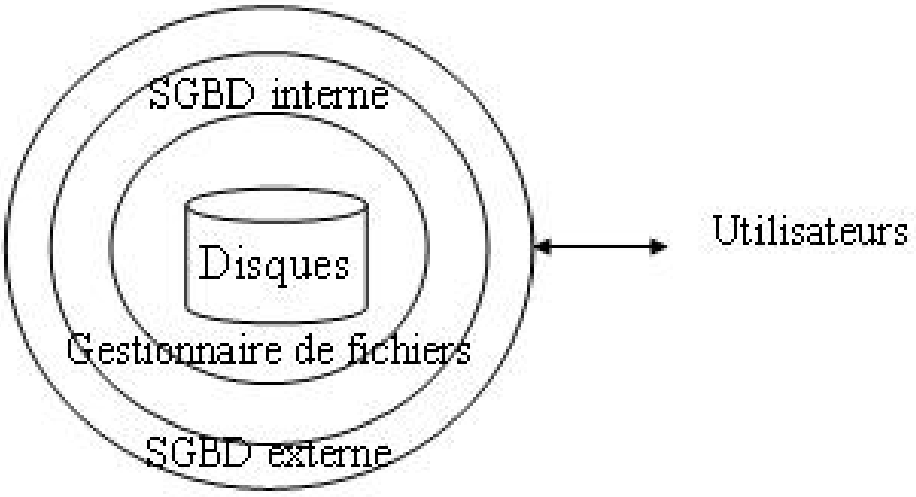


Figure 1.5 : Les trois couches d'un SGBD

- Au niveau le plus interne, le *système de gestion des fichiers* gère le stockage physique de l'information et est dépendant du matériel utilisé. Il fournit aux couches supérieures des mémoires addressables et capables de recherche sur le contenu.
- Le *SGBD interne* gère le placement et l'assemblage des données, ainsi que les liens et l'accès rapide aux données. Il repose généralement sur un modèle de données interne, par exemple des tables reliées par des pointeurs.
- Le *SGBD externe* s'occupe de la présentation et de la manipulation des données aux utilisateurs. Il s'occupe de la gestion des langages de requêtes et des outils de présentation des données (états, formulaires).

Ces trois couches fondamentales représentent environ 50% du code d'un SGBD. En effet, au delà de ces fonctions de recherche, de placement et de présentation des données, un SGBD doit gérer des problèmes plus délicats tels que :

- la cohérence des données,
- le partage des données,
- la protection des données, ...

Autant de problèmes qui vont prendre une part importante dans la conception d'un SGBD.

1.3.3 Les objectifs d'un SGBD

Le principal objectif d'un SGBD est bien évidemment d'assurer l'indépendance des applications de base de données par rapport aux données. Il faut pouvoir modifier la structure des données sans avoir à modifier les programmes d'applications (problèmes de maintenance).
De tels systèmes de gestion d'informations peuvent être réalisés sans outil spécifique. On peut alors se demander ce qui distingue ces systèmes, des systèmes de gestion de fichiers classiques. Nous présentons *neuf objectifs* qui différencient un SGBD d'un simple gestionnaire de fichiers.
Afin d'assurer l'indépendance des programmes par rapport aux données, il est apparu nécessaire de manipuler les données avec des langages de haut niveau spécifiant celles que lon veut traiter et non pas comment y accéder.

1.3.3.1 Indépendance physique

La machine, les méthodes d'accès, de placement, de tri, le codage des données, ne sont pas visibles pour l'utilisateur. Le SGBD permet une représentation des données sans se soucier de l'aspect matériel.

1.3.3.2 Indépendance logique

Cette indépendance est basée sur la notion de *schéma conceptuel*, qui permet d'avoir une vue logique complète de la base de données.
L'utilisateur doit pouvoir accéder aux données comme il le souhaite, sa perception des données n'est pas la même que celle des autres utilisateurs. On dit que chaque utilisateur possède sa propre *vue des données*.
L'utilisateur doit pouvoir faire évoluer sa vue sans remettre en cause le schéma global (au moins dans une certaine mesure).
Dans l'exemple du cinéma, on peut avoir un utilisateur chargé de la mise à jour des données qui accédera à l'ensemble des données, alors qu'un utilisateur se renseignant sur les films à l'affiche ne verra que certaines données et sous un format particulier.

1.3.3.3 Manipulable par des non-informaticiens

L'utilisateur doit accéder aux données, en décrivant ce qu'il veut, et non en décrivant comment l'obtenir. Les données doivent donc être accessibles au moyen d'un langage descriptif et non impératif. Ce langage est un langage non procédural basé sur les assertions logiques du 1°ordre.

1.3.3.4 Accès efficace aux données

Les performances en termes d'opérations exécutées et de temps de réponse sont un problème clé dans les SGBD qui peuvent contenir une masse très importante de données. Le SGBD doit utiliser des algorithmes de recherche de données efficaces.

1.3.3.5 Administration centralisée des données

Le SGBD doit offrir aux personnes qui administrent la base, ou *administrateurs* (DBA) :

- des outils de vérification de cohérence des données,
- de restructuration de la base,
- de sauvegarde ou de réplication.

Ces outils doivent être limités à un nombre restreint de personnes pour des raisons de sécurité.

1.3.3.6 Non redondance des données

Ce problème est essentiel pour la maintenance d'une base, et pour limiter sa taille mémoire. Une donnée stockée à plusieurs endroits est difficilement modifiable (toute modification de cette donnée suppose de retrouver et de mettre à jour toutes les autres données correspondantes).
Une règle simple en base de données est de dire : *Pour une donnée on a une place.* Cette règle peut cependant ne pas être respectée volontairement dans le cadre de bases de données réparties ou dans un souci d'optimisation de certaines bases de données (on parle de dénormalisation de base de données).

1.3.3.7 Cohérence des données

Cette cohérence est obtenue par la vérification de contraintes dites d'intégrité qui doivent être gérées automatiquement par le SGBD, et non par les programmes d'application.

Il ne doit pas être possible de mettre à l'affiche un film qui n'existe pas, ou de mettre à l'affiche un film dans un cinéma qui n'existe pas.

1.3.3.8 Partageabilité des données

Plusieurs utilisateurs doivent pouvoir utiliser simultanément une même base de données.

- L'utilisateur doit avoir l'impression qu'il travaille seul.
- Le SGBD gère les accès simultanés, et contrôle la cohérence des données.

1.3.3.9 Sécurité des données

Le SGBD doit s'assurer que les utilisateurs ne réalisent que des traitements autorisés. Il existe différents types d'utilisateurs : les administrateurs ou DBA, les développeurs, et les personnes qui consultent ou modifient les données. Le SGBD doit être capable aussi de restaurer la base de données après une panne.

Ces neuf objectifs correspondent à ce que l'on peut attendre d'un SGBD idéal. Dans la réalité les SGBD ne satisfont que certains de ces objectifs.

1.3.4 L'architecture ANSI/SPARC

Pour répondre aux objectifs décrits précédemment, Charles W. Bachman (qui a reçu le prix Turing pour ses contributions dans le domaine des bases de données) propose de déterminer dès 1965 des règles de normalisation des SGBD, et défini une architecture généralisée des SGBD : *l'architecture ANSI/SPARC* qui se compose de trois niveaux comme le montre la figure 1.6 .

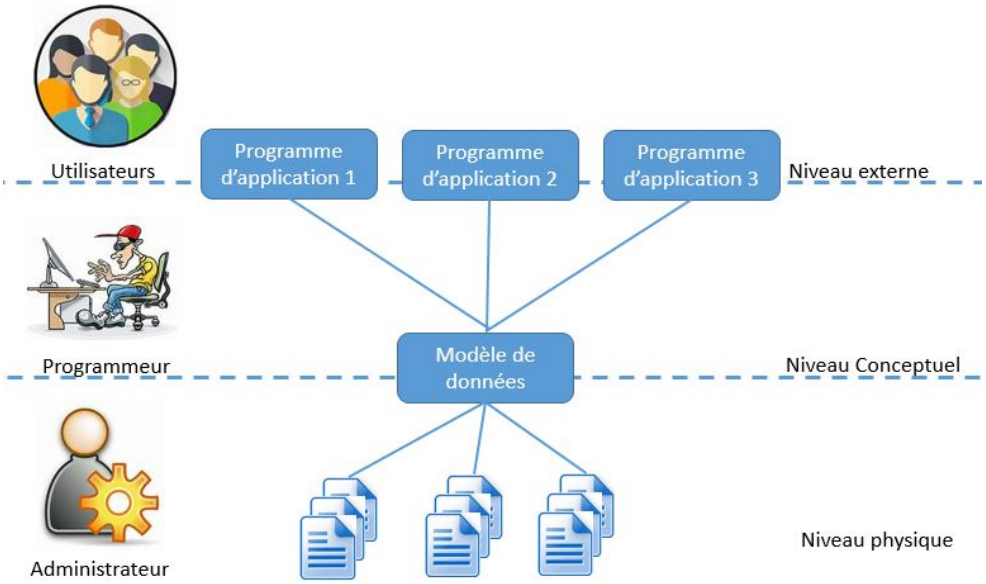


Figure 1.6 : L'architecture d'un SGBD

1.3.4.1 Le niveau externe

Il s'agit du niveau utilisateur. Il concerne la manière dont les utilisateurs voient les données. Chaque groupe d'utilisateur possède une description des données perçues appelée : *schéma ou vue externe*. Ces schémas décrivent seulement une partie des données.

1.3.4.2 Le niveau conceptuel

Ce niveau central permet de représenter la structure logique des données (entités et relations). Ce niveau relationnel correspond à ce que l'on retrouve dans la méthode Merise avec les modèles conceptuels de données (MCD), et ne se soucie pas de l'implantation.

1.3.4.3 Le niveau interne

Il correspond à la manière dont les données sont réellement stockées. Il n'est pas relationnel, et dépend du SGBD choisi. Il permet de décrire les données telles qu'elles sont stockées : noms des fichiers, articles des fichiers (longueur des champs, ...), chemins d'accès, modèle logique de données (MLD), ...

Ainsi le développement d'une base de données passe par trois étapes :

- La **phase conceptuelle** qui permet l'intégration des vues spécifiques de chaque utilisateur dans une description unique et cohérente qui élimine toute redondance d'information.
- La **phase d'implémentation logique** qui permet d'obtenir une structure plus efficace et de prendre en compte les éléments quantitatifs (volume des données, fréquence d'accès...).
- La **phase d'implémentation physique** qui est prise en charge par le SGBD.

Pour communiquer avec la base, l'utilisateur doit disposer d'un langage qui inclut plusieurs sous-langages :

- DDL : Langage de définition des données qui gère la définition et la déclaration des objets de la base
- DML : Langage de manipulation des données qui gère la manipulation ou les traitements sur les objets de la base
- DCL : Langage de contrôle de données
- TCL : Langage de contrôle des transactions

1.4 Les rôles dans les bases de données

Il existe différents types de personnes pouvant manipuler une base de données : les administrateurs, les concepteurs, les développeurs et les utilisateurs.

1.4.1 Administrateurs

On distingue deux types d'administrateurs, les administrateurs de données et les administrateurs de base de données. Ces deux rôles sont souvent réalisés par les mêmes personnes mais dans le cas de projets plus conséquents ils sont bien différenciés.

1.4.1.1 Administrateur de données ou AD

Les fonctions de cet administrateur sont essentiellement :

- la connaissance des données disponibles,
- l'acquisition des données,
- l'échange de données avec les acteurs,
- l'animation du dispositif autour de la base de données.

1.4.1.2 Administrateur de bases de données ou DBA

Cet administrateur est chargé de la mise en oeuvre technique de la base informatique, de la gestion de son intégrité et de l'intégration de données dans la base. Ces principales fonctions sont la gestion de :

- l'intégrité des données,
- la sécurité,
- la performance,
- l'aide au développement et au test,
- le recouvrement de données et la gestion des pannes,
- la validation et le conseil,
- la migration et les mises à jour.

1.4.2 Concepteurs

On peut distinguer deux types de concepteurs de bases de données : les concepteurs logiques et les concepteurs physiques.

1.4.2.1 Les concepteurs logiques

Ils s'intéressent à l'ensemble de l'organisation à mettre en place, et en particulier à l'identification des données (entités, attributs, associations, contraintes). Ils travaillent avec les utilisateurs futurs de la base de données.

1.4.2.2 Les concepteurs physiques

Ils traitent le passage du modèle logique de données vers un modèle physique. Ils représentent (ou "mappent") le schéma logique en un ensemble de relations et de contraintes d'intégrité, et s'intéressent à la sécurité des données. Le travail de ces concepteurs dépend fortement du SGBD choisi qu'ils doivent connaitre.

1.4.3 Développeurs d'applications

Lorsque la base de données est implémentée il reste à réaliser les applications de bases de données qui permettent aux utilisateurs finaux d'utiliser la base de données. Cette partie est à la charge des développeurs d'applications. Les applications de bases de données sont en générales écrites dans des langages tels que Java, php, ..., et ces programmes comportent des instructions qui permettent de consulter et de mettre à jour les données.

1.4.4 Utilisateurs

Les utilisateurs sont les *clients* de la base de données. Certains ne travaillent qu'à travers le biais des applications de base de données et ils n'ont aucune connaissance dans le domaine des bases de données. D'autres utilisateurs, plus expérimentés, peuvent écrire leur propres requêtes SQL pour interroger la base de données.

1.5 Les différents types de bases de données

Il existe de nos jours de nombreux types de bases de données. En voici quelques exemples. Cette liste n'est absolument pas exhaustive.

1.5.1 Les bases hiérarchiques

Ce sont les premiers SGBD qui sont apparus sur le marché. Citons en particulier GUAM (Generalized Update Access Method) développé par la NASA dans le cadre du projet lancé par JF Kennedy pour l'alunissage d'un homme sur la lune. Dans les années 60, IBM collabore avec la NASA pour étendre GUAM et met au point IMS (Information Management System) qui est l'un des premiers SGBD sur le marché.
Le schéma de la base est arborescent, la structure a la forme d'un arbre inversé comme le montre la figure 1.7 . Les enregistrements sont liés dans une structure arborescente de façon à ce que chaque enregistrement n'ait qu'un seul possesseur.

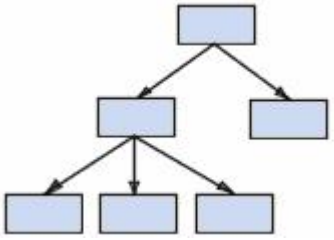


Figure 1.7 : Base de données hiérarchique

A cause de ses limitations internes, ce type de bases de données ne peut pas souvent être utilisé pour décrire des structures existantes dans le monde réel. Les liens hiérarchiques entre les différents

types de données permettent de répondre facilement à certaines questions, mais très difficilement à d'autres. Ils ne peuvent pas en particulier représenter des liens de type n-n.

1.5.2 Les bases réseaux

Le schéma des bases réseaux est plus ouvert que dans les bases hiérarchiques (cf. figure 1.8). Ce type de base est capable de résoudre certains problèmes rencontrés par le modèle hiérarchique grâce à sa possibilité d'établir des liens de type n-n, les liens entre objets pouvant exister sans restriction. Cependant, pour retrouver une donnée dans une telle modélisation, il faut connaître le chemin d'accès à la donnée, ce qui rend les programmes dépendants de la structure de données.

Ce modèle de bases de données a été introduit par C.W. Bachman en 1961.

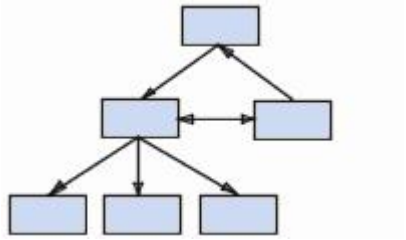


Figure 1.8 : Base de données réseau

Ces deux premières approches sont les plus anciennes et constituent des approches systèmes par rapport aux problèmes des bases de données. Une base de données est vue comme un ensemble de fichiers reliés par des pointeurs. Ces modèles sont performants, car très proches du niveau physique.

1.5.3 Les bases relationnelles

Une base de données relationnelle est une base de données structurée suivant les principes de l'algèbre relationnelle. Les données sont représentées sous la forme d'un ensemble de relations ou tables (cf. figure 1.9).

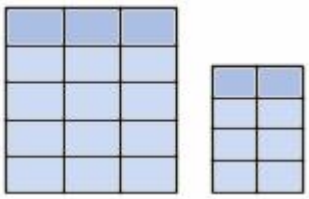


Figure 1.9 : Base de données relationnelle

Le créateur des bases de données relationnelles est Edgard F. Codd. Chercheur chez IBM à la fin des années 1960, il a étudié de nouvelles méthodes pour gérer de grandes quantités de données car les modèles et les logiciels de l'époque n'étaient pas satisfaisants. Mathématicien de formation, il a utilisé des branches spécifiques des mathématiques : la théorie des ensembles et la logique des prédicats du premier ordre, pour résoudre des difficultés telles que la redondance des données, l'intégrité des données ou l'indépendance de la structure de la base de données avec sa mise en oeuvre physique. En 1970, il publie un article [1] où il propose de stocker des données hétérogènes

dans des relations, permettant d'établir des liens entre elles. De nos jours, ce modèle est extrêmement répandu, mais en 1970, cette idée était considérée comme une curiosité intellectuelle. On pensait que les relations ne pourraient jamais être gérées de manière efficace par un ordinateur. Un premier prototype de Système de Gestion de Bases de Données Relationnelles (SGBDR) a été construit par IBM dans les années 80. Depuis cette technologie a mûri et a été très largement adoptée par l'industrie.

Remarque : La suite du cours présente ce type de base de données.

1.5.4 Les bases objets

Cette approche s'est développée dans les années 1990 - 2000 avec notamment la rédaction de la norme ODMG et la proposition d'un modèle relationnel étendu ou relationnel objet (SQL3).
La plupart des langages utilisés dans les applications de bases de données sont des langages objet. D'où l'apparition d'un problème fondamental le *mapping relationnel - objet*. L'approche base de données objet permet entre autres de traiter ce problème.
Les données sont représentées en tant qu'instances de classes hiérarchisées (cf. figure 1.10), ce qui permet de représenter des entités beaucoup plus complexes qu'avec le modèle relationnel. Chaque donnée possède ses propres méthodes d'interrogation et d'affectation.
Après un début prometteur ce type de base de données n'a finalement pas percé. On notera notamment le manque de respect de la norme dans les produits proposés, et par ailleurs le développement et l'implantation très importants des bases de données relationnelles.

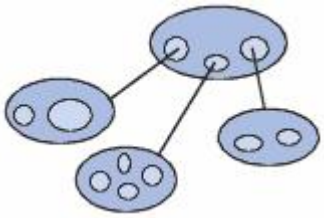


Figure 1.10 : Base de données Objet

1.5.5 Bases de données XML natives

Ces bases de données (fig. 1.11) s'appuient sur la structure offerte par le langage XML pour stocker les données et les repérer en s'appuyant sur les briques du langage XML relatives à la description et à la structuration de données (DTD, XML Schema, ...). Les langages de requêtes sont également décrits à l'aide de vocabulaires XML. Il s'agit principalement de XQuery et de XPath.
Ce type de bases de données représente une évolution importante du concept de base de données en permettant de stocker des volumes importants de données ou de documents, y compris multimédia et surtout non structurés.

1.5.6 Bases de données décisionnelles

Les bases de données décisionnelles sont utilisées dans le cadre de l'aide à la décision. Elles maintiennent et prennent en compte des versions historiques. Elles permettent d'écrire des requêtes complexes sur la base.
Les *entrepôts de données ou datawarehouse* sont des ensembles de données historisées variant dans le temps, organisé par sujets, consolidés dans une base de données unique, gérée dans un

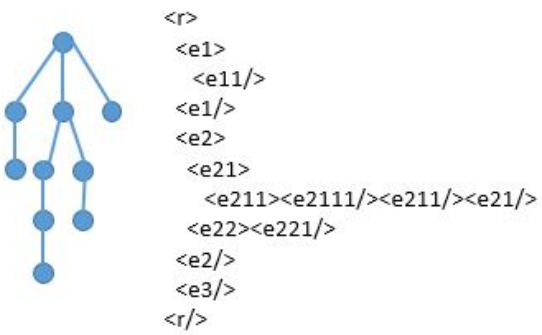


Figure 1.11 : Base de données XML

environnement de stockage particulier, et qui aident à la prise de décision dans lentreprise. Ils ont trois fonctions essentielles : la collecte de données de bases existantes et chargement, la gestion des données dans lentrepôt, et l'analyse de données pour la prise de décision.
Dans ce contexte l'objectif de *OLAP ou On-Line Analytical Processing* est de permettre une analyse multidimensionnelle sur des bases de données volumineuses afin de mettre en évidence une analyse particulière des données. Grâce à OLAP, les utilisateurs peuvent créer des représentations multidimensionnelles (appelées hypercubes ou ■ cubes OLAP ■) selon les critères qu'ils définissent afin de simuler des situations comme le montre la figure 1.12 .

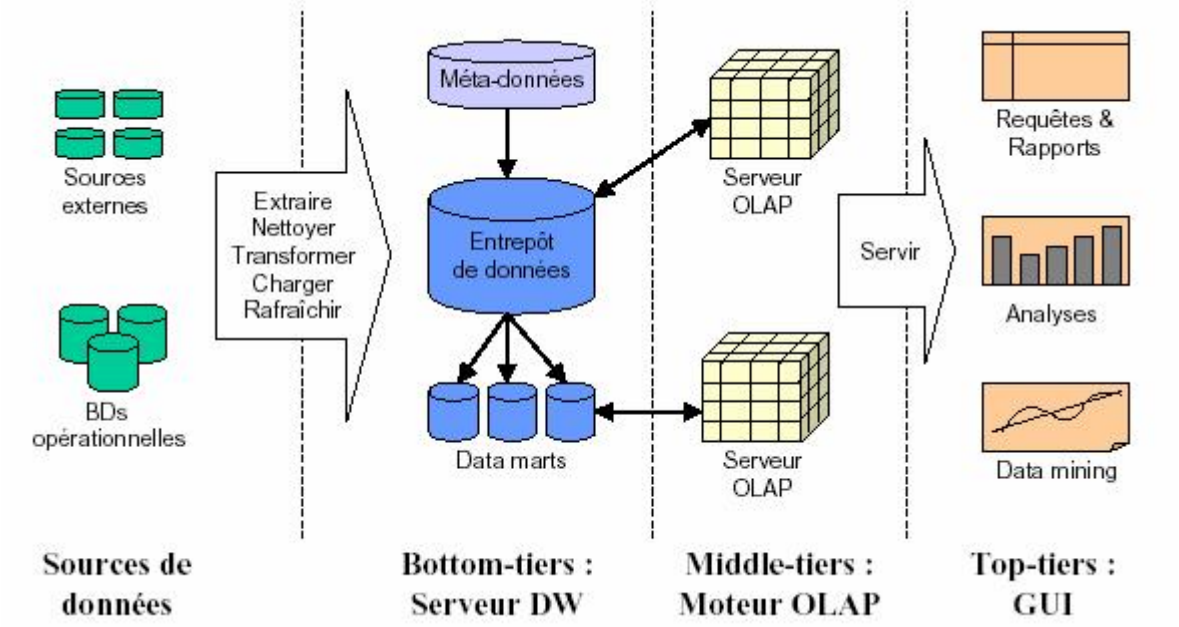


Figure 1.12 : Le décisionnel

1.5.7 Les bases déductives

Les données sont représentées sous la forme de tables. Le langage d'interrogation se base sur le calcul des prédicats et la logique du 1° ordre.

1.5.8 Les bases de données NoSQL

Le terme *NoSQL* ou *Not Only SQL* désigne de nombreuses base de données, apparues à partir de 2009, qui proposent des alternatives au langage SQL et au modèle de données relationnel en vue d'améliorer les performances et la capacité à traiter de très grands volumes de données. Dans l'industrie du web 2.0 beaucoup de sociétés ont abandonné les traditionnels SGBD relationnels pour des stockages de données appelés NoSQL qui fournissent une adaptabilité beaucoup plus importante, ou qui ont construit une couche cache distribuée par dessus les SGBDs.
De grand acteurs d'Internet, et notamment Google (BigTable), Amazon (Dynamo (en)), LinkedIn (Project Voldemort), Facebook (Cassandra Project puis HBase), SourceForge.net (MongoDB), Ubuntu One (CouchDB), xploitent des bases de données de type NoSQL. Ces bases de données sont difficiles à décrire, car elles évoluent de façon ad hoc avec différents modèles et capacités.
Il existe 4 grandes familles de bases de données NoSQL :

- **Clé-valeur :** Cette représentation, qui est la plus simple, est adaptée à la gestion de caches et à un accès rapide aux informations. Elle fonctionne comme un grand tableau associatif et retourne une valeur dont elle ne connaît pas la structure. Parmi les solutions les plus connues citons Redis, Riak et Voldemort.
- **Document :** Ajoute au modèle précédent, l'association d'une valeur à structure non plane, c'est-à-dire qui nécessite un ensemble de jointures en logique relationnelle. Pour cette famille, les implémentations les plus connues sont MongoDB, CouchDBase et CouchDB.
- **Colonne :** Correspond à une autre évolution du modèle clé-valeur, il permet de disposer un très grand nombre de valeurs sur une même ligne, permettant ainsi de stocker les relations de type 1-n. Contrairement au système Clé-Valeur, celui-ci permet d'effectuer des requêtes par clé. Comme solutions, on trouve principalement HBase (modèle BigTable publié par Google) et Cassandra.
- **Graphe :** Permet la modélisation, le stockage et la manipulation de données complexes liées par des relations non-triviales ou variables. La principale solution est Neo4J.

1.6 Les principaux SGBD

Il existe actuellement trois leaders dans le monde des constructeurs de SGBD : Oracle, IBM et Microsoft. Le site DB-Engine [8] propose en direct un classement des SGBD relationnels et NoSQL.

EN 2015 nous avions la répartition suivante [9] en terme de données gérées dans des SGBD (fig. 1.13).

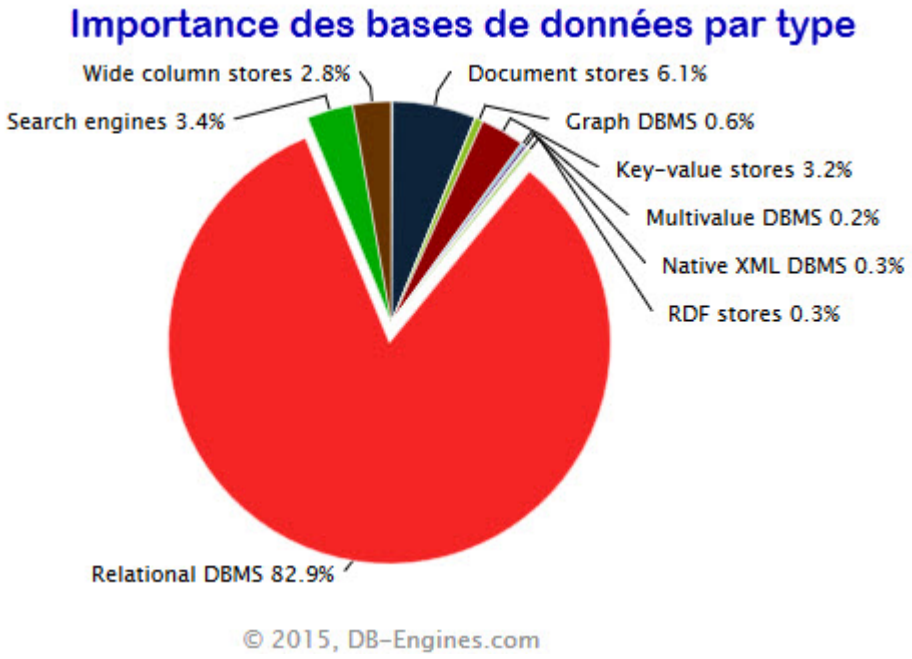


Figure 1.13 : La répartition DB-Engine des différents types de SGBD en 2015

En 2020 (fig. 1.14), l'importance des SBGD NoSQL augmente au détriment des SGBD relationnels qui restent cependant largement majoritaires.

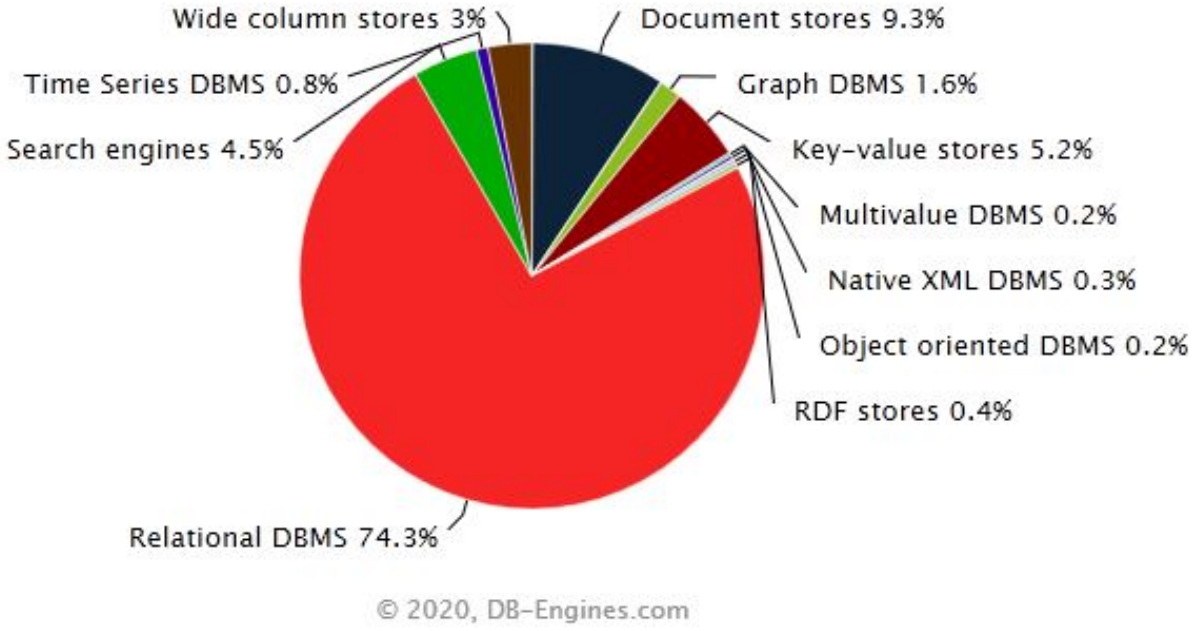


Figure 1.14 : La répartition DB-Engine des différents types de SGBD en 2020

Le classement DB-engine des SGBD les plus utilisés donne le résultat présenté dans la figure 1.15 .

Rank			DBMS	Database Model	Score		
Sep 2020	Aug 2020	Sep 2019			Sep 2020	Aug 2020	Sep 2019
1.	1.	1.	Oracle	Relational, Multi-model	1369.36	+14.21	+22.71
2.	2.	2.	MySQL	Relational, Multi-model	1264.25	+2.67	-14.83
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1062.76	-13.12	-22.30
4.	4.	4.	PostgreSQL	Relational, Multi-model	542.29	+5.52	+60.04
5.	5.	5.	MongoDB	Document, Multi-model	446.48	+2.92	+36.42
6.	6.	6.	IBM Db2	Relational, Multi-model	161.24	-1.21	-10.32
7.	7.	↑ 8.	Redis	Key-value, Multi-model	151.86	-1.02	+9.95
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	150.50	-1.82	+1.23
9.	9.	↑ 11.	SQLite	Relational	126.68	-0.14	+3.31
10.	↑ 11.	10.	Cassandra	Wide column	119.18	-0.66	-4.22

Figure 1.15 : Le classement DB-Engine des SGBD en 2020

Par ailleurs les SGBD libres continuent à voir leur usage augmenter. Ils rejoignent en terme d'utilisation, les SGBD propriétaires.

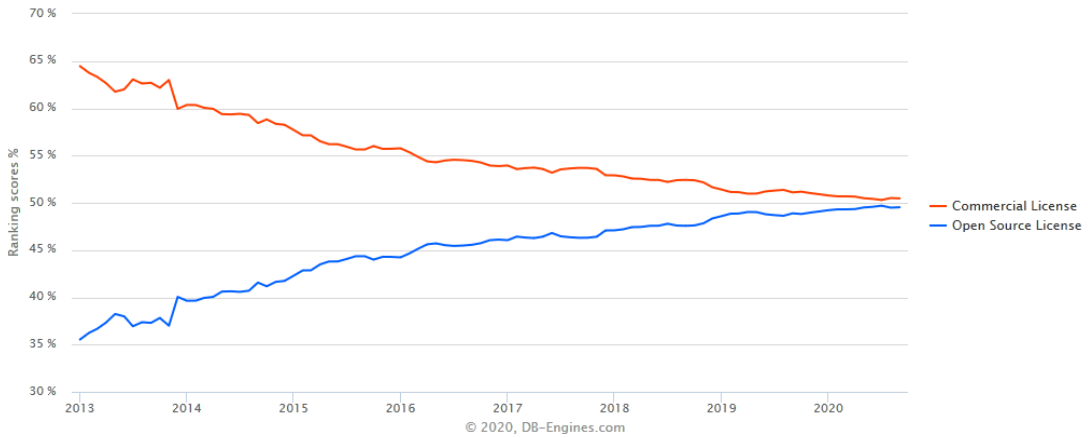


Figure 1.16 : Evolution des SGBD libres et propriétaires

Et un dernier classement DB-Engine donne les SGBD relationnels les plus utilisés.

Rank			DBMS
Sep 2020	Aug 2020	Sep 2019	
1.	1.	1.	Oracle
2.	2.	2.	MySQL
3.	3.	3.	Microsoft SQL Server
4.	4.	4.	PostgreSQL
5.	5.	5.	IBM Db2
6.	6.	↑ 7.	SQLite
7.	7.	↓ 6.	Microsoft Access
8.	8.	8.	MariaDB
9.	9.	↑ 10.	Teradata
10.	10.	↓ 9.	Hive

Figure 1.17 : Classement DB-Engine des SGBDR en 2020

Nous utiliserons pour ce cours indifféremment les SGBD MySQL ou MariaDB.

Part II

Le modèle relationnel

Chapter 2

La théorie Relationnelle

La modélisation relationnelle a été proposée pour la première fois en 1970 par Edgar F. CODD, dans l'article [1]. Cet article quoique rédigé il y a une trentaine d'années supporte bien les relectures. De nombreuses idées ont depuis été affinées, mais aucune modification révolutionnaire n'a été proposée. L'approche est basée sur la théorie mathématique des ensembles. Une base de données correspond à des ensembles. Pour extraire des données d'une base de données, on définit des opérateurs mathématiques dits relationnels, qui à partir d'un ou deux ensembles définissent un nouvel ensemble.
Ce chapitre qui a pour objectif de présenter la modélisation relationnelle est organisé en 2 parties :

- La première partie de ce chapitre a pour objectif de présenter les notions essentielles à la compréhension de la modélisation relationnelle, qui la différencient d'un simple ensemble de données. Nous verrons ainsi les définitions de base de la modélisation relationnelle ainsi que les propriétés des relations.
- L'ensemble des relations, représentant une base de données, est bien évidemment différent d'un ensemble de valeurs au sens mathématique. Ces valeurs doivent représenter à tout moment le monde réel. Nous verrons comment le modèle relationnel permet de gérer automatiquement la cohérence des données grâce aux contraintes d'intégrité.

Pour anticiper d'éventuelles remarques : la terminologie utilisée dans cette partie du cours est celle qui correspond à la théorie relationnelle. Les termes utilisés ici ne correspondent pas à ceux utilisés en analyse, et sont différents de ceux utilisés dans le contexte des SGBD.

Remarque : Les TDs correspondant à ce chapitre sont les exercices du chapitre 1 du document *TDs Base de Données*. Les corrigés des exercices de cette partie du cours sont dans le chapitre 2 du document *Corrigés des exercices du cours*.

2.1 Définitions de base

2.1.1 La notion de relation

La notion essentielle est bien évidemment la notion de relation qui est à la base de l'approche relationnelle.

*Une **relation**, R, sur un ensemble de domaines D1, D2, ..., Dn, est constituée de deux parties :*

- *l'en-tête : ensemble fixé d'attributs,*
- *le corps : ensemble de t-uplets.*

*Un **t-uplet** correspond à une ligne dans le corps de la relation, un **attribut** correspond à une colonne.*

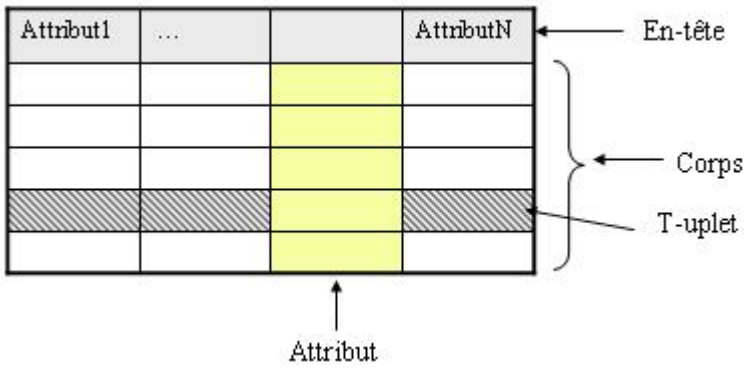


Figure 2.1 : Relation : tableau à deux entrées

Une relation (cf. figure 2.1) est vue comme une table à deux entrées. Les lignes correspondent à une information, les colonnes à une partie d'une information. Dans ce qui suit nous parlerons indifféremment de relation ou de table.

2.1.2 Autres définitions

Les propriétés suivantes définissent de façon plus précise une relation.

*Un **domaine** est un ensemble fini ou infini de valeurs admissibles pour un ou plusieurs attributs.* Chaque attribut d'une relation est défini dans un domaine.
*La **cardinalité** d'une relation est le nombre de t-uplets qui la composent.*

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

*Une **clé** est un groupe d'attributs dont la valeur permet d'identifier de manière unique un t-uplet de la relation.*

*Un **schéma de base de données relationnelle** ou **schéma relationnel** est un ensemble de relations qui ont toutes des noms différents.*

2.1.3 Quelques remarques sur ces définitions

- Les notions de cardinalité et de degré sont immédiates.
- Un domaine est composé de valeurs atomiques pour une base de données. C'est à dire qu'il n'est pas possible de décomposer une valeur d'un domaine. En effet si une valeur est décomposable cela signifie que si l'on désire atteindre à un moment donné une partie de l'information contenue dans cette valeur, cela risque de poser des problèmes au niveau de la recherche d'informations ou de la mise à jour des informations.
- La notion de clé est spécifique au domaine des bases de données. Nous reviendrons de façon plus complète sur cette notion essentielle en base de données.

