Contents

Chapter 1

Introduction

Cours de Bases de données - Introduction

Source: cours Introduction aux bases de données [Nicole Bidoit]

1.1 Introduction & Motivation

Que devez-vous connaître & savoir faire après ce cours?

- Qu'est-ce qu'une base de données?
- Notions élémentaires & vocabulaire
- Fonctionnalités de base d'un SGBD
- Un peu d'histoire
- Quelques systèmes
- Quelques métiers

1.1.1 Motivation

Nous vivons à l'ère du **Big Data** et du **Data Deluge**. Le terme **Data Scientist** est devenu courant. L'humanité produit une quantité exponentielle de données, estimée à 2,5 quintillions (10^{30}) d'octets par jour.

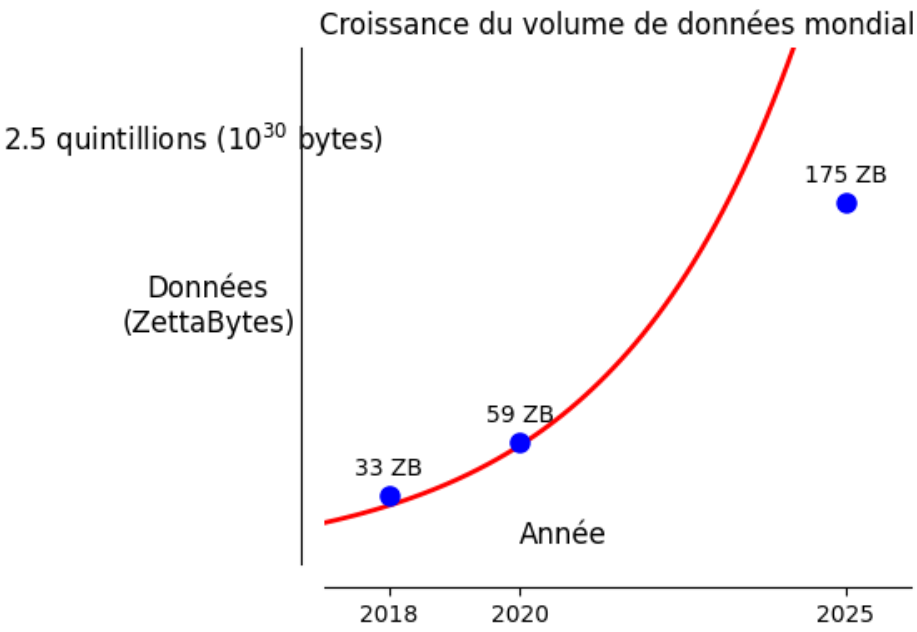

Figure 1.1: Évolution du volume de données généré.

2018	**2020**	**2025**
33 ZB	59 ZB	175 ZB
1 ZB (Zettabyte) = 1000 milliards de GB (Gigabytes) = 10^{21} octets		

Cette explosion de données est également illustrée par l'infographie "Data Never Sleeps" qui montre la quantité massive d'actions (recherches Google, messages envoyés, photos partagées, etc.) effectuées chaque minute sur Internet.

Les systèmes de gestion de bases de données (SGBD) sont au coeur de cette révolution, qui a commencé dans les années 1960. Cela fait plus de 50 ans ! Des applications courantes comme Izly, MyStudyLife, CAF, Parcoursup, SNCF, Too Good To Go, YouTube, Instagram, etc., reposent toutes sur des SGBD pour fonctionner. Elles manipulent constamment des données.

1.1.2 Introduction

Les données sont omniprésentes

- **Banque - Finance - Assurance** (Banking system)
- **Système d'Information Entreprise** : gestion du personnel, des clients, des stocks d'une entreprise (Information System)
- **Gestion de réservations** : Transports, Hotels, Spectacles (Airline reservation system)
- **Commerce électronique** : Culture, Agro-alimentaire, Bricolage (e-commerce)

Qu'est-ce qu'une base de données ?

Une base de données (Database) est une **collection d'informations** organisées.

- **Thématique/Activité** : Les données ont une sémantique et des liens abstraits entre elles.
- **Durée de vie** des données : variable selon le domaine (de quelques minutes à quelques années).

Qu'est-ce qu'un système de gestion de base de données ?

Un Système de Gestion de Base de Données (SGBD), ou Database Management System (DBMS), est un logiciel qui permet de gérer des bases de données. Ses fonctions principales sont :

- **Stocker les données** (Information storage) :
  Exemple : Enregistrer que "I. Adjani a joué dans L'été meurtrier".

  $$\text{I. Adjani a joué dans L'été meurtrier} \longrightarrow \boxed{\text{SGBD}}$$

- **Répondre à des questions = requêtes** (Query answering) :
  Exemple : Combien de César I. Adjani a-t-elle reçu ?

  $$\text{Combien de César I. Adjani a-t-elle reçu ?} \longrightarrow \boxed{\text{SGBD}} \longrightarrow 5 \text{ Césars}$$

- **Mise à jour les données** (Update) :
  Exemple : Enregistrer un nouveau tournage pour Adjani.

  $$\text{Nouveau tournage pour Adjani} \longrightarrow \boxed{\text{SGBD}}$$

1.1.3 Système de Gestion de Bases de Données

Que fait un SGBD (pour les applications)? Il assure le stockage, la mise à jour et l'accès (requêtes) **à de grandes masses** de données (*massive data*) en assurant :

- ✓ **efficacité** (*performance*)
- ✓ **persistance** (*persistency*)
- ✓ **fiabilité** (*reliability*)
- ✓ **simplicité d'utilisation** (*convenience*)
- ✓ **sécurité** (*safety*)
- ✓ **multi-utilisateur** (*multi-user*)

1.2 SGBD - Fonctionnalités

1.2.1 Grandes masses de données (Massive data)

Les SGBD sont conçus pour gérer :

- Gigabytes (10^9 octets) : c'est rien !
- Terabytes (10^{12} octets) / jour
- Petabytes (10^{15} octets)

Cela implique :

- $\curvearrowright$ Données stockées en mémoire secondaire (disques)
- $\curvearrowright$ Gestion optimisée des disques et du buffer (mémoire tampon)

Avec l'arrivée des Quintillions (10^{30} octets) (*very massive data*), il y a :

- $\curvearrowright$ Besoin de nouveaux supports de stockage
- $\curvearrowright$ De nouveaux mécanismes de stockage et d'accès aux données
- $\curvearrowright$ ...

Une courte parenthèse : Évolution des matériels et des besoins

Les machines et les besoins ont évolué :

- **1960** : Stocker un giga-octet → **plusieurs disques**.
  - SGBD → système complexe + coûteux + ordinateur puissant dédié
  - SGBD → **machine base de données**
- **2000** : Un giga-octet → **c'est rien !**
  - SGBD → installation sur un portable, aussi commun qu'un traitement de texte !
  - ... confusion avec la bureautique !
  - SGBD → **Client-Serveur**
- **2010** : Un tera-octet → **c'est rien !**
  - Data → **Cloud**
- **2024** : Un Zettabyte = 10^{21} bytes → **c'est beaucoup !**
  - Data → **Data Center** et besoin d'autres supports de stockage

1.2.2 Efficacité (Performance)

Le système doit être capable de gérer plusieurs milliers de requêtes par seconde ! Ces requêtes pouvant être complexes. L'optimisation est cruciale.

"Three things are important in the database world:
performance, performance, and performance"
- Bruce Lindsay, IBM (ACM SIGMOD Edgar F. Codd Innovations Award 2012)

1.2.3 Persistance (Persistency)

Exemple : Combien de Césars pour I. Adjani ?

- ↷ La consultation des palmarès à chaque demande peut donner une réponse différente en fonction des mises à jour !
- ↷ Toute modification des données faite par un programme **est enregistrée sur disque** et persiste au delà de la seule exécution du programme.

1.2.4 Fiabilité (pannes) (Reliability)

Les SGBD offrent une résistance aux pannes :

- Pannes matérielles, logicielles, électriques, ..., incendie, ...

Mécanismes clés :

- ↷ **Reprise sur panne** (failure recovery)
- ↷ **Journalisation** (logs, write ahead)
- ↷ Points de contrôle (checkpoints)

1.2.5 Sécurité (Safety)

Les données de la base peuvent être critiques :

- pour l'entreprise,
- pour les clients ...
- pour les états

Fonctionnalités :

- ↷ Protéger les données d'accès malveillants.
- ↷ **Contrôle d'accès, vues** (access grant)

✓ La protection de la vie privée (RGPD) est un autre sujet important (*privacy*).

1.2.6 Multi-utilisateur (Multi-user)

Accès à la base de données partagé par de nombreux utilisateurs et de nombreux programmes.

- Nécessité de contrôler l'accès (écriture/lecture) pour assurer la cohérence des données et des traitements.
- ↷ **Contrôle de la concurrence** (concurrency control)

Example 1.2.1 (Exécution concurrente)**.** M. et Mme Dupont ont 100 euros sur leur compte commun. Il retire 50 euros. Elle retire 20 euros. **En même temps**.

Opérations de M. Dupont		**Opérations de Mme Dupont**	
1	Lire_bd Cpte_Dupont $\rightarrow A_M$	i	Lire_bd Cpte_Dupont $\rightarrow A_{Mme}$
2	$A_M := A_M - 50$	ii	$A_{Mme} := A_{Mme} - 20$
3	Ecrire_bd $A_M \rightarrow$ Cpte_Dupont	iii	Ecrire_bd $A_{Mme} \rightarrow$ Cpte_Dupont

Une exécution concurrente possible : $1, i, 2, ii, 3, iii$. Si A_M et A_{Mme} lisent 100€ (étapes 1 et i), M. Dupont écrit 50€ (étape 3) puis Mme Dupont écrit 80€ (étape iii). Le solde final est 80€ au lieu des 30€ attendus. Le SGBD doit empêcher ce genre de situation via le contrôle de concurrence.

1.2.7 Fiabilité - Qualité (Data quality)

Données ≠ informations. Données + propriétés = **représentation** d'informations.

- ↷ **Contraintes d'intégrité** : Règles que les données doivent respecter.
- ↷ **Vérification automatique de l'intégrité** : Le SGBD assure que les contraintes sont respectées.

Example 1.2.2 (Contraintes d'intégrité)**.**

- Le César de la meilleure actrice est attribué à une femme.
- Le César de la meilleure actrice ne peut pas être attribué à deux actrices la même année.

1.2.8 Simplicité d'utilisation (Convenience)

Facilite le développement d'applications / Écriture de requêtes. Permet la maintenance et l'optimisation.

- ↷ **Langage de haut niveau & déclaratif** : SQL (Structured Query Language)

```
SELECT count(Palm.MA) FROM Prix WHERE Palm.Act='Adjani'
```

- ↷ **Schéma & Instance**
- ↷ **3 niveaux d'abstraction**
- ↷ **Principe d'indépendance physique**

1.3 SGBD - Principes

1.3.1 Schéma versus Instance

- ✓ **Schéma** = description de la structuration des données.
  - Le schéma est une donnée : ↷ stocké + modifiable + interrogeable
- ✓ **Instance** (du schéma) = données organisées en se conformant au schéma.

---

schéma = récipient — instance = contenu

1.3.2 Les Trois Niveaux d'Abstraction

Pour masquer la complexité du stockage physique et offrir une vue adaptée aux utilisateurs.

- **Niveau externe (vues)** : Description et manipulation des données dédiées à un groupe d'utilisateurs et applications. Plusieurs vues peuvent exister pour une même base.
- **Niveau logique** : Description et manipulation des données "haut niveau". Décrit quelles données sont stockées et quelles relations existent entre elles (ex: tables dans un SGBD relationnel). C'est le niveau du schéma conceptuel.
- **Niveau physique (interne)** : Organisation et stockage des données en mémoire secondaire. Décrit comment les données sont physiquement stockées (fichiers, index, etc.).