2.1.4 Exemple

Considérons l'exemple de la relation `CINEMA`, dont le contenu est donné ci-dessous (table 2.1). Les attributs de cette relation sont :

Numerocinema, NomCinema, AdresseCinema, VilleCinema, CodePostalCinema et TelephoneCinema.

*Numero cinema	Nom Cinema	Adresse Cinema	Ville Cinema	CodePostal Cinema	Telephone Cinema
1	Vox	Grande Rue	Besancon	25000	03 81 82 05 69
2	Plazza	Rue Des Granges	Besancon	25000	03 81 26 35 98
3	Plazza	Rue Ronchaux	Dijon	21000	03 80 26 53 25
4	Royal	Rue Villarceau	Dijon	21000	03 80 23 52 58

Table 2.1 : Relation CINEMA

La première ligne de la relation (cf. le tableau 2.1) représente l'en-tête de la relation CINEMA, le reste qui correspond à l'ensemble des informations stockées dans la relation est le corps de la relation.
La relation est de degré 6, et de cardinalité 4 (la relation contient la description de 4 cinémas).
La clé primaire de la relation est constituée de l'attribut NumeroCinema (le nom de l'attribut est précédé d'un astérisque lorsque celui-ci fait partie de la clé).
Les domaines des attributs sont :

- NumeroCinema : ensemble d'entiers strictement positifs
- NomCinema : chaine de caractères appartenant à l'ensemble des noms de cinémas
- CodePostalCinema : ensemble des entiers compris entre 1000 et 99999
- AdresseCinema ,VilleCinema : chaines de caractères représentant des adresses ou des noms de ville
- TelephoneCinema : chaines de caractères d'un format très spécifique représentant des n° de téléphone.

Remarque : Les termes employés dans cette partie du cours peuvent sembler inhabituels pour des personnes ayant déjà manipulé des bases de données. En fait il existe différentes terminologies selon que l'on se place au niveau mathématique, au niveau des bases de données ou à un niveau plus technologique. Le tableau 2.2 présente ces différents termes que vous pourrez trouver dans la littérature.

Modèle Math.	Bases de Données	SGBD
Relation	Table	Fichier
T-uplet	Ligne	Enregistrement
Attribut	Colonne	Champ

Table 2.2 : Terminologies correspondant aux termes du relationnel

---

Exercice 2.1 : Donner pour la relation FILM représentée en 2.3 les informations suivantes :

- Liste des attributs
- Cardinalité et degré de la relation
- Le domaine de l'attribut Genre

Remarque : Les corrigés des exercices de cours sont dans le document *Corrigés des exercices de cours.*

---

*Titre	MetteurEnScene	Genre
Alerte	Peterson	Aventure
Bad Boys	Bay	Policier
Braveheart	Gibson	Aventure
Gazon Maudit	Balasko	Comédie
Junior	Reitman	Comédie
L'Effaceur	Russel	Action
Les Voleurs	Techine	Comédie Dramatique
Ma Saison Préférée	Techine	Comédie Dramatique

Table 2.3 : Relation FILM

La notion de *relation* est fondamentale dans le théorie relationnelle qui est basée sur la théorie mathématique des ensembles.

2.2 Propriétés des relations

Les propriétés décrites ci-dessous proviennent de la définition mathématique de la notion de relation. Nous verrons plus loin que ces propriétés sont spécifiques aux relations et non aux *tables* manipulées dans les SGBD.

Les relations possèdent les propriétés suivantes :
- pas de duplication de t-uplets,
- les t-uplets ne sont pas ordonnés de haut en bas,
- les attributs ne sont pas ordonnés de gauche à droite,
- les valeurs d'un attribut font toutes parties d'un même domaine,
- les valeurs des attributs sont atomiques.

Dans un schéma relationnel nous avons de plus les propriétés suivantes :
- une relation a un nom unique dans le schéma,
- un attribut dans une relation a un nom unique.

2.2.1 Pas de duplication de t-uplets

Cette propriété provient du fait que l'ensemble des t-uplets d'une relation (le corps de la relation) est un ensemble au sens mathématique. Une même valeur ne peut pas apparaître deux fois dans un ensemble.
De cette propriété découle le fait que toute relation possède une clé.

La relation `CINEMA` ne peut pas contenir deux fois la description du même cinéma. Il est ainsi impossible d'avoir la relation `CINEMA` suivante :

*Numero cinema	Nom Cinema	Adresse Cinema	Ville Cinema	CodePostal Cinema	Telephone Cinema
1	Vox	Grande Rue	Besancon	25000	03 81 82 05 69
1	Vox	Grande Rue	Besancon	25000	03 81 82 05 69

Table 2.4 : Relation CINEMA invalide

On a effectivement ici deux t-uplets parfaitement identiques. Par contre, on peut avoir par exemple la relation CINEMA suivante :

*Numero cinema	Nom Cinema	Adresse Cinema	Ville Cinema	CodePostal Cinema	Telephone Cinema
1	Vox	Grande Rue	Besancon	25000	03 81 82 05 69
2	Vox	Grande Rue	Besancon	25000	03 81 82 05 69

Table 2.5 : Relation CINEMA : T-uplets tous différents

Pour un humain, consultant ces deux t-uplets, on pourrait considérer que l'on représente deux fois le même cinéma, mais d'un point de vue base de données on a bien deux t-uplets différents.

2.2.2 Les t-uplets ne sont pas ordonnés

Cette propriété provient également du fait que le corps d'une relation est un ensemble au sens mathématique, et cet ensemble n'est pas ordonné. Il n'existe pas de deuxième t-uplet dans une relation, pas plus que de t-uplet suivant.

La relation CINEMA a été présentée avec quatre cinémas dans l'ordre : Vox, Plazza, Plazza, Royal. Si les t-uplets sont présentés dans un ordre différent la relation CINEMA reste quand même identique.
Ainsi la relation CINEMA présentée dans le tableau 2.6 est équivalente à celle présentée dans le tableau 2.7 .

*Numero cinema	Nom Cinema	Adresse Cinema	Ville Cinema	CodePostal Cinema	Telephone Cinema
1	Vox	Grande Rue	Besancon	25000	03 81 82 05 69
2	Plazza	Rue Des Granges	Besancon	25000	03 81 26 35 98
3	Plazza	Rue Ronchaux	Dijon	21000	03 80 26 53 25
4	Royal	Rue Villarceau	Dijon	21000	03 80 23 52 58

Table 2.6 : Relation CINEMA

*Numero cinema	Nom Cinema	Adresse Cinema	Ville Cinema	CodePostal Cinema	Telephone Cinema
4	Royal	Rue Villarceau	Dijon	21000	03 80 23 52 58
2	Plazza	Rue Des Granges	Besancon	25000	03 81 26 35 98
3	Plazza	Rue Ronchaux	Dijon	21000	03 80 26 53 25
1	Vox	Grande Rue	Besancon	25000	03 81 82 05 69

Table 2.7 : Relation CINEMA avec un autre ordre des t-uplets

2.2.3 Les attributs ne sont pas ordonnés

L'en-tête d'une relation est elle-même un ensemble d'attributs. Comme pour les t-uplets, les attributs ne sont pas ordonnés.
Les attributs sont référencés par un nom et non par une position dans l'en-tête. On peut ainsi utiliser une même relation ADRESSE pour traiter des adresses dans des pays différents où le code postal apparaitra avant ou après la ville.
Les deux entêtes présentées en 2.8 et 2.9 correspondent à la même relation.

*Numero cinema	Nom Cinema	Adresse Cinema	Ville Cinema	CodePostal Cinema	Telephone Cinema

Table 2.8 : En-tête de la relation CINEMA

*Numero cinema	Nom Cinema	Telephone Cinema	Adresse Cinema	CodePostal Cinema	Ville Cinema

Table 2.9 : Autre en-tête de la relation CINEMA

2.2.4 Les valeurs des attributs sont dans un seul domaine

Dans une relation les valeurs possibles d'un attribut sont données par son domaine. On ne peut pas associer plusieurs domaines à un même attribut (pas d'union de type sur les domaines).

Ainsi si on définit par exemple une relation *OBSERVATION* qui est formée de 2 attributs : *Jour* et *NbInd*. L'attribut *Jour* donne la date de l'observation et est donc de type date et l'attribut *"nbInd"* représente le nombre d'individus observés ce jour et est donc de type entier (cf.2.10).

Jour	NbInd
01/09/2020	3
05/09/2020	10
10/09/2020	"ND"
...	

Table 2.10 : Relation OBSERVATION

La valeur de l'attribut *NbInd* à la date du *10/09/2020* est *"ND"*. Cette valeur signifie pour l'utilisateur *"Non défini"*. L'utilisateur a donné cette valeur car il ne connaît pas le nombre dindividus observés à la date du *10/09/2020*. Si cette pratique est réalisable dans une feuille de calcul (et assez courante) dans une base de données relationnelle, ceci n'est pas possible. L'attribut *NbInd* doit contenir une valeur de type entier, il ne peut pas être de type entier ou chaîne de caractères.

2.2.5 Les valeurs des attributs sont atomiques

Cette propriété provient du fait que les domaines contiennent des valeurs atomiques c'est à dire des valeurs qui ne peuvent pas être divisées en plusieurs sous-valeurs de même type. Elle peut aussi se traduire par le fait que lorsque l'on examine dans une relation l'intersection d'une ligne et d'une colonne il ne peut y avoir qu'une seule valeur, et non plusieurs. Une relation qui satisfait cette propriété est normalisée : *première forme normale*. Dans le contexte relationnel toutes les relations seront normalisées.

Par exemple la relation `DISTRIBUTION` présentée en 2.11 n'est pas en première forme normale. On a ici plusieurs nom d'acteurs pour un même film. Si on se permet de mettre ce genre d'information dans cette relation il devient difficile de faire des recherches sur les acteurs d'un film. On ne peut pas écrire dans ce cas directement une requête donnant les films ayant comme acteur *"Freeman"*.

2.2.6 Les noms des relations et des attributs sont uniques

Le schéma de base de données ainsi que les relations ont été définis comme des ensembles (de relations ou d'attributs (pour l'en-tête)), et il n'est pas possible d'avoir deux fois le même élément dans un ensemble. Si dans un schéma de base de données on avait deux relations qui portent

Titre	NomActeur
Alerte	Freeman, Hoffman, Russo
Bad Boys	Lawrence, Loni, Smith
Braveheart	Gibson, Marceau
Gazon Maudit	Abril, Balasko, Chabat
Junior	De Mornay, De Niro, Schwarzenegger
L'Effaceur	Schwarzenegger

Table 2.11 : Relation DISTRIBUTION

le même nom, ceci entrainerait une impossibilité de référencer sans ambiguïté ces relations. On retrouve le même problème pour les attributs au sein d'une même relation.

Exercice 2.2 : On considère la relation `AUTREAFFICHE` donnée en 2.12 , qui donne pour chaque cinéma, la liste des films qui y sont à l'affiche. Expliquez pourquoi cette relation n'est pas correcte, et transformez la.

NumeroCinema	Titre
1	Alerte, Junior
2	Alerte, Gazon Maudit, L'Effaceur
3	Alerte, Junior
4	Alerte, Braveheart, Les Voleurs

Table 2.12 : Relation AUTREAFFICHE

Remarque : Une relation est toujours définie en extension, contrairement aux ensembles mathématiques, qui peuvent être décrits par intention ou par extension.
On ne peut pas définir, par exemple, une relation `CINEMABESANCON` en précisant simplement qu'il s'agit des cinémas de Besançon. Dans une base de données on a besoin de décrire toutes les données.

Maintenant que nous avons vu la notion de relation, ainsi que les propriétés de cette notion, nous pouvons définir ce qu'est, dans la théorie relationnelle, une base de données :

Schéma de base de données relationnelle = Ensemble de relations normalisées

2.3 Notions nécessaires à la présentation de l'intégrité des données

Nous venons de définir un schéma de base de données comme étant un ensemble de relations normalisées. Dans la réalité il y a bien évidemment des différences entre un schéma de base de données, c'est-à-dire des ensembles au sens mathématique et une *base de données* qui correspond à un ensemble de données représenté grâce à un schéma relationnel. En particulier nous parlerons de *table* pour nommer la représentation d'une relation dans une base de données. Le terme de *relation* sera utilisé dans ce qui suit lorsque nous ferons allusion à la modélisation relationnelle, et le terme de *table* sera utilisé lorsque que nous manipulerons un ensemble de données dans une base.

Une base de données contient une configuration particulière des valeurs de données, qui est sensée représenter la *réalité*.

- Le code postal d'un cinéma ne peut pas, par exemple prendre la valeur 1. Cette valeur doit être comprise entre 1000 et 99999.
- De même on ne peut pas mettre à l'affiche d'un cinéma, un film qui n'existe pas.

*Les **contraintes d'intégrité** sont des règles qui permettent au SGBD de préserver automatiquement la cohérence de la base de données en :*

- *vérifiant les données lors de leur chargement,*
- *vérifiant les données lors de toute modification (saisie, mise à jour),*
- *répercutant certaines mises à jour entre tables,*
- *gérant les références entre tables.*

Ainsi la définition d'une base de données est étendue par l'ajout de **règles d'intégrité** de natures différentes :

- l'*unicité de clé* que nous avons déjà évoquée dans les propriétés des relations,
- les *contraintes de domaine*, qui permettent de vérifier l'intégrité des attributs,
- la *règle d'intégrité des entités* qui permet de préciser quelles valeurs doivent obligatoirement être définies,
- et enfin les *contraintes de références*, qui font appel aux notions de clés étrangères et qui permettent de vérifier la cohérence des informations entre différentes relations.

D'autres types de contraintes sont proposées par R. Gardarin, pour plus d'informations il est possible de consulter entre autres les documents [3] ou [4].

Les contraintes d'intégrité sont des règles qui permettent au SGBD de gérer automatiquement la cohérence de la base de données.

Avant de d'expliquer ces règles, nous présentons deux notions nécessaires pour leur compréhension.

2.3.1 Les clés relationnelles

Une clé est un groupe d'attributs qui identifie chaque t-uplet d'une relation ; en relationnel nous distinguons différents types de clés : *super-clé, candidate, primaire et étrangère.*

2.3.1.1 Les super-clés

*Une **super-clé** pour une relation R est un sous-ensemble de l'ensemble des attributs de R qui identifie de façon unique chaque t-uplet de la relation.*

On dit aussi qu'une super-clé possède la **Propriété d'unicité**.
***Propriété d'unicité :** Il n'existe pas deux t-uplets distincts de R ayant la même valeur pour cet ensemble d'attributs.*

Une super-clé identifie chaque t-uplet de la relation. On peut avoir pour une même relation plusieurs super-clés.

Exercice 2.3 : Sachant que chaque t-uplet de la relation `CINEMA` est identifié par l'ensemble d'attributs `(NumeroCinema)`, donnez pour cette relation toutes ses super-clés.

2.3.1.2 Les clés candidates

*Une **clé candidate** pour une relation R, est une super-clé qui possède la propriété d'irréductibilité.*
***Propriété d'irréductibilité :** Un ensemble d'attributs K est irréductible si aucun sous-ensemble strict de K n'est une super-clé.*

Considérons la relation CINEMA, nous avons vu qu'elle est constituée des attributs :
`(NumeroCinema, NomCinema, AdresseCinema, VilleCinema, CodePostalCinema, TelephoneCinema)`
et nous avons, de plus, précisé que sa clé est l'attribut `NumeroCinema`. Si `NumeroCinema` est une clé candidate de la relation `CINEMA` cela veut dire que deux cinémas distincts ne peuvent pas avoir le même numéro. De plus cette clé étant réduite à un seul attribut, elle est irréductible. Supposons maintenant que deux cinémas distincts ne puissent pas avoir le même nom, dans ce cas, l'attribut `NomCinema` serait aussi une clé candidate de la relation `CINEMA`.

Une même relation peut avoir plusieurs clés candidates. Considérons l'exemple d'une relation `ETUDIANT` (cf. table 2.13) contenant entre autres les attributs `NumInsee` et `NumEtudiant`. Le numéro `Insee` identifie de façon unique chaque personne, et pour l'université le numéro d'étudiant est unique pour chaque étudiant, et on ne peut pas avoir deux étudiants portant les mêmes nom et prénom.

NomEtudiant	PrenomEtudiant	NumEtudiant	NumInsee	NumDevoir
Alex	Dupont	12345	1856722	devoir01
Alain	Lands	67890	1826814	devoir01
Alex	Dupont	12345	1856722	devoir02
Alain	Pruys	11007	1772589	devoir01
Alain	Pruys	11007	1772589	devoir02
Maud	Dupont	54321	2867560	devoir02

Table 2.13 : La relation ETUDIANT

Pour déterminer de façon précise chaque t-uplet, on peut par exemple donner `(NomEtudiant, PrenomEtudiant, NumEtudiant, NumDevoir)`, mais cette super-clé n'est pas une clé candidate parce qu'elle est réductible. Elle a 2 sous-ensembles qui identifient chaque t-uplet. Ce sont :

- `(NomEtudiant, PrenomEtudiant, NumDevoir)`
- `(NumEtudiant, Numdevoir)`

Remarque : Attention avoir deux clés candidates est différent d'avoir une clé candidate formée de deux attributs. Reprenons l'exemple des étudiants, nous avons ici des clés candidates formées de 2 attributs:

- `(NumEtudiant, NumDevoir)`
- `(NumInsee, NumDevoir)`

et aussi une autre clé candidate, formée de trois attributs, qui est :

- `(NomEtudiant, PrenomEtudiant, NumDevoir)`

Titre	MetteurEnScene	Genre
Alerte	Peterson	Aventure
Bad Boys	Bay	Policier
Braveheart	Gibson	Aventure
Gazon Maudit	Balasko	Comédie
Junior	Reitman	Comédie
L'Effaceur	Russel	Action
Les Voleurs	Techine	Comédie Dramatique
Ma Saison Préférée	Techine	Comédie Dramatique

Table 2.14 : Relation FILM

Exercice 2.4 : On considère la relation `FILM` présentée en 2.14 . Donner pour cette relation les clés candidates pour chacun des cas suivants :

1. Le titre d'un film est unique.
2. Les titres de film ne sont pas uniques, il existe en effet plusieurs versions de King Kong. Mais pour un même titre on ne peut avoir plusieurs fois le même metteur en scène.
3. On suppose à nouveau que le titre d'un film est unique. Pourquoi le sous-ensemble d'attributs : `{Titre, MetteurEnScene}` n'est-il pas clé candidate de la relation `FILM` ?

2.3.1.3 Les clés primaires

Il est possible de définir plusieurs clés candidates pour une relation. D'où l'introduction de la notion de clé primaire, qui elle, est unique pour une relation donnée.

*Une **clé primaire** est une clé choisie arbitrairement parmi les clés candidates.*

Les clés primaires fournissent au SGBD un **mécanisme d'adressage** dans la table, car elles permettent de repérer un t-uplet dans celle-ci.

Dans le cas de la relation `CINEMA`, présentée ci-dessus (les noms de cinéma sont uniques) nous pouvons choisir comme clé primaire indifféremment : `{NumeroCinema}` ou `{NomCinema}`
Si nous choisissons la première clé comme clé primaire, cet attribut aura automatiquement des valeurs uniques, par contre il faudra alors préciser cette contrainte pour l'autre clé candidate.

Si l'on accède aux données correspondant à `NumeroCinema = 1`, on accède à un t-uplet. Maintenant si l'on accède aux données correspondant à `VilleCinema="Besançon"`, on obtient un nombre, à priori inconnu, de t-uplets.

2.3.1.4 Les clés étrangères

Certains attributs ou ensembles d'attributs peuvent apparaître dans plusieurs relations. Nous définissons ici cette *"liaison"* entre différentes relations.

*Soit R une relation, une **clé étrangère** dans R est un sous-ensemble C, de l'ensemble des attributs de R,tel que :*

- *Il existe une relation R' avec une clé primaire C', et*

– *à tout instant, chaque valeur de C dans la relation R, est identique à une valeur de C' dans la relation R'.*

On dit aussi que l'on définit un lien entre les relations R' et R.

Remarque : Cette définition aurait pu être donnée en considérant que C' est une clé candidate. Dans la pratique il s'agit de la clé primaire. En effet les SGBD ne gèrent pas explicitement la notion de clé candidate et si celle-ci est composée de plusieurs attributs les SGBD ne peuvent pas gérer l'unicité des valeurs d'un tel groupe d'attributs si celui-ci n'est pas déclaré en tant que clé primaire. Pour simplifier la notion de lien nous considèrerons donc dans ce cours que le lien est toujours construit à partir d'une clé primaire.

Reprenons par exemple les relations `CINEMA` et `AFFICHE`. L'attribut `NumeroCinema` est la clé primaire de la relation `CINEMA`, et c'est une clé étrangère pour la relation `AFFICHE`.

On a l'habitude de représenter un tel lien entre 2 relations au moyen d'une flèche allant de la clé primaire vers la clé étrangère, comme le montre la figure 2.2 .

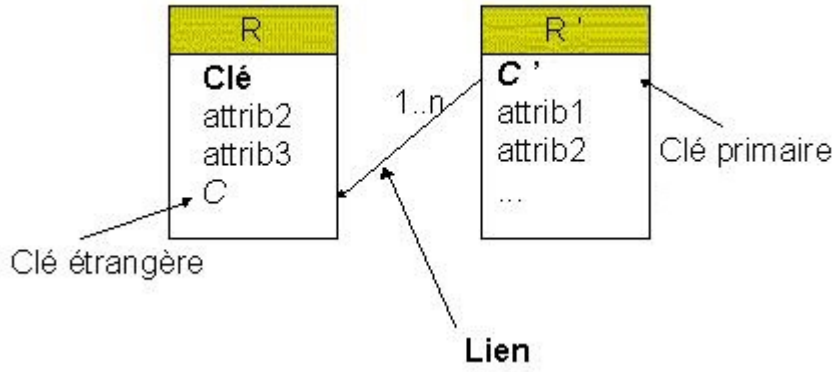


Figure 2.2 : Représentation graphique d'un lien

On note aussi ce lien de la façon suivante :

```
R'[C'] (? .. ?) -> [C] R
```

Notation : Nous prendrons l'habitude de noter les liens entre relations, en plaçant entre crochets les attributs impliqués dans chacune des relations.

Le lien entre les relations `CINEMA` et `AFFICHE`, noté :
CINEMA(NumeroCinema, AdresseCinema, VilleCinema, CodePostalCinema, TelephoneCinema)[NumeroCinema] (1..n) -> [NumeroCinema] AFFICHE(NumeroCinema,Titre)
est représenté par la figure 2.3 .

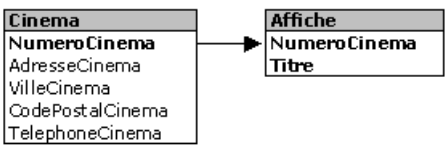


Figure 2.3 : Lien entre les relations CINEMA et AFFICHE

Un lien entre deux relations, peut être de différents types, selon le nombre de fois ou la valeur

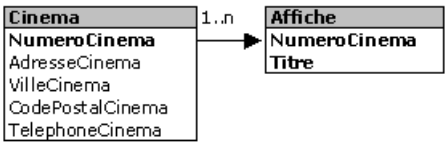


Figure 2.4 : Lien 1..n entre les relations CINEMA et AFFICHE

de la clé primaire de la relation source peut apparaître en tant que clé étrangère dans la relation cible (ce que nous avons représenté ci-dessus par *(? .. ?)*). On définit deux types de liens :

- **Lien 1..n :** une valeur de la clé primaire peut apparaître 0 à n fois en tant que clé étrangère.
- **Lien 1..1 :** une valeur de la clé primaire ne peut apparaître qu'une et une seule fois en tant que clé étrangère.

Dans l'exemple du lien que nous avons vu précédemment :
`CINEMA(*NumeroCinema, NomCinema, AdresseCinema, VilleCinema, CodePostalCinema, TelephoneCinema)[NumeroCinema] (1..n) -> [NumeroCinema] AFFICHE(*NumeroCinema, *Titre)`
On a un lien de type 1..n (cf. figure 2.4), ce qui signifie qu'un même numéro de cinéma peut apparaitre 0 ou plusieurs fois dans la relation `AFFICHE`, c'est à dire qu'un cinéma peut avoir plusieurs films à l'affiche.

Remarques :

- Un lien est toujours défini d'une clé primaire vers une clé étrangère.
- Un lien peut aller d'une relation R vers elle-même.
- Attention dans la modélisation relationnelle un lien est toujours de type ***1 .. x***. Nous ne sommes pas en analyse où l'on peut définir des associations de type ***0 ..x***. Il ne faut pas confondre ces deux notions.

Exercice 2.5 : Les genres de films
On introduit la relation `GENRE` dans le modèle relationnel des cinémas, et on modifie la relation `FILM` : l'attribut `Genre` devient l'attribut `NumeroGenre`. Les relations `FILM` et `GENRE` sont définies de la façon suivante :
`FILM(*Titre, MetteurEnScene, NumeroGenre)`
`GENRE(*NumeroGenre, Genre)`
1- Donner la nature du lien entre les relations `GENRE` et `FILM`.

Exercice 2.6 : Et les directeurs de cinéma
On introduit la relation `DIRECTEUR` dans le modèle relationnel des cinémas. Cette relation est définie de la façon suivante :
`DIRECTEUR(*NumeroDirecteur, Nom, Prenom, TelephoneDirecteur, NumeroCinema)`
1 - Sachant qu'un cinéma ne possède qu'un seul directeur, et qu'une personne ne peut être directeur que d'un seul cinéma. Définir le lien entre les relations `CINEMA` et `DIRECTEUR`.
2 - Est-il possible d'introduire la notion de directeur sans introduire la relation `DIRECTEUR`, mais en modifiant seulement la relation `CINEMA`? Si oui modifier la relation `CINEMA` en conséquence.
3 - Comparer les deux solutions.

2.3.2 La valeur NULL

Il n'est pas toujours possible de renseigner complètement toutes les données d'une base de données. Ce qui dans certains cas peut poser des problèmes et entraîner la saisie d'informations non identifiables. Pour palier à ce problème la valeur NULL a été introduite.

*Le **NULL** est une valeur conventionnelle introduite dans une relation lorsque que la valeur d'un des attributs est inconnue ou inapplicable. Il représente l'absence de valeur.*

L'introduction d'une telle valeur dans une base de données a pour objectif de permettre le traitement de l'ensemble de la base de données, même en l'absence de certaines données. Lors de la conception de la base de données on précise si un attribut est autorisé ou non à prendre la valeur NULL. Dans le cas où l'attribut peut prendre la valeur NULL, cette valeur lui sera attribuée automatiquement par le SGBD, dès que l'utilisateur ne précisera pas la valeur de l'attribut. Dans le cas contraire le SGBD rejettera tout t-uplet où la valeur de l'attribut ne serait pas renseignée.

Considérons la relation CINEMA, l'attribut TelephoneCinema peut être autorisé à prendre la valeur NULL. On pourrait ainsi obtenir un t-uplet de la forme :
`(1, Vox, Grande Rue, Besançon, 25000, NULL).`

2.4 Les règles d'intégrité des données

Maintenant que toutes les notions nécessaires ont été décrites, nous pouvons présenter les règles d'intégrité des données.

2.4.1 L'unicité de clé

Cette première règle d'intégrité, se base sur la notion de clés **candidate et primaire**.

Règle de l'unicité de clé : *Dans une base de données, toutes les relations doivent posséder une clé unique, appelée clé primaire.*

Dans un ensemble, au sens mathématique, les éléments sont tous distincts. Il en va de même pour les relations définissant une base de données relationnelle. Cependant cette première règle permet de mettre en place des mécanismes de contrôle, qui vérifient à tout instant que deux informations possédant les mêmes caractéristiques ne peuvent pas être présentes dans la base de données.

Il ne sera pas possible par exemple d'introduire dans la relation CINEMA (Table 2.6), le t-uplet suivant :

(1 , Stella, rue des roses, Paris, 75012, 01 56 41 23 78)

En effet le numéro 1 est déjà attribué.

Remarque : Dans la plupart des SGBD, lors de la création d'une relation, celui-ci demande toujours à l'utilisateur de définir la clé primaire de la relation.

Exercice 2.7 : On considère la relation FILM présentée dans l'exemple du cours (cf. 2.14), dont la clé primaire est {Titre}
Que se passe-t-il lorsque l'on ajoute les t-uplets suivants?
1 - (Alerte, Bay, Aventure)
2 - (Zorro, Peterson, Aventure)

2.4.2 Les contraintes de domaine

Chaque attribut d'une relation prend ses valeurs dans un domaine, dont nous avons vu qu'il s'agissait d'un ensemble. Dans la pratique, les domaines correspondent souvent à des types de base tels que : entier, réel, chaine de caractères, ...

Si l'on considère, par exemple un attribut tel que `CodePostalCinema`, dans la relation `CINEMA`, on constate que le domaine de cet attribut est l'ensemble des entiers compris entre 1000, et 99999.

*Une **contrainte de domaine** est une contrainte d'intégrité qui définit la propriété que doit vérifier toute valeur d'un attribut d'une relation donnée.*

Ce type de contrainte, traité par le SGBD, permet lors de la saisie ou la mise à jour d'information, de vérifier la cohérence des données.

Exercice 2.8 : On définit la relation `FILM1` de la façon suivante :
`FILM1(Titre, MetteurEnScene, Genre, AnneeSortie)`.
Donnez des contraintes de domaine possibles concernant l'année de sortie du film.

2.4.3 La gestion de `NULL`

Comme nous l'avons présenté `NULL` permet de spécifier le fait qu'une valeur n'est pas définie lors de la saisie d'un t-uplet. Mais lorsque l'attribut concerné fait partie de la clé primaire ou d'une clé étrangère que se passe-t-il ?

2.4.3.1 Les clés primaires et `NULL`

Il n'est pas possible d'enregistrer dans une base de données des informations que l'on ne peut pas identifier, ainsi tous les attributs ne peuvent prendre la valeur `NULL`. En effet la règle de l'unicité de clé (présentée comme première règle pour la gestion de l'intégrité des données), qui précise que chaque relation a une unique clé primaire, impose la connaissance de cette clé afin de vérifier l'unicité de sa valeur. D'où la règle suivante :

***Règle d'intégrité des entités :** Tous les attributs appartenant à la clé primaire d'une relation ne sont pas autorisés à prendre la valeur **NULL**.*

Considérons à nouveau la relation `CINEMA`, l'attribut `NumeroCinema` qui est la clé primaire de cette relation, ne peut être autorisé à prendre la valeur `NULL`.

2.4.3.2 Les clés étrangères et `NULL`

Modifions légèrement notre modèle relationnel de la base de données CINEMAS, en introduisant une relation `GENREFILM` définie par `(*NumGenre, NomGenre)`. Dans ce modèle on introduit un lien entre les relations `GENREFILM` et `FILM` :
`GENREFILM(*NumGenre, NomGenre) [NumGenre](1..n) -> [Genre] FILM(*TitreFilm, MetteurEnScene, Genre)`
Ce qui signifie que tout film a un genre, qui peut apparaître pour 0 ou plusieurs films. Maintenant supposons que l'on introduise le t-uplet suivant dans la relation `FILM`:
`(Godzilla, Emmerich, NULL)`
L'interprétation de ce t-uplet peut être la suivante : on a un film appelé Godzilla, mis en scène

par Emmerich, dont on ne connaît pas le genre.

Le problème est de savoir si un tel t-uplet respecte ou non la règle d'intégrité référentielle. Cette règle (que nous verrons en détail ci-après) précise que toute valeur d'une clé étrangère doit contenir une valeur unifiable. Cette définition peut être étendue pour prendre en compte la valeur `NULL`. Cette possibilité est laissée au concepteur de la base de données qui pourra préciser si oui ou non une clé étrangère peut prendre la valeur `NULL`. Dans ce qui suit nous supposons que sauf spécification contraire, une clé étrangère peut prendre la valeur `NULL`. Sauf, bien sûr, si celle-ci fait partie de la clé primaire.

Exercice 2.9 : On introduit la relation `DIRECTEUR` dans le modèle relationnel des cinémas. Cette relation est définie de la façon suivante :
`DIRECTEUR(*NumeroDirecteur, Nom, Prenom, TelephoneDirecteur, NumeroCinema)`
avec le lien suivant entre les relations `CINEMA` et `DIRECTEUR` :
`CINEMA(* NumeroCinema, NomCinema, ...) [NumeroCinema] (1 .. 1) -> NumeroCinema`
`DIRECTEUR(*NumeroDirecteur, Nom, Prenom, TelephoneDirecteur, NumeroCinema)`
1 - L'attribut `Nom` peut-il prendre la valeur `NULL`?
2 - L'attribut `NumeroDirecteur` peut-il prendre la valeur `NULL`?
3 - L'attribut `NumeroCinema` peut-il prendre la valeur `NULL`?

2.4.4 Les contraintes référentielles

Les contraintes référentielles permettent au SGBD de gérer automatiquement la présence de données référencées dans différentes relations de la base.

Reprenons l'exemple du cinéma, dans la relation `DISTRIBUTION` l'attribut `Titre` fait référence au film dans lequel joue l'acteur. Ainsi le t-uplet `("Junior", "Schwarzenegger")`, signifie que l'acteur *Schwarzenegger* joue dans le film *Junior*. La valeur prise par l'attribut `Titre` ne peut être qu'une valeur apparaissant dans la relation `FILM`. En effet il ne serait pas cohérent de décrire la distribution d'un film qui n'existe pas.

On associe à la notion de clé étrangère, la règle suivante qui permet de conserver un ensemble de données cohérent dans la base de données.

L'intégrité référentielle : *La base de données ne doit pas contenir de valeurs de clés étrangères* ***non unifiables****.*
indexIntégrité référentielle
Ceci signifie que si une clé étrangère prend une valeur alors cette valeur existe dans la relation référencée par le lien.

Dans l'exemple du lien : `FILM [Titre] (1..n) -> [Titre] DISTRIBUTION`,
ceci signifie qu'il ne peut exister dans la relation `DISTRIBUTION` de t-uplet (Zorro, acteur) si dans la table `FILM`, il n'existe pas de t-uplet (Zorro, ...).

La notion d'intégrité référentielle étant définie, il faut que le SGBD soit capable de gérer les situations où un utilisateur réalise des opérations qui invalident cette règle :

- une première possibilité est de refuser simplement toute modification de cette nature,
- une seconde est d'accepter cette opération, et de prendre en compte en plus un certain nombre d'opérations supplémentaires, nécessaires pour garantir la cohérence des données.

Les **règles de mises à jour** permettent de définir la stratégie de prise en compte de ce type d'opérations.

2.4.4.1 Les règles de mise à jour

Lors de la saisie de la valeur d'une clé étrangère, le contrôle permettant de savoir si la valeur existe dans la relation référencée est réalisé immédiatement. Par exemple lorsque l'on saisit l'affiche du film *Zorro*, si celui-ci n'existe pas dans la relation `FILM`, la saisie est refusée immédiatement.
Par contre que se passe-t-il lorsque l'on modifie les valeurs de la clé primaire ? Supposons maintenant que le film Zorro existe et qu'il soit à l'affiche de 4 cinémas. Que se passe-t-il si l'on veut changer le nom du film, ou supprimer ce film ?

Deux possibilités s'offrent à nous :
- On peut refuser de mettre à jour ce film tant qu'il existe dans la relation `AFFICHE`, des t-uplets ayant *Zorro* comme titre,
- ou mettre à jourle film *Zorro* dans tous les t-uplets de la relation `AFFICHE`.

Les règles de mise à jour concernent les *opérations de modification ou de suppression des données.*
- `RESTRICT ou NO ACTION` : *L'opération de mise à jour est "restreinte" au cas où aucune référence à cette valeur n'existe dans les autres relations en lien avec la relation de base, dans les autres cas elle est interdite.*
- `CASCADE` : *L'opération de mise à jour est réalisée en "cascade" dans les autres relations. Cette règle est en général choisie pour la modification d'une clé candidate.*
- *`SET NULL` : Lorsqu'une mise à jour de la clé candidate est réalisée, les valeurs des clés étrangères correspondantes sont mises à NULL. Dans ce cas les clés étrangères doivent pouvoir prendre la valeur NULL.*
- *`SET DEFAULT` : Lorsque l'on réalise la suppression d'un t-uplet dans la relation source, les t-uplets de la relation cible qui le référencent prennent une valeur* par défaut. *La clé étrangère doit pouvoir ici prendre plusieurs fois la même valeur.*
- *`NO CHECK` : Aucun contrôle n'est réalisé lors de la suppression d'un t-uplet dans la relation source.*

Remarque : Dans certains SGBD on distingue les règles de type RESTRICT et NO ACTION mais dans beaucoup de SGBD il n'y a pas de différence.

Dans l'exemple proposé ci-dessus, la solution consistant à refuser de modifier le titre du film Zorro, correspond à une règle de mise à jour `RESTRICT ou NO ACTION`, la deuxième solution à une règle de mise à jour `CASCADE`.

Lors de la définition d'un lien entre deux relations il est possible de choisir l'application de ces règles pour la modification, et la suppression. En règle générale seule la règle CASCADE est utilisée pour les modifications de la valeur de la clé primaire de la relation source. Par contre pour une suppression, ces différentes règles peuvent être utilisées en fonction du contexte.
Les **contraintes référentielles** permettent au SGBD de gérer automatiquement la présence de données référencées dans différentes relations de la base. La notion de lien permet de spécifier de telles propriétés.
Au niveau des SGBD différentes politiques de mises à jour peuvent être mises en oeuvre : **NO ACTION**, **CASCADE**, **SET NULL**, **SET DEFAULT** ou **NO CHECK**. Dans MariaDB (2.5) on propose les politiques RESTRICT, ACTION, CASCADE et SET NULL.

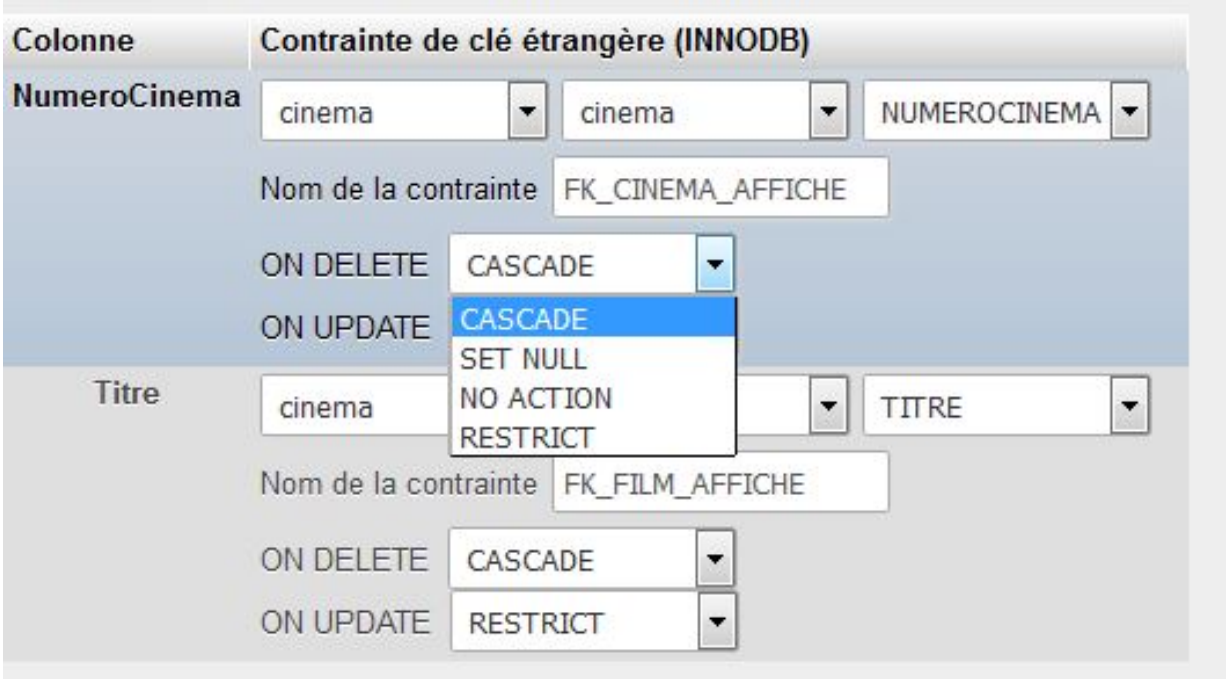


Figure 2.5 : Définition d'une politique de mise à jour sous MariaDB

Chapter 3

L'exemple des cinémas

Nous venons de voir la représentation des données dans la modélisation relationnelle, ainsi qu'un certain nombre de règles qui permettront au SGBD de gérer automatiquement la cohérence des données dans la base.

Nous introduisons maintenant de façon détaillée l'exemple de la base de données `CINEMAS` qui sert de support pour les exemples du cours.
La base de données CINEMAS doit permettre de gérer l'ensemble des cinémas français, ainsi que les films qui y sont à l'affiche. Intuitivement nous allons décrire les cinémas, les films, ainsi que les affiches de chaque cinéma, et la distribution de chaque film.
Il est bien évidemment possible de réaliser une base de données comprenant plus d'informations. Mais nous nous limiterons à une base de données nous permettant de connaître les films à l'affiche dans tel cinéma, les cinémas qui ont à l'affiche tel film, les films de tel genre, dans telle ville, les films avec tel acteur,...

La création d'un modèle relationnel suppose la réalisation d'une analyse. Les techniques et méthodes de conception d'un tel modèle seront vues dans le module d'analyse et modélisation des systèmes d'informations (AMSI). Nous n'insisterons donc pas dans l'enseignement de base de données sur cet aspect.

La présentation détaillée de cet exemple a pour rôle de préciser les notions que nous venons de présenter concernant la modélisation relationnelle.

Avant de décrire les relations de la base de données `CINEMAS`, nous présentons les types classiques d'attributs que l'on peut rencontrer dans un SGBD.

3.1 Les types de données dans les SGBD

Lors de la conception de la base de données, il faut définir complètement chacune des relations intervenant dans le modèle relationnel. Pour cela on défini, entre autres, chacun des attributs de la relation, en précisant *son nom, son rôle, et son type.*

Comme dans tous les langages de programmation les SGBD fournissent à l'utilisateur un certain nombre de types de base qu'il peut utiliser pour définir ses données. De légères variations peuvent apparaitre concernant ces types de base d'un SGBD à l'autre.
On trouve différents types de bases :

- numériques
- chaines de caractères

- dates
- autres types (types géométriques, valeur NULL)

Les données de type texte sont bien évidemment beaucoup plus largement employées que dans les langages de programmation. Nous présentons ci-dessous les types de données proposés dans le SGBD MariaDB (voir le site de MariaDB pour des informations plus complètes [10]).

3.1.1 Les chaines de caractères

Elles permettent de stocker des informations de type littéral. Dans l'exemple du cinéma, on utilisera ce type pour définir les noms des cinémas, les villes, les numéros de téléphone, ...
On précise de plus le nombre de caractères qui peut être utilisé. Dans un SGBD tel que MariaDB ([11]) on a les types de données caractères suivants (cf. Tab. 3.1):

Type	**Description**	**Remarques**
String Literals	Chaine de caractères mis entre quotes	ex : 'MariaDB'
CHAR[(n)]	Chaine fixe de n caractères, complétée à droite par des espaces. Si n est omis la longueur de la chaine est 1.	Taille fixe. 0 à 255 car.
VARCHAR(n)	Chaine de longueur variable de 0 à n car. ou octets	Taille variable Maxi 65635 octets ou car.
BINARY(n) CHAR BYTE	Similaire au type CHAR, mais stocke des chaînes d'octets binaires plutôt que des chaînes de caractères non binaires	n : longueur de la colonne en octets
VARBINARY(n)	Similaire au type VARCHAR, mais stocke des chaînes d'octets binaires plutôt que des chaînes de caractères non binaires	n : longueur de la colonne en octets
BLOB	Grand objet binaire. Peut contenir une quantité variablede données. TINYBLOB, BLOB, MEDIUMBLOB ou LONGBLOB	
TEXT[(n)]	Chaine de caractères non indexée. TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT ou JSON	Long. maxi 65 535 caractères
ENUM	Objet chaîne dont la valeur est choisie dans la liste des valeurs 'value1','value2', ..., NULL	maximum de 65 535 valeurs distinctes
SET('value1', 'value2',...)	Peut avoir 0 ou plusieurs valeurs, dont chacune est choisie dans la liste des valeurs	Maxi 64 éléments

Table 3.1 : Les chaines de caractères sous MariaDB

3.1.2 Les données numériques

Ces données sont essentiellement des entiers ou des réels (cf. tableau 3.2). Selon les SGBD on peut avoir différents types d'entiers ou de réels. Ces types numériques peuvent être définis comme `'SIGNED'`, `'UNSIGNED'` ou `'ZEROFILL'`.

- `'SIGNED'` : une partie du type numérique sera réservée pour le signe (plus ou moins).
- `'UNSIGNED'` : aucune partie du type numérique n'est réservée au signe, de sorte que pour les types entiers, la plage peut être plus grande.
- `'ZEROFILL'` : la colonne sera définie comme étant UNSIGNED et les espaces utilisés par défaut pour remplir le champ sont remplacés par des zéros.

Type	Description	Remarques
BIGINT	nombre entier étendu	de -9223372036854775808 à 9223372036854775807
BOOLEAN BOOL	synonyme de TINYINT(1)	zéro vaut FAUX (false) Autres valeurs valent VRAI (true)
SMALLINT	petit nombre entier	de -32768 à 32767 ou de 0 à 6553535
MEDIUMINT	entier de taille moyenne	de -8388608 à 8388607 ou de 0 à 16777215
INT INTEGER	entier de taille normale	de -2147483648 à 2147483647 ou de 0 à 4294967295
BIGINT	grand nombre entier	de -92233720362036854775808 à 9223372036854775807
DECIMAL[(M[,D]] DEC, NUMERIC, FIXED	M : nbre total de chiffres ; D : nbre de chiffres après le point décimal	
FLOAT	Petit nombre à virgule flottante simple précision	de -3.402823466E+38 à -1.175494351E-38
DOUBLE | REAL DOUBLE PRECISION	nombre à virgule flottante de taille normale	de -1.7976931348623157E+308 à -2.2250738585072014E-308
BIT	champ de bits	de 1 à 64

Table 3.2 : Les types numériques sous MariaDB

3.1.3 Les dates

Ce type de données est essentiel en base de données, les données de type date étant fréquentes à stocker dans une base de données.

Type	Description	Remarques
DATE	affiche les valeurs DATE dans le format'YYYYYY-MM-DD'	de'1000-01-01' à'9999-12-31'
TIME	affiche les valeurs de temps au format 'HH:MM:SS:SS.ssssss'	de '-838:59:59.999999' à '838:59:59.999999'
DATETIME	affiche des valeurs au format 'YYYY-MM-DD HH:MM:MM:SS'.	de '1000-01-01 00:00:00.000000' à '9999-12-31 23:59:59.999999'
TIMESTAMP	généralement utilisé pour définir à quel moment un enregistrement a été ajoutée ou mise à jour format : YYYY-MM-DD HH:MM:SS	de '1970-01-01 00:00:01' à '2038-01-19 05:14:07'
YEAR	format d'année à quatre chiffres	0000 et de 1901 à 2155

Table 3.3 : Les dates sous MariaDB

3.1.4 Les autres types de données

De plus en plus les bases de données sont amenées à stocker de nouveaux types de données tels que des documents, des images, des sons, des données spatiales... Les SGBD proposent aux utilisateurs des formats de données capables de stocker ce type d'informations.
Sous MariaDB on a les types `BLOB` qui permettent de stocker des données non structurées.

3.1.4.1 Les types géométriques

MariaDB fournit un moyen standard de créer des colonnes spatiales pour les types de géométrie. On trouve ainsi les différents types 3.4 .
WKB est l'abréviation de *'Well-Known Binary'*, un format de représentation des données géographiques et géométriques.

Type	Description
POINT	Définit par les valeurs des 2 coordonnées
LINESTRING	Définit des morceaux de droite à partir de points
POLYGON	Définit un polygone à partir de linestring
MULTIPOINT	Ensemble de points
MULTILINESTRING	Ensemble de linestring
MULTIPOLYGON	Ensemble de polygones
GEOMETRYCOLLECTION	Collection géométrique
GEOMETRY	N'importe quel type géométrique

Table 3.4 : Les types de géométrie

3.1.4.2 L'auto-incrément

L'attribut AUTO_INCREMENT peut être utilisé pour générer un identifiant pour les nouveaux enregistrements. Lorsque l'on insère un nouvel enregistrement dans la table et que le champ auto_increment est NULL ou DEFAULT, la valeur sera automatiquement incrémentée.
Les colonnes AUTO_INCREMENT commencent à partir de 1 par défaut. La valeur générée automatiquement ne peut jamais être inférieure à 0.
L'auto-incrément est un outil des SGBD qui permet leur de gérer de façon automatique des clés. Un attribut auto-incrément est toujours la clé primaire de la table. Et donc si il existe un lien de cette table vers une autre ce sera forcément avec cet attribut auto-incrément. L'intérêt de l'auto-incrément est que le SGBD fournit à chaque t-uplet de la table un identifiant unique et si ce t-uplet est détruit la valeur de l'identifiant ne pourra pas être ré-utilisée. Et cela est important car chaque t-uplet est ainsi identifié de façon unique et un identifiant ne représente qu'un seul t-uplet dans la table.
La déclaration d'un auto-incrément est montré dans la figure 3.1 .

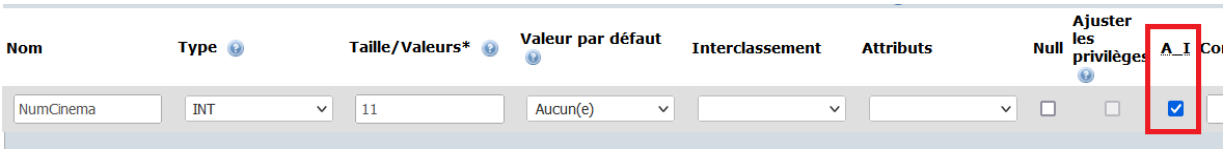


Figure 3.1 : Déclaration d'un attribut de type auto-incrément sous MariaDB

3.2 Les relations de CINEMAS

Les relations sont au nombre de 4. Pour chaque relation nous décrivons l'ensemble des attributs qui la constitue. Nous précisons son nom, le nom de ses attributs ainsi que leur type.

3.2.1 La relation CINEMA

Cette relation permet de décrire les informations générales concernant les cinémas, à savoir leur nom, adresse, et téléphone. On peut avoir par exemple les informations suivantes :

Nom du Cinéma	Adresse du Cinéma	Téléphone du Cinéma
Vox	Grande Rue 25000 Besancon	03 81 82 05 69
Plazza	Rue Des Granges 25000 Besancon	03 81 26 35 98
Plazza	Rue Ronchaux 21000 Dijon	03 80 26 53 25
Royal	Rue Villarceau 21000 Dijon	03 80 23 52 58

Table 3.5 : Relation CINEMA

La relation `CINEMA` proposée est la suivante : `CINEMA(*NumeroCinema, NomCinema, AdresseCinema, VilleCinema, CodePostalCinema, TelephoneCinema)`

Que peut-on dire de cette relation ?

- L'attribut `NumeroCinema` : C'est un nombre (nous supposerons ici un entier compris entre 0 et 999999). Cet attribut constitue la clé primaire de la relation `CINEMA`. Nous utiliserons un auto_incrément.
- Les attributs `AdresseCinema, VilleCinema, CodePostalCinema` : Pour décrire l'adresse d'un cinéma on utilise trois attributs et non pas un seul de façon à obtenir des informations plus détaillées. Le fait de distinguer les trois parties de l'adresse d'un cinéma permet par exemple de faire des recherches telles que : obtenir la liste des cinémas de Besançon, du doubs (Code postal commence par 25), ...
- Les attributs `AdresseCinema et VilleCinema` sont de type Texte.
- L'attribut `CodePostalCinema` est un entier compris entre 1000 et 99999.

Une description complète des attributs de cette relation est la suivante 3.6 :

Nom de l'attribut	**Type**	**Rôle**
NumeroCinema	Entier	Numéro servant de clé primaire pour la table cinéma
NomCinema	Texte	Nom du cinéma
AdresseCinema	Texte	Adresse du cinéma
VilleCinema	Texte	Ville du cinéma
CodePostalCinema	Entier	Code postal du cinéma
TelephoneCinema	Texte	téléphone du cinéma

Table 3.6 : Structure de la relation CINEMA

sous MariaDB on a la définition suivante de la table CINEMA :

3.2.2 La relation `FILM`

Cette relation permet de décrire les informations générales concernant les films, à savoir leur titre, le nom du metteur en scène, leur genre. On propose la relation `FILM` suivante :

`FILM(*Titre, MetteurEnScene, Genre)`

Nous supposons ici implicitement que :

- Deux films ne peuvent pas avoir le même titre (il n'existe pas plusieurs films s'appelant King Kong par exemple).
- Un film possède un seul metteur en scène, et n'appartient qu'à un seul genre.

Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
NUMEROCINEMA	int(11)			Non	*Aucune*	AUTO_INCREMENT
NOMCINEMA	varchar(30)	utf8_general_ci		Oui	*NULL*	
ADRESSECINEMA	varchar(30)	utf8_general_ci		Oui	*NULL*	
CODEPOSTALCINEMA	int(11)			Oui	*NULL*	
VILLECINEMA	varchar(20)	utf8_general_ci		Oui	*NULL*	
TELEPHONECINEMA	varchar(14)	utf8_general_ci		Oui	*NULL*	

Figure 3.2 : Table CINEMA

Nom	Type	Interclassement	Attributs	Null	Défaut
TITRE	varchar(100)	utf8_general_ci		Non	
METTEURENSCENE	varchar(30)	utf8_general_ci		Oui	*NULL*
GENRE	varchar(20)	utf8_general_ci		Oui	*NULL*

Figure 3.3 : Table FILM sous MariaDB

3.2.3 La relation `AFFICHE`

Cette relation permet de décrire les informations générales concernant les affiches des cinémas, à savoir le numéro du cinéma et le titre du film. On propose la relation `AFFICHE` suivante :

```
AFFICHE(*NumeroCinema, *Titre)
```

Nom	Type	Interclassement	Attributs	Null	Défaut
NumeroCinema	int(11)			Non	*Aucune*
Titre	varchar(100)	utf8_general_ci		Non	*Aucune*

Figure 3.4 : Table AFFICHE sous MariaDB

Nous supposons qu'une même film ne peut pas être à l'affiche plusieurs fois dans le même cinéma. Dans le cas contraire il aurait fallu introduire par exemple la notion de numéro de salle dans un cinéma.

3.2.4 La relation `DISTRIBUTION`

Cette relation permet de décrire les informations générales concernant les acteurs jouant dans les films, à savoir le titre des films et le nom des acteurs qui apparaissent dans le film. On propose la relation `DISTRIBUTION` suivante :

```
DISTRIBUTION(*Titre, *NomActeur)
```

Nous ne traitons pas le cas où un acteur joue plusieurs rôles dans un même film.

Nom	Type	Interclassement	Attributs	Null	D
TITRE	varchar(100)	utf8_general_ci		Non	
NOMACTEUR	varchar(30)	utf8_general_ci		Non	

Figure 3.5 : Table DISTRIBUTION

3.3 Les clés candidates et les clés primaires

Nous avons présenté les relations de la base `CINEMAS`, en donnant pour chacune d'elles sa clé primaire. Nous reprenons chacune des relations et explicitons le choix de ces clés primaires.

3.3.1 La relation `CINEMA`

La clé primaire proposée pour cette relation est l'attribut `NumeroCinema`, cependant il existe plusieurs clés candidates pour cette relation.
Si l'on considère qu'il ne peut pas exister dans une même ville deux cinémas portant le même nom, on en déduit que : `NomCinema, VilleCinema` et `NomCinema, CodePostal` sont des clés candidates de la relation `CINEMA`. De plus si l'on considère que deux cinémas ne peuvent pas avoir le même numéro de téléphone, `TelephoneCinema` est aussi une clé candidate.
Les trois clés candidates qui viennent d'être présentées auraient pu être choisies indifféremment comme clé primaire. Alors que dans le cas présent nous avons choisi d'introduire un attribut : `NumeroCinema`, qui ne représente aucune donnée du monde réel, mais qui a l'avantage d'être une clé "simple" plus facile à manipuler dans une base de données. De plus cet attribut a été défini en tant qu'auto-incrément, ses valeurs sont donc gérées par le SGBD.

Exercice 2.10 : La base Cinemas sans l'attribut NumeroCinema
Définir une autre version du modèle relationnel de la base `CINEMAS` sans introduire l'attribut `NumeroCinema` dans la relation `CINEMA`.
Nous proposons de choisir comme clé primaire de la relation `CINEMA` l'ensemble d'attributs : `NomCinema, VilleCinema`.
1 - Redéfinir si besoin les relations `FILM, DISTRIBUTION, et AFFICHE`.
2 - Expliquer l'intérêt de la solution proposée dans le cours par rapport à celle-ci.

3.3.2 La relation `FILM`

La clé primaire proposée pour cette relation est l'attribut `Titre`, puisque comme nous l'avons précisé un titre est spécifique à un film.
Il n'existe pas d'autre clé candidate pour cette relation.

Exercice 2.11 : les clés candidates de `FILM`
1 - Que signifierait au niveau des données le fait que `MetteurEnScene` soit une clé candidate de la relation `FILM`?
2 - Même question avec cette fois comme clé candidate `MetteurEnScene, Genre`.

3.3.3 La relation AFFICHE

La clé primaire proposée pour cette relation est `NumeroCinema, Titre`. Pour la première fois nous avons une clé primaire composée de plusieurs attributs, et non plusieurs clés. Ce qui signifie par exemple que l'on peut avoir les t-uplets suivants dans `AFFICHE` : *(1, Titre1), (1, Titre2), (2,Titre1).*

3.3.4 La relation DISTRIBUTION

La clé primaire proposée pour cette relation est `Titre, NomActeur`.

3.4 Les liens

Nous avons défini les relations et les clés primaires de la base de données `CINEMAS`, il reste maintenant à définir les liens existants entre relations.

3.4.1 Le modèle relationnel de la base CINEMAS

Le modèle relationnel complet proposé est schématisé de la façon suivante :

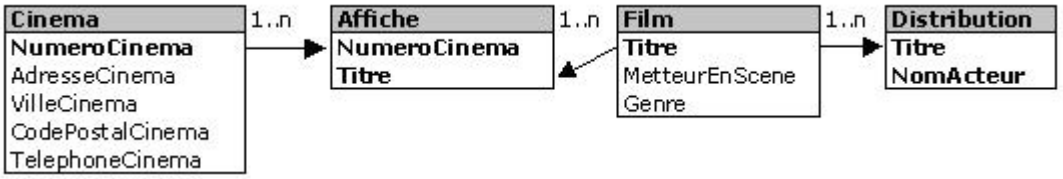


Figure 3.6 : Modèle de CINEMAS

3.4.2 Les clés étrangères

3.4.2.1 Le lien CINEMA - AFFICHE

On a le lien suivant :

```
CINEMA(*NumeroCinema, NomCinema, AdresseCinema,..)[NumeroCinema] (1 .. n) ->
                [NumeroCinema] AFFICHE(*NumeroCinema, *Titre)
```

`NumeroCinema` est la clé primaire de la relation `CINEMA`, et la clé étrangère de la relation `AFFICHE`. Ceci signifie que tout t-uplet introduit dans la relation `AFFICHE` doit faire référence à un cinéma qui existe (présence d'un t-uplet dans la relation `CINEMA` avec la même valeur pour l'attribut NumeroCinema).
De plus le lien est de type `(1..n)`, car un cinéma peut apparaître plusieurs fois dans la relation `AFFICHE`. Ce qui signifie qu'il peut y avoir plusieurs films à l'affiche d'un même cinéma.

3.4.2.2 Le lien FILM - AFFICHE

On a le lien suivant :

```
FILM(*Titre, MetteurEnScene, Genre) [Titre] (1 .. n) -> [Titre]
                AFFICHE(*NumeroCinema,*Titre)
```

Nous pouvons faire le même type de remarques que pour le lien précédent à savoir : Tout t-uplet introduit dans la relation `AFFICHE` doit faire référence à un film qui existe (présence d'un t-uplet dans la relation `FILM` avec la même valeur pour l'attribut `Titre`). De plus le lien est de type

(1..n),car un même titre peut apparaître plusieurs fois dans la relation AFFICHE. Ce qui signifie qu'un film peut être à l'affiche de plusieurs cinémas.

3.4.2.3 Le lien FILM - DISTRIBUTION

On a le lien suivant :

```
FILM(*Titre, MetteurEnScene, Genre) [Titre] (1 .. n) -> [Titre]
                DISTRIBUTION(*Titre, *NomActeur)
```

Ici nous pouvons constater qu'un film peut avoir plusieurs acteurs dans sa distribution, voire aucun. Et un même acteur peut apparaître dans plusieurs films.

Exercice 2.12 : L'intégrité référentielle On considère la base CINEMAS, décrire l'effet des opérations suivantes sur la base, en tenant compte entre autres, des contraintes d'intégrité référentielles.

Ajout de données

1- On veut ajouter dans la relation AFFICHE le t-uplet : (1, Bad Boys).

2 - On veut ajouter dans la relation DISTRIBUTION le t-uplet: (Alien, Weaver).

3 - On veut ajouter dans la relation AFFICHE le t-uplet : (8, Bad Boys).

Mise à jour de données

4 - On veut remplacer le numéro du Cinéma 2 par 1.

5 - On suppose que la règle de mise à jour est CASCADE, on veut remplacer le numéro du cinéma 2 par 8 dans la relation CINEMA.

Suppression de données

6 - On suppose que la règle de suppression est NO ACTION, on veut supprimer le cinéma 2 dans la relation CINEMA.

7 - On suppose que la règle de suppression est CASCADE, on veut supprimer le cinéma 2 dans la relation CINEMA.

8 - On suppose que la règle de suppression est SET DEFAULT, avec la valeur par défaut 0, on veut supprimer le cinéma 2 dans la relation CINEMA.

9 - On suppose que la règle de suppression est NO CHECK, on veut supprimer le cinéma 2 dans la relation CINEMA.

Chapter 4

Les opérateurs relationnels

Maintenant que nous avons vu comment les données sont représentées dans des relations, nous allons voir comment nous pouvons manipuler ces informations. Les données sont représentées sous la forme d'ensembles appelés relations, les opérateurs permettant de manipuler ces données sont des *opérateurs ensemblistes*. A partir d'une ou deux relations, un opérateur définit une nouvelle relation.

Les opérateurs sont les suivants :

- *Opérateurs ensemblistes classiques:* Union, Intersection, Différence, Produit cartésien.
- *Opérateurs relationnels spécifiques :* Sélection, Projection, Jointure.
- *Opérateurs dérivés :* Division, jointure externe, ...

Il existe d'autres opérateurs dérivés que nous ne présenterons pas dans ce cours. Nous présentons dans ce qui suit, chacun de ces opérateurs plus en détail. La description de la construction d'une relation avec les opérateurs relationnels est appelée requête.

Remarque : Il existe différentes notations des opérateurs relationnels. Nous avons choisi dans ce cours une notation simple qui peut être facilement exprimée dans des fichiers texte ; ce qui simplifie la rédaction des requêtes dans le cadre de ce cours.

4.1 Les opérateurs ensemblistes classiques

Les opérateurs ensemblistes, permettent de construire à partir de deux relations, une nouvelle relation. Ils correspondent aux opérateurs mathématiques classiques de la théorie des ensemble. Les opérateurs ensemblistes classiques tels que l'union, l'intersection,la différence et le produit cartésien sont utilisés. Dans tous les trois premiers cas, les deux relations doivent posséder la même en-tête.

4.1.1 L'Union

L'union *permet de créer une relation dans laquelle apparaissent tous les t-uplets apparaissant dans au moins une des deux relations de départ.*

```
T = R UNION S
```

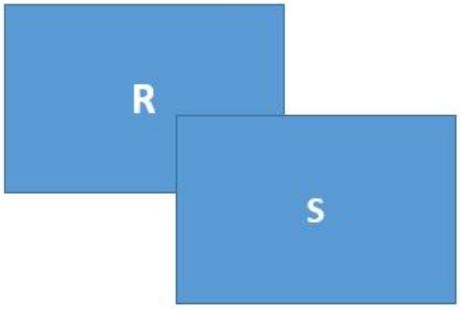


Figure 4.1 : Union

Considérons les deux relations `CINEBESANCON(*NomCinema)` et `CINEDIJON(*NomCinema)` qui donnent les noms des cinémas de Besançon et de Dijon. La requête :

```
CINEDIJONBESANCON = CINEBESANCON UNION CINEDIJON
```

donne une relation appelée `CINEDIJONBESANCON` (cf. Figure 4.2) qui donne la liste des noms de cinémas de Besançon et Dijon.

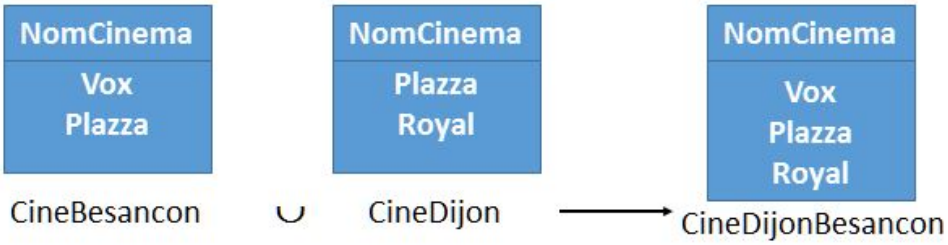


Figure 4.2 : Exemple d'union

Cet exemple permet de rappeler les éléments suivants :

- Les deux relations de départ ont le même en-tête. Elles ont comme attribut `NomCinema`.
- La relation résultat n'a que trois t-uplets, car les deux relations de départ comportaient un t-uplet identique.

Remarque : Attention à la compréhension des énoncés. Nous avons demandé ici la liste des noms de cinémas de Besançon et de Dijon, et nous avons traduit cette demande par une union, c'est-à-dire un **OU**. Le **ET** dans le langage courant signifiant souvent **et/ou**, et donc en logique **OU**. Pour demander la liste des noms de cinémas qui apparaissent à Dijon et à Besançon, on utilisera un **ET**.

4.1.2 L'intersection

L'intersection *permet de créer une relation contenant tous les t-uplets appartenant aux deux relations de départ (cf. figure 4.3).*

```
T = R INTER S
```

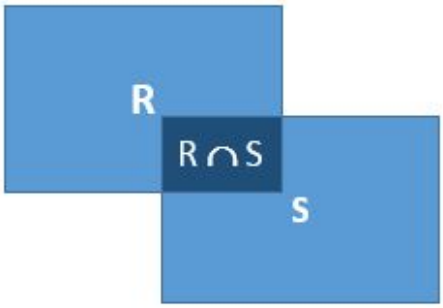


Figure 4.3 : Représentation d'une intersection

Reprenons les deux relations CINEBESANCON et CINEDIJON. La requête :

```
CINEDIJONETBESANCON = CINEBESANCON INTER CINEDIJON
```

donne la liste des noms de cinémas apparaissant à Besançon et à Dijon comme le montre la figure 4.4 .

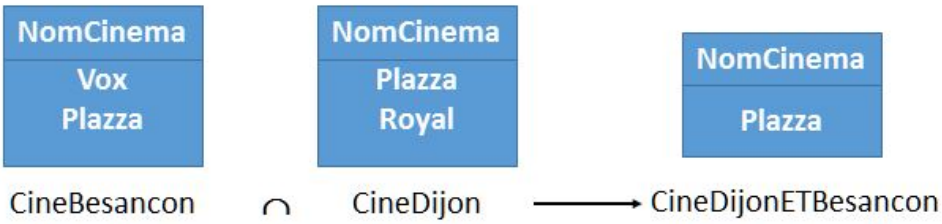


Figure 4.4 : Exemple d'intersection

4.1.3 La différence

*La **différence** permet de créer une relation consistant en l'ensemble de tous les t-uplets apparaissant dans la première relation mais n'apparaissant pas dans la deuxième relation.*

```
T = R - S
```

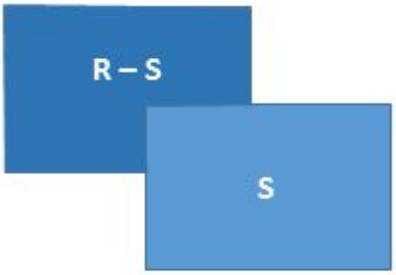


Figure 4.5 : Représentation d'une différence

Reprenons les deux relations `CINEBESANCON` et `CINEDIJON`. La requête :

```
CINEBESANCONPASDIJON = CINEBESANCON - CINEDIJON
```

nous donne la liste des noms de cinémas apparaissant à Besançon et n'apparaissant pas à Dijon.

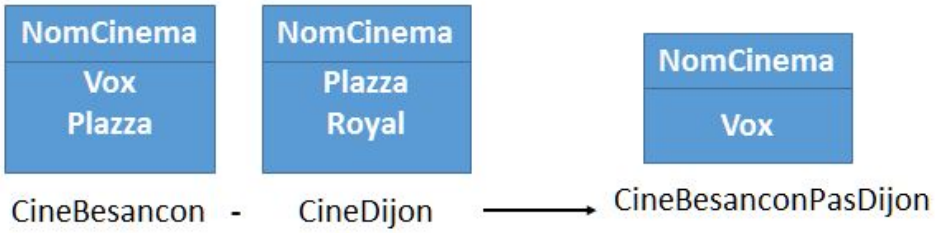


Figure 4.6 : Exemple de différence

4.1.4 Le produit cartésien

*Le **produit cartésien** permet de définir une nouvelle relation consistant en l'ensemble de tous les t-uplets résultant de la combinaison de 2 t-uplets provenant des 2 relations considérées.*
`T = R x S`

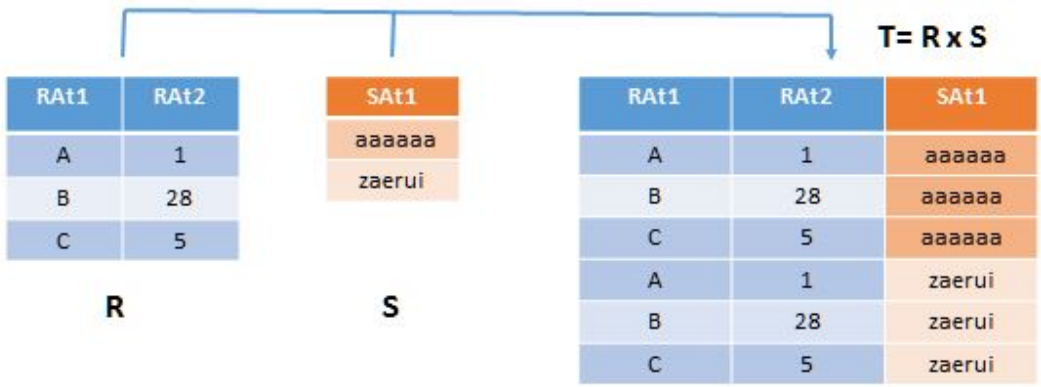


Figure 4.7 : produit Cartésien

Cet opérateur binaire correspond au produit cartésien au sens mathématique. A partir de 2 ensembles on définit un nouvel ensemble par composition d'éléments des 2 ensembles.

Considérons deux relations `CINEMA`, et `FILM` contenant les noms des cinémas d'une part, et et des titres de films d'autre part.
Le produit cartésien de ces deux tables donne la liste des affiches possibles. Comme le montre l'exemple présenté dans la figure 4.8 . `AFFICHE = CINEMA x FILM`

Figure 4.8 : Les affiches possibles

Exercice 2.13 : Les opérateurs ensemblistes : Union, Intersection et Différence
On considère les deux relations suivantes : `FILMA` et `FILMF` (table 4.1) qui représentent respectivement les titres des films américains et les films français.

1. Déterminer les relations obtenues en appliquant les opérateurs : Union, Intersection et Différence aux relations FILMA et FILMF. Préciser ce que représentent les relations obtenues.
2. Ces opérateurs sont-ils symétriques ? C'est-à-dire peut placer dans les opérations dans un ordre quelconque les deux relations : `FILMA op FILMF = FILMF op FILMA` ?

FILMA		**FILMF**
Alerte		Gazon maudit
Congo		Ma saison préférée
BadBoys		Les voleurs
Candyman2		Casper
Casper		Harcelement
Harcelement		

Table 4.1 : Les tables FILMA et FILMB

4.2 Les opérateurs relationnels spécifiques

Les opérateurs relationnels spécifiques sont les opérateurs unaires de *sélection* et de *projection*, et l'opérateur de *jointure*, opérateur essentiel dans la modélisation relationnelle.

4.2.1 La sélection

*La **sélection** appelée aussi restriction, permet d'extraire d'une relation les t-uplets satisfaisant une condition donnée.* `T = S (critère de sélection) R`

Figure 4.9 : Représentation d'une sélection

La relation T obtenue possède le même en-tête que la relation initiale R, mais une cardinalité différente. Seuls les t-uplets satisfaisant le critère de sélection apparaissent dans la relation T.
Le critère de sélection est une condition de la forme : `<Attribut> <Opérateur> <Valeur>`

L'opérateur est un opérateur de comparaison classique : =, >, <, <=, >=, <> (<> signifie différent).

Il est possible de composer des conditions de restriction avec les opérateurs ET et OU.
Sélectionnons par exemple les cinémas de Besançon. Cette requête est décrite par la sélection suivante :

```
CINEMABESANCON = S(VilleCinema='Besançon')CINEMA
```

Le résultat de cette requête est donné par la figure 4.10 .

Numero Cinema	Nom Cinema	Adresse Cinema	Ville Cinema	CodePostal Cinema	Telephone Cinema
1	Vox	Grande Rue	Besançon	25000	03 81 82 05 69
2	Plazza	Rue Des Granges	Besançon	25000	03 81 26 35 98
3	Plazza	Rue Ronchaux	Dijon	21000	03 80 26 53 25
4	Royal	Rue Villarceau	Dijon	21000	03 80 23 52 58
5	Vox	Rue saint Mandé	Paris	75012	01 25 12 56 78

S(VilleCinema='Besançon')CINEMA

Numero Cinema	Nom Cinema	Adresse Cinema	Ville Cinema	CodePostal Cinema	Telephone Cinema
1	Vox	Grande Rue	Besançon	25000	03 81 82 05 69
2	Plazza	Rue Des Granges	Besançon	25000	03 81 26 35 98

Figure 4.10 : Exemple de sélection

Autre exemple, pour sélectionner l'ensemble des cinémas de Besançon et Dijon (cf. figure 4.11), on peut proposer la requête suivante:

```
CINEMABESANCONDIJON = S(VilleCinema="Besancon" OU VilleCinema="Dijon")CINEMA
```

Numero Cinema	Nom Cinema	Adresse Cinema	Ville Cinema	CodePostal Cinema	Telephone Cinema
1	Vox	Grande Rue	Besançon	25000	03 81 82 05 69
2	Plazza	Rue Des Granges	Besançon	25000	03 81 26 35 98
3	Plazza	Rue Ronchaux	Dijon	21000	03 80 26 53 25
4	Royal	Rue Villarceau	Dijon	21000	03 80 23 52 58
5	Vox	Rue saint Mandé	Paris	75012	01 25 12 56 78

S(VilleCinema='Besançon' OU VilleCinema='Dijon))CINEMA

Numero Cinema	Nom Cinema	Adresse Cinema	Ville Cinema	CodePostal Cinema	Telephone Cinema
1	Vox	Grande Rue	Besançon	25000	03 81 82 05 69
2	Plazza	Rue Des Granges	Besançon	25000	03 81 26 35 98
3	Plazza	Rue Ronchaux	Dijon	21000	03 80 26 53 25
4	Royal	Rue Villarceau	Dijon	21000	03 80 23 52 58

Figure 4.11 : Exemple de sélection

Exercice 2.14 : La sélection Décrire les requêtes suivantes en utilisant l'opérateur de sélection, et donner la relation résultat

1. Les acteurs jouant dans Alerte
2. Les titres des films à l'affiche du cinéma N°1
3. Les films mis en scène par Téchiné
4. Les films d'aventure ou mis en scène par Téchiné
5. Les films ayant Balasko comme metteur en scène et qui sont des comédies.