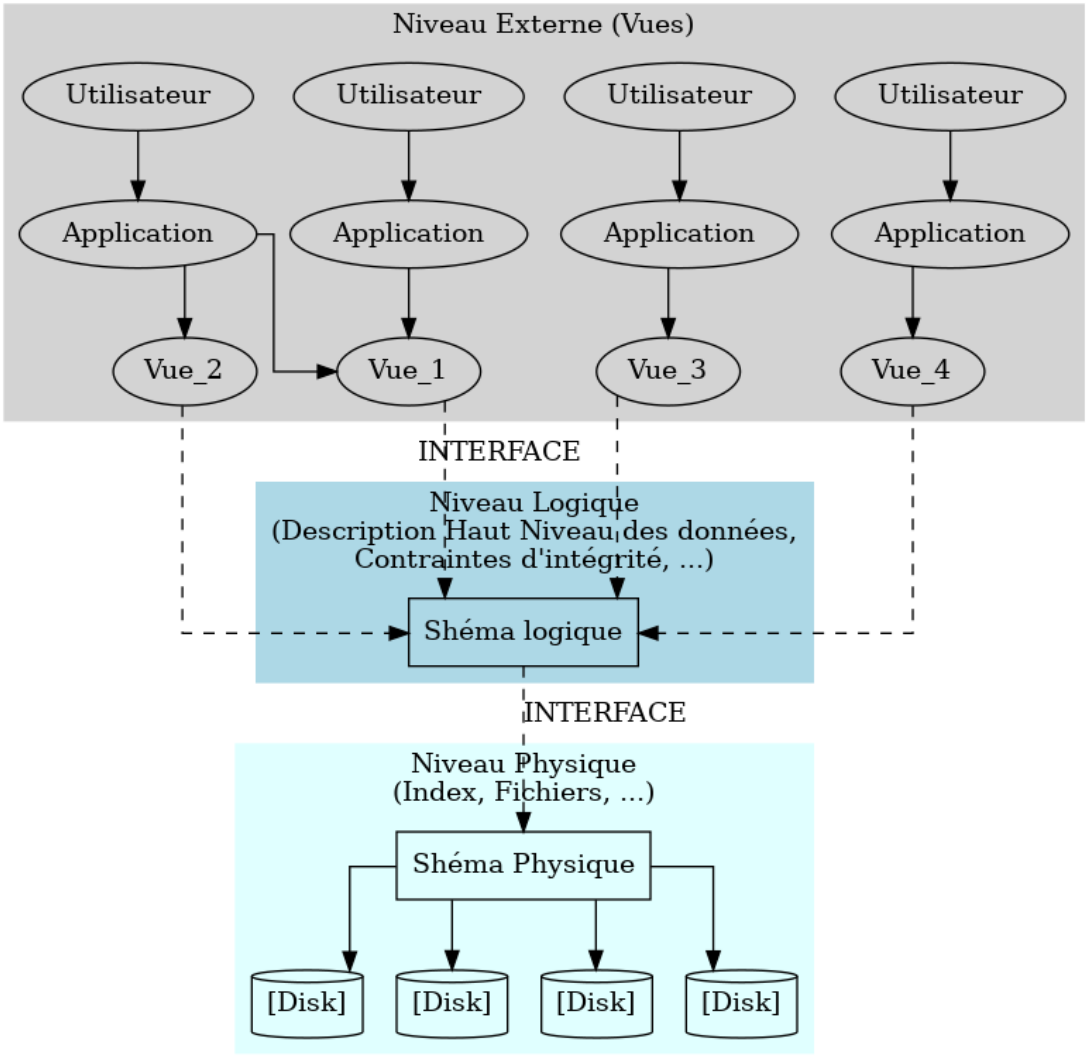

Figure 1.2: Les trois niveaux d'abstraction.

1.3.3 Utilisateurs versus Niveaux

- Utilisateurs λ, Développeurs d'Applications $\Longrightarrow$ Vues, Niveau Logique
- Développeurs BD, Administrateur BD $\Longrightarrow$ Niveaux Logique & Physique

1.3.4 Indépendance physique

Le principe d'indépendance physique signifie que les modifications apportées au niveau physique (comment les données sont stockées) ne doivent pas impacter les niveaux supérieurs (logique et externe), ni les applications qui les utilisent.

Programmes d'application	**Invariants**
Schéma Physique	**Modification**

Example 1.3.1 (Indépendance physique). Initialement = BD Palmarès festivals français

- ↷ schéma physique = fichier non trié, pas d'index (*heapfile*)

Requête :

```
Combien d'Oscars␣pour␣Adjani␣=␣SELECT␣count(Palm.MA)␣FROM␣Prix␣WHERE␣
    ↪ Palm.Act='Adjani'
```

Quelques années plus tard = BD Palmarès festivals internationaux

- ↷ schéma physique = fichier indexé (Arbre B+) (*index file*)

La même requête SQL fonctionne toujours, même si le stockage physique a changé. Le SGBD utilise l'index pour répondre plus efficacement, mais la requête elle-même reste inchangée.

```
Combien d'Oscars␣pour␣Adjani␣=␣SELECT␣count(Palm.MA)␣FROM␣Prix␣WHERE␣
    ↪ Palm.Act='Adjani'
```

Le SGBD assure cette indépendance via un processus de compilation/optimisation des requêtes. La requête (ex: SQL) est traduite en un plan d'exécution optimisé qui tient compte du schéma physique actuel.

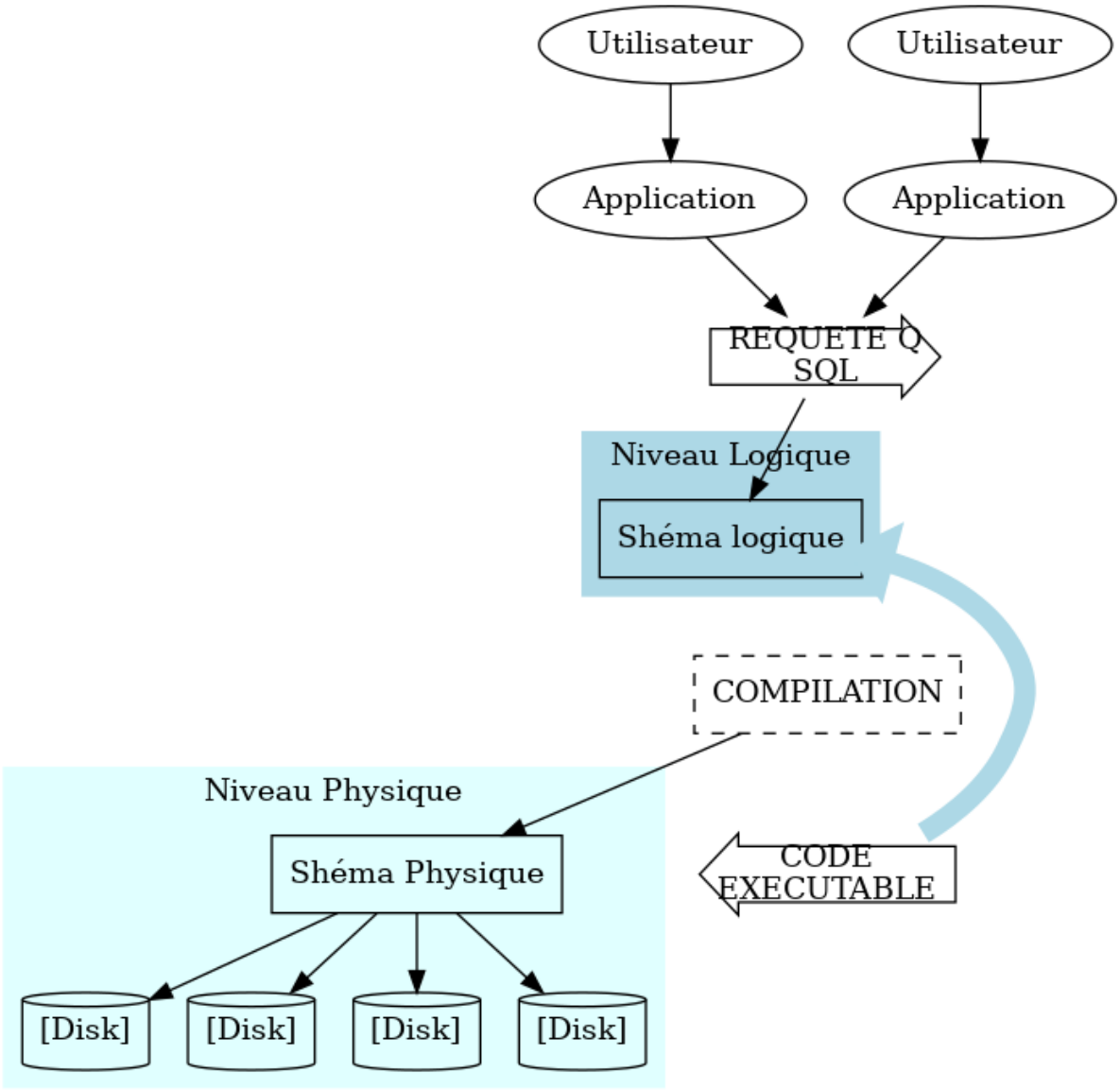

Figure 1.3: Indépendance physique et compilation des requêtes.

1.3.5 Les différents acteurs

Plusieurs rôles interviennent dans le cycle de vie d'une base de données :

- ✓ **Développeurs de SGBD** : Concevoir et développer les systèmes de gestion de bases de données (du futur !).
- ✓ **Concepteur de bases de données** : Concevoir le schéma logique de la base de données.
- ✓ **Développeur d'applications base de données** : Développer les programmes (requêtes, mises à jour et autres) qui opèrent sur la base de données et répondent aux besoins des applications.
- ✓ **Administrateur de la base de données** : Paramétrer le système, maintenance (tuning) des applications, gestion des sauvegardes, des droits d'accès, etc.

1.4 SGBD - Histoire

Un peu d'histoire des SGBDs : évolution des modèles de données.

- **Années 60-70 - SGBD Réseau** :
  - Exemples : IDS (General Electric), APL, DMS 1100, ..., ADABAS (Software AG)
  - Gestion basée sur des enregistrements reliés par des pointeurs.
- **Années 60-70 - SGBD Hiérarchique** :
  - Exemples : ISM, IBM (www.software.ibm.com)
  - Données organisées en arborescence. Gestion de pointeurs entre enregistrements.
  - Problème : pas d'indépendance physique.
- **Années 70-... - SGBD RELATIONNEL** (*Relational DBMS*) :
  - Proposé par Edgar F. Codd (IBM).
  - Modèle logique simple: données = **table**.
  - Formalisation mathématique : théorie des ensembles ou logique.
  - Un langage normalisé = **SQL** (Structured Query Language).
- **SGBD relationnel non normalisé** :
  - Une extension du modèle relationnel : des tables dans des tables !
- **SGBD orienté objet (OO)** :
  - Modèle inspiré par les concepts des langages objets (classe, héritage, etc.).
  - Identité d'objet, héritage, méthodes ...
- **Modèle semi-structuré et XML** :
  - Information incomplète, schéma souple.
  - Motivé par l'échange de données par les services web.
- **Modèle de graphes et RDF** :
  - Information incomplète, possibilité de raisonnement, schéma souple et interopérables.
  - Motivé par la publication et le partage de données sur le web (Web sémantique).
- **NoSQL** :
  - Adéquation aux besoins actuels (très grands volumes, haute disponibilité, schémas flexibles).
  - "SQL" = SGBDs relationnels classiques.
  - NoSQL= Ne pas utiliser les SGBDs relationnels classiques (pas toujours vrai).
  - NoSQL $\neq$ Ne pas utiliser le langage SQL.
  - NoSQL = Ne pas utiliser **seulement** les SGBDs relationnels (Not Only SQL). Regroupe différents modèles (clé-valeur, document, colonne, graphe).

1.5 SGBD – quelques systèmes

Focus sur les systèmes relationnels :

- ✓ **Systèmes historiques** :
  - System R (IBM-San José) - www.mcjones.org/System_R
  - Ingres, Postgres (post-Ingres) - www.ingres.com/products/ingres-database.php
- ✓ **Systèmes propriétaires** :
  - Oracle Database - www.oracle.com
  - Microsoft SQL Server - www.microsoft.com
  - DB2 (IBM), MaxDB (SAP), 4D, dBase, Informix (IBM), Sybase (SAP) ...
- ✓ **Systèmes libres** :
  - PostgreSQL - www.enterprisedb.com / www.postgresql.org
  - MariaDB (fork de MySQL) - www.mariadb.org
  - MySQL (Oracle) - www.mysql.com
  - Firebird, Ingres (Actian), HSQLDB, Derby (Apache) ...