4.2.2 La projection

*La **projection** permet de définir une nouvelle relation consistant en l'ensemble de tous les t-uplets de la relation de départ dans laquelle seuls les attributs de projection sont conservés.* `T =[Attribut1, Attribut2, ...]R`

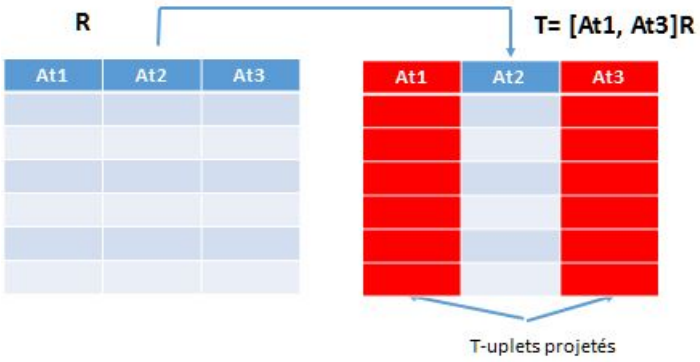


Figure 4.12 : Représentation d'une projection

La relation obtenue n'a plus le même en-tête que la relation R : l'en-tête de T contient seulement les attributs décrits pour la projection. Le degré des deux relations est différent. La cardinalité de T est inférieure ou égale à celle de la relation R. En effet les doublons sont supprimés.

Donnons la liste des noms des cinémas, ainsi que la ville dans laquelle ils sont. Cette requête est décrite par la projection suivante :
`CINEMAVILLE = [NomCinema, VilleCinema]CINEMA`

Le résultat de cette requête est donné par la figure **??** sur l'exemple du cinéma.

Numero Cinema	Nom Cinema	Adresse Cinema	Ville Cinema	CodePostal Cinema	Telephone Cinema
1	Vox	Grande Rue	Besançon	25000	03 81 82 05 69
2	Plazza	Rue Des Granges	Besançon	25000	03 81 26 35 98
3	Plazza	Rue Ronchaux	Dijon	21000	03 80 26 53 25
4	Royal	Rue Villarceau	Dijon	21000	03 80 23 52 58
5	Vox	Rue saint Mandé	Paris	75012	01 25 12 56 78

[NomCinema,VilleCinema]CINEMA

Nom Cinema	Ville Cinema
Vox	Besançon
Plazza	Besançon
Plazza	Dijon
Royal	Dijon
Vox	Paris

Figure 4.13 : Exemple de projection

On peut remarquer qu'il existe deux cinémas appelés *Plazza*. Si l'on veut maintenant simplement le nom des cinémas nous aurons alors la relation suivante en résultat :
`CINEMAS = [NomCinema]CINEMA`
Dans la relation `CINEMAS` les noms *Plazza* et *Vox* n'apparaissent qu'une fois, ce qui découle de façon

Nom Cinema
Plazza
Royal
Vox

Figure 4.14 : Projection sur le nom de cinéma

immédiate de la première propriété des relations que nous avons présenté : Il n'y a pas de duplication de t-uplets.

Exercice 2.15 : La projection Décrire les requêtes suivantes en utilisant l'opérateur de projection, et donner la relation résultat

1. La liste des noms des metteurs en scène.
2. Les titres des films à l'affiche du cinéma N°1.
3. La requête donnée ci-dessous est-elle identique à celle obtenue dans la question 2?
   `TITREFILM = [Titre]AFFICHE`
   `TITREFILMCINE1 = S(NumeroCinema=1)TITREFILM`
4. Le titre des films qui sont à l'affiche dans au moins un cinéma.
5. Le nom des metteurs en scène qui ont réalisé au moins un film d'aventure.

4.2.3 La jointure

La jointure est un opérateur essentiel pour la manipulation des informations dans une base de données.
*Une **jointure** consiste à construire une relation dont les t-uplets sont la concaténation d'un t-uplet d'une relation1, et d'un t-uplet d'une relation2 vérifiants une condition dite de jointure.*

`T = R [condition de jointure] S`

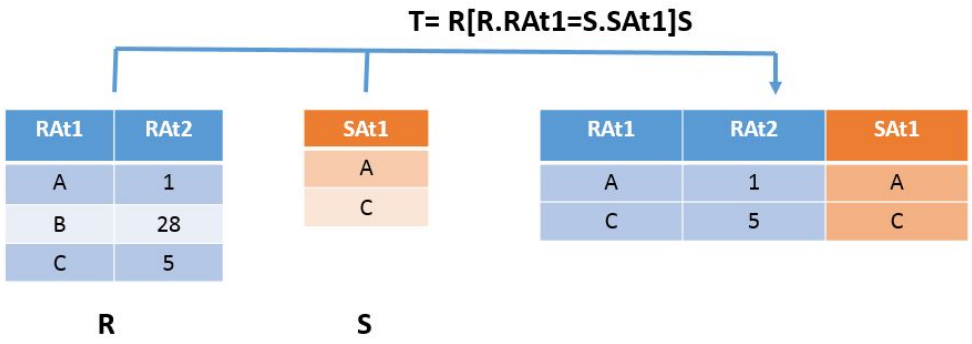


Figure 4.15 : Jointure

Pour réaliser une jointure entre deux relations, celles-ci doivent avoir au moins une colonne commune sémantiquement. Dans l'exemple de la figure 4.15 il s'agit des attributs RAt1 et SAt1.
La jointure peut être vue comme un produit cartésien suivi d'une sélection ayant comme condition, la condition de jointure.

Considérons par exemple le problème suivant : On désire donner pour chaque cinéma, la liste des titres de film qui y sont à l'affiche.

Pour cela nous allons réaliser une jointure des relations `AFFICHE` et `CINEMA` avec comme condition de jointure `AFFICHE.NumeroCinema = CINEMA.NumeroCinema`

`CINEMAFILM = CINEMA [CINEMA.NumeroCinema = AFFICHE.NumeroCinema]AFFICHE`

L'ensemble de t-uplets défini consiste en la concaténation de t-uplets provenant des deux relations de départ (comme lorsque l'on réalise un produit cartésien), mais ici seuls les t-uplets satisfaisant la condition de jointure sont gardés.

Et si maintenant nous réalisons sur la relation obtenue la projection suivante :

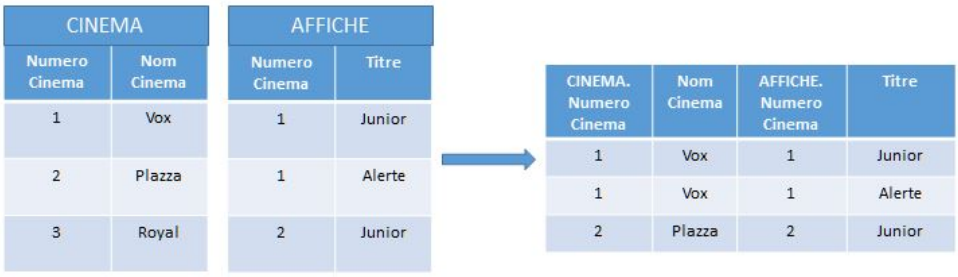


Figure 4.16 : Exemple de jointure

```
TITREAFFICHE = [NomCinema, Titre]CINEMAFILM
```

On obtient la relation 4.17 .

Nom Cinema	Titre
Vox	Junior
Vox	Alerte
Plazza	Junior

Figure 4.17 : Projection sur le nom des cinémas et les titres des films

On distingue trois types de jointure en fonction de la nature de la condition de jointure.

- La jointure naturelle
- L'équi-jointure
- La thêta-Jointure

Nous présentons dans ce qui suit ces trois types de jointures, par la suite nous utiliserons principalement le premier.

4.2.3.1 La jointure naturelle

*La **jointure naturelle** (Fig. 4.18) retourne une relation consistant en l'ensemble de tous les t-uplets qui sont obtenus par concaténation de t-uplets provenant des deux relations de départ, pour lesquels la valeur des attributs spécifiés dans la condition de jointure est la même.*

`T = R [R.Y = S.Y] S` ou `T = R [Y] S`

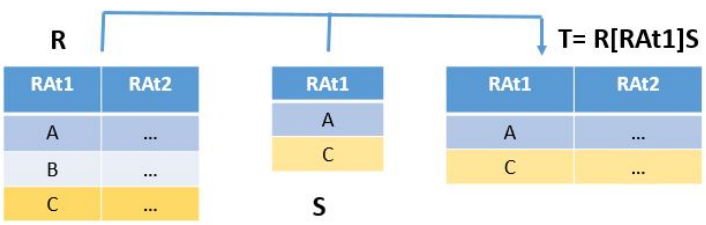


Figure 4.18 : Jointure Naturelle

L'exemple présenté en introduction est une jointure naturelle. Cependant dans la relation résultat l'attribut qui sert à réaliser la jointure n'apparaît qu'une seule fois. Pour l'exemple précédent nous obtenons donc la relation 4.19 :

Numero Cinema	Nom Cinema	Titre
1	Vox	Junior
1	Vox	Alerte
2	Plazza	Junior

Figure 4.19 : Jointure naturelle

Très souvent ce type de jointure fait intervenir une clé étrangère et une clé candidate, mais ceci n'est pas toujours le cas.
Il est possible de représenter cette jointure en ne mettant entre crochet que le nom de l'attribut impliqué car il n'existe ici pas d'ambiguïté. Ainsi la jointure :

```
Jointure = CINEMA [ CINEMA.NumeroCinema= AFFICHE.NumeroCinema ] AFFICHE
```

peut être écrite :

```
Jointure = CINEMA [ NumeroCinema ] AFFICHE
```

4.2.3.2 L'Equi-Jointure

L'équi-jointure *T de deux relations R et S, est une jointure dont la condition de jointure est l'égalité des valeurs de deux attributs i et j appartenant aux relations R et S.*

```
T = R [ R.Ati = S.Atj ] S
```

Considérons les deux relations `CINEMA` et `VILLE` :

NumeroCinema	NomCinema	VilleCinema
1	Vox	Besançon
2	Plazza	Besançon
3	Royal	Dijon

NomVille	CodePostal	NbreCinemas
Besançon	25000	7
Dijon	21000	6

Table 4.2 : Rel. CINEMA et VILLE

L'équi-jointure `CINEMA [CINEMA.VilleCinema = VILLE.NomVille]VILLE` permet de jointurer les deux relations qui n'ont pas d'attributs portant le même nom. Les attributs `VilleCinema` et `NomVille` sont de même nature (même type) et peuvent donc être comparés entre eux. La relation résultant de cette jointure est 4.3 :

NumeroCinema	NomCinema	VilleCinema	NomVille	CodePostal	NbreCinemas
1	Vox	Besançon	Besançon	25000	7
2	Plazza	Besançon	Besançon	25000	7
3	Royal	Dijon	Dijon	21000	6

Table 4.3 : Jointure CINEMA et VILLE

4.2.3.3 La thêta-Jointure

*La **thêta-Jointure** T de deux relations R et S selon une condition de jointure Q est l'ensemble des t-uplets du produit cartésien de R et S satisfaisant la condition Q.*

```
T = R [ Q ] S
```

Q est une condition quelconque permettant de comparer les valeurs de deux attributs provenant des relations R et S. cette condition est différente de l'égalité.
Reprenons les relations `CINEMA` et `VILLE`, la thêta-Jointure `CINEMA [CINEMA.VilleCinema <> VILLE.NomVille] VILLE` donne la relation 4.4 .

NumeroCinema	NomCinema	VilleCinema	NomVille	CodePostal	NbreCinemas
1	Vox	Besançon	Dijon	21000	6
2	Plazza	Besançon	Dijon	21000	6
3	Royal	Dijon	Besançon	25000	7

Table 4.4 : Thêta-Jointure CINEMA et VILLE

On peut utiliser les opérateurs <, <=, >, >=, ou <>.
Cette forme de jointure est la plus générale, ensuite lorsque l'on utilise l'opérateur d'égalité on défini une équi-jointure, et enfin si les attributs spécifiés dans la condition de jointure portent le même nom on parle alors de jointure naturelle.

Exercice 2.16 : La jointure

1. Qu'obtient-on avec la requête suivante :
   ```
   FILMAVENTURE = S(Genre="Aventure")FILM
   DISTAVENTURE = FILMAVENTURE [FILMAVENTURE.titre = DISTRIBUTION.titre]
   DISTRIBUTION
   DISTAVENTURE = [NomActeur] DISTAVENTURE
   ```

   Décrire les requêtes suivantes en utilisant les différents opérateurs présentés :
2. Les titres des films à l'affiche du Vox
3. Le nom des cinémas ayant à l'affiche Junior.

4.3 Les opérateurs relationnels dérivés

Les opérateurs dérivés qui peuvent être définis par composition des autres opérateurs. Nous ne présentons ici que trois opérateurs dérivés : *L'intersection*, la *division*, la *jointure externe.*

4.3.1 L'intersection

L'intersection a déjà été présentée avec les opérateurs ensemblistes, cet opérateur peut être défini à partir de l'opérateur de différence :

```
RELATION1 INTER RELATION2 = RELATION1 - (RELATION1 - RELATION2)
```

ou

```
RELATION1 INTER RELATION2 = RELATION2 - (RELATION2 - RELATION1)
```

A titre d'exemple on peut reprendre l'exercice sur l'intersection et montrer que l'on obtient effectivement la même chose.

Exercice 2.17 : L'intersection On considère les deux relations `FILMA` et `FILMF`, données en 4.1 , qui représentent respectivement les titres des films américains et les films français. En utilisant l'expression de l'intersection présentée ci-dessus recalculer la valeur de `FILMA INTER FILMB`.

4.3.2 La division

*La **division** de la relation `R(A1, ..., Ap, Ap+1, ... An)` par la relation `S(Ap+1,..., An)` retourne une relation `T(A1, A2, ...,Ap)` tels que tous les t-uplets de T concaténés à ceux de S apparaissent dans la relation R.*

```
T = R [R.Ap+1,...,R.An / S.Ap+1,...,S.An ] S
```

Cet opérateur permet de rechercher dans une relation tous les sous t-uplets qui sont complétés (par concaténation) par ceux d'une autre relation.

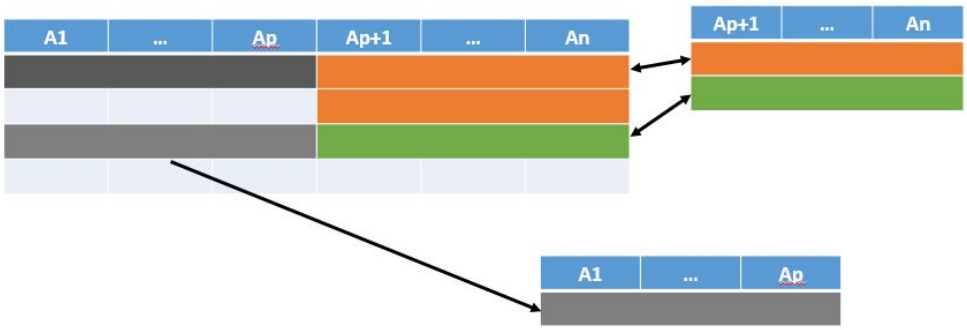


Figure 4.20 : Division

Cet opérateur permet de répondre à des questions de la forme : "*quelque soit x, trouver y*", de façon simple.
La relation S ne doit pas contenir d'autres attributs que ceux intervenant dans la division. D'où la possibilité de simplifier l'écriture de la division : $T = R\ Div\ S$.
Considérons l'expression suivante (cf. fig:4.21)

```
FILMPARTOUT = AFFICHE[ AFFICHE.NumeroCinema / CINE.NumeroCinema] CINE
```

le résultat est la relation `FILMPARTOUT` (cf. 4.21) contenant le titre des films passant dans tous les cinémas.

Exercice 2.18 : La division Décrire les requêtes suivantes en utilisant entre autres l'opérateur de division.

1. Les noms acteurs jouant dans tous les films
2. Les villes dans lesquelles tous les films sont à l'affiche.

4.3.3 La jointure externe

Nous avons présenté dans les opérateurs relationnels spécifiques la jointure qui permet de créer une relation par concaténation des t-uplets appartenant aux deux relations de départ et satisfaisant la condition de jointure.
Dans le cas où pour un t-uplet de la relation1 il n'existe pas dans la relation2 de t-uplets correspondant à la condition de jointure, ce t-uplet n'apparait pas dans la jointure. Une jointure définie

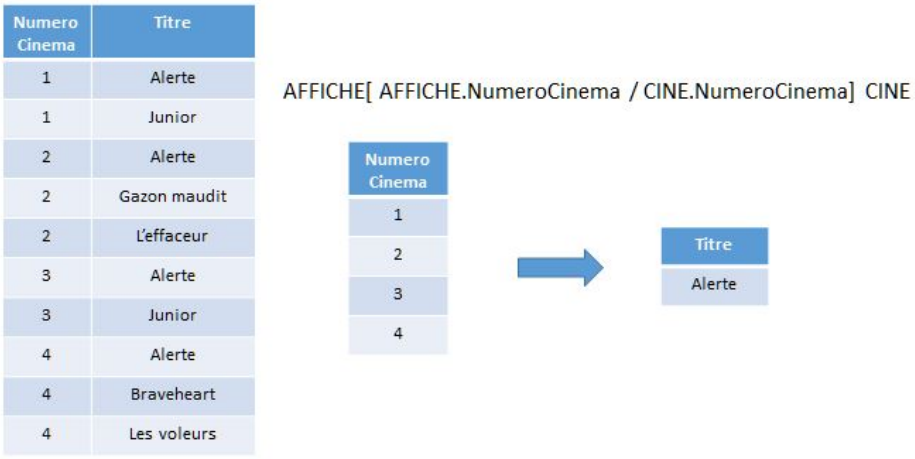


Figure 4.21 : Exemple de division

de cette façon est dite ***jointure interne****.*

Pour pouvoir garder les informations de la relation1, même si dans la relation2 aucun t-uplet correspondant n'existe, on défini la jointure externe.

La ***jointure externe*** *permet de créer une relation T à partir de deux relations R et S, par jointure des deux relations, et ajout de t-uplets des relations R ou S ne participant pas à la jointure, avec la valeur* `NULL` *pour les attributs de l'autre relation.*
Nous notons une jointure externe de la façon suivante :

`T = R+ [condition de jointure] S` ou `T = R [condition de jointure] +S`

Le symbole + est utilisé pour préciser si la jointure est externe à droite ou à gauche.
Pour mieux comprendre la différence entre une jointure interne et une jointure externe, reprenons l'exemple de jointure que nous avions présenté dans les opérateurs spécifiques.

La jointure proposée était :

`CINEFILM = CINEMA [CINEMA.NumeroCinema = AFFICHE.NumeroCinema]AFFICHE`

qui avec les relations ci-dessous donne la relation `CINEFILM` (cf. 4.22).

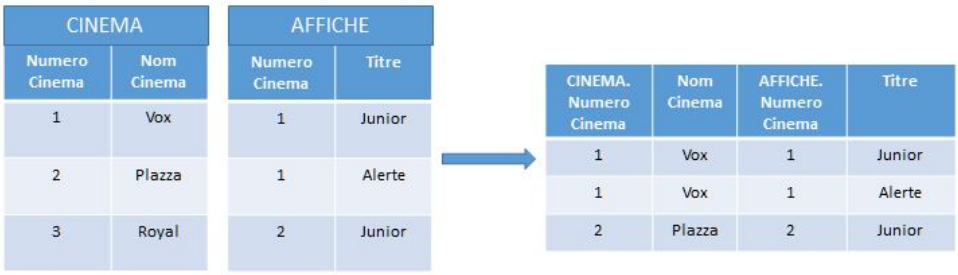


Figure 4.22 : Jointure interne

Le cinéma N°3 n'apparaît pas dans la relation `AFFICHE`, et donc n'apparaît pas dans la relation `CINEFILM`. Maintenant si nous réalisons une jointure externe, présentée comme suit, nous obtenons le résultat cf. fig. 4.23 .

`CINEFILM = CINEMA+ [CINEMA.NumeroCinema = AFFICHE.NumeroCinema] AFFICHE`

CINEMA. Numero Cinema	Nom Cinema	AFFICHE. Numero Cinema	Titre
1	Vox	1	Junior
1	Vox	1	Alerte
2	Plazza	2	Junior
3	Royal	NULL	NULL

Figure 4.23 : Exemple de jointure externe

Exercice 2.19 : La jointure externe Décrire la requête suivante en utilisant la jointure externe.
Donner pour chaque film la liste des cinémas dans lequel il est à l'affiche. Dans le cas où le film n'est à l'affiche nulle part, son titre doit quand même apparaître dans le résultat `(Titre, NomCinema)`.

Part III

Le langage SQL

Nous présentons dans cette partie le langage SQL. Ce langage est composé de cinq grandes parties :

- la définition des éléments d'une base de données (tables, attributs, clés, contraintes...),
- la manipulation des données (insertion, suppression, modification, extraction),
- la gestion des droits d'accès aux données (gestion des droits),
- la gestion des transactions
- le SQL intégré.

Ces cinq grandes parties ne sont pas toujours implémentées dans tous les SGBD, seul le langage de manipulation des données est systématiquement proposé et dans la plupart des cas la partie langage de définition des données. Nous ne présentons dans ce cours que les deux premières parties.

- Le langage de définition des données - DDL
- Les requêtes de sélection simple
- Les sous-requêtes
- Les opérateurs ensemblistes
- Les différentes expressions de la division en SQL
- Les fonctions et expressions dans les requêtes SQL
- Les requêtes de mise à jour des données.
- Le langage de contrôle des données - DCL

Attention je présente dans ce cours le langage standard SQL et les exemples sont écrits sous MariaDB. Il y a dans certains cas des différences entre les deux ou avec le SQl proposé par d'autres SGBD.

Chapter 5

Les Langages des SGBD

Après l'introduction de la modélisation relationnelle, différents langages basés sur ce modèle ont été proposés.
Citons des langages tels que : ALPHA, SQUARE, QBE , SEQUEL, et bien évidemment SQL.

A l'heure actuelle le langage SQL est le standard unanime en ce qui concerne, la manipulation de données dans les SGBD relationnels, cependant beaucoup de SGBD proposent un deuxième langage plus intuitif : QBE.

Nous présentons tout d'abord le langage QBE, et ensuite le langage standard des SGBD relationnels à savoir SQL. Cette présentation de SQL nous servira d'introduction pour ce langage qui sera présenté plus en détail au chapitre 4.

5.1 Le langage QBE

5.1.1 Pourquoi QBE ?

Les langages de manipulation de données "classiques" tels que SQL (que nous verrons dans les prochains chapitres), sont des langages complets qui permettent à un utilisateur de créer une base de données, de l'interroger, de définir des droits, ... Cependant l'utilisateur doit apprendre une syntaxe rigide et surtout non uniforme à travers l'ensemble des SGBD. Aussi dès les années 1970, on a recherché la possibilité de manipuler des données sans avoir à connaître un langage particulier.

Les informaticiens voulaient créer une possibilité de spécifier graphiquement tous les éléments d'une requête, c'est-à-dire choisir la ou les relations nécessaires, les critères de sélection et les champs concernés. Le standard QBE : *Query By Example* était né.

Une première version de QBE fut conçue par IBM en 1978 (Zloff). Son rôle était de faciliter la construction de requêtes relationnelles grâce à un aspect graphique, sans recours à un langage d'interrogation.

QBE, qui n'est pas standardisé, n'est pas implémenté de façon uniforme dans les différents SGBD. Chaque SGBD propose sa propre implémentation de ce langage.

5.1.2 La construction de requêtes en QBE

Une requête est construite en deux étapes :

- La première étape permet de sélectionner dans la base de données la ou les relations qui nous intéressent pour créer la requête.

– La deuxième, permet de définir les critères de la requête : attributs à projeter, sélections, regroupements de données ,...

Nous montrons ci-dessous comment ce langage se présente dans le SGBD MariaDB :
Lors de l'exécution d'une requête QBE, le SGBD interprète la requête et la transforme en code SQL afin de la traiter. Ce qui nous donne le schéma d'exécution suivant :

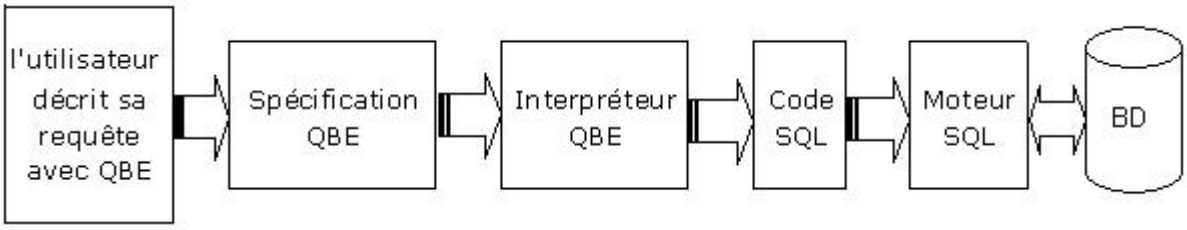


Figure 5.1 : Exécution d'une requête QBE

QBE peut ainsi être vu, plus comme une interface graphique placée au dessus du langage SQL, qu'un langage au même titre que SQL.
La définition d'une requête avec le SGBD MariaDB ressemble à ceci :

5.1.2.1 Première étape

L'utilisateur choisit (cf. figure 5.2) les tables nécessaires à l'écriture de la requête.

Figure 5.2 : Choix des tables

Comme nous l'avons vu ci-dessus lors de la première étape l'utilisateur précise les relations nécessaires à l'expression de sa requête. Nous pouvons remarquer que des requêtes peuvent être utilisées comme source pour une autre requête.

5.1.2.2 Deuxième étape

Cette interface permet à l'utilisateur de définir sa requête. La ligne `afficher` du tableau en bas, permet de savoir quels attributs seront dans la relation résultante (cf. figure 5.3). Ici seul l'attribut `NomCinema` apparaîtra dans le résultat. Une sélection avec le critère `CodePostalCinema = 25000` sera effectuée.
Cette requête donne la requête SQL suivante :

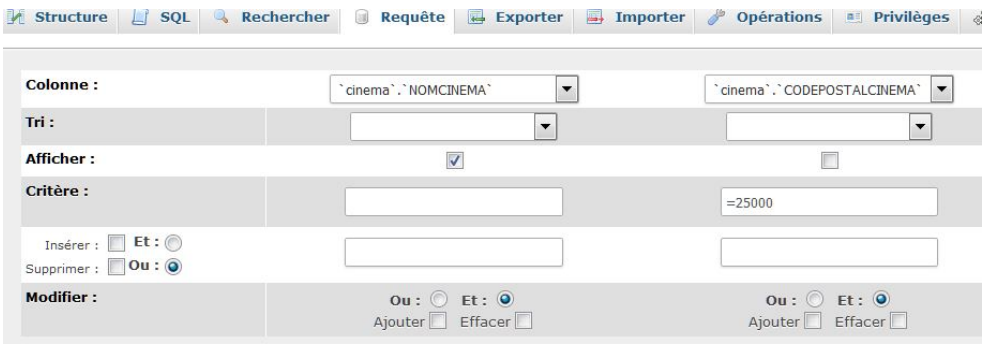


Figure 5.3 : Définition de la requête

```
SELECT cinema.NomCinema
FROM cinema
WHERE cinema.CodePostalCinema=25000
```

Sous MariaB on peut utiliser l'interface Requête pour construire graphiquement des requêtes. Et l'on obtient la requête SQL suivante (5.4).

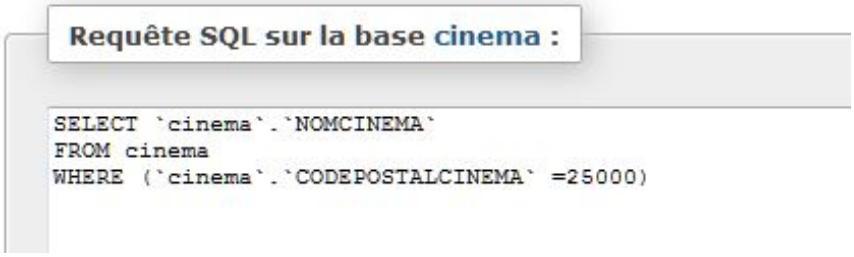


Figure 5.4 : Requête SQL générée par MariaDB

L'exécution de la requête donne le résultat présenté en figure 5.5 .

Figure 5.5 : Exécution de la requête SQL

Ainsi grâce à un langage tel que QBE, l'utilisateur peut construire graphiquement ses requêtes en 2 étapes :

- Choix des relations nécesaires
- Définitions des critères de choix.

5.2 Le langage SQL

5.2.1 SQL : Historique

SQL signifie : `Structured Query Langage`. Ce langage conçu dans les années 1970 par IBM, est le langage relationnel standard géré par presque tous les produits du marché.
Une première version de SQL est intégrée dans un SGBD en 1981, il s'agit du système ORACLE. Dans le même temps IBM propose la version SQL/DS puis DB2.

5.2.1.1 Les versions successives de SQL

SQL1 : En 1986 l'ANSI (American National Standard Institute) fait de SQL une norme : SQL1 niveau1. Ce langage est alors très incomplet, il permet de manipuler les données, mais ne permet de modifier un modèle relationnel. En 1989, la norme est modifiée pour définir SQL89 ou SQL 1 niveau2. Cette norme est ratifiée par l'ANSI et l'ISO (International Standard Organisation).

SQL2 : En 1992 une nouvelle version est proposée, SQL2 ou SQL92, qui est la version utilisée dans la plupart des SGBD actuels. Cette version complète très largement la version précédente : Elle la corrige, y ajoute des éléments mineurs, augmente le nombre de types de données, permet la mise à jour de modèles relationnels, ...

SQL3 : En 1998 une nouvelle version est présentée qui intègre un modèle objet-relationnel.

Dans ce qui suit nous nous intéresserons à SQL2.

5.2.2 Les sous-langages de SQL

SQL est un langage complet qui contient trois sous-langages permettant de réaliser différents types de manipulation sur une base de données.

5.2.2.1 DML : Data Manipulation Language

C'est un langage de manipulation des données, qui permet d'extraire et de mettre à jour les données dans la base. Dans certains références ce langage est séparé en deux parties : le langage de manipulation qui permet de réaliser seulement de la mise-à-jour des données et le langage d'extraction dennées avec la requête `SELECT`.

5.2.2.2 DDL : Data Definition Language

Ce langage permet de créer, modifier ou supprimer les relations de la base de données, de définir les liens entre ces relations, leurs clés primaires.

5.2.2.3 DCL : Data Control Language

Il permet de gérer des protections d'accès aux relations ou tables en environnement multi-utilisateurs. Ce sous-langage ne sera pas présenté dans ce cours d'introduction aux bases de données.

Chapter 6

Le langage de définition des données - DDL

Cette partie de SQL est utilisée pour spécifier le *schéma de la base de données*, à savoir le type des données, les tables qui permettront de les stocker ainsi que les contraintes. Ce langage permet de créer le schéma d'une base de données ou le modifier, mais il ne peut pas être utilisé pour modifier les données de la base de données (DML).

Une base de données est composée de différents types d'objets : les *tables*, les *clés*, les *index*, les *liens*, ... On parle aussi de connexion à une base de données, de session, de catalogue , ... Les ordres de ce langage portent sur les différents objets que sont :

- les tables,
- les index,
- les vues.

6.1 Règles de nommage

Comme tout langage informatique, SQL propose des règles de nommage pour les noms des objets d'une base de données. Un identifiant ne doit pas dépasser 128 caractères, il doit commencer par une lettre, peut être composé des caractères suivants : lettres, chiffres souligné bas. Et il ne doit pas correspondre à un mot réservé du langage.

Avant de manipuler le sous-langage de définition des données, nous allons créer notre base de données CINEMA sous MariaDB. Pour cela lancer phpmyadmin et choisissez l'onglet "Bases de données" et créez votre base de données vide (cf. figure 6.1). Si vous travaillez sur le serveur https://ctu.univ-fcomte.fr/bdd-cours), sélectionnez la base de donnée qui porte votre nom. Les bases de données : BDD_COURS, CHAT et BD_DIU ne sont accessibles qu'en lecture. Vous pouvez écrire des requêtes et tester le résultat. Mais vous ne pouvez pas modifier la structure de ces bases de données ni leur contenu.
La base de données créée est vide. Nous voyons maintenant les requêtes permettant de *'construire'* la base de données.

6.2 Les tables

Il est possible de créer, de modifier ou de supprimer des tables.
La création des tables suppose la définition des attributs de la table, et l'expression des contraintes

Figure 6.1 : Création d'une BD sous MariaDB

d'intégrité concernant la table. Lorsqu'une table est créée elle contient au moins un attribut et aucune ou plusieurs contraintes de table.

6.2.1 La définition des attributs de la table

```
CREATE TABLE} NOM_TABLE
(nomAttribut1 type1
nomAttribut2 type2
...)
```

Cette commande permet de définir une table appelée `NOM_TABLE`, avec les attributs `nomAttribut1` de type `type1`, `nomAttribut2` ...

La requête MariaDB suivante créé la table `CINEMA`, avec les attributs `NumeroCinema`, `NomCinema`, `AdresseCinema`, `VilleCinema`, `CodePostalCinema`, et `TelephoneCinema` :

```
CREATE TABLE ‘cinema‘ (
  ‘NUMEROCINEMA‘ int(11),
  ‘NOMCINEMA‘ varchar(30),
  ‘ADRESSECINEMA‘ varchar(30),
  ‘CODEPOSTALCINEMA‘ int(11),
  ‘VILLECINEMA‘ varchar(20),
  ‘TELEPHONECINEMA‘ varchar(14)
)
```

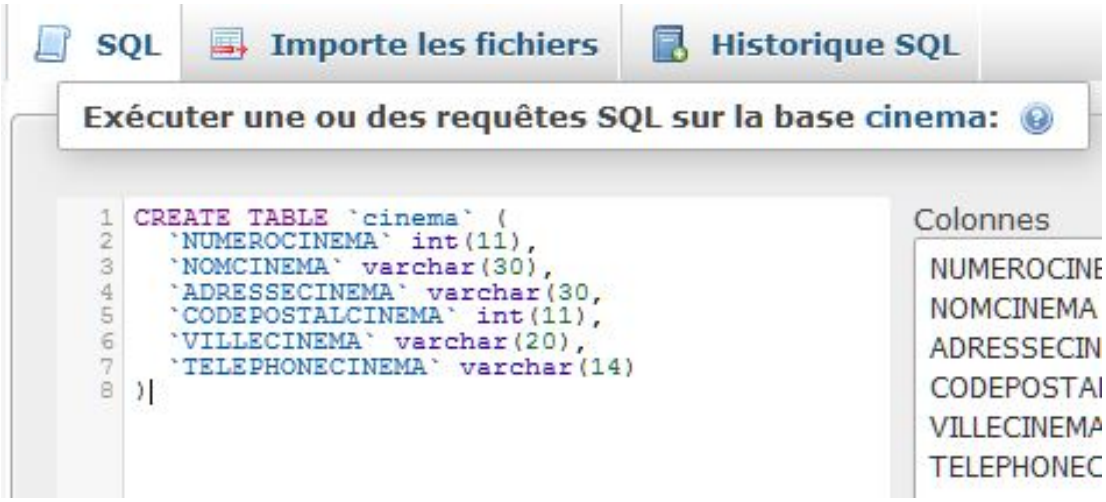


Figure 6.2 : Création de la table cinema

Exercice 4.1 : Créez sous MariaDB une première version des tables de la base de données CINEMAS, à savoir :
- CINEMA
- FILM
- AFFICHE
- DISTRIBUTION

6.2.2 L'expression des contraintes d'intégrité

Nous avons présenté dans le chapitre 2, les contraintes d'intégrité qui sont des règles prises en charge par le SGBD, pour préserver automatiquement la cohérence des données lors de la saisie, la modification, la suppression de données, ... Ces règles permettent d'alléger le développement des applications de base de données. Le programmeur n'a pas à se soucier de ces règles et lors de l'exécution d'une requête, le respect de ces règles est contrôlé par le serveur de base de données. Le langage de définition des données permet bien évidemment d'exprimer ces règles, lors de la création d'une table.
Nous distinguons deux grands types de clauses permettant la gestion de ces règles : *Les règles portant sur un attribut* et *celles portant sur une table.*

La syntaxe générale pour la création d'une contrainte peut prendre deux formes : la contrainte peut être déclarée en même temps que l'attribut. On dit que la contrainte est *en ligne*, ou alors une fois l'attribut déclaré la contrainte peut être déclarée et nommée avec la syntaxe suivante :

```
CONSTRAINT  Nom_Contrainte
expression de la contrainte
```

Le nom d'une contrainte est optionnel, mais il est conseillé de nommer les contraintes ; ceci améliore la lisibilité de la base de données. Si l'utilisateur ne nomme pas la contrainte un nom généré automatiquement est proposé par le SGBD.

6.2.2.1 Contrainte portant sur un attribut

Une telle clause est associée à la définition d'un attribut. Il est possible d'en utiliser zéro ou plusieurs. Ce type de contrainte permet de restreindre l'ensemble des valeurs de l'attribut sans aucune référence aux autres attributs.
Les contraintes portant sur un attribut sont :

```
NOT NULL | UNIQUE | DEFAULT | CHECK
```

- La contrainte `NOT NULL` permet de forcer la saisie d'une valeur pour l'attribut.
- La contrainte `UNIQUE` vérifie que la valeur donnée est différente de toutes celles déjà présentes dans la table. Les valeurs `NULL` étant une exception sauf si cette valeur est interdite pour l'attribut en question.
- La contrainte `DEFAULT` donne une valeur par défaut à l'attribut. Si l'utilisateur ne donne pas de valeur, la valeur par défaut sera automatiquement introduite.
- La contrainte `CHECK` vérifie que la valeur satisfait la condition spécifiée. Cette contrainte n'est vérifiée que si la valeur saisie est différente de `NULL`.

Si l'on reprend la définition de la table `CINEMA`, on peut avoir par exemple la définition des contraintes suivantes qui spécifient qu'un nom de cinéma ne peut pas être `NULL`, que le code postal d'une ville est compris entre 1000 et 99999, et que par défaut la ville du cinéma est *PARIS* :

```
CREATE TABLE IF NOT EXISTS ‘cinema1‘ (
  ‘NUMEROCINEMA‘ int(11) NOT NULL,
  ‘NOMCINEMA‘ varchar(30),
  ‘ADRESSECINEMA‘ varchar(30) DEFAULT NULL,
  ‘CODEPOSTALCINEMA‘ int(11) CHECK (CODEPOSTALCINEMA BETWEEN 1000 AND 99999),
  ‘VILLECINEMA‘ varchar(20) DEFAULT ’PARIS’,
  ‘TELEPHONECINEMA‘ varchar(14)
);
```

6.2.2.2 Les contraintes portant sur des tables

Ces clauses concernent un ensemble d'attributs de la table. Elles permettent de définir la clé primaire d'une table ainsi que les liens avec d'autres tables. La syntaxe SQL est :

```
[ CONSTRAINT  Nom_Contrainte ]
   PRIMARY KEY (attribut1 [, attribut2] ...)
   FOREIGN KEY (attribut1 [, attribut2] ...)
     REFERENCES [schema.]nomTableSource (attribut1 [, attribut2] ...)
      [MATCH {FULL | PARTIAL | SIMPLE } ]
      [ON UPDATE {NO ACTION | RESTRICT | CASCADE | SET NULL | SET DEFAULT}]
      [ON DELETE {NO ACTION | RESTRICT | CASCADE | SET NULL | SET DEFAULT}]
```

Convention de nommage : les contraintes de clé primaire sont souvent nommées *PK_nomTable*, et les contraintes de clé étrangères : *FK_nomTableSource_nomTableCible*. En respectant cette convention la contrainte définissant la clé primaire de la table `FILM` se nomme *PK_FILM* et la contrainte qui définit le lien entre `CINEMA` et `AFFICHE` : *FK_CINEMA_AFFICHE*.

- **La contrainte** `PRIMARY KEY` permet de déclarer qu'un ensemble d'attributs est la **clé primaire** de la table. Il n'y a qu'une clause de ce genre par table. Automatiquement les attributs ne peuvent plus prendre la valeur `NULL`, et chaque valeur de la clé doit être unique.
- **La contrainte** `FOREIGN KEY` permet de déclarer qu'un ensemble d'attributs est une **clé étrangère**. La clause `FOREIGN KEY (attribut1, ...) REFERENCE TABLE (Attribut1', ...)` signifie que les attributs (`attribut1, ...`) de la table en cours référencent la clé primaire `(attribut1', ...)` de la table `TABLE` référencée. La clause `MATCH` permet de définir le type de correspondance existant entre les valeurs dans la table cible et celles de la table source. Cette clause est prise en compte au moment de l'ajout de données dans la table cible, elle ne présente d'intérêt que si l'intégrité référentielle porte sur plusieurs attributs.
  Il y a trois types de correspondance : `FULL, PARTIAL, SIMPLE`.
  - La correspondance `FULL` ne permet pas à une partie de la clé étrangère de prendre la valeur `NULL` à moins que tous les attributs formant la clé étrangère ne soient `NULL`.
  - La correspondance `PARTIAL` s'applique à tous les attributs formant la clé étrangère qui ont une valeur non `NULL`.
  - La correspondance `SIMPLE` vérifie, si tous les attributs ont une valeur différente de `NULL`, que ces valeurs sont bien issues de la table source. Par contre si certains attributs de clé étrangère ont pour valeur `NULL`, les valeurs des autres attributs faisant partie de la clé étrangère ne sont pas vérifiées. Par défaut c'est cette correspondance qui s'applique.

Les clauses `ON DELETE` **et** `ON UPDATE` décrivent les modifications automatiques à faire sur les clés étrangères en cas de changement de valeur de la clé primaire associée. On trouve les options suivantes :

- `NO ACTION` : la suppression ou la modification échouent, si la valeur apparaît dans la table en clé étrangère.
- `RESTRICT` : la suppression ou la modification échouent, si la valeur apparaît dans la table en clé étrangère.
- `CASCADE` : les t-uplets correspondant sont effacés ou mis à jour en cascade.
- `SET NULL` : place la valeur Null, en cas d'effacement d'un t-uplet dans la table référencée.
- `SET DEFAULT` : place la valeur par défaut, en cas d'effacement d'un t-uplet dans la table référencée.

Exemple : Pour mieux comprendre la clause `MATCH`. Considérons les relations `AFFICHE` et SEANCE suivantes :

`AFFICHE(*NumeroCinema, * Titre)`
`SEANCE(*NumeroSeance, HoraireSeance, NumeroCinema, Titre)`
avec un lien entre `AFFICHE` et `SEANCE`. Nous avons dans `AFFICHE` *((2,Alerte))*

Nous insérons dans `SEANCE` les t-uplets suivants :

- T1 : (1, 14:00:00, 2, Alerte)
- T2 : (2, 16:00:00, NULL, NULL)
- T3 : (3, 14:00:00, NULL, Alerte)
- T4 : (4, 16:00:00, 3, NULL)
- T5 : (5, 14:00:00, 3, Alerte)

Avec un `MATCH SIMPLE` :

- T1 : pas de problème
- T2 : Il y a un NULL –> la contrainte ne s'applique pas –> pas de problème
- T3 : Il y a un NULL –> la contrainte ne s'applique pas –> pas de problème
- T4 : Il y a un NULL –> la contrainte ne s'applique pas –> pas de problème
- T5 : Pb car pas de t-uplet (3, Alerte) dans AFFICHE

Avec un `MATCH FULL` :

- T1 : pas de problème
- T2 : Il y a que des NULL –> pas de problème
- T3 : Une partie clé étrangère (NumeroCinema) est à NULL –> Pb
- T4 : il y a un NULL –> Pb
- T5 : Pb car pas de t-uplet (3, Alerte) dans AFFICHE

Avec un `MATCH PARTIAL` :

- T1 : pas de problème
- T2 : Il y a que des NULL –> pas de problème
- T3 : Il y a un NULL –> la contrainte ne s'applique que sur les attributs renseignés –> pas de problème
- T4 : Il y a un NULL –> la contrainte ne s'applique que sur les attributs renseignés –> Pb car pas de t-uplet de la forme (3, ?) dans AFFICHE

– T5 : Pb car pas de t-uplet (3, Alerte) dans AFFICHE

Remarque : La différence entre les options `NO ACTION` et `RESTRICT` provient du moment où le blocage de l'action demandée sera effectué. Dans le cas de l'option `RESTRICT` la commande SQL qui a déclenchée le problème échoue (`UPDATE` ou `DELETE`), dans le cas de l'option `NO ACTION` le blocage intervient en fin de transaction.

MariaDB : Par défaut, MariaDB met les options suivantes `ON DELETE RESTRICT` et `ON UPDATE RESTRICT`. Il est aussi possible d'utiliser les options `ON DELETE CASCADE`. Par contre MariaDB ne supporte pas les options `SET DEFAULT`. La mise en place de telles politiques ne peut se faire sous MariaDB qu'avec des déclencheurs ou des procédures stockées, voire dans le code de l'application.

La requête suivante permet de créer la table `DIRECTEUR` qui a pour clé primaire `NumeroDirecteur`, et comme clé étrangère `NumeroCinema` vers la table `CINEMA`. La mise à jour lors de suppression se fait en cascade à partir de la table `CINEMA`. L'attribut `NomDirecteur` n'a pas le droit de prendre la valeur `NULL`.

```
CREATE TABLE DIRECTEUR
(NomDirecteur CHAR(30) NOT NULL,
NumeroDirecteur CHAR(3),
NumeroCinema INTEGER,
CONSTRAINT PK_Directeur PRIMARY KEY (NumeroDirecteur),
CONSTRAINT FK_Cinema_Directeur FOREIGN KEY (NumeroCinema)
   REFERENCES CINEMA (NumeroCinema) ON DELETE CASCADE);
```

6.2.3 Modification d'une table

La modification d'une table peut être réalisée par l'ajout d'un attribut, la modification d'un attribut existant, ou la suppression d'un attribut. La syntaxe simplifiée d'une requête `ALTER TABLE` est la suivante :

```
ALTER TABLE nom_Table
  [ ADD ( col1 type1 [, col2 type2 [, ...] ] ) ]
  [ DROP col1 [, col2] , ... ]
  [ ALTER (attrib1 SET DEFAULT val_attrib1 ]
  [ ALTER (attrib1 DROP DEFAULT ]
  [ ADD CONSTRAINT définition_Contrainte ]
  [ DROP CONSTRAINT nom_Contrainte ]
```

Cette requête permet d'ajouter les attributs décrits dans la clause `ADD` en fin de table, de supprimer des attributs (`DROP`), de définir (`ALTER SET DEFAULT`) ou de supprimer (`ALTER DROP DEFAULT`) la valeur par défaut d'un attribut, d'ajouter (`ADD CONSTRAINT`) ou de supprimer (`DROP CONSTRAINT`) des contraintes d'intégrité. Par contre elle ne permet pas de modifier le nom ou le type d'un attribut, ni d'ajouter ou de supprimer une contrainte d'attribut `NOT NULL`.

MariaDB : Sous MariaDB on retrouve l'instruction ALTER TABLE avec les options suivantes :

– `ADD` : Ajout de colonnes
– `RENAME COLUMN` : Renommage de colonnes
– `MODIFY` : Changement de la taille d'un attribut, ou modification d'une contrainte
– `DROP COLUMN` : Suppression d'attributs
– `ADD CONSTRAINT` : Ajout de contraintes

– `DROP CONSTRAINT` : Suppression de contraintes

Pour obtenir la spécification complète de ce type de requête est présentée dans le manuel de référence de mysql (voir doucument mis à disposition sur moodle).
On trouve aussi sous Oracle des options permettant de désactiver ou de réactiver des contraintes (`DISABLE CONSTRAINT` ou `ENABLE CONSTRAINT`).

Exercice 4.2 : Créez sous MariaDB les requêtes de MODIFICATION des tables de la base de données CINEMAS de façon à définir les contraintes suivantes :

- Clé primaire de CINEMA : NumeroCinema
- Clé primaire de AFFICHE : NumeroCinema, Titre
- Clé primaire de FILM : Titre
- Clé primaire de DISTRIBUTION : Titre, NomActeur
- Clé étrangère : NumeroCinema dans AFFICHE (Table source CINEMA). Politique en suppression CASCADE
- Clé étrangère : Titre dans AFFICHE (Table source FILM). Politique en suppression CASCADE
- Clé étrangère : Titre dans DISTRIBUTION (Table source FILM). Politique en suppression CASCADE

6.2.4 Suppression d'une table

La commande `DROP TABLE` permet de supprimer une table. On peut lui associer une option `RESTRICT` ou `CASCADE`. Avec l'option `RESTRICT` la table n'est supprimée que si elle n'est pas référencée par une vue, un déclencheur, ... Dans le cas de l'option `CASCADE` les différents éléments utilisant cette table sont aussi détruits.
La requête suivante permet de détruire la table `DIRECTEUR`.

```
DROP TABLE DIRECTEUR
```

6.3 Les index

Les index ne font pas partie de SQL au même titre que les tables, mais ils sont primordiaux dans l'optimisation de la recherche des informations.

6.3.1 Qu'est ce qu'un index ?

Considérons la requête suivante qui donne les informations concernant les cinémas ayant pour nom : *Plazza*.

```
SELECT *
FROM CINEMA
WHERE NomCinema = "Plazza"
```

Pour retrouver les t-uplets pour lesquels NomCinema vaut *Plazza*, il faut consulter tous les t-uplets de la table `CINEMA`. Un tel traitement risque de conduire à des temps d'accès très longs dès que l'on aura beaucoup d'informations dans cette table.
Une solution pour remédier à ce problème d'accès aux informations est la création d'un **index**.
L'index est une notion que nous ne développerons pas dans le cours. Il permet d'indexer les données d'une table pour optimiser l'accès aux données. Par défaut le SGBD créé pour chaque table un index correspondant à la clé primaire. Mais pour les clés étrangères il n'y a aucune

obligation.
C'est ainsi une construction des données qui permet d'accéder rapidement et directement à certains t-uplets dans la table. Il est décrit au moyen de un ou plusieurs attributs de la table. Un index peut être vu comme un ensemble d'informations organisées physiquement pour permettre la recherche de données par rapport à certains critères (attributs). L'index doit connaître la valeur des attributs qui servent à la recherche et la référence du ou des t-uplets correspondant dans la table. Un index peut être créé juste après la création d'une table ou lorsque celle-ci contient déjà des t-uplets. Il est mis à jour automatiquement à chaque mise à jour de la table. D'un SGBD à l'autre la gestion des index peut être différente. Mais la pluaprt des SGBD s'appuient sur des structures classiques pour organiser les index. Les structures les plus courantes sont :

- l'index séquentiel
- l'index organisé en arbre
- l'index en cluster
- l'index en hachage
- l'index bitmap
- l'index textuel

Dans beaucoup de SGBD la déclaration d'une clé primaire implique la création d'un index sur cette même clé.
Les requêtes SQL sont transparentes au fait qu'il existe ou non des index. C'est l'optimiseur de SQL qui s'intéresse à leur existence.

6.3.2 Création d'un index

La syntaxe de la création des index est la suivante :

```
CREATE [UNIQUE] INDEX NomINDEX
ON NomTABLE (nomAttribut1, nomAttribut2 [ASC | DESC],...)
```

Cette commande permet de définir un index sur la table `NomTABLE` appelé `NomINDEX`, avec les attributs `nomAttribut1, nomAttribut2,...` , par ordre croissant ou décroissant.
La requête suivante créé un index sur la table `CINEMA` par rapport au nom de cinéma :

```
CREATE INDEX NomCine
ON CINEMA (NomCinema)
```

L'option `UNIQUE` précise si les valeurs des attributs indexés sont uniques. Dans ce cas l'index permet de gérer l'unicité de clé (contrainte d'intégrité vue dans le chapitre 2).
Oracle : Plusieurs structures d'index sont proposées sous Oracle :

- l'index organisé en arbre (B-tree)
- l'index en cluster (reverse key)
- l'index bitmap
- des index basés sur les fonctions

Sous MariaDb, la plupart des index sont des B-tree, cependant les index portant sur des données spatiales utilisent des R-tree. On dispose aussi d'index fulltext pour la recherche plein texte.

6.3.3 Suppression d'un index

La syntaxe de la suppression d'un index est la suivante :

```
DROP INDEX NomINDEX
```

Cette requête a pour effet de supprimer l'index appelé `NomINDEX`.

6.4 Les vues

6.4.1 Qu'est ce qu'une vue?

Nous avons présenté au chapitre 1, l'architecture ANSI/SPARC des SGBD. Dans cette architecture le niveau externe offre à chaque groupe d'utilisateurs, une vue externe des données. Dans le langage SQL, la vue est introduite pour permettre cette description.

Une **vue** est une table dont les données ne sont pas physiquement stockées, mais qui se réfère à d'autres tables réelles. ***Seule la définition de la vue est enregistrée dans la base, mais pas les données***. On parle aussi de table virtuelle.
La notion de vue permet :

- de dissocier la façon dont les utilisateurs voient les données et les tables réelles. On sépare ainsi les aspects externes (vues externes) et les aspect conceptuels (le modèle),
- de favoriser l'indépendance des données et des programmes,
- de restreindre les droits d'accès à certaines parties de tables,
- de permettre de protéger l'accès aux tables en fonction des utilisateurs.

6.4.2 Création d'une vue

La création d'une vue se fait à l'aide d'une requête SELECT appelée *requête de définition*, qui sélectionne dans la base les informations pour la vue.

```
CREATE [OR REPLACE] VIEW NomVUE
AS Requête_Select
[WITH [ CASCADED | LOCAL | CHECK OPTION]]
```

Cette commande permet de créer une vue appelée `NomVUE`, à partir d'une requête SELECT ou de remplacer une vue existante si la clause `OR REPLACE` est utilisée. Elle peut avoir différentes options concernant la gestion des mises à jour de données :

- `CASCADED`: il y a vérification de l'intégrité de la vue et de chaque vue dépendante,
- `LOCAL` : cette option lance la vérification de l'intégrité de la vue,
- `CHECK OPTION` : cette option fonctionne avec les vues pouvant être mises à jour. Toutes les commandes de mise à jour de données sur la vue seront vérifiées pour s'assurer que les données obtenues satisfont les conditions définissant la vue. Si ce n'est pas le cas, la mise à jour est refusée.

La requête suivante créé une vue appelée `CineBesac`, qui donne les informations concernant les cinémas de Besançon :

```
CREATE VIEW CineBesac
AS SELECT *
FROM CINEMA
WHERE VilleCinema = 'Besançon'
```

Oracle : Sous Oracle on trouve la commande suivante pour créer une vue :

```
CREATE [OR REPLACE] [[NO] FORCE] VIEW NomVUE
[ alias [ContrainteInLine] | [ContrainteOutLine] ]
AS SELECT ...
[WITH [ READ ONLY | CHECK OPTION [CONSTRAINT NomContrainte]]]
```

- `OR REPLACE` : remplace la vue si elle exsite déjà,
- `FORCE` : ne vérifie pas l'existence des tables ou les droits du créateur de la vue sur ces tables,
- `NOFORCE` : vérifie l'existence des tables et les droits (par défaut),

- `alias` : donne le nom des attributs de la vue, par défaut le nom correspond à celui de l'attribut dans la requête de définition,
- `ContrainteInLine` : décrit une contrainte en ligne,
- `ContrainteOutLine` : décrit une contrainte nommée,
- `READ ONLY` : déclare que la vue est non modifiable,
- `CHECK OPTION` : décrit un prédicat qui devra être satisfait lors de toute mise à jour de la vue.

6.4.3 Les contraintes des vues

Une vue ne peut pas être considérée de la même façon qu'une table. Ainsi on ne peut pas créer d'index sur une vue et la mise à jour à partir d'une vue ne peut être réalisée que sous certaines conditions.
Toute modification réalisée sur une ou plusieurs tables liées à une vue se répercute sur celle-ci. Le contraire n'est pas tout à fait vrai. Les opérations de mise à jour telles l'ajout ou la modification sont possibles sur une vue mais elles doivent respecter quelques règles :

- le `SELECT` définissant la vue ne fait référence qu'à une seule table,
- le `SELECT` ne comporte pas de `DISTINCT`,
- les colonnes résultats du `SELECT` correspondent à des attributs de la table, et non à des résultats d'expression,
- il n'y a ni clause `GROUP BY` ni clause `HAVING`,
- la clause `WHERE` ne doit pas contenir de sous-requête synchronisée.

Pour qu'une vue soit modifiable chaque élément de la vue (t-uplet, attribut) doit pouvoir être associé par le SGBD à un t-uplet et un attribut dans la table source. Dans le cas où la vue ne respecte pas une des ces conditions, elle est en lecture seule. Cette définition est standard mais d'un SGBD à un autre elle peut varier.

6.4.4 Suppression d'une vue

Une vue peut être supprimée en utilisant la commande DROP.

```
DROP VIEW [IF EXISTS] nom_vue1 [, nom_vue2] ...
[RESTRICT | CASCADE]
```

La commande DROP VIEW permet de supprimer une ou plusieurs vues à condition d'avoir les privilèges DROP sur chaque vue. La clause IF EXIST permet d'éviter un message d'erreur si la vue n'existe pas.

Chapter 7

Le langage de manipulation des données - DML

Nous présentons dans ce chapitre le langage de manipulation de SQL. Pour cela nous verrons les différentes leçons :

- Les requêtes de sélection simples
- Les sous-requêtes
- Les opérateurs ensemblistes
- Les différentes expressions de la division en SQL
- Les fonctions et expressions dans les requêtes SQL
- Les requêtes de mise à jour des données.

Le langage de manipulation de données (Data Manipulation Language - DML) est l'ensemble des commandes concernant la manipulation des données dans une base de données. Le DML permet l'ajout, la suppression et la modification de t-uplets, et surtout la consultation du contenu des tables.

7.1 Les requêtes de sélection simples

7.1.1 La requête de sélection de données

L'obtention de données se fait exclusivement avec l'ordre `SELECT`. Cet ordre possède 6 clauses différentes dont seules les deux premières sont obligatoires.
Syntaxe de l'ordre SELECT

```
SELECT Liste-Attributs
FROM Liste-tables
WHERE critère de sélection
GROUP BY liste-attributs-groupe
HAVING critère-groupe
ORDER BY Liste-attributs
```

La syntaxe minimale de cet ordre est :

```
SELECT Liste-Attributs
FROM Liste-tables
```

Mais il peut être enrichi de nombreuses clauses qui permettent d'exprimer des sélections, des jointures entre autres. Les paramètres des différentes clauses de l'ordre SELECT sont les suivants :

- *liste-attributs* : nom des attributs à extraire

- *liste-tables* : noms des tables utilisées
- *critère de sélection* : expression qui détermine les t-uplets qui sont sélectionnés
- *liste-attributs-groupe* : liste des attributs qui définissent le regroupement
- *critère-groupe* : expression qui détermine les groupes qui sont sélectionnés
- *liste-attributs* : liste d'attributs permettant d'ordonner la présentation des résultats.

Nous reviendrons plus en détail dans ce qui suit sur chacune des clauses de l'ordre SELECT, et donnerons des exemples de ces différents paramètres. L'évaluation d'une telle requête est réalisé dans l'ordre suivant :

- clause WHERE
- clause GROUP BY
- fonctions d'agrégation
- clause HAVING
- clause SELECT
- clause ORDER

Attention la clause SELECT ne correspond pas à l'opérateur relationnel de sélection que nous avons présenté au chapitre précédent.

7.1.2 La clause SELECT

La clause `SELECT` permet d'indiquer quels attributs ou quelles expressions doivent être retournés par la requête. Il s'agit en fait de l'expression SQL de l'opérateur de projection que nous avons présenté avec les opérateurs relationnels.
Syntaxe de la clause SELECT

```
SELECT [DISTINCT | ALL | DISTINCTROW] expr1 [[AS] nom1], expr2 [[AS] nom2], ...
SELECT [DISTINCT | ALL | DISTINCTROW] *
```

- Le mot réservé `ALL` permet de retourner tous les t-uplets correspondant à la requête.
- Le mot réservé `DISTINCT` permet d'éliminer les doublons, c'est-à-dire les t-uplets identiques dans le résultat de la requête. Car contrairement à ce qui se passe avec les opérateurs relationnels, une requête SQL peut fournir des t-uplets identiques.
- Le mot réservé `DISTINCTROW` est un synonyme de `DISTINCT`.
- Les expressions peuvent être des noms d'attributs, tels que `NomCinema,VilleCinema` ou le résultat de fonctions que nous détaillerons plus loin dans ce cours.

7.1.2.1 Mot réservé DISTINCT

Reprenons un exemple utilisant l'opérateur de projection, à savoir la liste des noms des acteurs. Cette requête s'écrit :

```
SELECT NomActeur
FROM distribution
```

Nous voyons que le résultat 7.1 contient deux fois le nom `Schwarzenegger`, ce qui n'était pas possible avec les opérateurs relationnels. Pour éviter ce doublon la requête doit être écrite de la façon suivante :

```
SELECT DISTINCT NomActeur
FROM distribution
```

Remarque : Le mot réservé `DISTINCT` ne doit pas être utilisé de façon systématique. En effet pour pouvoir éliminer les doublons le moteur du SGBD devra examiner tous les t-uplets résultats ce qui peut prendre beaucoup de temps.

SELECT NomActeur
FROM DISTRIBUTION

NOMACTEUR
Freeman
Hoffman
Russo
Lawrence
Loni
Smith
Gibson
Marceau
Abril
Balasko
Chabat
De Mornay
De Niro
Schwarzenegger
Schwarzenegger

SELECT DISTINCT NomActeur
FROM DISTRIBUTION

NOMACTEUR
Freeman
Hoffman
Russo
Lawrence
Loni
Smith
Gibson
Marceau
Abril
Balasko
Chabat
De Mornay
De Niro
Schwarzenegger

Figure 7.1 : Requêtes SELECT sans et avec DISTINCT

7.1.2.2 Mot réservé *

L'utilisation du caractère * permet de sélectionner tous les attributs d'une table. La requête

```
SELECT *
FROM cinema
```

retourne toutes les données contenues dans la table CINEMA 7.2 .

NUMEROCINEMA	NOMCINEMA	ADRESSECINEMA	CODEPOSTALCINEMA	VILLECINEMA	TELEPHONECINEMA
1	Vox	Grande Rue	25000	Besançon	03 81 82 05 69
2	Plazza	Rue des granges	25000	Besançon	03 81 26 35 98
3	Plazza	Rue Ronchaux	21000	Dijon	03 80 26 53 25
4	Royal	Rue Villarceau	21000	Dijon	03 80 23 52 58

Figure 7.2 : Requête SELECT avec *

7.1.2.3 Mot réservé AS

L'utilisation du mot réservé `AS` suivi d'un nom permet de renommer les résultats. Ce renommage peut aussi être réalisé en mettant simplement le nouveau nom du résultat juste derrière le nom de l'attribut ou l'expression.

```
SELECT NomCinema AS NomCine
FROM cinema
```

ou

```
SELECT NomCinema NomCine
FROM cinema
```

retourne le résultat 7.3 où l'attribut se nomme `NomCine`.

NomCine
Vox
Plazza
Plazza
Royal

Figure 7.3 : Requête Select avec AS

7.1.2.4 Nommage d'un attribut

Le nom complet d'un attribut d'une table est le nom de la table suivi d'un point et du nom de l'attribut. Le nom de la table peut être omis lorsqu'il n'y a aucune ambiguïté.

```
NOM-TABLE.nom-attribut
```

Par exemple `cinema.NomCinema` permet de caractériser l'attribut `NomCinema` de la table `CINEMA`. La première requête présentée ci-dessus peut alors s'écrire :

```
SELECT cinema.NomCinema
FROM cinema
```

Par défaut dans les SGBD, les requêtes générées utiliseront cette façon sans ambiguïté pour nommer les attributs.

7.1.3 La clause FROM

La clause `FROM` permet :

- d'indiquer quelles relations sont utilisées par la requête,
- de définir une relation qui est le produit cartésien des relations décrites,
- de déclarer les jointures utilisées dans la requête.

Sa syntaxe est :

```
FROM TABLE1 [AS Nom1], TABLE2 [AS Nom2], ...
```

Oracle : Le renommage d'une table se fait avec la syntaxe suivante `TABLE Nom`. Le mot réservé `AS` ne peut pas être utilisé dans la clause `FROM`.

Considérons deux tables `CINEMA`, et `FILM` contenant les noms des cinémas d'une part, et des titres des films d'autre part. Le produit cartésien de ces deux tables donne la liste des affiches possibles. Comme le montre l'exemple ci-contre **??**.
En SQL cette requête est traduite par :

```
SELECT *
FROM cinema, film
```

Pour éviter les confusions lorsque par exemple on utilise deux fois la même relation dans une requête, on peut renommer une table. Pour cela on écrit le nom de la table suivi du nouveau nom. Dans la requête ci-dessous, on donne une liste contenant des couples de titres de films, formés à partir d'une même relation `AFFICHE`.

```
SELECT A1.titre, A2.titre
FROM affiche A1, affiche A2
```

Dès que nous utiliserons plusieurs relations dans une clause `SELECT`, leur produit cartésien sera défini, sauf indication contraire.

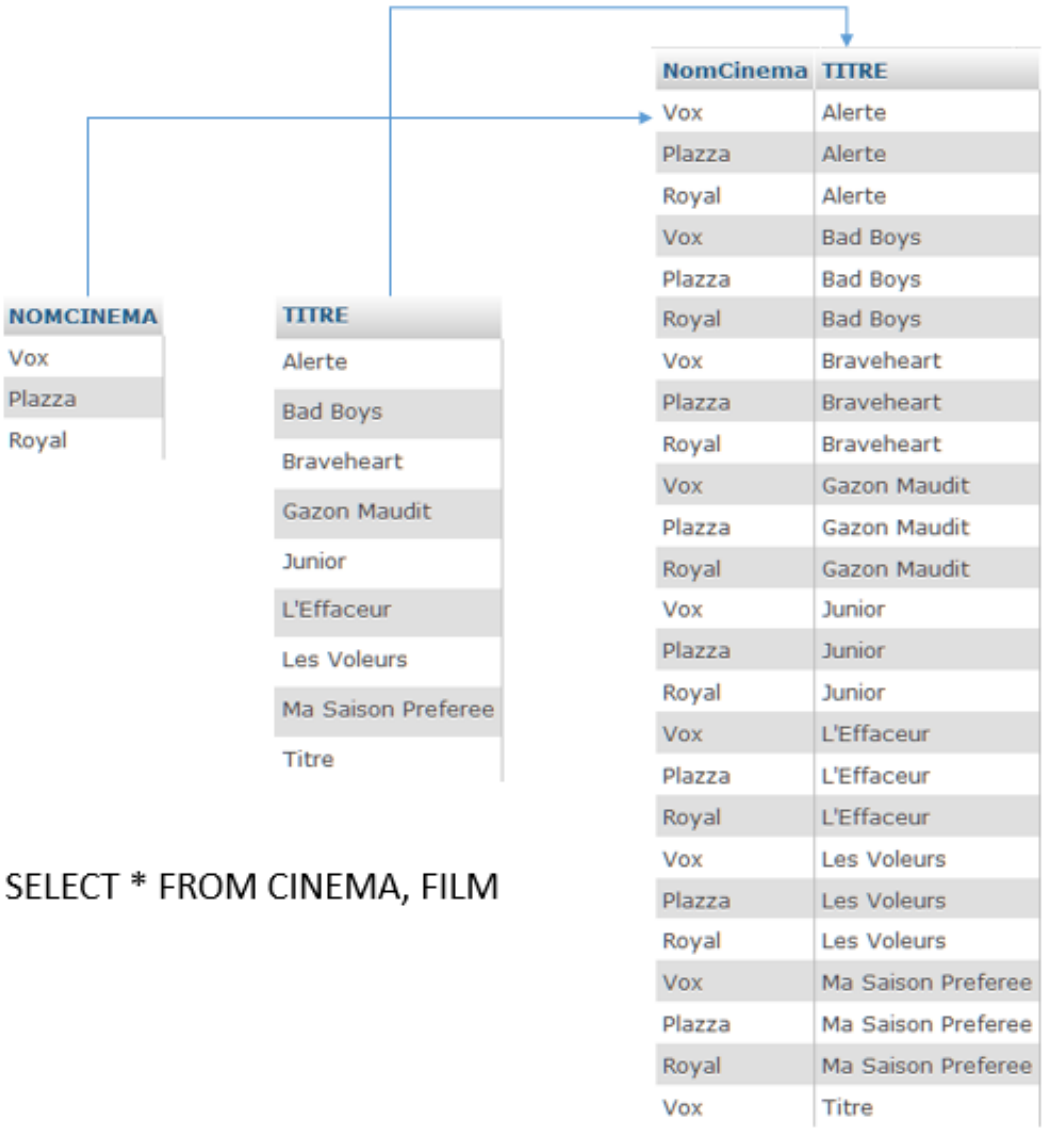


Figure 7.4 : Produit cartésien avec la clause FROM

Exercice 5.1 : Décrire les requêtes suivantes en utilisant les clauses `SELECT` et `FROM` :

1. La liste des noms des metteurs en scène.
2. Le titre des films qui sont à l'affiche dans au moins un cinéma. Donner une version de la requête qui ne duplique pas les titres de films obtenus.
3. La liste de toutes les affiches possibles. Le résultat donnera seulement le numéro des cinémas et le titre des films.

7.1.4 La clause WHERE

La clause `WHERE` permet :

- de spécifier les t-uplets à sélectionner dans une relation ou dans le produit cartésien de plusieurs relations,
- de définir le critère de sélection : un prédicat qui est évalué pour chaque t-uplet.

Syntaxe de la clause `WHERE` :

```
WHERE critère de sélection
```

Sélectionnons par exemple les noms des cinémas de Besançon. Cette requête est décrite par la sélection suivante :

```
SELECT NomCinema
FROM cinema
WHERE VilleCinema = 'Besançon'
```

7.1.4.1 L'expression des critères de sélection

Il est possible d'exprimer des critères de sélection plus complexes, en utilisant :

- des opérateurs de comparaison : <, >, >=, <=, =, <>
- des opérateurs logiques : AND, OR, NOT, XOR
- d'autres prédicats qui servent à définir des ensembles : IN, BETWEEN ... AND ..., LIKE
- la comparaison avec la valeur NULL : IS NULL, IS NOT NULL
- l'utilisation de caractères génériques : "_" qui remplace un caractère, et "%" qui remplace une chaîne de caractères de longueur quelconque. Attention ces caractère génériques peuvent varier en fonction du SGBD utilisé.

Pour sélectionner les noms des cinémas de Besançon et de Paris, on peut proposer la requête suivante:

```
SELECT NomCinema
FROM cinema
WHERE VilleCinema = 'Besançon'
OR VilleCinema = 'Paris'
```

ou la requête ci-dessous qui utilise l'opérateur IN :

```
SELECT NomCinema
FROM cinema
WHERE VilleCinema IN ('Besançon','Paris')
```

Exercice 5.2 : Décrire les requêtes suivantes en utilisant la clause `WHERE` :

1. Les acteurs jouant dans Alerte
2. Les titres des films à l'affiche du cinéma N°1
3. Les titres des films mis en scène par Téchiné
4. Les films d'aventure ou mis en scène par Téchiné
5. Les informations sur les films ayant Balasko comme metteur en scène et qui sont des comédies.
6. Le nom des acteurs qui commencent par "D".
7. Le nom et le numéro des cinémas pour lesquels on ne connaît pas le numéro de téléphone.
8. Le nom des cinémas dont le code postal est compris entre 25000 et 25999.

7.1.5 La Jointure en SQL

Nous avons défini dans le chapitre 4, la jointure comme étant la composition d'un produit cartésien et d'une sélection. Dans les différentes versions de SQL on trouve des expressions différentes de cet opérateur.

7.1.5.1 Expression de la jointure avec la clause WHERE

Cette expression a été proposée dans SQL1, elle permet de réaliser une jointure interne, mais pas externe.

La jointure : `T = R [condition de jointure] S` s'exprime en SQL par la requête :

```
SELECT *
FROM R, S
WHERE condition de jointure
```

Reprenons l'exemple suivant qui donne des informations sur les films et leurs affiches. La jointure :

```
FILM [FILM.Titre = AFFICHE.Titre] AFFICHE
```

est traduite en SQL par :

```
SELECT *
FROM film, affiche
WHERE film.Titre = affiche.Titre
```

La condition de jointure apparaît dans la clause `WHERE`.

7.1.5.2 Expression de la jointure avec l'opérateur JOIN

L'opérateur JOIN a été introduit dans SQL2, il permet de spécifier les relations intervenant dans la jointure ainsi que la condition de jointure.

La jointure : T = R [condition de jointure] S s'exprime en SQL par la requête :

```
SELECT *
FROM R Op Jointure S ON Condition de jointure
```

L'opérateur de jointure peut prendre les valeurs suivantes selon le type de jointure que l'on désire réaliser.

– – **Jointure interne** : `INNER JOIN` ou `JOIN`

Cet opérateur permet d'exprimer une jointure interne. L'exemple précédent se traduit par :

```
SELECT *
FROM film INNER JOIN affiche on film.Titre = affiche.Titre
```

Nous pouvons aussi traduire cette jointure naturelle de la façon suivante :

```
SELECT *
FROM cinema NATURAL JOIN affiche
```

Ici les attributs servant à la jointure naturelle sont les attributs communs aux deux tables, à savoir `NumeroCinema`.

Remarque : Attention cette expression peu utilisée de la jointure peut poser des problèmes. Elle est présentée ici pour que vous sachiez qu'elle existe. Mais il vaut mieux éviter de l'utiliser, surtout lorsque les jointures portent sur plus de 2 tables.

– – **Jointure externe** : `OUTER JOIN`

Il est possible de spécifier différents types de jointure externe, gauche, droite ou complète, selon que l'on désire garder tous les t-uplets en provenance de la relation de droite dans l'expression, de gauche ou des deux.

```
SELECT *
FROM film FULL OUTER JOIN affiche on film.Titre = affiche.Titre
```

Pour cela on utilise les termes :

- `LEFT OUTER JOIN`
- `RIGHT OUTER JOIN`
- `FULL OUTER JOIN`

La requête :

```
SELECT *
FROM film LEFT OUTER JOIN affiche on film.Titre = affiche.Titre
```

avec les tables 7.5 et 7.6 donne le résultat 7.7 :

NumeroCinema	Titre
1	Alerte
2	Alerte
3	Alerte
4	Alerte
4	Braveheart
2	Gazon Maudit
1	Junior
3	Junior
2	L'Effaceur
4	Les Voleurs

Figure 7.5 : Table AFFICHE

TITRE	METTEURENSCENE	GENRE
Alerte	Peterson	Aventure
Bad Boys	Bay	Policier
Braveheart	Gibson	Aventure
Gazon Maudit	Balasko	Comedie
Junior	Reitman	Comedie
L'Effaceur	Russel	Action
Les Voleurs	Techine	Comedie Dramatique
Ma Saison Preferee	Techine	Comedie Dramatique

Figure 7.6 : Table FILM

Tous les t-uplets de la table FILM réapparaissent dans le résultat, mais les film de Titre (`Casper, Harcelement, Harcelement`) qui ne sont à l'affiche d'aucun cinéma apparaissent sans valeur pour : `AFFICHE.NumeroCinema` et `AFFICHE.Titre`.

TITRE	METTEURENSCENE	GENRE	NumeroCinema	Titre
Alerte	Peterson	Aventure	1	Alerte
Alerte	Peterson	Aventure	2	Alerte
Alerte	Peterson	Aventure	3	Alerte
Alerte	Peterson	Aventure	4	Alerte
Bad Boys	Bay	Policier	*NULL*	*NULL*
Braveheart	Gibson	Aventure	4	Braveheart
Gazon Maudit	Balasko	Comedie	2	Gazon Maudit
Junior	Reitman	Comedie	1	Junior
Junior	Reitman	Comedie	3	Junior
L'Effaceur	Russel	Action	2	L'Effaceur
Les Voleurs	Techine	Comedie Dramatique	4	Les Voleurs
Ma Saison Preferee	Techine	Comedie Dramatique	*NULL*	*NULL*

Figure 7.7 : Exemple de jointure externe

Exercice 5.3 : Jointure

Décrire en SQL les requêtes suivantes, en utilisant les deux façons de traduire la jointure :

1. Le nom des cinémas ayant au moins un film à l'affiche.
2. Les titres des films à l'affiche du Vox.
3. Le nom des cinémas ayant à l'affiche Junior.
4. Décrire avec la jointure externe adéquate la requête : Donner pour chaque film, les noms des acteurs faisant partie de la distribution. Dans le cas où le film n'a aucun acteur dans sa distribution, on affichera seulement le titre du film.