Il existe une généalogie complexe retraçant l'évolution et les liens entre ces différents systèmes (voir "Genealogy of Relational Database Management Systems" du Hasso Plattner Institut).

1.6 Objectif du Cours

Un SGBD relationnel est un **système complexe** ! Il est souvent **facile à utiliser** (grâce à SQL et aux abstractions), mais **difficile à bien utiliser** !

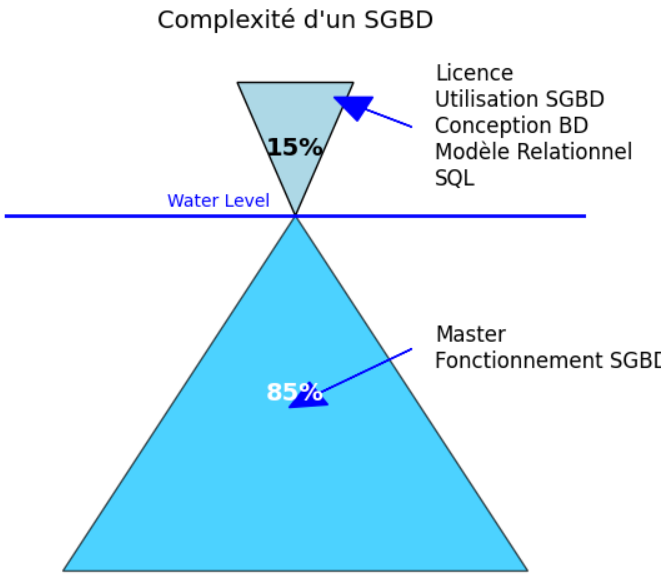

Figure 1.4: Illustration de la complexité cachée d'un SGBD.

La partie visible (Licence) représente ce qui est facile d'accès : utilisation basique, conception simple, modèle relationnel, SQL. La partie immergée (Master) représente la complexité interne : fonctionnement détaillé du SGBD (optimisation de requêtes, gestion de la concurrence, reprise sur panne, etc.).

1.7 Plan du cours

Ce cours abordera les thèmes suivants :

1 **Introduction & Motivation** (Ce chapitre)

2 **SGBD-Fonctionnalités** (Ce chapitre)

3 **SGBD-Principes** (Ce chapitre)

4 **SGBD-Histoire** (Ce chapitre)

5 **SGBD – quelques systèmes** (Ce chapitre)

6 **Plan du cours détaillé** :

- ✓ **Conception d'une base de données**
  - Processus de conception
  - Le modèle Entité-Association
- ✓ **Modèle de données**
  - Le Modèle Relationnel
  - Du Modèle E-A au modèle relationnel
  - SQL : Langage de définition (LDD)
- ✓ **Interrogation d'une base de données**
  - Algèbre relationnelle
  - SQL : Langage de manipulation (LMD)
- ✓ **Dépendances & Conception de schéma**
  - Dépendances fonctionnelles
  - Schéma sous forme normale

Chapter 2

Conception et modele entité relation

Cours 2 : Conception et Modèle Entité-Association

Introduction

La conception d'une base de données est une étape cruciale dans le développement d'applications qui nécessitent de manipuler de grandes quantités de données persistantes. Elle fait partie intégrante du processus de génie logiciel et vise à produire les schémas qui structureront la base de données.

Objectifs et Défis

L'objectif principal est de développer des applications (programmes) faisant un usage intensif de grandes masses de données persistantes.

- **Exemples d'applications :** gestion des stocks et des employés, agences de voyages, services hospitaliers, marketing, traitement des infractions, cinémas, agences spatiales, scolarité à l'université, laboratoires de recherche, etc.

La conception de bases de données diffère du génie logiciel "classique" (centré sur le développement de programmes) par son focus sur l'usage intensif de données persistantes. C'est une tâche complexe qui nécessite :

- Une **interaction** étroite avec les experts du domaine d'application et les futurs utilisateurs pour analyser et comprendre le domaine et les besoins métier.
- L'**identification** précise des données nécessaires aux traitements, aujourd'hui et demain.
- La capacité à **coder** le "monde réel", avec sa complexité et ses exceptions, à l'aide d'un modèle informatique. Ce passage de l'information brute aux données structurées demande abstraction et simplification.
- La **maîtrise de la complexité** liée au grand nombre d'informations à représenter.
- Une attention constante aux **performances** du système.

Les Étapes de la Conception

Le processus de conception se déroule généralement en trois phases principales :

1. **Modélisation Conceptuelle :**
   - Analyse du domaine d'application et des besoins.
   - Modélisation abstraite des données à l'aide d'un modèle conceptuel (par exemple, le modèle Entité-Association).

- Cette étape est indépendante du modèle de données cible (relationnel, objet, etc.) et du Système de Gestion de Base de Données (SGBD) utilisé.

2. **Modélisation Logique :**
   - Transformation du modèle conceptuel en un modèle logique, adapté au type de SGBD choisi (ex: modèle relationnel).
   - Cette étape dépend du modèle de données (Relationnel, Objet, Semi-structuré) mais reste généralement indépendante du SGBD spécifique.
3. **Modélisation Physique :**
   - Création de la base de données.
   - Définition du schéma physique en fonction du SGBD cible et des performances attendues.
   - Comprend l'organisation physique des données, les structures de stockage, et la création de structures accélératrices comme les index.
   - Cette étape dépend fortement du SGBD utilisé.

Ce chapitre se concentre sur la première étape : la **Modélisation Conceptuelle** à l'aide du **Modèle Entité-Association (E/A)**.

Élaborer un Modèle Conceptuel

L'élaboration d'un modèle conceptuel vise à :

- **Isoler les concepts fondamentaux :** Identifier ce que les données vont représenter, découvrir les concepts élémentaires du monde réel et décrire les sous-concepts pertinents.
- **Faciliter la visualisation du système :** Utiliser des diagrammes avec des notations simples et précises pour permettre une compréhension visuelle, et pas seulement intellectuelle, de la structure des données.

2.1 Le Modèle Entité-Association (E/A)

Introduit par Peter Chen en 1976 (*The Entity-Relationship Model – Towards a Unified View of Data*, ACM TODS, Vol 1, No 1), le modèle E/A fournit un ensemble de **concepts** pour modéliser les données d'une application et un ensemble de **symboles graphiques** pour les représenter.

2.1.1 Les Concepts Centraux

Le modèle E/A repose sur trois concepts centraux :

- **Entité**
- **Attribut**
- **Association**

À ces concepts s'ajoutent des notions importantes pour affiner le modèle :

- Contraintes de cardinalité
- Clés (identifiants) et entités faibles
- Règles métier (ex: prix ¿= 0)
- Héritage (spécialisation/généralisation)

Example 2.1.1 (Gestion de réservations de trains)**.** Supposons que le besoin soit de développer une application de gestion de réservations de trains .

- **Entités** potentielles : Voyage, Client, Train.
- **Attributs** potentiels : nom-client, date-départ, ville-départ, prix (pour Voyage), etc.
- **Association** potentielle : un Client *réserve* un Voyage.
- **Contraintes de cardinalité** : Une réservation est effectuée par un seul client ; Un client peut effectuer plusieurs réservations.
- **Clés** : Le numéro de réservation identifie une réservation unique.
- **Règles** : prix ¿= 0, date/heure d'arrivée ¿ date/heure départ.

Example 2.1.2 (Mini-application Cinéma - Schéma)**.** Considérons une application simple sur le cinéma, parlant d'acteurs et de films.

- **Entités :** Acteur, Film.
- **Attributs :**
  - Pour Acteur : Prénom, Nom, Date de Naissance (Ddn).
  - Pour Film : Titre, Année de sortie, Metteur en Scène (MeS).
- **Association :** Un Acteur *joue* dans un Film.

Le diagramme E/A (schéma) représente ces concepts :

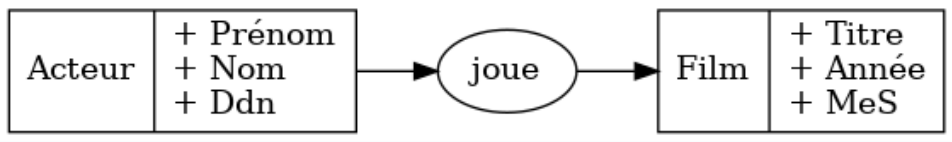

Figure 2.1: Schéma Entité-Association pour l'exemple du cinéma.

Example 2.1.3 (Mini-application Cinéma - Instance)**.** Une instance du schéma représente des données concrètes :

- L'actrice Isabelle Adjani (Prénom: Isabelle, Nom: Adjani, Ddn: 27-06-1955) a joué dans "L'été meurtrier" (Titre: L'été meurtrier, Année: 1983, MeS: J. Becker).
- Isabelle Adjani a aussi joué dans "La journée de la Jupe" (Titre: La journée de la Jupe, Année: 2009, MeS: J-P. Lilienfeld).

Cette instance peut être visualisée comme des liens entre les objets spécifiques :

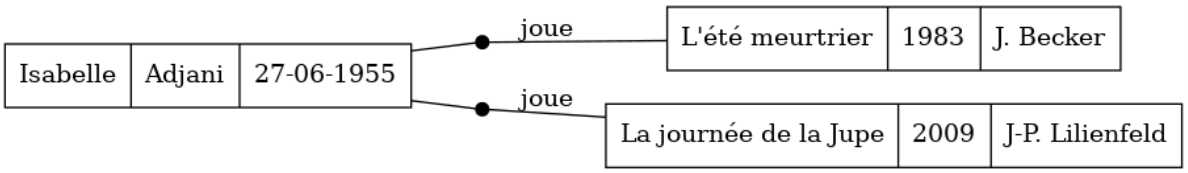

Figure 2.2: Instance du schéma Entité-Association pour l'exemple du cinéma.

Le schéma est la structure (le "plan"), tandis que l'instance représente les données réelles à un moment donné.

2.1.2 Entité (ou Type Entité)

Definition 2.1.4. Une **entité** (ou type d'entité) correspond à un ensemble d'objets (concrets ou abstraits, ex: événements) qui sont centraux dans le domaine de l'application et pour lesquels des informations doivent être stockées.

- Une entité représente une classe d'objets partageant les mêmes propriétés.
- Les objets individuels de cet ensemble sont appelés **instances** de l'entité.
- Une entité est spécifiée par un **nom**, généralement un substantif au singulier.
- Graphiquement, une entité est représentée par un rectangle contenant son nom.

Example 2.1.5.
- Acteur, Film (objets plutôt concrets)
- Tournage (événement, plus abstrait)

Figure 2.3: Représentation graphique des entités Acteur, Film, Tournage.

2.1.3 Attribut

Definition 2.1.6. Un **attribut** est une propriété ou une caractéristique d'une entité (ou parfois d'une association). Il sert à décrire l'entité.

- Un attribut est spécifié par un **nom** (A) et possède un **domaine de valeurs** (D(A)), qui définit l'ensemble des valeurs possibles pour cet attribut (ex: string, integer, date).
- Graphiquement, les attributs sont souvent listés à l'intérieur du rectangle de l'entité qu'ils décrivent.
- Un attribut peut être **simple** (atomique, ex: Nom) ou **complexe** (composé de sous-attributs, ex: Ddn = [jour, mois, an]). Le modèle E/A de base se concentre souvent sur les attributs simples.

Example 2.1.7. Pour l'entité Acteur : Prénom, Nom, Ddn. Domaines : D(Prénom)=string, D(Nom)=string, D(Ddn)=date. L'instance "Isabelle Adjani, 27-06-1955" a les valeurs d'attributs : Prénom="Isabelle", Nom="Adjani", Ddn="27-06-1955".

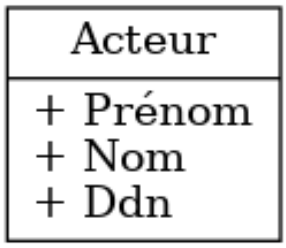

Figure 2.4: Entité Acteur avec ses attributs.