7.1.6 Les fonctions de groupe

Les fonctions de groupe ou fonctions d'agrégation permettent d'obtenir des informations sur un ensemble de t-uplets dans une table. On a les fonctions suivantes 7.1 :

Calcul de la valeur moyenne	AVG
Somme	SUM
Plus petite valeur	MIN
Plus grande valeur	MAX
Variance	VARIANCE
Écart type	STDDEV
Nombre de lignes	COUNT
Chaînes de caractères concaténées	GROUP_CONCAT
Booléen qui vaut vrai si toutes les occurrences sont vraies	EVERY
Booléen qui vaut vrai si au moins une occurrence est vraie	ANY | SOME

Table 7.1 : Fonctions de groupe

Lorsque l'on utilise une telle fonction on travaille sur une colonne et non pas sur une ligne (t-uplet) comme dans une clause `WHERE`.
On peut de plus utiliser les mots réservés `ALL` ou `DISTINCT` pour les fonctions statistiques. La

première requête 7.8 donne le nombre de films à l'affiche du cinéma N°1, la suivante donne le plus petit code postal répertorié dans la relation `CINEMA`.

```
SELECT COUNT(Titre) AS NbFilms
FROM affiche
WHERE NumeroCinema = 1
```

Le résultat est le suivant :

NUMEROCINEMA	TITRE
1	Alerte
1	Junior

NBFILMS
2

Figure 7.8 : Fonction COUNT

Autre exemple de requête avec une fonction de groupe :

```
SELECT MIN(CodePostalCinema) AS PlusPetit
FROM cinema
```

Dans les deux requêtes ci-dessous, on a renommé le résultat en utilisant le terme AS. Lorsque l'on utilise des fonctions de groupe, le résultat est nommé automatiquement par le SGBD, ce qui donne des noms du type `MIN(CodePostalCinema)`. Pour obtenir un résultat qui soit en particulier réutilisable il convient de le renommer.

Enfin dernier exemple avec la fonction de concaténation de chaines de caractères :

```
SELECT nomcinema, GROUP_CONCAT(CodePostalCinema)
FROM cinema
WHERE nomcinema='Plazza'
```

On obtient le résultat :

nomcinema	GROUP_CONCAT(codepostalcinema)
Plazza	25000,21000

Figure 7.9 : Requête avec la fonction GROUP_CONCAT

Particularités de la fonction `COUNT`

Attention, il existe différentes syntaxes pour utiliser la fonction `COUNT`.

- `COUNT(*)` : compte le nombre de t-uplets obtenus dans la requête (lignes).
- `COUNT(attribut)` : compte le nombre de t-uplets dont la valeur de l'attribut est différente de NULL
- `COUNT(DISTINCT attribut)` : compte le nombre de valeurs différentes de l'attribut.

Ainsi la requête :

```
SELECT count(*)
FROM affiche
```

donne le nombre de t-uplets dans la table AFFICHE à savoir 10.

```
SELECT count(Titre)
FROM affiche
```

donne le nombre de t-uplets dans la table `AFFICHE` pour lesquels l'attribut `Titre` n'a pas la valeur `NULL`, à savoir 10.

```
SELECT count(DISTINCT Titre)
FROM affiche
```

donne le nombre de titres différents dans la relation AFFICHE à savoir 5.

Exercice 5.4 : Les fonctions de groupe

Décrire en SQL les requêtes suivantes :

1. Donner le nombre de films d'aventure.
2. Donner le nombre de films à l'affiche à Besançon.
3. Donner le plus petit numéro de cinéma dans lequel est à l'affiche le film "Alerte".

7.1.7 La clause GROUP BY

7.1.7.1 Présentation de la clause GROUP BY

La clause `GROUP BY` permet :

- de subdiviser la table en groupes
- une seule ligne représente l'ensemble des t-uplets de la table regroupés, tous les t-uplets regroupés ont la même valeur pour les expressions du regroupement.

Syntaxe de la clause `GROUP BY`

```
GROUP BY expr1, expr2, ..
```

Considérons la relation `AFFICHE` le regroupement de cette table par rapport à l'attribut `NumeroCinema` s'écrit en SQL

```
SELECT ...
FROM affiche
GROUP BY NumeroCinema
```

et donne la table intermédiaire 7.10 .

Le regroupement est réalisé ici par rapport à l'attribut `NumeroCinema`. On retrouve dans

NumeroCinema	Titre
1	Alerte
	Junior
2	Alerte
	Gazon Maudit
	L'Effaceur
3	Alerte
	Junior
4	Alerte
	Braveheart
	Les Voleurs

Figure 7.10 : Regroupement

chaque ligne du regroupement tous les éléments correspondant à une même valeur de l'attribut

NumeroCinema. On obtient 4 groupes.
On peut se rendre compte que cette table n'est pas "correcte", en ce sens que les valeurs de l'attribut Titre ne sont plus atomiques. A ce titre les informations que l'on va pouvoir extraire d'une telle table sont limitées. On ne peut pas par exemple donner le titre d'un film pour le premier regroupement.

7.1.7.2 Les restrictions liées à la clause GROUP BY

Les seules opérations réalisables sur une "table regroupée" sont des opérations qui prennent en compte un ensemble de lignes, comme les fonctions de groupe, ou alors des opérations de projection portant sur des attributs impliqués dans le regroupement.

Dans l'exemple précédent, on peut par exemple :
- appliquer des fonctions de groupe sur l'attribut Titre,
- réaliser une projection sur l'attribut NumeroCinema.

La requête suivante permet de compter pour chaque cinéma le nombre de films qui y sont à l'affiche 7.11 .

```
SELECT NumeroCinema, COUNT(Titre)As NBFilms
FROM affiche
GROUP BY NumeroCinema
```

NumeroCinema	NBFilms
1	2
2	3
3	2
4	3

Figure 7.11 : Requête de regroupement

Exercice 5.5 : Le regroupement
Décrire en SQL les requêtes suivantes, et présenter à chaque fois la table intermédiaire définie par le regroupement :

1. Donner pour chaque acteur le nombre de films dans lesquels il joue.
2. Donner pour chaque film, le premier acteur par ordre alphabétique.
3. Donner pour chaque cinéma, le nombre de films à l'affiche. On donnera le nom du cinéma et le nombre de titres à l'affiche dans celui-ci, et si aucun film n'est à l'affiche, le nombre sera zéro.

7.1.8 La clause HAVING

La clause HAVING permet de sélectionner des groupes définis par la clause GROUP BY.
Syntaxe de la clause HAVING

```
HAVING prédicat
```

Le prédicat associé à une clause HAVING est défini de la même façon que les prédicats de la clause WHERE, mais il ne peut porter que sur les groupes définis par la clause GROUP BY.

Reprenons l'exemple vu dans la présentation de la clause GROUP BY, qui permet de compter pour chaque cinéma, le nombre de films qui y sont à l'affiche. Et supposons qu'ici nous voulions

sélectionner le numéro de cinémas dans lesquels il y a au moins trois films à l'affiche. En SQL cette requête est traduite par :

```
SELECT NumeroCinema
FROM affiche
GROUP BY NumeroCinema
HAVING COUNT(titre) > 2
```

Cette condition ne peut pas être exprimée dans une clause `WHERE`, car comme nous l'avons dit dans la présentation d'une requête `SELECT`, la clause `WHERE` est évaluée avant le regroupement. Reprenons la requête que nous venons de présenter et regardons comment elle est évaluée. Lors de l'évaluation d'une requête, la première clause traitée est la clause `WHERE`, ici il n'y en a pas. Ensuite on évalue la clause `GROUP BY`, ce qui donne la table intermédiaire 7.10 . Puis on évalue les fonctions d'agrégation 7.11 et enfin la clause `HAVING` 7.12 .

NumeroCinema
2
4

Figure 7.12 : Clause HAVING

Exercice 5.6 : La clause HAVING

Décrire en SQL les requêtes suivantes :

1. Donner le nom des acteurs qui jouent dans au moins deux films.
2. Donner les titres des films qui ont au plus deux acteurs dans leur distribution.

7.1.9 La clause ORDER

La clause `ORDER` permet de préciser dans quel ordre les t-uplets sélectionnés seront donnés. Le tri peut s'effectuer selon plusieurs critères, en allant du premier au dernier.
Syntaxe de la clause `ORDER`

```
ORDER BY Expr1 [DESC], Expr2 [DESC], ...
```

Cette clause ne modifie l'ensemble des données retournées par la requête, mais simplement leur ordre de présentation. Les expressions sont généralement des noms d'attributs. On peut trier par ordre croissant par défaut, ou par ordre décroissant, en utilisant dans ce cas le mot réservé `DESC`.
Pour donner la liste des noms de cinémas par ordre alphabétique, on a la requête :

```
SELECT NomCinema, VilleCinema
FROM cinema
ORDER BY NomCinema
```

NomCinema ▲ 1	VilleCinema
Plazza	Besançon
Plazza	Dijon
Royal	Dijon
Vox	Besançon

Figure 7.13 : Clause ORDER

Et la requête ci-dessous donne le nom et la ville des cinémas, en respectant l'ordre alphabétique des villes, puis des cinémas.

```
SELECT NomCinema, VilleCinema
FROM cinema
ORDER BY VilleCinema, NomCinema
```

NomCinema ▲ 2	VilleCinema ▲ 1
Plazza	Besançon
Vox	Besançon
Plazza	Dijon
Royal	Dijon

Figure 7.14 : Clause Order

Exercice 5.7 : La clause ORDER
Décrire en SQL les requêtes suivantes, et présenter à chaque fois la table obtenue :

1. Donner pour chaque acteur le nombre de films dans lesquels il joue, la liste sera triée par ordre décroissant du nombre de films.
2. Afficher les informations concernant les films en les triant par genre, et par titre.

7.2 Les sous-requêtes

Les sous-requêtes permettent l'expression d'un prédicat à l'aide du résultat d'un `SELECT`. Leur syntaxe est la suivante :

```
SELECT colonnes
FROM TABLE1
WHERE Col op (SELECT col
              FROM TABLE2
              WHERE condition)
```

Comment par exemple exprimer que l'on veut les titres des films qui ont été réalisés par le même metteur en scène que le film "les voleurs" ? Nous disposons de deux types de solutions. La première utilise une auto-jointure de la table `FILM` :

```
SELECT F1.Titre
FROM FILM F1 INNER JOIN FILM F2 ON F1.MetteurEnScene = F2.MetteurEnScene
WHERE F2.Titre = 'Les Voleurs'
```

La deuxième solution, qui s'exprime avec une sous-requête, permet de sélectionner les titres de films, appartenant à l'ensemble des titres de films qui ont pour metteur en scène, le metteur en scène du film "les voleurs". En d'autres termes, nous définissons une requête qui donne le nom de ce metteur en scène, et nous l'utilisons dans notre requête principale, ce qui nous donne la requête suivante :

```
SELECT Titre
FROM FILM
WHERE MetteurEnScene = (SELECT MetteurEnScene
                        FROM FILM
                        WHERE Titre = 'Les Voleurs')
```

Cette deuxième solution s'exprime beaucoup plus naturellement.
Nous distinguons différents types de sous-requêtes selon qu'elles sont indépendantes ou non de la requête principale, ou qu'elles ramènent un ou plusieurs t-uplets.

7.2.1 Les sous-requêtes indépendantes

Les sous-requêtes indépendantes, sont évaluées en premier indépendamment de la requête principale, qui sera quant à elle évaluée ensuite.
Reprenons l'exemple des films mis en scène par le même metteur en scène que 'les voleurs'. La sous-requête :

```
SELECT MetteurEnScene
FROM FILM
WHERE Titre = 'Les Voleurs'
```

donne la relation suivante à partir de la relation `FILM` 7.2.1. L'évaluation de la requête principale

TITRE	METTEURENSCENE	GENRE
Casper	Silberling	Comédie
Harcelement	Levinson	Comédie Dramatique
Junior	Reitman	Comédie
Alerte	Peterson	Aventure
Gazon Maudit	Balasko	Comédie
L'Effaceur	Russel	Action
Les Voleurs	Téchiné	Comédie dramatique
Ma saison préférée	Téchiné	Comédie dramatique

METTEURENSCENE
Téchiné

Figure 7.15 : Sous-requête

se ramène alors à :

```
SELECT Titre
FROM FILM
WHERE MetteurEnScene = 'Téchiné'
```

et donne comme résultat 7.16 . Selon le nombre de t-uplets ramenés par la sous-requête les

TITRE
Les Voleurs
Ma saison préférée

Figure 7.16 : Résultat de la requête

opérateurs utilisés sont différents. Nous présentons dans ce qui suit ces différents types de sous-requêtes. Le nombre d'attributs retourné par la sous-requête est de 1.

7.2.1.1 Les sous-requêtes ramenant un t-uplet

Les opérateurs utilisés sont des opérateurs permettant de comparer deux valeurs, à savoir : = <> < > <= et >=.

La requête suivante donne par exemple la liste des noms et Numéro de téléphone des cinémas dont le nom est inférieur par ordre alphabétique à celui du cinéma N°1.

```
SELECT NomCinema,TelephoneCinema
FROM CINEMA
WHERE NomCinema < (SELECT Nomcinema FROM CINEMA WHERE NumeroCinema = 1)
```

Ce qui donne le résultat 7.17

NOMCINEMA	TELEPHONECINEMA
Plazza	03 81 26 35 98
Eldorado	01 23 56 98 56
Pax	01 23 56 98 78

Figure 7.17 : Sous-requête indépendante retournant 1 t-uplet

7.2.1.2 Les sous-requêtes ramenant plusieurs t-uplets

Les opérateurs utilisés sont des opérateurs ensemblistes qui permettent de comparer une valeur avec une liste de valeurs.

– – **L'opérateur `IN`**

Cet opérateur ensembliste classique permet de vérifier que l'attribut spécifié dans la requête principale comme étant lié avec la sous-requête, doit avoir une valeur qui appartient à l'ensemble des valeurs retournées par la sous-requête.

Cet opérateur peut être composé avec un `NOT`. Dans ce cas la valeur de l'attribut ne doit pas appartenir à l'ensemble des valeurs retournées par la sous-requête.

La requête suivante retourne les titres et les genres des films qui ne sont pas à l'affiche du cinema N°1 7.18 .

```
SELECT Titre, Genre
FROM FILM
WHERE Titre NOT IN (SELECT Titre FROM AFFICHE WHERE NumeroCinema = 1)
```

TITRE	GENRE
Casper	Comédie
Harcelement	Comédie Dramatique
Gazon Maudit	Comédie
L'Effaceur	Action
Les Voleurs	Comédie dramatique
Ma saison préférée	Comédie dramatique

Figure 7.18 : Sous-requête avec IN

– – **Les opérateurs `ANY` et `ALL`**

Ces opérateurs sont associés aux opérateurs classiques de comparaison.

- Lorsque l'on utilise l'opérateur `ANY`, la comparaison sera vraie, si elle est vraie pour au moins un élément de l'ensemble (et fausse si l'ensemble est vide).
- Lorsque l'on utilise l'opérateur `ALL` la comparaison sera vraie, si elle est vraie pour tous les éléments de l'ensemble (vraie si l'ensemble est vide).

La requête suivante donne le nom du premier cinéma par ordre alphabétique 7.19 :

```
SELECT NomCinema
FROM CINEMA
WHERE NomCinema <= ALL (SELECT NomCinema FROM CINEMA)
```

NOMCINEMA
Eldorado

Figure 7.19 : Sous-requête avec ALL

Remarque : L'opérateur `IN` est équivalent à `=ANY`, et l'opérateur `NOT IN` est équivalent à `<> ALL`. Ainsi l'exemple donné ci-dessus pour illustrer l'opérateur IN, peut s'écrire :

```
SELECT Titre, Genre
FROM FILM
WHERE Titre <> ALL (SELECT Titre FROM AFFICHE WHERE NumeroCinema = 1)
```

Exercice 5.8 : Les sous-requêtes indépendantes

Décrire les requêtes suivantes en utilisant des sous-requêtes.

1. Les noms des acteurs des films mis en scène par Balasko
2. Les titres des films à l'affiche du cinéma Vox
3. Les noms des acteurs qui n'ont jamais joué dans un film mis en scène par Téchiné
4. Les acteurs jouant dans un film avec un acteur qui a joué dans L'Effaceur

7.2.2 Les sous-requêtes synchronisées

Ces sous-requêtes font référence à une colonne de la requête principale, et nécessitent la réévaluation de la sous-requête pour chaque t-uplet de la requête principale.
Ce type de sous-requête est plus complexe que le précédent, car ici l'évaluation de la sous-requête se fait pour chaque ligne obtenue par la requête principale.
Considérons l'exemple suivant qui nous donne, les noms des cinémas apparaissant dans plusieurs villes :

```
SELECT DISTINCT C1.NomCinema
FROM CINEMA C1
WHERE C1.NomCinema = (SELECT Distinct C2.NomCinema
                      FROM CINEMA C2
                      WHERE C2.NomCinema = C1.NomCinema
                      AND C2.VilleCinema <> C1.VilleCinema)
```

L'évaluation de cette requête est réalisée de la façon suivante :

- La première valeur de C1.NomCinema est "Eldorado", la sous-requête est donc évaluée avec cette valeur, ce qui nous donne un résultat vide.
- La deuxième valeur de C1.NomCinema est "Vox"; la sous-requête est évaluée avec cette valeur, et donne un résultat vide,
- La troisième valeur de C1.NomCinema est "Plazza"; la sous-requête est évaluée avec cette valeur, et donne un résultat vide,
- La deuxième valeur de C1.NomCinema est "Pax"; la sous-requête est évaluée avec cette valeur, et donne un résultat vide.

Il est possible d'écrire différents types de sous-requêtes synchronisées : des requêtes synchronisées *simples* et des sous-requêtes introduites par `EXISTS`.

7.2.2.1 Sous-requêtes synchronisées simples

L'exemple que nous venons de voir fait partie de ces sous-requêtes. La sous-requête :

- est introduite comme dans le cas d'une requête indépendante, avec des opérateurs de comparaison, et/ou des opérateurs ensemblistes (selon la sous-requête peut retourner un ou plusieurs t-uplets)
- elle contient des références aux colonnes de la requête principale → **synchronisation avec la requête principale**

7.2.2.2 Sous-requêtes avec la clause `EXISTS`

La clause `EXISTS` est utilisée dans la clause `WHERE`, où elle est suivie d'une sous-requête entre parenthèse. Elle est évaluée à `VRAI`, si au moins un t-uplet satisfait les conditions de la sous-requête. La sous-requête doit être synchronisée avec la requête principale.

La syntaxe est la suivante :

```
SELECT ...
WHERE EXISTS (SELECT ...)
```

La sous-requête peut renvoyer 0 ou plusieurs valeurs, pour obtenir un succès, la sous-requête doit retourner au moins un t-uplet.

Il est possible d'utiliser l'opérateur `NOT` devant la clause `EXISTS`.

Reprenons l'exemple précédent qui nous donne, les noms des cinémas apparaissant dans plusieurs villes :

```
SELECT DISTINCT C1.NomCinema
FROM CINEMA C1
WHERE EXISTS (SELECT C2.NomCinema
              FROM CINEMA C2
              WHERE C2.NomCinema = C1.NomCinema
              AND C2.VilleCinema <> C1.VilleCinema)
```

Exercice 5.9 : Les sous-requêtes synchronisées
Décrire les requêtes suivantes en utilisant des sous-requêtes synchronisées

1. Les noms des acteurs des films mis en scène par Téchiné
2. Les titres des films à l'affiche du cinéma Vox
3. Les noms des acteurs qui n'ont jamais joué dans un film mis en scène par Téchiné
4. Le nom du premier acteur par ordre alphabétique

7.3 Les opérateurs ensemblistes en SQL

Nous retrouvons en SQL les trois opérateurs ensemblistes vus dans les opérateurs relationnels, à savoir : l'union, l'intersection, et la différence. Pour chacun des ces opérateurs la syntaxe est la même, ils permettent de composer le résultats de deux requêtes `SELECT`.

```
SELECT ...
OP
SELECT ...
```

Les deux requêtes doivent retourner des relations ayant le même en-tête.

Remarque : Attention ces opérateurs n'existent pas dans tous les SGBD. En particulier sous MariaDB. Mais il existe alors d'autres solutions pour les exprimer.

7.3.1 L'Union : UNION

Considérons la requête suivante qui nous donne le nom des cinémas de Besançon et (ou) de Paris.

```
SELECT NomCinema
FROM CINEMA
WHERE VilleCinema = 'Besançon'
UNION
SELECT NomCinema
FROM CINEMA
WHERE VilleCinema ='Paris'
```

NOMCINEMA
Eldorado
Pax
Plazza
Vox

Figure 7.20 : Requête UNION

Oracle : Il est possible d'utiliser l'opérateur `UNION` avce le mot réservé `ALL`. dans ce cas la requête peut retourner des doublons.
Cette requête aurait pu être écrite en utilisant l'opérateur `OR` dans la clause `WHERE`, ce qui donne :

```
SELECT NomCinema
FROM CINEMA
WHERE VilleCinema = 'Besançon'
OR VilleCinema = 'Paris'
```

7.3.2 L'Intersection : INTERSECT

La requête ci-dessous donne le nom des cinémas qui apparaissent à Besançon et à Paris.

```
SELECT NomCinema
FROM CINEMA
WHERE VilleCinema = 'Besançon'
INTERSECT
SELECT NomCinema
FROM CINEMA
WHERE VilleCinema = 'Paris'
```

Remarque : Attention le choix des attributs de projection est fondamental lorsque l'on réalise une intersection.

7.3.3 La différence : MINUS

La requête ci-dessous donne le nom des cinémas qui apparaissent à Besançon mais pas à Paris.

```
SELECT NomCinema
FROM CINEMA
WHERE VilleCinema = 'Besançon'
MINUS
SELECT NomCinema
FROM CINEMA
WHERE VilleCinema ='Paris'
```

Exercice 5.10 : Les opérateurs ensemblistes

Décrire les requêtes suivantes en utilisant les opérateurs ensemblistes

1. Les noms des acteurs des films mis en scène par Téchiné ou Peterson
2. Les titres des films à l'affiche du cinéma Vox et du cinéma N° 3.
3. Les noms des acteurs qui n'ont jamais joué dans un film mis en scène par Reitman

7.4 La division en SQL

Nous avons vu l'opérateur de division dans la modélisation relationnelle. Sa traduction n'est pas immédiate en SQL, même s'il existe différentes façons de la traduire. Rappelons le principe de la division :

```
X(At1, At2) / Y(At2) --> T(At1)
```

Signifie que toute valeur t de l'attribut At1 appartenant à T est telle que : pour toute valeur y de At2 dans Y, le couple (t,y) appartient à la relation X.

7.4.1 La division avec la fonction d'agrégation COUNT

Dans ce cas nous considérons que t appartient à T si le nombre de valeurs distinctes de At2 apparaissant avec t dans X est égal au nombre de valeurs de At2 dans Y. Cette solution suppose que les valeurs de l'attribut At2 qui apparaissent dans la relation X ne peuvent pas prendre des valeurs différentes de celles qui apparaissent dans la relation Y.
Dans ces conditions la division peut s'exprimer de la façon suivante :

```
SELECT At1
FROM X
GROUP BY At1
HAVING COUNT(DISTINCT At2) = (SELECT COUNT(*) FROM Y)
```

La requête qui donne le titre des films à l'affiche de tous les cinémas, s'exprime ici par :

```
SELECT Titre
FROM AFFICHE
GROUP BY Titre
HAVING COUNT(DISTINCT NumeroCinema) = (SELECT COUNT(*) FROM CINEMA)
```

TITRE
Alerte

Figure 7.21 : Requête de division avec COUNT

7.4.2 La division avec la clause EXISTS

Pour définir cette solution reprenons la définition de l'opérateur de division. Nous avons dit que :
$\forall$ at1 $\in$ T on a : $\forall$ at2 $\in$ Y, (at1, at2) $\in$ X
Ce que nous pouvons réécrire de la façon suivante :
$\forall$ at1 $\in$ T on a : $\neg \exists$ at2 $\in$ Y, (at1, at2) $\notin$ X
C'est à dire que pour toute valeur at1 de T, il n'existe pas de valeur at2 de Y pour laquelle (at1, at2) n'appartienne pas à X.
Ce que nous traduisons en SQL par :

```
SELECT At1
FROM X X1
WHERE NOT EXISTS
    (SELECT At2
     FROM Y
     WHERE NOT EXISTS
         (SELECT X2.At1, X2.At2
          FROM X X2
          WHERE X1.At1 = X2.At1
          AND Y.At2 = X2.At2)
```

Cette deuxième façon de traduire la division est valable quelque soit le contexte.

La requête qui donne le titre des films à l'affiche de tous les cinémas, s'exprime ici par :

```
SELECT Titre
FROM FILM F
WHERE NOT EXISTS
   (SELECT NumeroCinema
    FROM CINEMA C
    WHERE NOT EXISTS
       (SELECT A.Titre, A.NumeroCinema
        FROM AFFICHE A
        WHERE F.titre = A.Titre
        AND C.NumeroCinema = A.NumeroCinema))
```

Exercice 5.11 : La division
Décrire les requêtes suivantes, en proposant chaque fois deux solutions.

1. Les noms acteurs jouant dans tous les films
2. Les villes dans lesquelles tous les films sont à l'affiche.
3. Donner le nom des acteurs qui ont déjà joué dans tous les genres de films.

7.5 Les expressions et fonctions en SQL

Nous avons vu les fonctions d'aggrégation qui permettent de réaliser des calculs "verticaux" sur une relation. Les expressions et les fonctions que nous présentons ici permettent de réaliser des calculs *horizontaux*, c'est-à-dire des calculs ne faisant intervenir que les éléments d'un unique t-uplet.

Par exemple la requête :

```
SELECT COUNT(*)
FROM CINEMA
```

compte le nombre de cinémas, c'est-à-dire le nombre de t-uplets dans la relation `CINEMA`.

Supposons maintenant que chaque film soit noté par trois critiques et que ces notes soient stockées dans la relation `FILM` à l'aide des trois attributs `note1, note2` et `note3`. Nous obtenons la table FILM suivante :

TITRE	METTEURENSCENE	GENRE	NOTE1	NOTE2	NOTE3
Casper	Silberling	Comédie	8	10	8
Harcelement	Levinson	Comédie Dramatique	12	13	14
Junior	Reitman	Comédie	9	14	10
Alerte	Peterson	Aventure	13	15	13
Gazon Maudit	Balasko	Comédie	15	15	14
L'Effaceur	Russel	Action	9	13	12
Les Voleurs	Téchiné	Comédie dramatique	10	10	11
Ma saison préférée	Téchiné	Comédie dramatique	11	12	13

Figure 7.22 : Table FILM avec notes

Si nous voulons sélectionner les films *bien notés* nous pouvons par exemple sélectionner les films tels que la somme des trois notes soit supérieure ou égale à 40 :

```
(note1 + note2 + note3) >=40
```

Pour exprimer cette condition nous avons besoin de pouvoir écrire des expressions.

7.5.1 Les expressions

On peut décrire des expressions en utilisant des opérateurs tels que : `+, -, *, /` entre autres.

Les fonctions et les expressions peuvent être utilisées dans certaines clauses :

- `SELECT`
- `WHERE`
- `GROUP BY`
- `ORDER`

La liste des titres des films bien notés sera par exemple exprimée par la requête suivante :

```
SELECT Titre
FROM FILM
WHERE (note1 + note2 + note3) >= 40
```

Le calcul de la moyenne des films donne :

```
SELECT Titre,(note1 + note2 + note3) /3 AS Moyenne
FROM FILM
```

TITRE
Alerte
Gazon Maudit

Figure 7.23 : Films bien notés

TITRE	MOYENNE
Casper	8,666666666666666666666666666666666666667
Harcelement	13
Junior	11
Alerte	13,666666666666666666666666666666666666667
Gazon Maudit	14,666666666666666666666666666666666666667
L'Effaceur	11,333333333333333333333333333333333333333
Les Voleurs	10,333333333333333333333333333333333333333
Ma saison préférée	12

Figure 7.24 : Moyenne des films

Et enfin l'affichage des titres de film par ordre décroissant de *mérite* donne :

```
SELECT Titre
FROM FILM
ORDER BY (note1 + note2 + note3) DESC
```

TITRE
Gazon Maudit
Alerte
Harcelement
Ma saison préférée
L'Effaceur
Junior
Les Voleurs
Casper

Figure 7.25 : Film per ordre décroissant de mérite

Exercice 5.11 : Les expressions
Supposons que la table Film on ait placé les deux attributs suivants : NoteCritique, NotePublic qui sont tous deux des valeurs comprises entre 0 et 20 qui correspondent respectivement à la note attribuée par les critiques au film, et celle attribuée par le public. Décrire en SQL les requêtes suivantes:

1. Donner la liste des titres mieux notés par le public que par la critique.
2. Donner la liste des films qui ont une note moyenne supérieure à 12, avec leur moyenne.

TITRE	GENRE	METTEURENSCENE	NOTEPUBLIC	NOTECRITIQUE
Casper	Comédie	Silberling	10	8
Harcelement	Comédie Dramatique	Levinson	12	13
Junior	Comédie	Reitman	13	10
Alerte	Aventure	Peterson	13	14
Gazon Maudit	Comédie	Balasko	15	14
L'Effaceur	Action	Russel	12	9
Les Voleurs	Comédie dramatique	Téchiné	10	13
Ma saison préférée	Comédie dramatique	Téchiné	12	12

Figure 7.26 : Table FILM avec notes Public et Critique

7.5.2 Les différents types de fonctions

Il existe différents types de fonctions : Arithmétique, caractères, dates, et de conversion. Nous présentons dans ce qui suit les principales fonctions, en sachant bien évidemment que d'un SGBD à l'autre ces fonctions peuvent être très différentes.

7.5.2.1 Les fonctions arithmétiques

On peut trouver dans ces fonctions des fonctions mathématiques telles que la valeur absolue, puissance, plusgrand, ..., et des fonctions financières (calcul d'ammortissement, ...)
Oracle : Les fonctions numériques suivantes 7.2 sont disponibles

7.5.2.2 Les fonctions chaînes

Les fonctions de type chaine de caractères permettent de calculer la longueur d'un chaîne, de transformer en majuscule ou en minuscule une chaîne, d'extraire une sous-chaîne, ...
Oracle : Les fonctions chaîne suivantes 7.3 sont disponibles

7.5.2.3 Les fonctions dates

Il est possible avec ce type de valeur de construire comme avec les types vus précédemment des expressions. Nous avons les possibilités suivantes :

- Date + Nombre –> Date
- Date - Nombre –> Date
- Date - Date –> Nombre

De plus on dispose de fonctions retournant la date système, ... **Oracle :** Les fonctions date suivantes 7.4 sont disponibles

Fonction	Objectif
ABS(n)	Valeur absolue de n
ACOS(n)	Arc cosinus
ATAN(n)	Arc tangente de n
CEIL(n)	Plus petit entier supérieur ou égal à n
COS(n)	Cosinus de n
COSH(n)	Cosinus hyperbolique de n
EXP(n)	Exponentiel n
FLOOR(n)	plus grnad entier inférieur ou égal à n
LN(n)	Logarithme népérien de n
LOG(n)(m,n)	Logarithme de n en base m
MOD(m,n)	Division entière de m par n
POWER(m,n)	m à la puissance n
ROUND(m,n)	Arrondi de m à n décimales
SIGN(n)	Signe de n
SIN(n)	Sinus de n
SINH(n)	Sinus hyperbolique de n
SQRT(n)	Racine carrée de n
TAN(n)	Tangente de n
TANH(n)	Tangente hyperbolique de n
TRUNC(m,n)	Troncature de m à n décimales
WIDTH_BUCKET (exp, min, max,num)	Construction d'histogrammes exp : nbre ou date, min : limte inf., max : limite sup. num : nbre d'intervalles à construire

Table 7.2 : Fonctions numériques sous Oracle

7.5.2.4 Les fonctions de conversion

Ces fonctions permettent de passer d'un type à un autre, pour pouvoir effectuer des calculs ou pour permettre des affichages. **Oracle :** Les fonctions de conversion suivantes 7.5 sont disponibles

7.6 Les requêtes de mise à jour des données

Nous avons vu jusqu'à présent des requêtes permettant d'extraire de l'information d'une base de données, avec les requêtes `SELECT`. Les requêtes proposées ici permettent de supprimer, modifier et ajouter des données dans une base de données.

7.6.1 La suppression de données

La suppression de données est réalisée grâce à une requête `DELETE` qui a la syntaxe suivante:

```
DELETE FROM TABLE
WHERE Condition
```

Cette requête permet de supprimer de la table spécifiée, les t-uplets satisfaisant la condition.

La requête suivante, par exemple, permet de supprimer de la table Affiche tous les t-uplets faisant référence au film 'Alerte'.

```
DELETE FROM AFFICHE
```

Fonction	Objectif
ASCII(c)	Code ASCII équivalent
CHR(n)	Caractère correspondant dans le jeu de caractères
CONCAT(c1,c2)	Concaténation de c1 et c2
INITCAP(c)	Mot avec première lettre en majuscule
INSTR(c1, c2 [,p [,o]])	1er indice de la ss-chaine c2 dans c1 après le oième caractère
LOWER(c)	Mot en minuscule
LENGTH(c)	Longueur de la chaine
LPAD(c1, n, c2)	Insertion de la chaine c2 à gauche de c1 pour obtenir une chaine de longueur n
LTRIM(c1, c2)	Enlève la chaine c2 à la gauche de c1
REPLACE(c1, c2, c3)	Remplace la chaine c2 par c3 dans c1
RPAD((c1, n, c2)	Insertion de la chaine c2 à droite de c1 pour obtenir une chaine de longueur n
RTRIM(c1, c2)	Enlève la chaine c2 à la droite de c1
SUBSTR(c, n [, t])	Extrait de c la ss-chaine commençant à la position n de longueur t
TRANSLATE(c, de, vers)	Traduit chaque caractère de c avec le correspondant de 'de' dans 'vers'
TRIM(c1 FROM c2)	Enlève le caractère c1 dans c2
UPPER(c)	Mot en majuscule

Table 7.3 : Fonctions chaine sous Oracle

Fonction	Objectif
ADD_MONTHS(d, n)	Ajoute n mois à la date d
CURRENT_DATE	Date courante
EXTRACT(YEAR | MONTH |DAY HOUR | MINUTE | SECOND) FROM d |i	Extrait une partie de la date d ou de l'intervalle i
LAST_DAY(n)	Dernier jour du mois n
MONTHS_BETWEEN(d1, d2)	Nbre de mois entre les dates d1 et d2
NEW_TIME(d, z1, z2)	Date et heure de d donnée dans le fuseau horaire z1 en date dans le fuseau z2
NEXT_DAY(d, jour)	Date du prochain jour après d où *jour* est un jour de la semaine
ROUND(d, f)	Arrondi la date d selon le format f
SYSDATE	Date système
TRUNC(d, f)	Tronque la date dselon le format f

Table 7.4 : Fonctions Date sous Oracle

```
WHERE Titre = 'Alerte'
```

7.6.2 La modification de données

La modification de données est réalisée grâce à une requête `UPDATE` qui a la syntaxe suivante:

```
UPDATE TABLE
SET attribut1 = valeur1, ..., attributn = valeurn
WHERE Condition
```

Cette requête permet de modifier dans la table spécifiée, les t-uplets satisfaisant la condition.

Fonction	Objectif
BIN_TO_NUM(b1, b2, ...)	Convertit les bits en number
CAST(exp AS t)	Convertit exp dans le type t
CHARTOROWID(c)	Convertit c en `ROWID`
COMPOSE(c)	Convertit c en Unicode
CONVERT(c, d, s)	Convertit la chaine c du jeu de caractères s dans le jeu de caractères d
NUMTODSINTERVAL(n, i)	Convertit n en intervalle i de type DAY TO SECOND
NUMTOYMINTERVAL(n,i)	Convertit n en intervalle i de type YEAR TO MONTH
ROWIDTOCHAR(r)	Convertit le ROWID r en VARCHAR2
TO_CHAR(c)	Convertit la chaine c en VARCHAR2
TO_CHAR(d [,f])	Convertit la date d en VARCHAR2 par rapport à un format f
TO_CHAR(n [,f])	Convertit le numérique n en VARCHAR2
TO_DSINTERVAL(c [, f])	Convertit la chaine c en intervalle de type format f jour à seconde
TO_NUMBER(c)	Convertit c en number
TO_YMINTERVAL(c)	Convertit c en intervalle de temps d'années et mois
UNISTR(c)	Convertit la chaine c en Unicode

Table 7.5 : Fonctions de conversion sous Oracle

Cette modification ne porte que sur les attributs décrits dans la clause SET.
Supposons que le code postal de la ville de Besançon soit modifié, et qu'il passe de 25000 à 25001. Cette modification peut être réalisée dans la base de données CINEMAS grâce à la requête suivante :

```
UPDATE CINEMA
SET CodePostalCinema = 25001
WHERE VilleCinema = 'Besançon'
```

Il est possible d'utiliser dans la clause `WHERE` une sous-requête qui permet de définir une sélection plus fine de t-uplets à modifier.

7.6.3 L'ajout de données

L'ajout de données dans une table est réalisé grâce à une requête `INSERT` qui a la syntaxe suivante :

```
INSERT INTO TABLE(NomAttr1, NomAttr2, ...)
VALUES (val1, val2...)
```

Cette requête permet d'ajouter dans la table spécifiée un t-uplet dont les attributs ont les valeurs données dans la clause `VALUES`. Si certains attributs ne sont pas renseignés dans cette clause, leur valeur est mise à `NULL`. La liste des noms de colonne est optionnelle. Si elle est n'est pas mise, la liste des colonnes sera par défaut la liste de l'ensemble des colonnes de la table dans l'ordre de la création de la table.
Il est possible d'insérer dans une table plusieurs t-uplets provenant d'une autre table. La syntaxe est la suivante :

```
INSERT INTO TABLE(NomAttr1, NomAttr2, ...)
SELECT ...
```

La clause `SELECT` peut contenir n'importe quelle clause sauf une clause `ORDER BY`.