2.1.4 Association (ou Type d'Association)

Definition 2.1.8. Une **association** (ou type d'association) représente un lien, une relation sémantique entre N entités (souvent N=2, on parle alors d'association binaire). Elle correspond à des liens concrets entre des instances de ces entités.

- Une association est spécifiée par un **nom** (souvent un verbe), les **entités** qu'elle relie, et éventuellement les **rôles** joués par les entités dans l'association (surtout si l'association est réflexive ou si la même entité participe plusieurs fois).
- Une association peut aussi avoir des **attributs** qui décrivent le lien lui-même.
- Graphiquement, une association est souvent représentée par une ellipse (ou un losange) reliée par des traits aux rectangles des entités qu'elle associe.

Example 2.1.9 (Association Binaire)**.** L'association 'joue' relie les entités 'Acteur' et 'Film'. Une instance de cette association est un lien entre une instance d'Acteur (ex: Isabelle Adjani) et une instance de Film (ex: L'été meurtrier). (Voir Figure 2.1 et 2.2).

Example 2.1.10 (Association Ternaire)**.** Supposons qu'un metteur en scène ('MeS') dirige un 'Acteur' dans un 'Film'. Si le metteur en scène est considéré comme une entité à part entière (avec ses propres attributs, comme un nom), on pourrait avoir une association ternaire (N=3) 'Dirige'. *Note: Les associations ternaires ou d'arité supérieure sont souvent plus complexes à gérer et on essaie parfois de les remplacer par des entités et des associations binaires.*

Ici, si 'MeS' n'est qu'un attribut du film (comme dans le premier schéma), l'association reste binaire. Si 'MeS' devient une entité, on pourrait avoir:

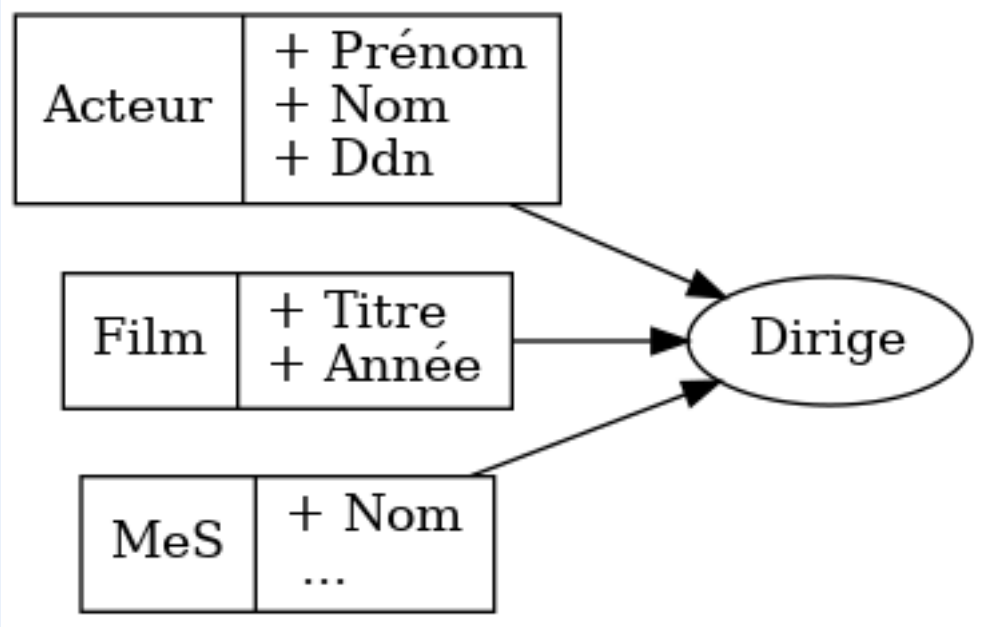

Figure 2.5: Schéma possible si MeS est une entité et liée par une association 'Dirige'. La représentation exacte peut varier.

Attention : La diapositive 16 suggère que MeS est une ENTITÉ liée à une association ternaire 'Dirige'. Cependant, la décision de modéliser MeS comme attribut ou entité dépend des besoins (voir section Principes de Conception).

Example 2.1.11 (Attribut d'Association)**.** Si un acteur peut jouer un rôle spécifique dans un film, l'attribut 'Rôle' ne décrit ni l'acteur en général, ni le film en général, mais bien le lien "joue" entre un acteur et un film particulier. 'Rôle' est alors un attribut de l'association 'joue'.

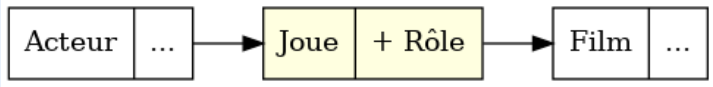

Figure 2.6: Association 'Joue' avec un attribut 'Rôle' (représentation approximative).

Question soulevée : Un acteur peut-il jouer plusieurs rôles dans un même film ? Si oui, ce modèle simple avec 'Rôle' comme attribut pourrait ne pas suffire (voir section Principes de Conception).

Example 2.1.12 (Association Réflexive)**.** Une association peut relier une entité à elle-même. On parle d'association réflexive ou récursive. Il est alors crucial de spécifier les rôles. Par exemple, un film peut être un remake d'un autre film.

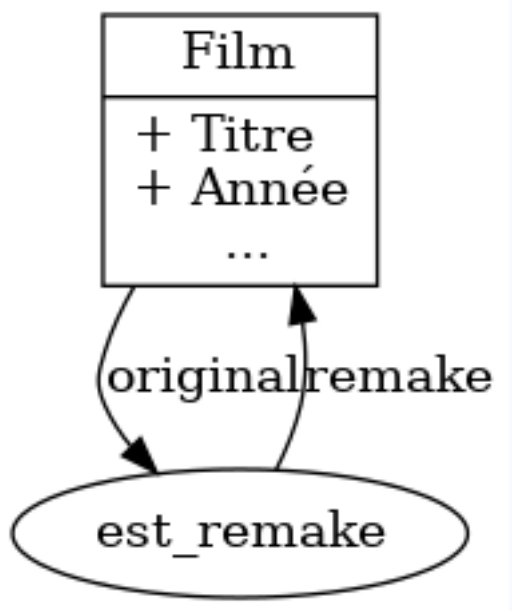

Figure 2.7: Association réflexive ‘est_remake‘ sur l’entité ‘Film‘ avec rôles.

Instance : Le film ”Les Choristes” (2004, Ch. Barratier) est un remake du film ”La Cage aux Rossignols” (1946, G. Chaperot).

2.1.5 Contraintes de Cardinalité

Definition 2.1.13. Les **contraintes de cardinalité** d’une association précisent le nombre minimum et maximum de liens auxquels une instance d’une entité peut participer dans cette association. Elles affinent la sémantique du lien entre les entités.

Pour une association binaire entre Entité A et Entité B :

- On définit une paire de cardinalités (min, max) du côté de chaque entité participante.
- **(min_A, max_A)** : Une instance de B est liée à au moins ‘$\min_A$‘$etauplus$‘max_A‘$instancesde A$.**(min_B, max_B)** : $Uneinstancede Aestliéeàaumoins$‘$min_B$‘$etauplus$‘$max_B$‘$inst$

- Les valeurs possibles pour min sont typiquement 0 ou 1.
- Les valeurs possibles pour max sont typiquement 1 ou ’m’ (ou ’n’, signifiant ”plusieurs”, ”many”).
- On doit avoir ‘$\min_x <= max_x$‘. Graphiquement, les cardinalités sont indiquées sur les traits reliant les entités à l’association.

Figure 2.8: Représentation générique des cardinalités.

Example 2.1.14 (Cardinalités Acteur-Film)**.** Reprenons l’association ‘joue‘ entre ‘Acteur‘ et ‘Film‘.

- (a) Un acteur peut ne jamais avoir joué dans un film $\implies$ min = 0 côté Film.
- (b) Un acteur peut jouer dans plusieurs films $\implies$ max = m côté Film.

- (c) Certains films sont tournés sans acteur (ex: film d'animation, documentaire) $\implies$ min = 0 côté Acteur.
- (d) La distribution d'un film est constituée de plusieurs acteurs (en général) $\implies$ max = m côté Acteur.

Les cardinalités sont donc (0, m) pour 'Acteur' (un film a 0 à plusieurs acteurs) et (0, m) pour 'Film' (un acteur joue dans 0 à plusieurs films). On parle d'une association de type "plusieurs-à-plusieurs" (m:m).

Figure 2.9: Association 'joue' avec cardinalités (0,m) - (0,m).

Une instance de cette association pourrait ressembler à ceci (où chaque point représente une instance d'acteur ou de film, et les lignes représentent les liens "joue") :

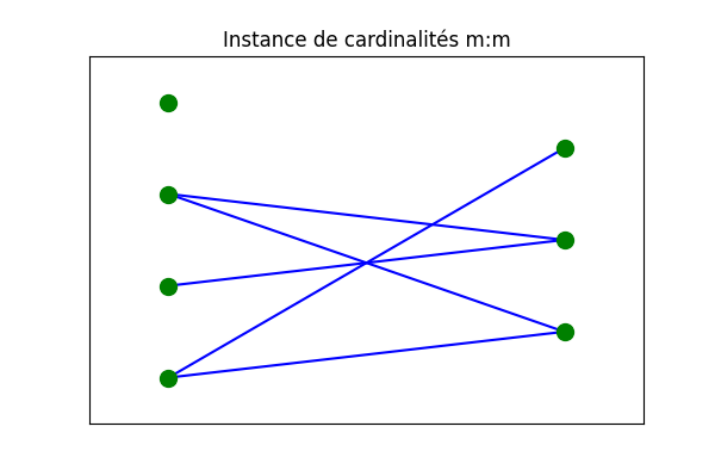

Figure 2.10: Illustration d'une instance avec cardinalités m:m.

Types d'Associations Binaires selon Cardinalités Maximales

On classifie souvent les associations binaires selon leurs cardinalités maximales :

- **Association m:m (plusieurs-à-plusieurs) :** Max_A = m et Max_B = m. (Exemple: Acteur joue dans Film).
- **Association 1:m (un-à-plusieurs) :** Max_A = 1 et Max_B = m. Une instance de A est liée à au plus un B, une instance de B peut être liée à plusieurs A. (Exemple: Un Employé travaille dans un Département ; un Département a plusieurs Employés).
- **Association 1:1 (un-à-un) :** Max_A = 1 et Max_B = 1. Une instance de A est liée à au plus un B, et une instance de B est liée à au plus un A. (Exemple: Un Utilisateur a un Profil ; un Profil appartient à un Utilisateur).

2.1.6 Clé (ou Identifiant)

Definition 2.1.15. Une **clé** (ou identifiant) d'une entité E est un ensemble minimal d'attributs de E dont la valeur permet d'identifier de manière unique chaque instance de E.

Il est difficile de désigner ou manipuler une instance spécifique d'une entité sans disposer d'une valeur unique pour l'identifier.

- **Cas simple :** La clé K est constituée d'un seul attribut A de E. Pour deux instances distinctes e et e' de E, on a $A(e) \neq A(e')$.
- **Cas général :** La clé K est composée de plusieurs attributs $A_1, A_2, ..., A_n$ de E. Pour deux instances distinctes e et e' de E, on a : $A_1(e) \neq A_1(e')$ OU $A_2(e) \neq A_2(e')$ OU ... OU $A_n(e) \neq A_n(e')$. Autrement dit, la combinaison des valeurs des attributs de la clé est unique pour chaque instance.
- **Minimalité :** Aucun sous-ensemble propre de K ne peut être une clé. Si $\{A_1, A_2\}$ est une clé, alors $\{A_1, A_2, A_3\}$ n'est pas considérée comme une clé (même si elle garantit l'unicité), car elle n'est pas minimale.
- Une entité peut avoir plusieurs clés candidates. On en choisit généralement une comme **clé primaire**.
- Graphiquement, les attributs formant la clé primaire sont souvent soulignés.

Example 2.1.16 (Clés pour Acteur et Film). • Pour l'entité 'Acteur' : {Prénom, Nom, Ddn} pourrait être une clé candidate (peu probable qu'il y ait deux acteurs avec exactement les mêmes). Cependant, il est plus courant d'introduire un identifiant artificiel unique, par exemple 'Num$_A$'($numérod'acteur$).Si'Num_A'$estajouté, ildevientlaclé primaire.Pourl'entité$'$Film$' : $\{Titre, Année\}pourraitêtreuneclécandidate(ilpeutyavoirdesfilmsaveclemêmetitremaissortisdesannéesdifférentes, ou$
Supposons que nous choisissons 'Num$_A$'$pour$'$Acteur$'$etlecouple\{Titre, Année\}pour$'$Film$'.

Figure 2.11: Entités avec clés primaires indiquées (conventionnellement soulignées) : `Acteur(Num_A, Prénom, Nom, Ddn)` et `Film(Titre, Année, MeS)`.

2.1.7 Entité Faible (ou Identification Relative)

Definition 2.1.17. Une **entité faible** est une entité B dont les instances ne peuvent pas exister ou être identifiées de manière unique sans être associées à une instance d'une autre entité A (dite entité propriétaire ou identifiante). L'existence de B dépend de A.

- Une entité faible B ne possède pas suffisamment d'attributs pour former une clé primaire par elle-même.
- Sa clé primaire est composite : elle inclut la clé primaire de l'entité propriétaire A et un ou plusieurs attributs propres à B (appelés discriminateur ou clé partielle).
- Il existe une **association identifiante** (R) entre A et B, qui est généralement de type 1:m (une instance de A peut être liée à plusieurs instances de B) et avec une cardinalité (1,1) du côté de B (chaque instance de B doit être liée à exactement une instance de A).
- Graphiquement, une entité faible est souvent représentée par un rectangle double, et l'association identifiante par une ellipse (ou losange) double.

Figure 2.12: Schéma générique : Entité forte A, Entité faible B, Association identifiante R. Clé de B = {Key_A, B1}. (Conventions graphiques : double rectangle pour B, double losange pour R non utilisées ici, alternatives possibles).

2.1.8 Héritage (is_a - Spécialisation/Généralisation)

Definition 2.1.19. L'**héritage** (ou relation 'is$_a$', *spécialisation/généralisation*)*permetd'introduireunehiérarchieentreent* *classeousous − entité*)*héritedesattributsetassociationsd'uneautreentité*(*super − classeousur − entité*), *toutenayantpotentiellementsespropresattributsetassociationsspécifiques*.

- C'est une situation très courante dans la réalité.
- Permet de modéliser des concepts qui sont des "cas particuliers" d'autres concepts.

- Graphiquement, on utilise souvent un triangle pointant de la (ou des) sous-classe(s) vers la super-classe, étiqueté 'is$_a$'(*ouparfoisISA*, *ousansétiquette*).

Example 2.1.20 (Types de Films)**.** Un film d'animation est un type de film. Un documentaire est aussi un type de film.

- 'Film' est la super-classe (sur-entité).
- 'Animation' et 'Documentaire' sont des sous-classes (sous-entités).
- 'Animation' et 'Documentaire' héritent des attributs de 'Film' (ex: Titre, Année) et peuvent avoir des attributs spécifiques (ex: Technique pour Animation, Sujet pour Documentaire).

heritage_film.png

Figure 2.14: Hiérarchie d'héritage : 'Documentaire' et 'Animation' sont des 'Film'.

Contraintes sur l'Héritage

Il est important de préciser la nature de la relation d'héritage :

- **Disjonction (Disjoint vs Overlapping) :** Une instance de la super-classe peut-elle appartenir à plusieurs sous-classes ?
  - *Disjoint :* Non (ex: un film ne peut pas être à la fois un documentaire ET un film d'animation). Souvent implicite ou indiqué par 'd' dans le triangle.
  - *Overlapping :* Oui (ex: une Personne peut être à la fois un Employé ET un Client). Indiqué par 'o'.
- **Totalité (Total vs Partial) :** Toute instance de la super-classe doit-elle appartenir à au moins une des sous-classes ?
  - *Total :* Oui (ex: un Film est forcément soit un Documentaire, soit une Animation, soit un Film de Fiction - si Fiction est aussi une sous-classe). Souvent indiqué par une double ligne vers la super-classe ou 't' dans le triangle.
  - *Partial :* Non (ex: un Film peut être autre chose qu'un Documentaire ou une Animation). Ligne simple (par défaut).

Dans l'exemple de base (Figure 2.14), la spécialisation est généralement considérée comme disjointe et partielle par défaut.

2.2 Principes de Conception E/A

La modélisation conceptuelle est un art autant qu'une science. Voici quelques principes directeurs :

1. Se Focaliser sur les Besoins et Faire Simple

- **Utilité :** Les entités, attributs et associations doivent refléter la réalité pertinente *pour l'application cible*. N'ajoutez pas d'éléments "au cas où" s'ils ne servent aucun besoin identifié.
  - *Exemple Attribut :* Pour une mini-application de cinéma listant les films et acteurs, l'attribut 'GENRE' d'un film pourrait être utile, mais l''adresse' de l'acteur ne l'est probablement pas. Pour les applications Amazon, l'adresse de l'acteur n'a aucun intérêt. En revanche, pour un studio de cinéma gérant ses acteurs, l'adresse et le portable sont importants.
  - *Exemple Attribut :* Si l'application n'utilise jamais la 'pointure' d'un acteur, inutile de la modéliser.
- **Réalité Applicative :** Les associations et leurs cardinalités doivent refléter la réalité et la situation correspondant à l'application.
  - *Exemple Cardinalité :* Un film est-il dirigé par un seul metteur en scène (1:m depuis MeS vers Film) ou potentiellement par plusieurs (m:m) ? La modélisation doit correspondre à la règle métier choisie pour l'application.
  - *Exemple Association :* A-t-on besoin de deux associations entre 'Studio' et 'Film' : une pour "produit" et une pour "a participé au tournage" ? Seulement si cette distinction est nécessaire pour l'application.
- **Simplicité :** Préférez les modèles plus simples quand ils suffisent à couvrir les besoins.

2. Éviter les Redondances

Definition 2.2.1. La **redondance** signifie que la même information apparaît sous des formes différentes ou à plusieurs endroits dans le modèle.

- **Problèmes :**
  - *Efficacité :* Gaspillage d'espace de stockage et temps de mise à jour potentiellement plus long (il faut mettre à jour l'information partout où elle se trouve).
  - *Qualité des données :* Risque d'inconsistances si une mise à jour n'est effectuée que sur une partie des occurrences de l'information redondante.
- **Objectif :** Concevoir le schéma de manière à minimiser la redondance non contrôlée. (Certaines redondances peuvent être introduites volontairement pour des raisons de performance au niveau physique, mais elles doivent être gérées).

3. Choisir Judicieusement Entités, Associations et Attributs

La décision de modéliser un concept comme une entité, un attribut ou une association n'est pas toujours immédiate.

Example 2.2.2 (Attribut vs Entité : Metteur en Scène)**.** Doit-on modéliser le metteur en scène (MeS) comme un simple attribut de l'entité 'Film' ou comme une entité à part entière ?

- **MeS comme Attribut :** Simple, si la seule information nécessaire est le nom du MeS associé au film. Suffisant pour le schéma de la Figure 2.1.
- **MeS comme Entité :** Nécessaire si on veut stocker plus d'informations sur le MeS (date de naissance, nationalité, autres films réalisés) ou si le MeS doit être lié à d'autres entités (ex: un MeS peut être aussi un Acteur). Cela conduit à un schéma comme celui de la Figure 2.5 (avec

une association ‘Dirige‘).

Le choix dépend des besoins de l’application.

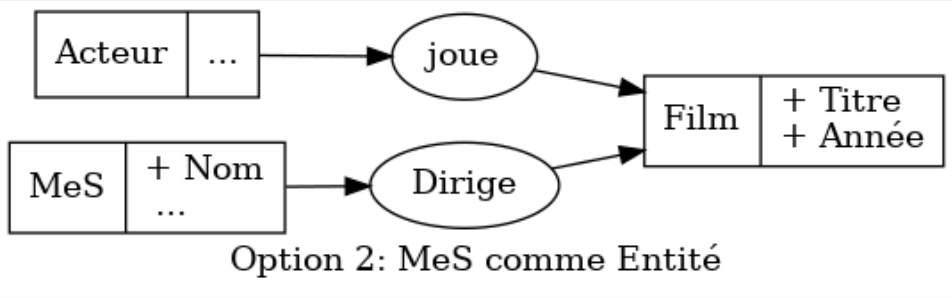

Figure 2.15: Option 2 : Modélisation de ‘MeS‘ comme une entité distincte.

Example 2.2.3 (Attribut d’Association vs Entité : Rôle)**.** Un acteur peut-il jouer plusieurs rôles dans le même film ?

- **Rôle comme Attribut de ’joue’ (Figure 2.6):** Fonctionne seulement si un acteur joue au plus un rôle par film. Le couple {Acteur, Film} identifie implicitement le rôle.
- **Si plusieurs rôles possibles :** Le modèle précédent ne suffit pas. Il faut pouvoir distinguer les différents rôles joués par le même acteur dans le même film. Deux options principales :
  1. **Promouvoir l’Association en Entité :** Créer une entité ‘RoleJoue‘ (ou ‘Participation‘) avec comme attributs le rôle, le costume, le texte, etc. Cette entité serait liée à ‘Acteur‘ et ‘Film‘. Sa clé serait probablement {Id_Acteur, Id_Film, Rôle}.
  2. **Créer une Entité Rôle :** Si les rôles eux-mêmes ont des propriétés ou existent indépendamment (ex: ”Hamlet”, ”James Bond”), on peut créer une entité ‘Role‘. L’association ‘joue‘ deviendrait alors ternaire entre ‘Acteur‘, ‘Film‘, et ‘Role‘, ou serait décomposée. La diapositive 39 suggère une entité ‘Rôle‘ liée à l’association ‘joue‘, ce qui est une modélisation possible mais potentiellement complexe.

Le choix dépend encore une fois de la complexité requise par l’application. La première option (Rôle comme attribut de ’joue’) est la plus simple si elle suffit.

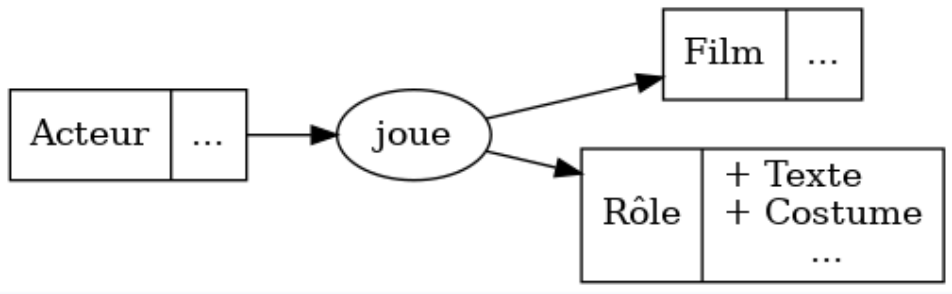

Figure 2.16: Modélisation possible si un acteur peut jouer plusieurs rôles dans un film (basée sur la diapositive 39).

4. Introduire Judicieusement les Entités Faibles et l’Héritage

- N’utilisez les entités faibles que lorsque l’identification dépend clairement d’une autre entité.

- N'utilisez l'héritage que lorsqu'il y a une claire relation "est un type de" avec des spécificités au niveau des sous-classes.
- Parfois, une relation 1:1 peut remplacer l'héritage ou vice-versa. Évaluez la solution la plus naturelle et la plus maintenable.

2.2.1 Conclusion et Perspectives

Le modèle Entité-Association est un outil puissant et largement utilisé pour la modélisation conceptuelle des bases de données. Il permet de décrire la structure des données d'une application de manière abstraite et graphique, en se concentrant sur les entités principales, leurs attributs et les associations qui les relient, enrichies par des contraintes de cardinalité, des clés, et des concepts comme les entités faibles et l'héritage.