La requête suivante permet d'ajouter dans la table AFFICHE, le t-uplet décrivant que le film "les voleurs" est mis à l'affiche du cinéma N°1.

```
INSERT INTO AFFICHE
VALUES (1, 'Les Voleurs')
```

ou

```
INSERT INTO AFFICHE(NumeroCinema, Titre)
VALUES (1, 'Les Voleurs')
```

La requête ci-dessous permet d'ajouter dans une table ACTEURDECOMEDIE le nom des acteurs apparaissant dans la table DISTRIBUTION comme jouant dans une comédie.

```
INSERT INTO ACTEURDECOMEDIE
SELECT NomActeur FROM DISTRIBUTION
WHERE Titre IN (SELECT Titre FROM FILM WHERE Genre = 'Comédie')
```

Part IV

La normalisation

Chapter 8

La normalisation

Le problème de la conception d'une base de données est le suivant : *Etant donné un ensemble de données à représenter comment décider d'une structure logique adaptée à ces données ? Quels sont les modèles relationnels les plus adaptés pour éviter des problèmes de cohérence pouvant intervenir lors de la mise à jour de la base de données ?*

Dans ce chapitre nous présentons la normalisation qui est un outil pour répondre à ces questions. Attention cet outil ne se substitue pas à l'analyse, il permet plutôt à partir d'un modèle existant et des règles de gestion de ce modèle de dire si oui ou non le modèle relationnel proposé est correct et dans le cas ou celui-ci ne l'est pas de proposer des modifications.

Nous présentons dans ce chapitre

1. Pourquoi la normalisation ?
2. Les notions essentielles dans la théorie de la normalisation

8.1 Pourquoi la normalisation ?

Pourquoi dans une base de données proposons-nous plusieurs relations et pas une seule ? Une mauvaise conception d'un modèle relationnel peut entrainer des anomalies d'insertion, de suppression ou de modification.
Considérons l'exemple de la base de données CINEMAS, et supposons que les données soient stockées au moyen d'une unique relation de la forme :
`CINEMA(*NumeroCinema, NomCinema, AdresseCinema, VilleCinema, CodePostalCinema, TelephoneCinema, *Titre, MetteurEnScene, Genre, *NomActeur)`
On pourrait avoir par exemple la relation CINEMA suivante : Cette relation est naturellement soumise à certaines contraintes d'intégrité telles que :

- Le metteur en scène d'un film est toujours le même, ainsi que le genre du film.
- Un cinéma a toujours la même adresse,
- Un film a toujours les même acteurs dans sa distribution, ...

De plus on remarque des redondances d'informations telles que :

- (NumeroCinema, NomCinema, AdresseCinema, VilleCinema, CodePostalCinema)
- (Titre, MetteurEnScene, Genre)

Numero Cinema	Nom Cinema	Adresse Cinema	... ...	Titre Titre	... ...	NomActeur NomActeur
1	Vox	Grande Rue		Junior		Schwarzenegger
1	Vox	Grande Rue		Junior		De Mornay
1	Vox	Grande Rue		Junior		De Niro
1	Vox	Grande Rue		Alerte		Hoffman
1	Vox	Grande Rue		Alerte		Russo
1	Vox	Grande Rue		Alerte		Freeman
2	Plazza	Rue Des Granges		Alerte		Hoffman
2	Plazza	Rue Des Granges		Alerte		Russo
2	Plazza	Rue Des Granges		Alerte		Freeman
3	Eldorado	Rue St. Lambert		Alerte		Hoffman
3	Eldorado	Rue St. Lambert		Alerte		Russo
3	Eldorado	Rue St. Lambert		Alerte		Freeman
4	Pax	Avenue d'Italie		Alerte		Hoffman
4	Pax	Avenue d'Italie		Alerte		Russo
4	Pax	Avenue d'Italie		Alerte		Freeman
2	Plazza	Rue Des Granges		L'Effaceur		Schwarzenegger
2	Plazza	Rue Des Granges		Gazon Maudit		Balasko
2	Plazza	Rue Des Granges		Gazon Maudit		Abril
2	Plazza	Rue Des Granges		Gazon Maudit		Chabat
3	Eldorado	Rue St. Lambert		Casper		Pullman
3	Eldorado	Rue St. Lambert		Casper		Ricci
4	Pax	Avenue d'Italie		Casper		Pullman
4	Pax	Avenue d'Italie		Casper		Ricci
4	Pax	Avenue d'Italie		Harcelement		Moore
4	Pax	Avenue d'Italie		Harcelement		Douglas

Ces redondances d'informations engendrent immanquablement des difficultés de mise à jour, qui peuvent se produire lors de l'ajout de nouvelle données, de la suppression ou la modification de données.

8.1.1 Anomalie d'insertion

La relation proposée ne permet pas de décrire un film qui n'est à l'affiche d'aucun cinéma. Ou alors il faudrait autoriser la présence de la valeur `NULL` dans des attributs tels que `NumeroCinema`. On pourrait avoir par exemple les t-uplets suivant dans notre relation `CINEMA`.

Numero Cinema	Nom Cinema	Adresse Cinema	... ...	Titre	... ...	NomActeur
Null	Null	Null	...	Les Voleurs		??
8	Vox	Rue des alouettes		Null		Null
Null	Null	Null		Null		Acteur inconnu

8.1.2 Anomalie de suppression

La disparition d'un film qui est le seul à l'affiche d'un cinéma donné aura pour effet la perte des informations concernant ce cinéma.

Par exemple si l'on retire de l'affiche les films Junior et Alerte, nous n'aurons plus aucune information concernant la cinéma numéro 1.

8.1.3 Anomalie de modification

Toute modification sur un cinéma doit être répercutée sur tous les titres et les acteurs liés à ce cinéma. Ce qui risque d'entrainer un temps de traitement très long.

Si le cinéma *numéro 1*, appelé *Vox* change de nom pour devenir le *nouveau Vox*, il va falloir modifier cette information dans la table `CINEMA` partout où elle apparait. Ici nous aurons 6 t-uplets à modifier, alors que logiquement une telle information ne devrait être stockée qu'une fois.

8.1.4 La normalisation des relations

Pour définir des modèles relationnels, dont les tables ne posent pas de problèmes tels que ceux présentés ci-dessus, on utilise une approche par décomposition. En partant d'une relation unique (comme la relation cinema présentée ci-dessus), appelée ***relation universelle***, on décompose petit à petit la relation en sous-relations ne posant pas d'anomalies.

Le but de la normalisation est de :

- Supprimer tout risque de redondance d'informations,
- Réaliser une décomposition des informations sans perte d'informations. Les décompositions doivent être réversibles.

Nous précisons ici les deux notions introduites dans la présentation de la normalisation.

8.1.4.1 La relation universelle

Une **relation universelle** est une relation unique dont l'ensemble des attributs est défini par union de tous les attributs des relations de la base de données.

Dans le cas de la base de données cinema, nous avons défini notre relation universelle, en prenant tous les attributs des tables : `AFFICHE, CINEMA, FILM`, et `DISTRIBUTION`.
Pour définir cette relation, il faut bien évidemment que deux attributs représentant le même concept portent le même nom.

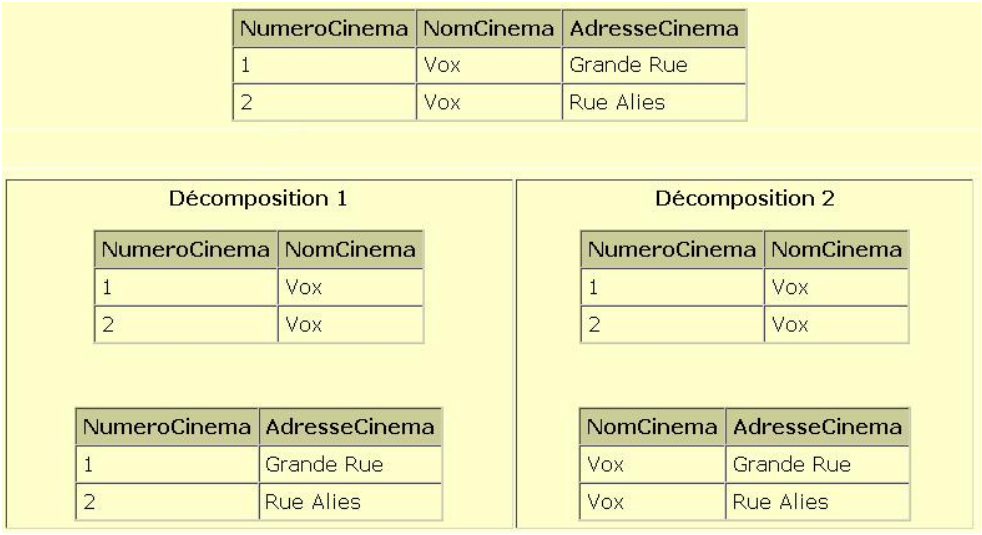
8.1.4.2 La décomposition

La décomposition des relations est basée sur l'utilisation de la projection et de la jointure.

La `décomposition` d'une relation est le remplacement d'une relation `R(Att1, Att2,..., Attn)` par un ensemble de relations `R1, R2, ..., Rp` obtenu par des projections de `R` sur des sous-ensembles d'attributs dont l'union contient l'ensemble des attributs de `R`.

Considérons la relation `R` ci-dessous, nous pouvons proposer les deux décompositions suivantes :
Si nous observons ces deux décompositions que pouvons-nous remarquer ?

- **Décomposition 1 :** Cette décomposition n'entraîne aucune perte d'information. Les deux relations nous permettent de retrouver à partir de n'importe quel numéro de cinéma, le nom et l'adresse correspondants.
- **Décomposition 2 :** Ici nous pourrons donner le nom d'un cinéma à partir d'un numéro, mais nous serons incapable de donner l'adresse correspondante. Nous avons perdu de



NumeroCinema	NomCinema	AdresseCinema
1	Vox	Grande Rue
2	Vox	Rue Alies

Décomposition 1

NumeroCinema	NomCinema
1	Vox
2	Vox

NumeroCinema	AdresseCinema
1	Grande Rue
2	Rue Alies

Décomposition 2

NumeroCinema	NomCinema
1	Vox
2	Vox

NomCinema	AdresseCinema
Vox	Grande Rue
Vox	Rue Alies

Figure 8.1 : Exemple de décomposition

l'information.

Il paraît immédiat que toute décomposition réalisée doit être sans perte d'informations.

8.1.4.3 Décomposition sans perte d'informations

La **décomposition sans perte d'informations** d'une relation `R` est la décomposition d'une relation `R(Att1, Att2,..., Attn)` en un ensemble de relations `R1, R2, ..., Rp` tel que `R` est égal à la jointure des relations `R1, ..., Rp`.

Dans le cas de la décomposition 1, la jointure de deux relations obtenues par rapport à l'attribut `NumeroCinema` permet de réobtenir la relation de départ. Par contre dans le deuxième cas la jointure des deux tables par rapport à l'attribut commun nous donne la table suivante : Le problème

NumeroCinema	NomCinema	AdresseCinema
1	Vox	Grande Rue
1	Vox	Rue Alies
2	Vox	Grande Rue
2	Vox	Rue Alies

Figure 8.2 : Décomposition avec perte d'informations

général de la conception d'un modèle peut donc être vu comme celui d'une décomposition sans perte d'informations d'une relation universelle, de manière à obtenir un ensemble de sous-relations ne présentant pas d'anomalies d'insertion, de suppression ou de modification.

Dans un modèle relationnel normalisé, on a l'habitude de définir des relations dont nous dirons qu'elles sont en 3NF (forme normale qui sera présentée plus loin dans ce chapitre).

8.2 Les notions essentielles pour la théorie de la normalisation

L'idée de base de la normalisation est de pouvoir caractériser les relations pouvant être décomposées sans perte d'informations. La notion de dépendance fonctionnelle permet de "capturer" l'idée de dépendances entre informations.

8.2.1 Les dépendances Fonctionnelles

Cette notion essentielle dans la théorie de la normalisation consiste en une association plusieurs-vers-un entre deux ensembles d'attributs à l'intérieure d'une relation donnée.
Les dépendances fonctionnelles sont des notions sémantiques. Rechercher les dépendances fonctionnelles fait partie du processus de compréhension de la signification des informations.

> Soit `R` une relation, et soient `X` et `Y` deux sous-ensembles quelconques de l'ensemble des attributs de `R`. On dit que `Y` est en dépendance fonctionnelle avec `X` dénoté :
> `X` $\rightarrow$ `Y`
> Si et seulement si à chaque valeur de `X` dans `R` correspond une seule valeur de `Y` dans `R`.

On appelle **déterminant** le membre gauche d'une dépendance fonctionnelle.
Une dépendance fonctionnelle est différente d'une relation au sens mathématique, car elle peut varier avec le temps.
Si `X` est une clé candidate alors tous les attributs de la relation sont en dépendance fonctionnelle avec `X`. Ce qui est logique, car si l'on pouvait avoir des valeurs différentes pour un attribut `Y` pour une valeur de la clé `X`, alors `X` ne serait plus une clé.

Application au Cinéma
Considérons notre relation `CINEMA` présentée dans l'introduction à la normalisation. Nous avons par exemple les deux dépendances fonctionnelles suivantes :

- `NumeroCinema` $\rightarrow$ `NomCinema`
- `NumeroCinema` $\rightarrow$ `AdresseCinema, VilleCinema, CodePostalCinema`

Ce qui signifie que le numéro d'un cinéma défini un et un seul nom de cinéma, ainsi qu'une seule adresse. Par contre un même nom de cinéma peut appartenir à plusieurs cinémas, et donc correspondre à plusieurs numéros de cinéma.

8.2.2 Les dépendances fonctionnelles totales ou DFT

On parle aussi de dépendance irréductible, que l'on note DFT.

> Soit R une relation, et soient X et Y deux sous-ensembles quelconques de l'ensemble des attributs de R. On dit que Y est en **dépendance fonctionnelle totale** avec X dénoté :
> X DFT $\rightarrow$ Y
> si X $\rightarrow$ Y et aucun sous-ensemble d'attributs de X ne défini fonctionnellement Y.

X est une clé candidate alors X est une super-clé irréductible.
Application au Cinéma : Considérons notre relation `CINEMA` présentée dans l'introduction à la normalisation. Nous avons par exemple la dépendance fonctionnelle suivante :
`NumeroCinema, Titre` $\rightarrow$ `NomCinema`
Cette dépendance n'est pas totale car `NomCinema` est en DF avec une partie du déterminant, à savoir `NumeroCinema`. Par contre `NumeroCinema DFT` $\rightarrow$ `NomCinema.`

Une façon pratique de manipuler ces 'relations' est d'utiliser des schémas comme ceux ci-dessous. On voit ici que `NomCinema` qui est défini fonctionnellement par `(NumeroCinema, Titre)` l'est

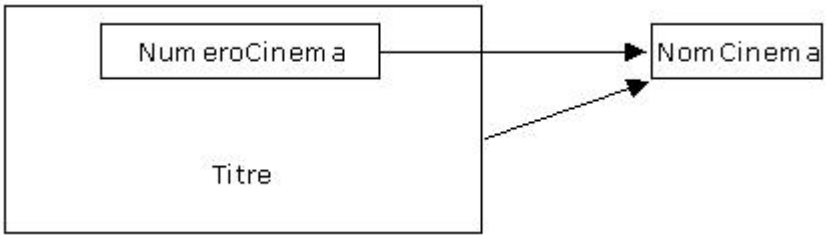


Figure 8.3 : Représentation des DFs

aussi par une partie de cet ensemble d'attributs. Ce qui signifie que l'on a pas de dépendance fonctionnelle totale liant `(NumeroCinema, Titre)` et `(NomCinema)`

Exercice 7.1 :

1. Donner l'ensemble des dépendances fonctionnelles de la relation `CINEMA`.
2. Dire si les dépendances obtenues sont totales ou pas.

8.2.3 Comment définir l'ensemble des DFs?

Certaines dépendances fonctionnelles en impliquent d'autres. Par exemple la dépendance fonctionnelle :

`NumeroCinema` → `NomCinema, AdresseCinema`

implique les deux dépendances fonctionnelles suivantes :
`NumeroCinema` → `NomCinema`
`NumeroCinema` → `AdresseCinema`

Ou si nous prenons l'exemple suivant :
`NumeroCinema` → `VilleCinema et VilleCinema` → `CodePostalCinema`
Le numero de cinéma défini alors un seul code postal, d'où la dépendance fonctionnelle transitive :
`NumeroCinema` → `CodePostalCinema`
Le problème est donc de savoir définir complètement un ensemble de dépendances fonctionnelles.

8.2.3.1 Fermeture d'un ensemble de dépendances fonctionnelles

L'ensemble de toutes les dépendances fonctionnelles qui sont impliquées par un ensemble `S` de dépendances fonctionnelles est appelé **fermeture** de `S`, et est dénoté `S+`.

Pour déterminer la fermeture d'un ensemble des dépendances fonctionnelles (DFs), on dispose des **axiomes d'Amstrong**, qui permettent de déduire nouvelles DFs de l'ensemble de départ.

8.2.3.2 Les règles d'inférences d'Amstrong

Ces règles permettent à partir d'un ensemble de dépendances fonctionnelles d'en dériver de nouvelles.

Soient `A, B`, et `C`, trois sous-ensembles d'attributs de la relation `R`

- **Réflexivité :** Si `B` est un sous-ensemble de `A` alors `A` → `B`.
- **Augmentation :** Si `A` → `B` alors `A, C` → `B, C`
- **Transitivité** : Si `A` → `B` et `B` → `C` alors `A` → `C`

Intuitivement, nous retrouvons le fait que : Tout ensemble d'attributs se définit lui-même, et définit toute partie de lui-même. On peut augmenter les deux ensembles d'attributs apparaissant dans la dépendance, par un troisième.

La fermeture d'un ensemble S de dépendances fonctionnelles peut être obtenue par application des règles d'inférence d'Amstrong. (On dit que cet ensemble de règles est complet). D'autres règles peuvent être proposées, mais elles sont dérivées de ces trois premières.
Application au Cinéma : Considérons notre relation `CINEMA` précédente. Nous avons la dépendance fonctionnelle suivante :

`NumeroCinema` → `AdresseCinema, VilleCinema, CodePostalCinema`

Qui peut être décomposée en trois DFs en utilisant la réflexivité :
`NumeroCinema` → `AdresseCinema`
`NumeroCinema` → `VilleCinema`
`NumeroCinema` → `CodePostalCinema`
Par augmentation nous obtenons :
`NumeroCinema, Titre` → `AdresseCinema, Titre`

Par transitivité :
`NumeroCinema` → `VilleCinema` → `CodePostalCinema`

d'où :
`NumeroCinema` → `CodePostalCinema`

8.2.3.3 Calcul de la fermeture d'un ensemble d'attributs

Nous avons parlé de la fermeture d'un ensemble de dépendances fonctionnelles, nous présentons ici la notion de fermeture d'un ensemble d'attributs X par rapport à un ensemble de DFs, S.

Soient `X` un ensemble d'attributs d'une relation `R`, et `S` l'ensemble des DFs concernant cette relation. On appelle **fermeture de X pour S**, l'ensemble `X+` des attributs qui dépendent fonctionnellement de `X`.

Cette définition de la fermeture d'un ensemble d'attributs pour un ensemble de DFs, permet de caractériser les super-clés d'une relation.

Soit `X` un ensemble d'attributs d'une relation `R`, `X` est une **super-clé** pour `R`, si `X` contient au moins une clé candidate de `R`.

Une super-clé X d'une relation R, est telle que tout sous-ensemble d'attributs Y de R satisfait la dépendance fonctionnelle : X –> Y. Ainsi si l'on définit la fermeture de X pour R, on obtient l'ensemble des attributs de R.

Une **clé candidate** d'une relation `R`, est une super-clé irréductible de la relation `R`.

Soient R une relation, dont l'ensemble d'attributs est T, K un sous-ensemble d'attributs de T, et S l'ensemble des DFs, pour déterminer la fermeture K+ de l'ensemble d'attributs K pour S, on applique l'algorithme suivant :

```
K+ := K
Changé <-- VRAI
Tant Que Changé ET il existe au moins une DF non marquée Faire
```

```
 Changé <-- FAUX
 Pour Chaque DF : X-->Y de S Faire
  Si X appartient à K+ alors
  K+ := K+ Union Y
   Marquer la DF "Utilisée"
   Changé <-- VRAI
  Finsi
  FinPour
FinTantQue
```

Application au Cinéma : Utilisons l'algorithme proposé ci-dessus pour montrer que l'ensemble d'attributs (`NumeroCinema, Titre, NomActeur`) est une super-clé de la relation `CINEMA`. Nous avons défini les dépendances fonctionnelles suivantes :

- DF1 : `NumeroCinema` → `NomCinema, AdresseCinema, CodePostalCinema, TelephoneCinema`
- DF2 : `CodePostalCinema` → `VilleCinema`
- DF3 : `Titre` → `MetteurEnScene, Genre`

Au départ nous avons : `K+ = NumeroCinema, Titre, NomActeur`

Premier Passage

DF1 : `NumeroCinema` ∈ K+, donc nous ajoutons à K+, les attributs : `NomCinema, AdresseCinema, CodePostalCinema, TelephoneCinema` (partie droite de la DF).
`K+ = NumeroCinema, Titre, NomActeur, NomCinema, AdresseCinema, CodePostalCinema, TelephoneCinema`

DF2 : `CodePostalCinema` ∈ K+, donc nous ajoutons à K+, l'attribut : `VilleCinema`.
`K+ = NumeroCinema, Titre, NomActeur, NomCinema, AdresseCinema, CodePostalCinema, TelephoneCinema, VilleCinema`

DF3 : `Titre` ∈ K+, donc nous ajoutons à K+, les attributs : `MetteurEnScene, Genre`.
`K+ = NumeroCinema, Titre, NomActeur, NomCinema, AdresseCinema, CodePostalCinema, TelephoneCinema, VilleCinema, MetteurEnScene, Genre`

K+ a été modifié durant ce premier passage, et toutes les DFs ont été marquées. K+ représente donc l'ensemble de tous les attributs de la relation `CINEMA` en dépendance avec les attributs `K+ = NumeroCinema, Titre, NomActeur`.

L'ensemble K+ obtenu contient tous les attributs de la relation `CINEMA`, `(NumeroCinema, Titre, NomActeur)` est donc une super-clé de la relation `CINEMA`.
De plus cet ensemble d'attributs est irréductible. Si l'on retire un des attributs alors la fermeture obtenue ne contient plus tous les attributs de la relation `CINEMA`. `(NumeroCinema, Titre, NomActeur)` est une clé candidate de la relation `CINEMA`.

Exercice 7.2 : 1 - Soit `R` une relation contenant les attributs suivants `(A, B, C, D, E, F)` avec les dépendances fonctionnelles suivantes :

- `DF1 : A` $\rightarrow$ `B, C`
- `DF2 : E` $\rightarrow$ `C, F`
- `DF3 : B` $\rightarrow$ `E`
- `DF4 : C, D` $\rightarrow$ `E, F`

En utilisant l'algorithme pour déterminer la fermeture d'un ensemble d'attributs pour un ensemble de DFs déterminer l'ensemble `A+` des attributs qui dépendent fonctionnellement des attributs `A, B`. Et en déduire si cet ensemble d'attributs est une clé candidate de la relation `R`.

8.2.4 Le théorème de décomposition

Le principe de base de la normalisation d'une relation, est de réaliser une décomposition de l'information sans perte d'information. Reprenons la relation `CINE` suivante :

NumeroCinema	**NomCinema**	**AdresseCinema**
1	Vox	Grande Rue
2	Vox	Rue Alies

Table 8.1 : Relation CINE

Nous avons vu que les deux décompositions ci-dessous ne donnent pas les mêmes "résultats". En particulier la deuxième nous fait perdre l'information : quelle est l'adresse du cinéma N° 1?

NumeroCinema	NomCinema
1	Vox
2	Vox

NumeroCinema	AdresseCinema
1	Grande Rue
2	Rue Alies

Table 8.2 : Décomposition 1

NumeroCinema	NomCinema
1	Vox
2	Vox

NomCinema	AdresseCinema
Vox	Grande Rue
Vox	Rue Alies

Table 8.3 : Décomposition 2

Dans les deux cas la décomposition a consisté en la réalisation de deux projections à partir de la relation de départ. Dans le premier cas la jointure de ces deux projections redonne la relation initiale, pas dans le deuxième. Nous pouvons donc considérer que l'**opérateur de décomposition** est la **projection** et celui de **recomposition** la **jointure**.

Maintenant il est intéressant de comprendre pourquoi la première décomposition permet de ne pas perdre d'informations alors que la deuxième ne le permet pas. Reprenons les DFs présentées précédemment, nous avons :
`NumeroCinema` $\rightarrow$ `NomCinema`
`NumeroCinema` $\rightarrow$ `AdresseCinema`

Nous pouvons remarquer que la décomposition correcte, et celle qui laisse dans une même relation les attributs liés par des dépendances fonctionnelles. Ce qui nous amène au théorème de décomposition.

8.2.4.1 Le théorème de décomposition

Soit `R` une relation d'ensemble d'attributs `T` et `(X, Y, Z)` une partition de `T` avec `X` → `Y`.
`R` peut être décomposée en deux relations `R1` et `R2` telles que :
`R1 = [X, Y] R`
`R2 = [X, Z] R`
et alors `R = R1 [ R1. X = R2.X] R2`

Ce que nous pouvons schématiser de la façon suivante :

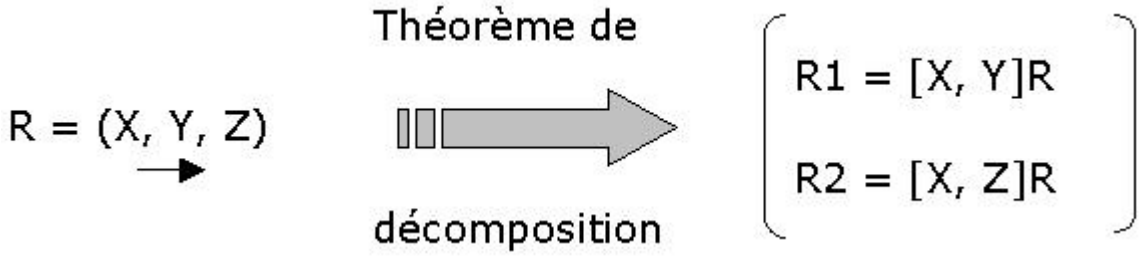


Figure 8.4 : Théorème de décomposition

Remarque : `(X1, X2, ..., Xn)` est une partition de `X` signifie :
- les `Xi` sont disjoints 2 à 2,
- la réunion de tous les `Xi` donne `X`.

Exercice 7.3 : Le théorème de décomposition En utilisant le théorème de décomposition et les deux dépendances fonctionnelles proposées ci-dessous, décomposer la relation `CINEMA` sans perte d'information.

DF1 : `NumeroCinema` → `NomCinema, AdresseCinema, VilleCinema, CodePostalCinema, TelephoneCinema`
DF2 : `Titre` → `MetteurEnScene, Genre`

Chapter 9

Les formes normales

La normalisation est réalisée par réductions successives d'un ensemble de relations en une forme plus satisfaisante. Les réductions sont toujours réversibles. Cette procédure est construite autour du concept de **forme normale**.
Nous présentons dans ce chapitre

1. La notion de forme normale
2. Les formes normales issues de la notion de dépendance fonctionnelle
3. Les 4° et 5° formes normales.

9.1 Qu'est-ce qu'une forme normale ?

Une relation est dans une **forme normale** particulière si elle satisfait un ensemble de conditions prédéfinies.

On distingue différentes formes normales dont les principales sont représentées dans la figure ci-dessous.

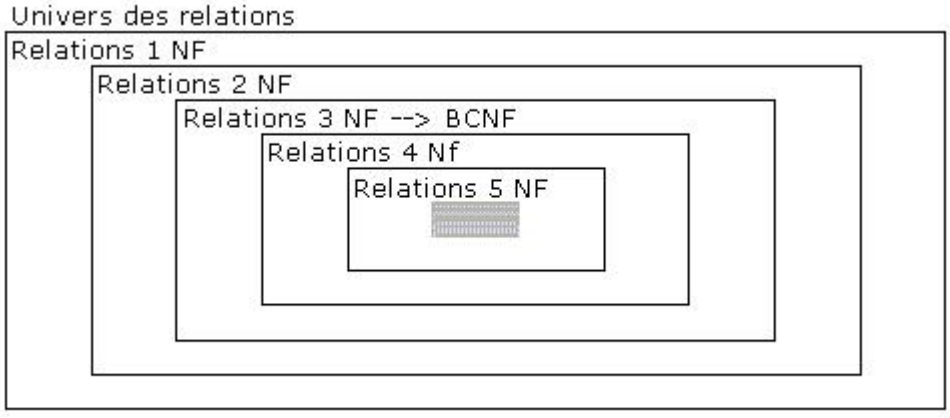


Figure 9.1 : Les formes normales

Nous présentons dans ce qui suit ces différentes formes normales, ainsi que les règles qui permettent de transformer un ensemble de relations non *'normalisées'* en un ensemble de relations normalisées.

9.2 La première forme normale - 1NF

Les trois premières formes normales ont été introduites par Codd. Elles ont pour rôle de permettre la décomposition d'un ensemble de relations sans perte d'information, en utilisant les dépendances fonctionnelles.

Une relation est en **première forme normale**, notée 1NF, si : Tous ses attributs sont atomiques.

Ceci signifie que pour qu'une relation soit en 1NF aucun de ses attributs ne doit être lui-même une relation. Considérons par exemple la relation `FILM1` suivante :

Titre	NotesCritiques	...
Alerte	10 10.5 12	
Casper	8 12 11	

Table 9.1 : FILM1

Cette relation n'est pas en 1NF, car l'attribut `NotesCritiques` est "décomposable". Pour obtenir une relation 1NF, plusieurs solutions sont possibles : Si le nombre de notes est connu et fixe on peut alors proposer de définir autant d'attributs que de notes ce qui nous donne par exemple la relation suivante 9.2 .

Titre	Note1Critique	Note2Critique	Note3Critique	...
Alerte	10	10.5	12	
Casper	8	12	11	

Table 9.2 : Relation 1NF

Dans le cas où le nombre de notes n'est pas connu on peut proposer la relation 9.3 , où chaque t-uplet permet de décrire une note :

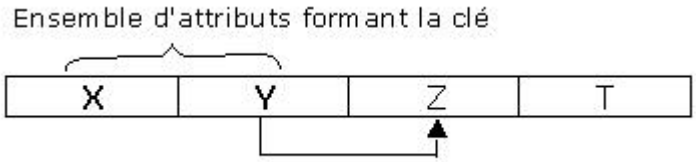
Titre	NoteCritique	...
Alerte	10	
Alerte	10.5	
Alerte	12	
Casper	12	
Casper	8	
Casper	11	

Table 9.3 : Relation 1NF

9.3 La deuxième forme normale - 2NF

Une relation est en **deuxième forme normale**, notée 2NF, si et seulement si :
- elle est en première forme normale,
- tout attribut n'appartenant pas à la clé est en dépendance fonctionnelle totale avec la clé

Cette forme normale concerne les relations où la clé n'est pas réduite à un seul attribut. Une relation non 2NF peut être schématisée de la façon suivante 9.3 :
On a ici une partie de la clé qui définit fonctionnellement une partie du reste des attributs.



9.3.1 Transformation en 2NF

Le théorème de décomposition permet de passer à une relation 2NF de la façon suivante 9.3.1 :
Ce que nous pouvons représenter de la façon suivante 9.3.1.

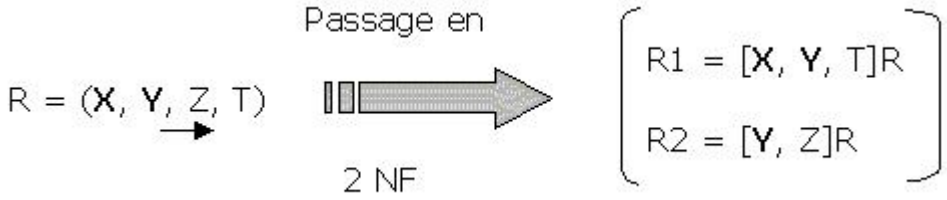
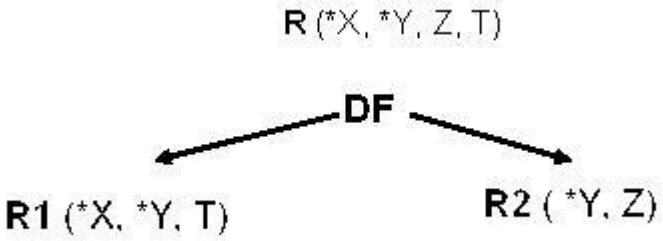


Figure 9.2 : Théorème de décomposition et 2NF



9.3.2 Application au Cinéma

La relation `CINEMA (*NumeroCinema, NomCinema, AdresseCinema, VilleCinema, CodePostalCinema, TelephoneCinema, *Titre, MetteurEnScene, Genre, *NomActeur)`
posssède les dépendances fonctionnelles suivantes :

- DF1 : `NumeroCinema` → `NomCinema, AdresseCinema, CodePostalCinema, TelephoneCinema`
- DF2 : `CodePostalCinema` → `VilleCinema`
- DF3 : `Titre` → `MetteurEnScene, Genre`

Les DFs DF1 et DF3 font que notre relation n'est pas 2NF. Utilisons le théorème de décomposition pour transformer cette relation en un ensemble de relations 2NF.

DF1 nous donne les relations suivantes :

`CINEMA1 (*NumeroCinema, *Titre, MetteurEnScene, Genre, *NomActeur)` et
`CINEMA (*NumeroCinema, NomCinema, AdresseCinema, VilleCinema, CodePostalCinema,`

```
TelephoneCinema)
```

Nous transformons enfin la relation `CINEMA1` en tenant compte de DF3, ce qui nous donne :

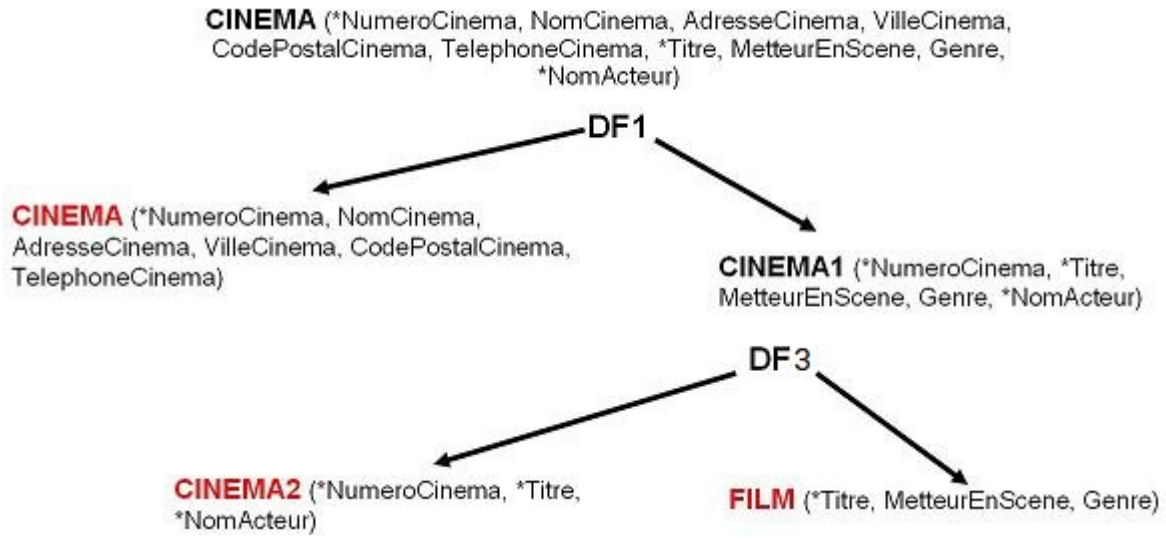
```
CINEMA2 (*NumeroCinema, *Titre, *NomActeur)
FILM (*Titre, MetteurEnScene, Genre)
```

Nous obtenons ainsi les relations suivantes :

```
CINEMA (*NumeroCinema, NomCinema, AdresseCinema, VilleCinema,
CodePostalCinema, TelephoneCinema)
FILM (*Titre, MetteurEnScene, Genre)
CINEMA2 (*NumeroCinema, *Titre, *NomActeur)
```

qui sont 2NF.

Nous pouvons représenter cette décomposition de la façon suivante 9.3.2 où les relations représentées en rouge sont 2NF.



Exercice 7.4 : La forme 2NF 1 - Les prêts de K7
Considérons la relation suivante :
`PRETK7(*NumCli, NomCli, AddrCli, *NumK7, TitreK7, *DatePret)`

qui permet de gérer les prêts de cassettes aux clients. Les DFs suivantes ont été mises à jour :
DF1 : `NumCli` → `NomCli, AddrCli`
DF2 : `NumK7` → `TitreK7`

Montrer que cette relation n'est pas 2NF, et la transformer en un ensemble de relations 2NF.

2 - Etude de relations non 2NF
Considérons la relation `CINEMA1` suivante :
`CINEMA1 (*NumeroCinema,*Titre, MetteurEnScene, Genre, *NomActeur)`

Avec le contenu 9.4 .
Quels problèmes se posent lorsque l'on supprime par exemple tous les t-uplets de la relation `CINEMA1` faisant référence au titre *Alerte*?
Que se passe-t-il si on modifie un t-uplet de la table `CINEMA1` en faisant passer son Metteur en scène de *Peterson* à *Peetrson* ?

NumeroCinema	Titre	Metteur	Genre	NomActeur
1	Junior	Reitman	Comedie	Schwarzenegger
1	Junior	Reitman	Comedie	De Mornay
1	Junior	Reitman	Comedie	De Niro
1	Alerte	Peterson	Aventure	Hoffman
1	Alerte	Peterson	Aventure	Russo
1	Alerte	Peterson	Aventure	Freeman
2	Alerte	Peterson	Aventure	Hoffman
2	Alerte	Peterson	Aventure	Russo
2	Alerte	Peterson	Aventure	Freeman
3	Alerte	Peterson	Aventure	Hoffman
3	Alerte	Peterson	Aventure	Russo
3	Alerte	Peterson	Aventure	Freeman

Table 9.4 : Relation CINEMA1

9.4 La troisième forme normale - 3NF

Une relation est en **troisième forme normale**, notée 3NF, si et seulement si :
- Elle est en deuxième forme normale,
- Tout attribut n'appartenant pas à la clé, ne dépend pas d'un attribut non clé.