La maîtrise des principes de conception (se focaliser sur les besoins, simplicité, éviter la redondance, choisir judicieusement les concepts) est essentielle pour produire un schéma conceptuel de qualité, qui servira de base solide pour les étapes ultérieures de modélisation logique et physique.

D'autres approches existent pour la modélisation conceptuelle, notamment :

- Les **Diagrammes de Classes UML** (Unified Modeling Language), inspirés du génie logiciel classique.
- Les **Modèles Sémantiques** et les **Ontologies**, issus du domaine de la représentation des connaissances.

Des outils graphiques dédiés (comme WinDesign, Looping, Lucidchart, Visual Paradigm, etc.) aident à la création et à la gestion des modèles E/A. La modélisation de données reste un métier à part entière.

Chapter 3

Passage schéma relationnel

3.1 Introduction

La conception d'une base de données passe typiquement par plusieurs étapes de modélisation. On commence souvent par une **modélisation conceptuelle** qui permet d'analyser le domaine d'application et les besoins de manière abstraite. Le modèle Entité-Association (EA) est fréquemment utilisé à ce niveau. Ensuite, on procède à une transformation vers une **modélisation logique**, qui dépend du type de Système de Gestion de Base de Données (SGBD) cible. Le modèle relationnel est le modèle logique le plus répandu. Enfin, la **modélisation physique** s'occupe des détails d'implémentation comme les index et les paramètres spécifiques au SGBD.

Ce chapitre se concentre sur l'étape cruciale du passage du modèle conceptuel EA au modèle logique relationnel. Comprendre ce processus est essentiel pour construire des bases de données bien structurées et efficaces.

3.1.1 Rappel des Concepts Fondamentaux

Modèle Entité-Association (EA)

Le modèle EA repose sur les concepts suivants :

- **Entité :** Représente un objet ou un concept du monde réel (ex : Acteur, Film, Studio). Une entité possède des attributs.
- **Association :** Représente un lien entre deux ou plusieurs entités (ex : un Acteur *joue* dans un Film). Une association peut aussi avoir des attributs.
- **Attribut :** Une propriété d'une entité ou d'une association (ex : Nom d'un Acteur, Titre d'un Film, Rôle dans une association Joue).
- **Clé :** Un attribut ou un ensemble d'attributs permettant d'identifier de manière unique une occurrence d'entité (ex : Num-A pour Acteur).
- **Contraintes de Cardinalité :** Indiquent le nombre minimum et maximum d'occurrences d'une entité qui peuvent être liées à une occurrence d'une autre entité via une association (ex : (0,n), (1,1), (1,n)).

Modèle Relationnel

Le modèle relationnel repose sur les concepts suivants :

- **Relation (ou Table) :** Un ensemble de tuples (lignes), où chaque tuple représente une occurrence et chaque colonne représente un attribut.
- **Attribut :** Une colonne dans une relation.

- **Clé Primaire :** Un attribut ou un ensemble d'attributs qui identifie de manière unique chaque tuple dans une relation. Une clé primaire est associée à une *dépendance fonctionnelle*.
- **Clé Étrangère :** Un attribut ou un ensemble d'attributs dans une relation qui référence la clé primaire d'une autre relation (ou de la même relation). Elle assure l'intégrité référentielle et est associée à une *dépendance d'inclusion*.
- **Schéma de Relation :** La définition d'une relation, incluant son nom, ses attributs et ses clés. Exemple : Acteur(Num_A, Prénom, Nom, Ddn).

3.2 Règles de Passage du Modèle EA au Modèle Relationnel

Le passage du modèle EA au modèle relationnel suit un ensemble de règles systématiques.

3.2.1 Règle 1 : Conversion des Entités Fortes

Pour chaque type d'entité forte E dans le modèle EA, on crée un schéma de relation ayant le même nom E.

- Les **attributs** de l'entité E deviennent les attributs de la relation E.
- La **clé** (identifiant) de l'entité E devient la **clé primaire** de la relation E.

Example 3.2.1 (Conversion d'entités). En reprenant l'exemple du diagramme EA (voir figure conceptuelle sur les diapositives) :

- L'entité **Acteur** avec les attributs Num-A (clé), Prénom, Nom, Ddn devient la relation : **Acteur**(Num_A, Prénom, Nom, Ddn)
- L'entité **Film** avec les attributs Num_F (clé), Titre, Année devient la relation : **Film**(Num_F, Titre, Année)
- L'entité **Président** avec les attributs Num_P (clé), Nom, An devient la relation : **Président**(Num_P, Nom, An)
- L'entité **Studio** avec les attributs Nom (clé), Adr devient la relation : **Studio**(Nom, Adr)

3.2.2 Règle 2 : Conversion des Associations

Pour chaque type d'association A (binaire ou n-aire) entre les entités E1, E2, ..., En, on crée un schéma de relation A.

- Les **attributs** de la relation A incluent :
  - Tous les attributs propres à l'association A.
  - Les attributs formant les clés primaires K1, K2, ..., Kn des relations E1, E2, ..., En correspondantes. Ces attributs deviennent des clés étrangères dans la relation A.
- La **clé primaire** de la relation A est généralement composée de l'ensemble des attributs clés K1, K2, ..., Kn provenant des entités participantes. (Des exceptions existent, notamment pour les cardinalités max = 1).
- Pour chaque clé Ki provenant d'une entité Ei, on ajoute une **contrainte d'inclusion** (clé étrangère) pour garantir l'intégrité référentielle : $A(Ki) \subseteq Ei(Ki)$. Cela signifie que la valeur d'une clé étrangère dans A doit exister comme valeur de clé primaire dans la table référencée Ei.

Example 3.2.2 (Conversion d'associations). En reprenant l'exemple :

- L'association **Joue** entre Acteur et Film, avec l'attribut Rôle :
  Relation : **Joue**(Num_A, Num_F, Rôle)
  Clés étrangères :
  - Joue(Num_A) $\subseteq$ Acteur(Num_A)
  - Joue(Num_F) $\subseteq$ Film(Num_F)

  La clé primaire est composite : {Num_A, Num_F}.
- L'association **Distribue** entre Film et Studio (sans attribut propre) :
  Relation : **Distribue**(Num_F, Nom)
  Clés étrangères :
  - Distribue(Num_F) $\subseteq$ Film(Num_F)
  - Distribue(Nom) $\subseteq$ Studio(Nom)

  La clé primaire est composite : {Num_F, Nom}.
- L'association **Dirige** entre Président et Studio (sans attribut propre) :
  Relation : **Dirige**(Num_P, Nom)
  Clés étrangères :
  - Dirige(Num_P) $\subseteq$ Président(Num_P)
  - Dirige(Nom) $\subseteq$ Studio(Nom)

  La clé primaire est composite : {Num_P, Nom}.

3.2.3 Règle 3 : Raffinement du Schéma Relationnel

Après l'application des deux premières règles, on obtient un schéma relationnel "brut". L'étape suivante consiste à le raffiner en tenant compte des contraintes de cardinalité. Ce raffinement peut conduire à :

- **Fusionner** des schémas de relation (typiquement une relation issue d'une association avec une relation issue d'une entité) pour optimiser la structure et réduire le nombre de tables.
- Analyser et potentiellement **normaliser** les relations pour éviter les redondances et les anomalies de mise à jour (non détaillé ici).

L'analyse des cardinalités est cruciale pour décider quelles fusions sont possibles et judicieuses.

3.3 Prise en Compte des Contraintes de Cardinalité (Raffinement)

Les cardinalités maximales (Max1, Max2) des associations binaires guident la fusion éventuelle des relations.

3.3.1 Cas [1:M] (Cardinalités Max = 1 et Max = N)

Soit une association A entre E1 et E2, avec des cardinalités (min1, 1) du côté de E2 et (min2, N) du côté de E1. La cardinalité maximale est 1 pour E1 participant à A, et N pour E2.

- **Règle :** On fusionne la relation A issue de l'association avec la relation E1 issue de l'entité côté 'N'. La relation E2 reste inchangée. On obtient une nouvelle relation que l'on peut nommer AE1 (ou garder le nom E1 et y ajouter les attributs).
- **Attributs de AE1 :** Attributs de E1 + Attributs de A + Clé primaire K2 de E2 (devient clé étrangère dans AE1).

- **Clé primaire de AE1 :** La clé primaire K1 de E1.
- **Clé étrangère :** AE1(K2) $\subseteq$ E2(K2).
- **Contrainte de Nullité :** Si la cardinalité minimale côté E1 (min1) est 1 (participation obligatoire), alors l'attribut K2 (clé étrangère) dans AE1 ne peut **pas** avoir de valeur nulle (NOT NULL). Si min1 est 0, K2 peut être NULL.

Example 3.3.1 (Cas [1:M]. - Film-Distribue-Studio] L'association **Distribue** lie Film et Studio. Cardinalités : Film –(1,1)– Distribue –(1,n)– Studio. Maximums : 1 côté Studio, N côté Film. C'est donc un cas M:1 (ou 1:N vu depuis Studio). Le côté '1' est Film.

Application de la règle [1:M] (en considérant l'inverse, M côté Studio, 1 côté Film) : On fusionne la relation **Distribue** avec la relation **Film** (côté '1'). Relations initiales (issues règles 1 et 2) :

- Film(Num_F, Titre, Année)
- Studio(Nom, Adr)
- Distribue(Num_F, Nom)

Après fusion de Film et Distribue :

- **FilmDist**(Num_F, Titre, Année, Nom)
- **Studio**(Nom, Adr)

Clé primaire de FilmDist : Num_F (clé de Film). Clé étrangère : FilmDist(Nom) $\subseteq$ Studio(Nom). Nullité : La cardinalité minimale de Film dans Distribue est 1. Donc, l'attribut 'Nom' (la clé étrangère référençant Studio) dans FilmDist **ne peut pas être nul** (Nom non Null).

Schéma relationnel final après raffinement :

- FilmDist(Num_F, Titre, Année, *Nom*)
- Studio(Nom, Adr)

Note : L'attribut Nom dans FilmDist est une clé étrangère référençant Studio.Nom.

Example 3.3.2 (Cas [0,1 : M]. - Variante avec 0 minimum] Si l'association A entre E1 et E2 avait des cardinalités (0,1) côté E2 et (0,N) côté E1. Max = 1 côté E1, Max = N côté E2. On fusionnerait A avec E1 (côté N). Relation AE1 : Attrs(E1) + Attrs(A) + Key(E2). Clé primaire = Key(E1). Clé étrangère : AE1(K2) $\subseteq$ E2(K2). Nullité : Comme la cardinalité minimale côté E1 est 0, la clé étrangère K2 dans AE1 **peut être nulle** (valeur nulle possible).

3.3.2 Cas [1:1] (Cardinalités Max = 1 et Max = 1)

Soit une association A entre E1 et E2, avec des cardinalités maximales de 1 des deux côtés.

Sous-cas : (0,1) – (0,1)

- **Règle :** On peut fusionner A avec E1 **ou** avec E2. Le choix est arbitraire, mais peut être guidé par la logique métier. Supposons qu'on fusionne A et E1 en AE1.
- **Attributs de AE1 :** Attributs de E1 + Attributs de A + Clé primaire K2 de E2.
- **Clé primaire de AE1 :** La clé primaire K1 de E1.
- **Clé étrangère :** AE1(K2) $\subseteq$ E2(K2).

- **Contrainte de Nullité :** Puisque les cardinalités minimales sont 0, la clé étrangère K2 dans AE1 **peut être nulle**. De même, si on avait fusionné avec E2, K1 aurait pu être nulle dans AE2.

Example 3.3.3 (Cas [1:1]. - (0,1)–(0,1) - Président-Dirige-Studio] Association **Dirige** entre Président et Studio. Cardinalités (0,1) des deux côtés. Relations initiales :

- Président(Num_P, Nom, An)
- Studio(NomS, Adr)
- Dirige(Num_P, NomS)

Fusionnons Dirige avec Président (choix arbitraire) :

- **Presidirige**(Num_P, Nom, An, NomS)
- **Studio**(NomS, Adr)

Clé primaire de Presidirige : Num_P. Clé étrangère : Presidirige(NomS) $\subseteq$ Studio(NomS). Nullité : L'attribut NomS dans Presidirige peut être nul (un président peut ne diriger aucun studio). Attention aux noms : Il faut parfois renommer les attributs lors de la fusion pour éviter les conflits (ex: Nom du Président vs Nom du Studio renommé NomS).

Schéma relationnel final :

- Presidirige(Num_P, Nom, An, *NomS*)
- Studio(NomS, Adr)

Note : NomS dans Presidirige est une clé étrangère référençant Studio.NomS.

Sous-cas : (1,1) – (1,1)

- **Règle :** On fusionne les relations E1, E2 et A en une seule relation R.
- **Attributs de R :** Attributs de E1 + Attributs de E2 + Attributs de A.
- **Clé primaire de R :** On peut choisir K1 (clé de E1) **ou** K2 (clé de E2) comme clé primaire. L'autre devient une clé candidate (unique et non nulle).
- **Contrainte de Nullité :** Aucun des attributs K1 ou K2 ne peut être nul dans R, car les participations sont obligatoires (minimum 1).

Example 3.3.4 (Cas [1:1]. - (1,1)–(1,1) - Acteur-Assure-Contrat] Supposons une association **Assure** (1,1)–(1,1) entre Acteur et Contrat (ex: un contrat d'assurance spécifique lie exactement un acteur et couvre exactement un contrat de travail). Relations initiales :

- Acteur(Num_A, Prénom, Nom, Adr)
- Contrat(Num_C, Type, Durée)
- Assure(Num_A, Num_C)

Fusion des trois relations :

- **Assurance**(Num_A, Prénom, Nom, Adr, Num_C, Type, Durée)

Clé primaire : Num_A (choix arbitraire). Clé candidate : Num_C (doit être UNIQUE et NOT NULL). Clés étrangères : implicites dans la fusion. Num_A référence Acteur et Num_C référence Contrat, mais comme tout est fusionné, elles deviennent des clés candidates.

Schéma relationnel final :

- Assurance(Num_A, Prénom, Nom, Adr, Num_C, Type, Durée) * Contrainte : Num_C UNIQUE NOT NULL

3.3.3 Cas [M:N] (Cardinalités Max = N et Max = M)

Soit une association A entre E1 et E2 avec des cardinalités maximales M et N (M¿1, N¿1).

- **Règle :** On ne fusionne **pas**. On conserve la relation A créée lors de la Règle 2.
- **Relation A :** Attrs(A) + Clé K1 de E1 + Clé K2 de E2.
- **Clé primaire de A :** La combinaison des clés étrangères {K1, K2}.
- **Clés étrangères :** A(K1) $\subseteq$ E1(K1) et A(K2) $\subseteq$ E2(K2).

C'est le cas traité par la Règle 2 standard, sans raffinement par fusion. L'exemple **Joue**(Num_A, Num_F, Rôle) correspond à ce cas si les cardinalités max sont N et M.

3.4 Cas des Entités Faibles

Une entité faible F est une entité dont l'existence dépend d'une autre entité, dite forte E1. L'entité faible est identifiée par une association AF (dite identifiante) avec l'entité forte E1. La cardinalité de F dans AF est toujours (1,1). L'entité faible possède une clé partielle (ou discriminant).

- **Règle :** On crée une relation RF pour l'entité faible F.
- **Attributs de RF :** Attributs de F + Clé primaire K1 de l'entité forte E1.
- **Clé primaire de RF :** Elle est composite, formée par la clé primaire K1 de l'entité forte E1 et la clé partielle KF de l'entité faible F : {K1, KF}.
- **Clé étrangère :** RF(K1) $\subseteq$ E1(K1).
- **Association Identifiante AF :** Elle n'est **pas** traduite en une relation séparée. Sa sémantique est capturée par l'inclusion de K1 dans la clé primaire de RF.
- **Autres Associations :** Toute autre association (non identifiante) A liée à l'entité faible F doit utiliser la clé primaire **complète** {K1, KF} de RF comme clé étrangère.

Example 3.4.1 (Entité Faible - Cinéma et Salle)**.** Une Salle (entité faible) ne peut exister sans un Cinéma (entité forte). L'association identifiante est "Est-dans" (implicite ou explicite) avec cardinalité (1,1) côté Salle. La clé partielle de Salle est Num_S (numéro de salle dans un cinéma donné).

- Entité Forte : Cinéma(Id_C, Nom_C, Tél)
- Entité Faible : Salle (Attributs : Num_S, Nb_Pl)

Conversion :

- **Cinéma**(Id_C, Nom_C, Tél)
- **Salle**(Id_C, Num_S, Nb_Pl)

Clé primaire de Salle : {Id_C, Num_S}. Clé étrangère : Salle(Id_C) $\subseteq$ Cinéma(Id_C).

Considérons maintenant une association "Projète" entre Salle et Film. Supposons que c'est M:N (une salle projette plusieurs films, un film est projeté dans plusieurs salles).

- Film(Num_F, Titre, Année)

- L'association Projète devient une relation :
  Projète(Id_C, Num_S, Num_F)

Clé primaire de Projète : {Id_C, Num_S, Num_F}. Clés étrangères :

- Projète(Id_C, Num_S) ⊆ Salle(Id_C, Num_S)
- Projète(Num_F) ⊆ Film(Num_F)

Schéma relationnel final :

- Cinéma(Id_C, Nom_C, Tél)
- Film(Num_F, Titre, Année)
- Salle(Id_C, Num_S, Nb_Pl)
- Projète(Id_C, Num_S, Num_F)

3.5 Cas des Associations d'Héritage (ISA)

L'héritage modélise une relation "est-un" (ISA) entre une entité générique (super-classe) et des entités plus spécifiques (sous-classes). Par exemple, Film est une super-classe, et Documentaire et Animation sont des sous-classes.

- **Règle :** On crée une relation pour la super-classe et une relation pour **chaque** sous-classe.
- **Relation Super-classe (ex: Film) :** Traduite selon la Règle 1.
- **Relation Sous-classe (ex: Documentaire) :**
  - **Attributs :** Attributs spécifiques à la sous-classe + Clé primaire de la super-classe.
  - **Clé primaire :** La clé primaire de la super-classe devient la clé primaire de la relation sous-classe.
  - **Clé étrangère :** La clé primaire de la sous-classe référence la clé primaire de la super-classe. SousClasse(Clé) ⊆ SuperClasse(Clé).

Example 3.5.1 (Héritage - Film, Documentaire, Animation).

- Super-classe : Film(Num_F, Titre, Année)
- Sous-classe : Documentaire (Attributs : Sujet, Pays)
- Sous-classe : Animation (Attribut : Style)

Conversion :

- **Film**(Num_F, Titre, Année)
- **Documentaire**(Num_F, Sujet, Pays)
- **Animation**(Num_F, Style)

Clés primaires : Num_F pour les trois relations. Clés étrangères :

- Documentaire(Num_F) ⊆ Film(Num_F)
- Animation(Num_F) ⊆ Film(Num_F)

Chaque documentaire ou animation doit d'abord exister comme film dans la table Film.

3.6 Exemple Complexe

Les diapositives présentent un autre exemple de diagramme EA plus complexe impliquant DEPARTEMENT, EMPLOYE, PROJET, PERIODE, avec diverses associations (CONTROLE, A DIRIGE, TRAVAILLE-POUR, ESTCHEFDE, TRAVAILLEDANS). L'application des règles décrites précédemment (conversion des entités, conversion des associations, raffinement basé sur les cardinalités (ex: (0,n), (1,1), (1,n)), gestion des éventuelles entités faibles ou héritages) permettrait de dériver le schéma relationnel correspondant. La dernière diapositive montre un diagramme relationnel complexe qui pourrait être le résultat d'une telle conversion, illustrant comment de nombreuses relations interconnectées par des clés étrangères peuvent émerger d'un modèle EA détaillé.

3.7 Conclusion

Le passage du modèle Entité-Association au modèle relationnel est un processus structuré basé sur des règles précises. Il implique la transformation des entités et des associations en relations, suivie d'un raffinement essentiel qui prend en compte les contraintes de cardinalité pour optimiser le schéma, souvent en fusionnant des relations. Des cas spécifiques comme les entités faibles et l'héritage nécessitent des traitements particuliers pour garantir l'intégrité et la cohérence du modèle relationnel final. Une bonne compréhension de ces règles permet de concevoir des bases de données relationnelles robustes, non redondantes et qui reflètent fidèlement la sémantique du domaine d'application modélisé initialement avec le modèle EA.

Chapter 4

Modèle de données

CHAPITRE 2 : MODÈLE de DONNÉES

Plan du Chapitre

- Le Modèle Relationnel
- SQL : Langage de définition (LDD)
- Du Modèle E-A au modèle relationnel

4.1 Le Modèle Relationnel

4.1.1 Idée simple

Le modèle de données relationnel repose sur une idée simple pour structurer les données.

- **Schéma** = description des colonnes d'une table.
- **Instance** = lignes (contenu) de la table.

4.1.2 Formalisation simple

La formalisation de ce modèle s'appuie sur des concepts issus de :

- La théorie des ensembles.
- La logique.

Ce modèle a été proposé par Tedd Codd en 1970.

4.1.3 Un petit exemple avant les définitions

Considérons une base de données simple sur le cinéma. Une table pourrait représenter les films :

Example 4.1.1 (Table FILM).

Titre	Metteur-en-scène	Acteur
Speed 2	Jan de Bont	S. Bullock
Speed 2	Jan de Bont	J. Patric
Speed 2	Jan de Bont	W. Dafoe
Marion	M. Poirier	C. Tetard
Marion	M. Poirier	M-F Pisier

4.1.4 Schéma de base de données Cinéma

La structure globale (le schéma) de cette base de données pourrait être :

- **FILM**(Titre, Metteur-en-scène, Acteur)
- **CINE**(Nom-Ciné, Adresse, Téléphone)
- **PROG**(Nom-Ciné, Titre, Horaire)

(Note: Les soulignements indiquent ici les attributs, pas nécessairement les clés à ce stade).

4.2 Définitions Formelles du Modèle Relationnel

4.2.1 Schéma de relation

Definition 4.2.1 (Schéma de relation)**.** Pour décrire la "forme" d'une table (relation), on a besoin :

- D'un **nom de relation** R (par exemple, FILM).
- D'un **ensemble fini d'attributs** $Att(R) = \{A, B, C, ...\}$ (par exemple, {Titre, Metteur-en-scène, Acteur}). L'arité de R est la cardinalité de $Att(R)$.
- Pour chaque attribut A, d'un **domaine** $Dom(A)$, qui est l'ensemble des valeurs possibles pour cet attribut (par exemple, $Dom(\text{Titre}) = \text{string}[20]$).

Notation : On note un schéma de relation $R(A, B, C)$. Par exemple, FILM(Titre, Metteur-en-scène, Acteur).

4.2.2 Schéma d'une base de données relationnelle

Definition 4.2.2 (Schéma de BD relationnelle)**.** Un schéma de base de données relationnelle (BD) est un ensemble fini de schémas de relation.

$$\text{BD} = \{R_1(A_{11}, ..., A_{1n_1}), R_2(A_{21}, ..., A_{2n_2}), ..., R_m(A_{m1}, ..., A_{mn_m})\}$$

Par exemple, pour notre base Cinéma :

$\text{BD}_{\text{Cinema}} = \{\text{FILM(Titre, Metteur-en-scène, Acteur)}, \text{CINE(Nom-Ciné, Adresse, Téléphone)}, \text{PROG(Nom-Ciné, Titre, H}$

On peut aussi avoir un exemple plus abstrait : $\text{BD} = \{R(A, B, C), S(B, D)\}$.

4.2.3 Instance d'un schéma de relation

Definition 4.2.3 (n-uplet)**.** Qu'est-il autorisé de mettre dans une "ligne d'une table" ? Un **n-uplet** u (tuple ou enregistrement) sur un schéma de relation $R(A_1, A_2, ..., A_n)$ est une séquence de n valeurs $(a_1, a_2, ..., a_n)$ telles que pour tout $i = 1..n$, $a_i \in Dom(A_i)$. **Notation :** $u[A_i]$ désigne la composante (valeur) du n-uplet u pour l'attribut A_i. Donc, $u[A_i] = a_i$. *Remarque :* Un n-uplet peut être défini formellement comme une fonction associant à chaque attribut A_i une valeur dans $Dom(A_i)$, ou comme un élément du produit cartésien des domaines $Dom(A_1) \times ... \times Dom(A_n)$.