La définition d'une relation 3NF permet d'éliminer les redondances dues aux dépendances transitives. Une relation non 3NF peut être schématisée de la façon suivante 9.3 :

9.4.1 Transformation en 3NF

Le théorème de décomposition permet de passer à une relation 3NF de la façon suivante 9.4

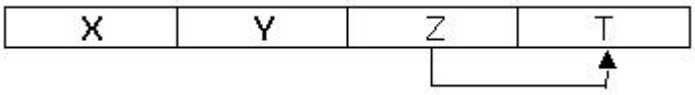


Figure 9.3 : Forme non 3NF

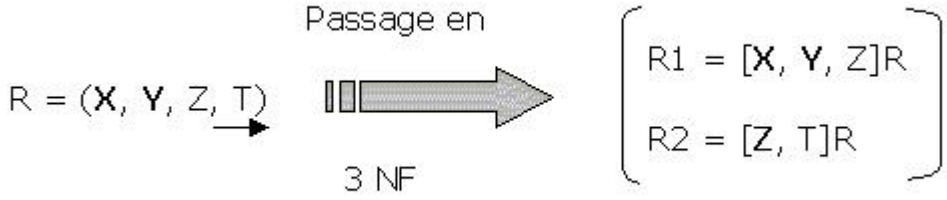


Figure 9.4 : Théorème de décomposition et 3NF

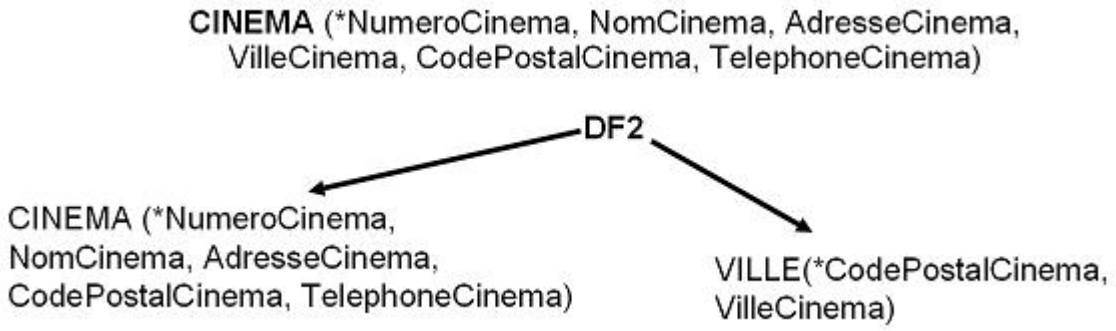
Nous décomposons la relation non 3NF en deux relations. La première contient tous les attributs de la relation de départ, sauf ceux apparaissant en partie droite de la DF. La deuxième relation contient tous les attributs apparaissant dans la DF impliquée par la transformation, et les attributs déterminants de la DF constituent la clé de cette nouvelle relation.

9.4.2 Application au Cinéma

L'étape précédente de normalisation nous a conduit à l'ensemble de relations suivant :
`CINEMA (*NumeroCinema, NomCinema, AdresseCinema, VilleCinema, CodePostalCinema, TelephoneCinema)`
`FILM (*Titre, MetteurEnScene, Genre)`
`CINEMA2 (*NumeroCinema, *Titre, *NomActeur)`

La dépendance DF2 : `CodePostalCinema` → `VilleCinema`
implique que la relation `CINEMA` n'est pas 3NF. D'où la transformation suivante :
`CINEMA (*NumeroCinema, NomCinema, AdresseCinema, CodePostalCinema, TelephoneCinema)`
`VILLE(*CodePostalCinema, VilleCinema)`

que nous pouvons schématiser par :
Nous obtenons donc l'ensemble de relations 3NF suivantes :



```
CINEMA (*NumeroCinema, NomCinema, AdresseCinema, CodePostalCinema,
        TelephoneCinema)
VILLE(*CodePostalCinema, VilleCinema)
FILM (*Titre, MetteurEnScene, Genre)
CINEMA2 (*NumeroCinema, *Titre, *NomActeur)
```

Dans la relation `CINEMA` de départ, le fait d'avoir cette dépendance fonctionnelle entre `CodePostalCinema` et `VilleCinema`, pose quelques problèmes :

- On ne peut pas décrire le nom de ville correspondant à un code postal tant que ce code n'apparait pas dans la relation `CINEMA`.
- S'il y a un seul t-uplet ayant comme code postal *75000*, par exemple, la suppression de ce t-uplet nous fera perdre l'information correspondant à ce code postal (quelle est la ville correspondant au code postal *75000*?)
- Si la ville de *Saint-Marin* devient *SanMarino* comment va-t-on mettre à jour cette information dans la relation `CINEMA` ?

Exercice 7.5 : Les voitures

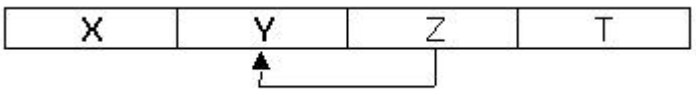
Considérons la relation suivante :
`VOITURE(*Immatriculation, Marque, Modele, Puissance, Couleur)`
Avec la DF suivante :
DF1 : `Modele` → `Marque, Puissance`

Montrer que cette relation n'est pas 3NF, et la transformer en un ensemble de relations 3NF.

9.5 La forme normale de Boyce-Codd- BCNF

Une relation est en **forme normale de Boyce-Codd**, notée **BCNF**, si et seulement si :
- Elle est en troisième forme normale,
- Chaque dépendance fonctionnelle irréductible a une clé candidate dans son déterminant.

Une relation non BCNF peut être schématisée de la façon suivante 9.5 :

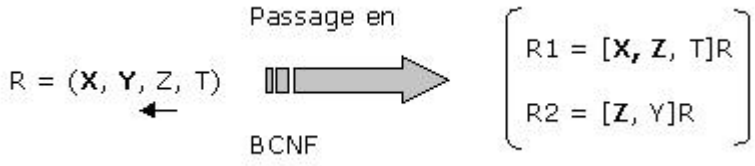


Ce problème peut se poser lorsque l'on a à faire à une relation qui a au moins deux clés candidates telles que :

- elles sont composées
- elles se chevauchent (au moins un attribut commun).

9.5.1 Transformation en BCNF

Le théorème de décomposition permet de passer à une relation BCNF de la façon suivante :



La décomposition permet de créer deux relations, la première contient tous les attributs de la relation de départ, sauf ceux apparaissant en partie droite de la DF, de plus la clé de la relation est modifiée. La deuxième relation,comme dans le cas des formes 3NF, contient tous les attributs apparaissant dans la DF impliquée par la transformation, et les attributs déterminants de la DF constituent la clé de cette nouvelle relation.

9.5.2 Application au Cinéma

L'étape précédente de normalisation nous a conduit à l'ensemble de relations suivant :

```
CINEMA (*NumeroCinema, NomCinema, AdresseCinema, CodePostalCinema,
        TelephoneCinema)
VILLE(*CodePostalCinema, VilleCinema)
FILM (*Titre, MetteurEnScene, Genre)
CINEMA2 (*NumeroCinema, *Titre, *NomActeur)
```

Dans le cas présent nous avons la dépendance fonctionnelle suivante dans la relation `CINEMA`: `TelephoneCinema` → `NumeroCinema`

Si nous appliquons la transformation proposée nous obtenons :

`CINEMA(*TelephoneCinema, NomCinema, AdresseCinema, CodePostalCinema)`
`NUMCINEMA(*TelephoneCinema, NumeroCinema)`
et la relation `NUMCINEMA` pose le même problème que la relation `CINEMA` précédente. La décomposition proposée n'est donc pas satisfaisante.
En fait nous n'avons pas étudié correctement les hypothèses de la décomposition d'une relation BCNF. Un ensemble d'attributs non clé doit définir fonctionnellement une partie de la clé, pour qu'il y ait un problème. Ce qui n'est pas le cas ici.

Exercice 7.6 : Les crus
Considérons la relation 9.5 : `LESCRUS(*Cru, *Pays, Region, Qualite)`
Avec les DFs suivantes :

- DF1 : `Cru, Pays` → `Region`
- DF2 : `Cru, Pays` → `Qualite`
- DF3 : `Region` → `Pays`

Montrer que cette relation n'est pas BCNF, et la transformer en un ensemble de relations BCNF.

---

* Cru	* Pays	Region	Qualite
Pinot Gris	France	Alsace	Bien
Pinot Gris	USA	Californie	Bien
Chablis	France	Beaujolais	Excellent

Table 9.5 : relation LESCRUS

9.6 Les 4e et 5e formes normales

Nous avons vu les premières formes normales qui dépendent toutes de la notion de dépendance fonctionnelle. Les 4e et 5e formes normales sont définies à partir de dépendances plus générales que la dépendance fonctionnelle :

- Les dépendances multivaluées
- les dépendances de jointure

9.6.1 Les dépendances multivaluées

Certaines relations sont BCNF, et pourtant contiennent des redondances d'informations.

9.6.1.1 Un premier exemple : La relation BAGUE

Considérons la relation `BAGUE` 9.6 .
Nous pouvons constater qu'il n'y a aucune dépendance fonctionnelle dans cette relation, par

*NumeroBague	*Taille	*Matière
1	40	Or
1	41	Or
1	42	Or
2	40	Or
2	42	Or
2	40	Argent
2	42	Argent

Table 9.6 : Relation BAGUE

contre nous voyons bien qu'il y a des redondances d'informations : Les bagues *Numéro 1* existent dans les tailles *40, 41* et *42* en *Or*. De même, les bagues *numéro 2* existent dans les tailles *40* et *42* dans les matières *Or* et *Argent*.
Les redondances d'informations proviennent du fait que :

- Chaque bague existe dans certaines tailles et pour chaque taille dans les mêmes matières,
- De même, chaque bague existe dans certaines matières, et pour chaque matière dans les mêmes les tailles.

Ce que nous pouvons exprimer de la façon suivante :
Si les t-uplets *(NB, T1, M1)* et *(NB, T2, M2)* sont dans la relation `BAGUE`, alors les t-uplets *(NB, T1, M2)* et *(NB, T2, M1)* le sont aussi.
Par exemple :
les t-uplets *(2, 40, Or)* et *(2, 42, Argent)* sont dans la relation `BAGUE` 9.7 alors les t-uplets*(2, 40, Argent)* et *(2, 42, Or)* le sont aussi.

Cette relation pose des problèmes de mise à jour, en effet supposons que l'on créé pour la bague *Numéro 2* une nouvelle taille de bague: *43*, il faudra ajouter dans la relation `BAGUE`, les t-uplets suivants : Intuitivement les problèmes posés par la relation `BAGUE`, proviennent du fait que les attributs `Taille` et `Matière` sont indépendants, et il semble immédiat que la décomposition de la relation `BAGUE` en deux relations séparant ces deux attributs nous éviterait les redondances d'informations de cette relation et les problèmes d'insertion de données.
Ainsi la décomposition 9.10 et 9.9 semble correcte.

*NumeroBague	*Taille	*Matière
1	40	Or
1	41	Or
1	42	Or
2	***40***	***Or***
2	42	Or
2	40	Argent
2	***42***	***Argent***

Table 9.7 : Relation BAGUE

*NumeroBague	*Taille	*Matière
1	40	Or
1	41	Or
1	42	Or
2	40	Or
2	***42***	***Or***
2	***40***	***Argent***
2	42	Argent

Table 9.8 : Relation BAGUE

9.6.1.2 Définitions

Soient :
- R une relation
- (X, Y, Z) une **partition** de l'ensemble des attributs de R

On dit que **Y et Z sont multi-dépendants de X** et on note **X** $\rightarrow\!\!\rightarrow$ `Y / Z` :
Si et seulement si
- l'ensemble des valeurs de Y correspondant à un couple de valeurs (X, Z) de R dépend seulement de X et est indépendant de Z.
- l'ensemble des valeurs de Z correspondant à un couple de valeurs (X, Y) de R dépend seulement de X et est indépendant de Y

Une dépendance multivaluée caractérise une indépendance entre deux ensembles d'attributs, qui sont liés à un même attribut. Ce que nous pouvons exprimer de la façon suivante :

`X` $\rightarrow\!\!\rightarrow$ `Y / Z` est équivalent à :
`(x, y, z)` et `(x, y', z')` sont des t-uplets de R
alors :
`(x, y', z)` et `(x, y, z')` sont aussi des t-uplets de `R`.

Remarque : Dans certains cas on parlera de la dépendance multi-valuée : `X` $\rightarrow\!\!\rightarrow$ `Y`. Alors les dépendances multivaluées iront toujours par paires. Dans le cas de la partition `(X, Y, Z)`, si nous avons la dépendance multivaluée : `X` $\rightarrow\!\!\rightarrow$ `Y`, alors nous aurons aussi la dépendance multivaluée : `X` $\rightarrow\!\!\rightarrow$ `Z`. Ces deux dépendances correspondent à la notation vu ci-dessus : `X` $\rightarrow\!\!\rightarrow$ `Y / Z`.

On peut remarquer que les dépendances fonctionnelles sont des cas particuliers de dépendances multivaluées. En effet si `X` $\rightarrow$ `Y` : `(x, y, z)` et `(x, y', z')` sont des t-uplets de `R` implique que `y = y'`, d'après la définition d'une dépendance fonctionnelle.
Par suite nous avons donc : `(x,y,z)` et `(x,y,z')` sont des t-uplets de `R`, ce qu'il fallait démontrer.
Donc `X` $\rightarrow$ `Y` implique que `X` $\rightarrow\!\!\rightarrow$ `Y`.

*NumeroBague	*Taille	*Matière
2	43	Or
2	43	Argent

*NumeroBague	*Taille
1	40
1	41
1	42
2	40
2	42

Table 9.9 : Décomposition correcte : table 2

9.6.1.3 Quelques propriétés

Comme les dépendances fonctionnelles, les dépendances multivaluées ont des propriétés qui permettent de définir de nouvelles dépendances à partir d'un ensemble de dépendances.

Soient `R` une relation, et `(X, Y, Z)` des sous-ensembles de l'ensemble des attributs de `R` : `A`
- **Complémentation** : `X` $\rightarrow\!\!\rightarrow$ `Y` alors `X` $\rightarrow\!\!\rightarrow$ `A - X - Y`
- **Augmentation** : `X` $\rightarrow\!\!\rightarrow$ `Y` et `V` $\subset$ `W` alors `X, W` $\rightarrow\!\!\rightarrow$ `Y, V`
- **Transitivité** : `X` $\rightarrow\!\!\rightarrow$`Y` et `Y` $\rightarrow\!\!\rightarrow$ `Z` alors `X` $\rightarrow\!\!\rightarrow$ `Z - Y`

Dans l'exemple de la relation `BAGUE`, nous avons par exemple la dépendance multivaluée :
`NumeroBague` $\rightarrow\!\!\rightarrow$ `Taille`

et la règle de complémentarité nous donne une deuxième dépendance multivaluée qui est :
`NumeroBague` $\rightarrow\!\!\rightarrow$ `NumeroBague, Taille, Matière - NumeroBague - Taille`
Ce qui nous donne :
`NumeroBague` $\rightarrow\!\!\rightarrow$ `Matière` ou en dénotant les deux dépendances multivaluées ensemble :
`NumeroBague` $\rightarrow\!\!\rightarrow$ `Taille / Matière`

Ce qui paraissait immédiat de façon intuitive.

9.6.1.4 Application au Cinéma

Considérons la relation `CINEMA2` définie dans les étapes de normalisation précédentes. On se rend compte que pour une valeur de `Titre`, on peut avoir plusieurs valeurs de `NumeroCinema`, et plusieurs valeurs de `NomActeur`. On pourrait avoir envie par exemple de représenter la relation `CINEMA2` (Cf. 9.12) par la "relation" non 1NF donnée dans la table 9.11 .

Nous voyons ici qu'un même titre peut être à l'affiche dans plusieurs cinémas, et qu'il a plusieurs acteurs dans sa distribution. De plus ces informations sont complètement indépendantes, en effet les acteurs qui jouent dans un film ne dépendent pas des cinémas dans lesquels les films sont à l'affiche et vice versa. Mais par contre elles sont toutes deux liées au film.

Nous avons la dépendance multivaluée :
`Titre` $\rightarrow\!\!\rightarrow$ `NumeroCinema / NomActeur`

La relation `CINEMA2` proposée est bien 3NF ainsi que BCNF. Nous voyons cependant que certaines informations sont redondantes.

*NumeroBague	*Matière
1	Or
2	Or
2	Argent

Table 9.10 : Décomposition correcte

*NumeroCinema	*Titre	*NomActeur
1	Junior	Schwarzenegger De Mornay De Niro
1, 2, 3	Alerte	Hoffman Russo Freeman
2	L'Effaceur	Schwarzenegger
2	Gazon Maudit	Balasko Abril Chabat
3, 4	Casper	Pullman Ricci
4	Harcelement	Moore Douglas

Table 9.11 : Relation CINEMA2 non 1NF

Remarque : Nous avons bien ici : *(1, Alerte, Hoffman)* et *(2, Alerte, Russo)* dans la relation `CINEMA2`, et par définition d'une dépendance multivaluée : les t-uplets *(2, Alerte, Hoffman)* et *(1, Alerte, Russo)* appartiennent aussi à la relation `CINEMA2`.

La forme normale présentée ci après, 4NF, permet de prendre en compte ce genre de configuration. La notion de **dépendance multivaluée** correspond à une **généralisation de la notion de dépendance fonctionnelle**, qui permet de caractériser une indépendance entre 2 ensembles d'attributs liés à un troisième.

9.6.2 La quatrième forme normale : forme 4NF

Une relation R est en quatrième forme normale, notée 4NF, si et seulement si :
- Elle est en BCNF
- Les dépendances multivaluées dans R ont pour déterminant une super-clé.

Une relation est 4NF, si toute dépendance multivaluée a pour déterminant une clé de la relation (ou un groupe d'attributs contenant une clé).

9.6.2.1 Transformation en 4NF

Le **théorème de Fagin** permet de passer à une relation 4NF de la façon suivante :

Soit `R =X, Y, Z` une relation où `X, Y, Z` sont des ensembles d'attributs. `R` est égale à la jointure de ses projections `X, Y` et `X, Z` Si et seulement si `R` satisfait la dépendance multivaluée : `X` $\rightarrow\!\!\rightarrow$ `Y / Z`

*NumeroCinema	*Titre	*NomActeur
1	Junior	Schwarzenegger
1	Junior	De Mornay
1	Junior	De Niro
1	Alerte	Hoffman
1	Alerte	Russo
1	Alerte	Freeman
2	Alerte	Hoffman
2	Alerte	Russo
2	Alerte	Freeman
3	Alerte	Hoffman
3	Alerte	Russo
3	Alerte	Freeman
4	Alerte	Hoffman
4	Alerte	Russo
4	Alerte	Freeman
2	L'Effaceur	Schwarzenegger
2	Gazon Maudit	Balasko
2	Gazon Maudit	Abril
2	Gazon Maudit	Chabat
3	Casper	Pullman
3	Casper	Ricci
4	Casper	Pullman
4	Casper	Ricci
4	Harcelement	Moore
4	Harcelement	Douglas

Table 9.12 : Relation CINEMA2

Ce que nous pouvons schématiser par 9.5 :
Nous décomposons la relation non 4NF en deux relations.

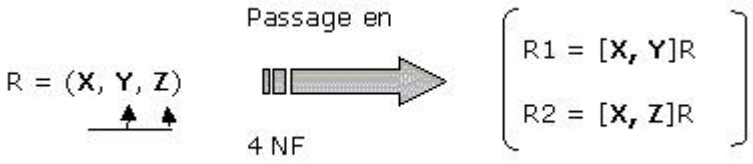


Figure 9.5 : Décomposition d'une relation non 4NF

- La première contient tous les attributs de la relation correspondant à la première partie de la dépendance multivaluée.
- La deuxième relation contient tous les attributs apparaissant dans la deuxième dépendance multivaluée. (partie complémentaire)

9.6.2.2 Application au Cinéma

L'étape précédente de normalisation nous a conduit à l'ensemble de relations suivant :

```
CINEMA(*NumeroCinema, NomCinema, AdresseCinema, CodePostalCinema,
```

```
       TelephoneCinema)
VILLE(*CodePostalCinema, VilleCinema)
FILM(*Titre, MetteurEnScene, Genre)
CINEMA2(*NumeroCinema, *Titre, *NomActeur)
```

Il est immédiat que la relation `CINEMA2` n'est pas 4NF, car nous avons la dépendance multivaluée :

DM1 : `Titre` $\rightarrow\rightarrow$ `NumeroCinema / NomActeur`

Or `Titre` n'est pas une super-clé de la relation `CINEMA2`. Nous proposons donc la décomposition suivante de cette relation, en utilisant le théorème de FAGIN :

```
AFFICHE(*NumeroCinema,* Titre)
DISTRIBUTION(*Titre, *NomActeur)
```

Ce que nous pouvons shématiser par 9.6 : Nous obtenons donc l'ensemble de relations suivantes

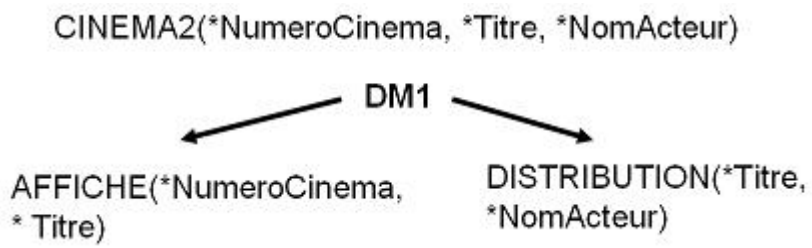


Figure 9.6 : Décomposition de CINEMA2

qui est 4NF.

```
CINEMA(*NumeroCinema, NomCinema, AdresseCinema, CodePostalCinema,
       TelephoneCinema)
VILLE(*CodePostalCinema, VilleCinema)
FILM(*Titre, MetteurEnScene, Genre)
AFFICHE(*NumeroCinema,* Titre)
DISTRIBUTION(*Titre, *NomActeur)
```

Exercice 7.7 : Les cours et les sports
Considérons la relation suivante :

`ETUDESPORT(*NumeroEtudiant, *Cours, *Sport)`

Où `Cours` et `Sport` permettent respectivement de décrire les cours suivis et les sports pratiqués par l'étudiant de numéro `NumeroEtudiant`. Notre relation peut contenir par exemple 9.13
. Montrer que cette relation n'est pas 4NF, et la transformer en un ensemble de relations 4NF.

---

*NumeroEtudiant	*Cours	*Sport
10	BD1	Tennis
10	MCOO	Tennis
10	BD1	Vélo
10	MCOO	Vélo
11	BD1	Tennis
12	MCOO	Vélo
12	MCOO	Tennis

Table 9.13 : Relation ETUDESPORT

9.6.3 Les dépendances de jointure

Soient :
- `R` une relation
- `(X, Y,..., Z)` des sous-ensemble quelconques des attributs de `R`,
On dit qu'il existe une **dépendance de jointure (JD)** et on note `*(X, Y,..., Z)` Si et seulement si `R` est égale à la jointure de ses projections sur les sous-ensembles d'attributs `X, Y, ..., Z`.

En d'autres termes la dépendance de jointure `*(X,Y,...Z)` est valide si `X, Y, ..., Z` est une décomposition sans perte d'informations de `R`.

9.6.3.1 La relation BAGUE

Pour mieux comprendre cette notion reprenons la relation `BAGUE` présentée dans le cours sur les dépendances multivaluées. Supposons que pour chaque bague on puisse avoir plusieurs tailles et plusieurs matières, mais que l'on ne dispose pas sytématiquement de toutes les tailles avec toutes les matières. Nous pourrions avoir par exemple la relation `BAGUE` suivante 9.16 : Largent

*NumeroBague	*Taille	*Matière
1	41	Or
1	42	Or
2	41	Or
2	42	Or
2	41	Argent

Table 9.14 : Relation BAGUE

ne se travaille que sur les tailles inférieures à *42*. Si maintenant nous essayons comme dans toutes les décompositions précédentes, de décomposer cette relation en deux relations, nous pouvons proposer par exemple la décomposition suivante :
Une première décomposition

*NumeroBague	*Taille
1	41
1	42
2	41
2	42

Table 9.15 : Relation BAGUE-Taille

*NumeroBague	*Matière
1	Or
2	Or
2	Argent

Et si nous essayons de "recomposer" notre relation par jointure, nous obtenons la relation `BAGUE1` ci-dessous :

*NumeroBague	*Taille	*Matière
1	41	Or
1	42	Or
2	41	Or
2	42	Or
2	41	Argent
2	**42**	**Argent**

Table 9.16 : Relation BAGUE1 recomposée

Le t-uplet *(2,42, Argent)* n'appartenait pas à la relation initiale `BAGUE`. La décomposition réalisée ici ne fait pas perdre de l'information, au contraire elle introduit des données parasites. Il n'y a pas ici de dépendance de jointure entre `BAGUE` et les sous-ensembles d'attributs : `(NumeroBague, Taille)`, et `(NumeroBague, Matière)`.

Considérons maintenant la décomposition de la relation BAGUE, non pas en deux relations, mais en trois relations.

Une deuxième décomposition

*NumeroBague	*Taille
1	41
1	42
2	41
2	42

*NumeroBague	*Matiere
1	Or
2	Or
2	Argent

*Taille	*Matiere
41	Or
42	Or
41	Argent

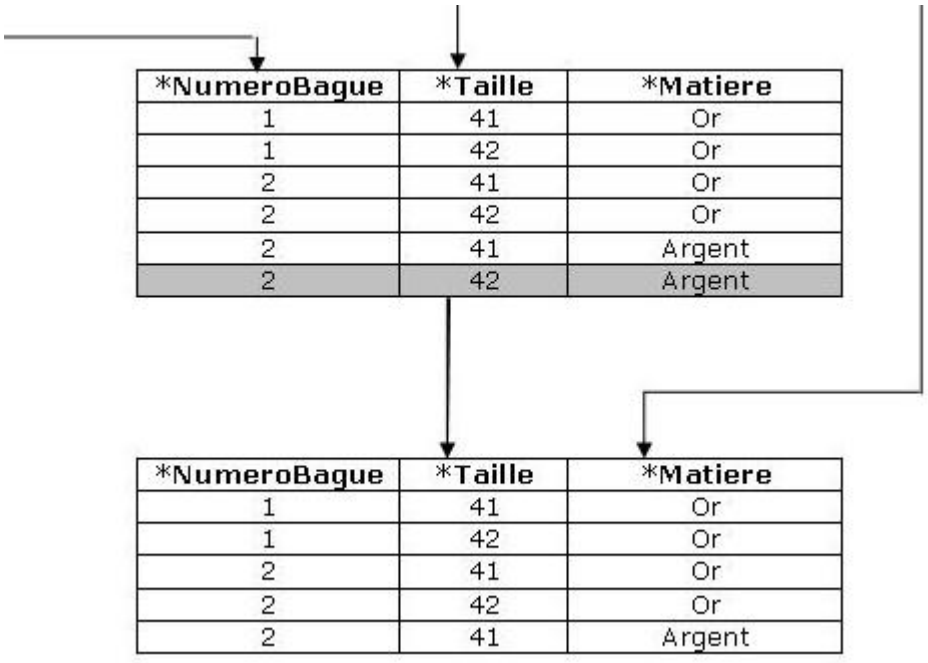
Figure 9.7 : Décomposition de BAGUE en 3 relations

Essayons maintenant de recomposer notre relation `BAGUE` à partir de ces trois relations : - Nous commençons pas effectuer une jointure sur `NumeroBague` des deux premières relations, - ensuite nous faisons une jointure sur `Taille` et `Matiere`. Nous obtenons alors bien la relation `BAGUE`. Il y a ici une dépendance de jointure entre `BAGUE` et les sous-ensembles d'attributs : `(NumeroBague, Taille)`, et `(NumeroBague, Matière)` et `(Taille, Matière)`.

Le fait que la relation `BAGUE` soit égale à la jointure de ces trois relations est équivalent au fait que l'on a :

- Si le t-uplet *(1, 41)* apparaît dans la relation `(NumeroBague, Taille)`
- Si le t-uplet *(1, Or)* apparaît dans la relation `(NumeroBague, Matière)`
- Si le t-uplet *(41, Or)* apparaît dans la relation `(Taille, Matière)`

Alors le t-uplet *(1, 41, Or)* apparaît dans la relation `BAGUE`.



*NumeroBague	*Taille	*Matiere
1	41	Or
1	42	Or
2	41	Or
2	42	Or
2	41	Argent
2	42	Argent

*NumeroBague	*Taille	*Matiere
1	41	Or
1	42	Or
2	41	Or
2	42	Or
2	41	Argent

Figure 9.8 : Décomposition correcte de BAGUE

Considérons le t-uplet *(2, 42, Argent)* qui n'appartient pas à la relation `BAGUE`. On voit effectivement que :

- le t-uplet *(2, 42)* apparaît dans la relation `(NumeroBague, Taille)`
- le t-uplet *(2, Argent)* apparaît dans la relation `(NumeroBague, Matière)`
- Par contre, le t-uplet **(42, Argent)** n'apparaît pas dans la relation `(Taille, Matière)`

La notion de dépendance de jointure correspond à une généralisation de la notion de dépendance multivaluée, elle permet de décomposer une relation sans perte d'informations en plus de deux relations.

Exercice 7.8 : Fournisseurs - Pièces et Projets
Considérons la relation suivante :

`FPP(*Fournisseur, *Piece, *Projet)`

Telle que la jointure des trois relations `(*Fournisseur, *Piece)`, `(*Fournisseur, *Projet)` et `(*Piece, *Projet)` redonne la relation FPP. Supposons que notre relation contienne les informations 9.17 . Expliquer pourquoi si l'on introduit le t-uplet *(Durand,X23, P02)* alors on doit aussi introduire le t-uplet *(Dupont, X23, P02)*.

---

*Fournisseur	*Piece	*Projet
Dupont	X23	P01
Dupont	X30	P02

Table 9.17 : Relation FOURNISSEURS

9.6.4 La cinquième forme normale : forme 5NF

Avant de définir plus avant la cinquième forme normale, nous introduisons la notion de dépendance dérivable d'une clé candidate.

9.6.4.1 Dépendance de jointure triviale

Soit `R` une relation et une dépendance de jointure `*X1, X2,..., Xp`. La dépendance de jointure est dite **triviale** si l'une des relations `Xi` est `R` elle-même.

9.6.4.2 Forme 5NF : Définition

Une relation `R` est en **cinquième forme normale**, notée **5NF**, si et seulement si : Toute dépendance de jointure `*X1, X2, ...,Xp` de `R` est triviale ou tout `Xi` est une super-clé de `R`.
Cette forme appelée aussi **forme normale de projection-jointure**, correspond à la décomposition la plus poussée que l'on puisse réaliser, si l'opérateur de décomposition est la projection.

9.6.4.3 Transformation en 5NF

Soit `R` une relation avec une dépendance de Jointure `*X1, X2,..., Xp` non triviale. On peut décomposer la relation `R` sous la forme de p relations `R1, R2, ..., Rp` telles que l'ensemble d'attributs de `Ri` est `Xi`.

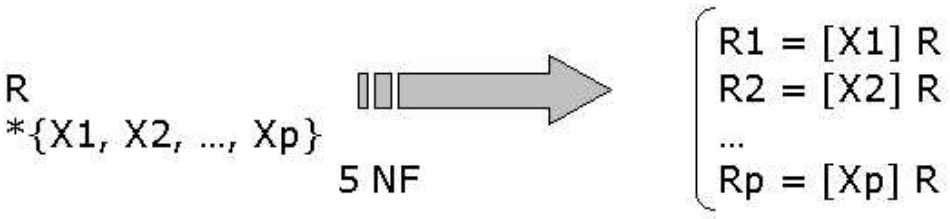


Figure 9.9 : Décomposition en 5NF

9.6.4.4 La relation BAGUE : 5NF ou pas ?

La relation `BAGUE` n'est pas 5NF, car `(NumeroBague, Taille)`, `(NumeroBague, Matière)` et `(Taille, Matière)` ne sont pas des super-clés de la relation `BAGUE`, et aucun de ces ensembles d'attributs n'est égal à `(NumeroBague, Taille, Matière)`. Il faut donc décomposer cette relation en les trois relations présentées ci-dessus afin d'éviter les anomalies de mises à jour.

Part V

Annexes

Bibliography

[1] Codd, E.F. *A relationnal model of data for large shared data banks.* CACM 13 N°6 Juin 1970.
C'est l'article de base de proposition du modèle relationnel. Il introduit les concepts fondamentaux de cette modélisation, et présente l'algèbre relationnelle comme langage de manipulation des relations.

[2] Connolly t.m. et Begg c.e. *Systèmes de bases de données* Les éditions Reynald Goulet INC. - 2005 - ISBN 2-89377-267-6

[3] Gardarin, Georges *Supports de cours sur les bases de données (livre, transparents).* http://georges.gardarin.free.fr/
Site complet sur les bases de données réalisé par G. Gardarin. Il y présente des éléments non présentés dans le cours.

[4] Gardarin, Georges *Bases de données : Objet et Relationnel.*. Eyrolles.
Livre de G. Gardarin, qui présente le lien entre le relationnel et l'objet. La présentation des notions de bases de la modélisation relationnelle y ait très claire. La suite du livre est plus réservée au module de base de données avancées qui suivra en master 1.

[5] Silberschatz a., Stonebraker m. et Ullman j. *Database systems : Achievements and opportunities.* .ACM SIGMOD Record, 19(4) - 1990 doi:10.1145/125223.125272

[6] Silberschatz a., Stonebraker m.r. et Ullman j. *Database Research: Achievements and Opportunities into the 21st century.* Technical Report CS-TR-96-1563, Department of Computer Science, Stanford - University, Stanford, CA.(1996).

[7] Soutou C. *SQL pour Oracle* Eyrolles ISBN 978-2-212-12299-2 (2008).

[8] DB-Engine *Classement des SGBD* https://db-engines.com/en/ranking

[9] Informatics News - Guy Hervier *Quelle base de données pour quel usage ?* http://www.informatiquenews.fr/quelle-base-de-donnees-pour-quel-usage-33500

[10] Site MariaDB *Documentation MariaDB en français* https://mariadb.com/kb/fr/documentation-de-mariadb/

[11] Site MariaDB *Documentation MariaDB sur les types de données* https://mariadb.com/kb/en/library/data-types/

Acronymes

Voici la liste des principaux acronymes utilisés dans ce cours.

A

- AD : Administrateur de Données
- ANSI : American National Standards Institute

B

- BD : Base de données
- BI : Business Intelligence

C

- CRM : Customer Relationship Management

D

- DA : Data Administrator.
- DBA : Database administrator. En français Administrateur de bases de données.
- DCL : Data Control Language
- DDL : Data Definition Language
- DML : Data Manipulation Language
- DTD : Document Type Definition

E

- ERP : Enterprise Resource Planning

G

- GUAM : Generalized Update Access Method

I

- IBM : International Business Machines
- IMS : Information Management System
- ISO : International Standard Organisation

M

- MCD : Modèle conceptuel de données
- MLD : Modèle logique de données

N

- NASA : National Aeronautics and Space Administration
- NoSQL : Not Only Structured Query Language

O

- ODMG : Object Database Management Group
- OLAP : On-Line Analytical Processing

Q

- QBE : Query By Example

S

- SGBD : Système de Gestion de Base de Données
- SGBDR : Système de Gestion de Base de Données Relationnel
- SI : Système d'information
- SPARC : Standards Planning And Requirements Committee
- SQL : Structured Query Language

T

- TCL : Transaction Control Language

X

- XML : Extensible Markup Language
- XQuery : XML Query
- XPath : XML Path Language

W

- WKB : Well-Known Binary

Index

*, 89

Administrateur, 14
Ajout de données, 113
ALL, 102
ALTER TABLE, 82
ANY, 102
Application de base de données, 5
AS, 89
Atomicité des attributs, 30
Attribut, 25, 90
AVG, 95

Base de données, 8
BETWEEN, 92

Cardinalité, 26
Clé, 26, 32
Clé étrangère, 34, 38, 39
Clé candidate, 33
Clé primaire, 34, 38
CONSTRAINT, 79
Contrainte d'intégrité, 79
Contrainte de domaine, 38
Contrainte référentielle, 39
Contraintes d'intégrité, 32
COUNT, 95
CREATE INDEX, 84
CREATE TABLE, 78
CREATE VIEW, 85

Dépendance de jointure, 141
Dépendance fonctionnelle, 121
Dépendance multivaluée, 135
Déterminant, 121
DCL, 76
DDL, 76, 77
Degré, 26
DELETE, 111
Différence, 55
DISTINCT, 88
Division, 65, 106
DML, 76, 87
Domaine, 26
DROP INDEX, 84
DROP TABLE, 83
DROP VIEW, 86

Equi-jointure, 63
EXISTS, 104
Expression en SQL, 107

Fonction d'agrégation, 95
Fonction de groupe, 95
Fonctions en SQL, 110
FOREIGN KEY, 80
Forme 1NF, 128
Forme 2NF, 128
Forme 3NF, 131
Forme 4NF, 138
Forme 5NF, 144
Forme BCNF, 133
Forme normale, 127
FROM, 90

GROUP BY, 97
GROUP_CONCAT, 95

HAVING, 98

IN, 92, 102
Index, 83
INNER JOIN, 93
INSERT, 113
Intégrité des entités, 38
INTERSECT, 105
Intersection, 54, 64

JOIN, 93
Jointure, 61, 92
Jointure interne, Jointure externe, 65
Jointure naturelle, 62

Lien, 35
LIKE, 92

MAX, 95

MIN, 95
MINUS, 105

NATURAL JOIN, 93
Normalisation, 117, 127
NULL, 37, 38

Opérateur ensembliste en SQL, 104
Opérateur relationnel, 53
Opérateur relationnel dérivé, 64
Opérateur relationnel ensembliste, 53
Opérateur relationnel spécifique, 57
ORDER BY, 99
OUTER JOIN, 93

PRIMARY KEY, 80
Produit cartésien, 56
Projection, 59
Propriété d'irréductibilité, 33
Propriété d'unicité, 32

QBE, 73

Régles d'inférence de Amstrong, 122
Règle de mise à jour : Cascade, 40
Règle de mise à jour : No action, 40
Règle de mise à jour : No Check, 40
Règle de mise à jour : Restrict, 40
Règle de mise à jour : Set Default, 40
Règle de mise à jour : Set Null, 40
Règles de mise à jour, 40, 79
Regroupement, 97
Relation, 25
Requête, 87

Sélection, 57
Schéma de base de données, 31
Schéma de base de données relationnelle, 31
SELECT, 87, 88
SGBD, 9
Sous-requête, 100
Sous-requête indépendante, 101
Sous-requête synchronisée, 103
SQL, 76
STDDEV, 95
Super-clé, 32
Système d'information, 5

T-uplet, 25
Table, 31, 77
Théorème de décomposition, 125
Thêta-Jointure, 64
Type de données, 43
Type de lien, 36

Unicité de clé, 37
UNION, 105
Union, 53
UPDATE, 112

VARIANCE, 95
Vue, 85

WHERE, 91