Definition 4.2.4 (Instance de relation)**.** Qu'est-il autorisé de mettre dans une table ? Une **instance** r d'un schéma de relation $R(A_1, ..., A_n)$ est un **ensemble fini** de n-uplets sur $(A_1, ..., A_n)$. Une instance est donc un sous-ensemble fini du produit cartésien des domaines :

$$r \subseteq Dom(A_1) \times Dom(A_2) \times ... \times Dom(A_n)$$

Une instance représente un état particulier de la table à un moment donné, c'est-à-dire l'ensemble des lignes (le contenu) de la table.

Example 4.2.5 (Instance d'un schéma de relation – exercice)**.** Considérons le schéma de relation STOCK(Habit, Taille) avec :

- $Dom(\text{Habit}) = \{\text{Tailleur, Costume}\}$
- $Dom(\text{Taille}) = \{38, 40\}$

Donnez toutes les instances possibles de STOCK(Habit, Taille).

Solution. Le produit cartésien $Dom(\text{Habit}) \times Dom(\text{Taille})$ est :

$$\{(\text{Tailleur}, 38), (\text{Tailleur}, 40), (\text{Costume}, 38), (\text{Costume}, 40)\}$$

Une instance est un sous-ensemble fini de ce produit cartésien. Il y a $2^4 = 16$ sous-ensembles possibles (donc 16 instances possibles). Voici quelques exemples d'instances :

- L'instance vide : $\emptyset$
- Une instance avec un seul n-uplet : $\{(\text{Tailleur}, 38)\}$
- Une instance avec deux n-uplets : $\{(\text{Tailleur}, 40), (\text{Costume}, 38)\}$
- L'instance "complète" : $\{(\text{Tailleur}, 38), (\text{Tailleur}, 40), (\text{Costume}, 38), (\text{Costume}, 40)\}$
- ... et ainsi de suite pour tous les autres sous-ensembles.

□

4.2.4 Instance d'une base de données

Definition 4.2.6 (Instance de BD)**.** Qu'est-il autorisé de mettre dans une base de données complète ? Une **instance** $\mathcal{I}$ d'un schéma de base de données BD $= \{R_1, ..., R_m\}$ est composée d'une instance r_i pour chaque schéma de relation R_i dans BD, pour $i = 1..m$. Donc, $\mathcal{I} = (r_1, ..., r_m)$. Une instance d'une base de données est un m-uplet d'instances de relations.

Example 4.2.7 (Instance de Base de Données Cinéma)**.** Voici un exemple d'instance pour notre base de données Cinéma.

Instance r_{FILM} de FILM(Titre, Metteur-en-scène, Acteur)

Titre	Metteur-en-scène	Acteur
Speed 2	Jan de Bont	S. Bullock
Speed 2	Jan de Bont	J. Patric
Speed 2	Jan de Bont	W. Dafoe
Marion	M. Poirier	C. Tetard
Marion	M. Poirier	M-F Pisier

Instance r_{PROG} de PROG(Nom-Ciné, Titre, Horaire)

Nom-Ciné	Titre	Horaire
Français	Speed 2	18h00
Français	Speed 2	20h00
Français	Speed 2	22h00
Français	Marion	16h00
Trianon	Marion	18h00

Instance r_{CINE} de CINE(Nom-Ciné, Adresse, Téléphone)

Nom-Ciné	Adresse	Téléphone
Français	9, rue Montesquieu	05 56 44 11 87
Gaumont	9, c. G-Clémenceau	05 56 52 03 54
Trianon	6, r. Franklin	05 56 44 35 17
UGC Ariel	20, r. Judaique	05 56 44 31 17

L'instance $\mathcal{I}_{\text{Cinema}}$ de la base de données est alors le triplet $(r_{FILM}, r_{PROG}, r_{CINE})$.

4.3 Contraintes d'intégrité - Dépendances

Une base de données ne contient pas seulement des données brutes, mais elle doit aussi respecter certaines règles pour garantir la cohérence et la validité de l'information.

- Une base contient des informations.
- L'information n'est pas simplement les données stockées dans les tables.
- L'information est approximativement : données + contraintes.

4.3.1 Différents types de contraintes

- **Contraintes structurelles (liées au modèle)** : Ces contraintes découlent directement de la définition du modèle relationnel. Par exemple, la valeur d'un attribut pour un n-uplet doit appartenir au domaine de cet attribut : $u[A_i] \in Dom(A_i)$.
- **Contraintes d'intégrité liées à l'application** : Ce sont des propriétés spécifiques que les données doivent satisfaire pour refléter correctement la réalité modélisée. Elles doivent être vérifiées à chaque état de la base de données, notamment lors des mises à jour (insertions, modifications).
- **Contraintes dynamiques** : Elles portent sur l'évolution de la base de données, c'est-à-dire les transitions entre états (non abordé en détail ici).

4.3.2 Exemples de Contraintes liées à l'application

Example 4.3.1 (Contrainte 1 : Unicité des coordonnées par cinéma)**. Mini-Application :** On souhaite garantir qu'il y a *une seule adresse* et *un seul numéro de téléphone* par cinéma.

Considérons l'instance r_{CINE} suivante :

Nom-Ciné	Adresse	Téléphone
Français	9, rue Montesquieu	05 56 44 11 87
Français	9, rue Montesquieu	08 01 68 04 45
UGC	20, rue Judaique	05 56 44 31 17
UGC	20, rue Judaique	08 01 68 70 14
UGC	22, rue Judaique	08 01 68 70 14

Cette instance est *bien une instance* de CINE (chaque ligne respecte les types de données définis par les domaines). Cependant, elle n'est **pas cohérente** avec la règle de l'application : le cinéma "Français"

a deux numéros de téléphone différents, et le cinéma "UGC" a plusieurs adresses et numéros. Elle ne satisfait pas la propriété souhaitée.

Une instance **cohérente** serait :

Nom-Ciné	Adresse	Téléphone
Français	9, rue Montesquieu	05 56 44 11 87
UGC	20, rue Judaique	05 56 44 31 17

Cette instance est bien une instance de CINE et elle satisfait la propriété (un seul tuple par Nom-Ciné).

Example 4.3.2 (Contrainte 2 : Référencement des films projetés)**. Mini-Application :** Les films projetés (dans la table PROG) doivent être des films répertoriés (dans la table FILM).

Considérons les instances r_{PROG} et r_{FILM} suivantes : **Instance r_{PROG} (partielle)**

Nom-Ciné	Titre	Horaire
Français	Speed 2	18h00
Français	Speed 2	20h00
Français	Western	18h00
Français	Western	20h00

Instance r_{FILM} (partielle)

Titre	Metteur-en-scène	Acteur
Speed 2	Jan de Bont	S. Bullock
Speed 2	Jan de Bont	J. Patric
Speed 2	Jan de Bont	W. Dafoe
Marion	M. Poirier	C. Tetard
Marion	M. Poirier	M-F Pisier

Ces instances sont bien des instances de PROG et FILM respectivement. Cependant, l'instance globale n'est **pas cohérente** avec la règle de l'application, car le film "Western" est projeté (dans r_{PROG}) mais n'existe pas dans la liste des films répertoriés (r_{FILM}). Elle ne satisfait pas la propriété.

Pour rendre l'instance **cohérente**, il faudrait soit retirer les projections de "Western", soit ajouter "Western" à la table FILM. Par exemple, si on ajoute "Western" à FILM : **Instance r'_{FILM} (cohérente avec r_{PROG})**

Titre	Metteur-en-scène	Acteur
Speed 2	Jan de Bont	S. Bullock
Speed 2	Jan de Bont	J. Patric
Speed 2	Jan de Bont	W. Dafoe
Marion	M. Poirier	C. Tetard
Marion	M. Poirier	M-F Pisier
Western	M. Poirier	M-F Pisier

Avec r'_{FILM} et r_{PROG}, l'instance globale est cohérente.

4.4 Dépendances Fonctionnelles (DF)

Les dépendances fonctionnelles sont un type de contrainte d'intégrité très important, souvent utilisé pour définir les clés. Elles expriment des relations de détermination entre ensembles d'attributs au sein d'une *même* relation. (Elles correspondent aux contraintes de l'exemple 1).

Definition 4.4.1 (Dépendance Fonctionnelle)**.** Soit $R(A_1, ..., A_n)$ un schéma de relation, et X, Y deux sous-ensembles des attributs de R ($X, Y \subseteq \{A_1, ..., A_n\}$). Une **dépendance fonctionnelle (DF)** est

une assertion de la forme :

$$X \to Y$$

On lit : "X détermine fonctionnellement Y" ou "Y dépend fonctionnellement de X".

Sémantique : Une instance r de R satisfait la DF $X \to Y$ (noté $r \models X \to Y$) si et seulement si : Pour tout couple de n-uplets u, v dans r,

$$\text{Si } u[A] = v[A] \text{ pour tout } A \in X, \text{ alors } u[B] = v[B] \text{ pour tout } B \in Y.$$

Autrement dit : Si deux tuples coïncident sur tous les attributs de X, alors ils doivent aussi coïncider sur tous les attributs de Y.

$$(\forall u, v \in r) \quad [(\forall A \in X, u[A] = v[A]) \implies (\forall B \in Y, u[B] = v[B])]$$

Remark 4.4.2 (DF Triviales). Une DF $X \to Y$ est dite **triviale** si $Y \subseteq X$. Les DF triviales sont toujours satisfaites par n'importe quelle instance, car si deux tuples coïncident sur X, ils coïncident a fortiori sur toute partie Y de X. Par exemple, $A \to A$ ou $AB \to A$.

Example 4.4.3 (Reprise de l'Exemple 1). La contrainte "une seule adresse par cinéma" sur la relation CINE(Nom-Ciné, Adresse, Téléphone) s'écrit :

$$\{\text{Nom-Ciné}\} \to \{\text{Adresse}\}$$

Vérification sur l'instance (incohérente) de la page 11 : Soient $u = (\text{UGC}, 20, \text{rue Judaique}, 0556443117)$ et $v = (\text{UGC}, 22, \text{rue Judaique}, 0801687014)$. On a $u[\text{Nom-Ciné}] = v[\text{Nom-Ciné}] = \text{UGC}$. Mais $u[\text{Adresse}] = 20, \text{rue Judaique} \neq v[\text{Adresse}] = 22, \text{rue Judaique}$. La DF n'est pas satisfaite par cette instance.

La contrainte "un seul numéro de téléphone par cinéma" s'écrit :

$$\{\text{Nom-Ciné}\} \to \{\text{Téléphone}\}$$

Vérification sur l'instance (incohérente) de la page 11 : Soient $u = (\text{Français}, 9, \text{rue Montesquieu}, 0556441187)$ et $v = (\text{Français}, 9, \text{rue Montesquieu}, 0801680445)$. On a $u[\text{Nom-Ciné}] = v[\text{Nom-Ciné}] = \text{Français}$. Mais $u[\text{Téléphone}] = 0556441187 \neq v[\text{Téléphone}] = 0801680445$. La DF n'est pas satisfaite par cette instance.

Example 4.4.4 (Exercice sur les DF). L'instance r de $R(A, B, C, D)$ ci-dessous satisfait-elle les DFs suivantes ?

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

1. $A \to C$? Oui. Pour $A = a_1$, C est toujours c_1. Pour $A = a_2$, C est toujours c_2. Pour $A = a_3$, C est c_2.
2. $C \to A$? Non. Pour $C = c_1$, A est a_1. Mais pour $C = c_2$, A peut être a_2 ou a_3. (Contre-exemple : les deux dernières lignes ont $C = c_2$ mais A différent).
3. $AB \to D$? Oui. Chaque paire (A, B) unique correspond à une seule valeur de D. ($a_1, b_1 \to d_1$; $a_1, b_2 \to d_2$; $a_2, b_2 \to d_2$; $a_2, b_3 \to d_3$; $a_3, b_3 \to d_4$).

4. $A \to A$? Oui. C'est une DF triviale ($Y \subseteq X$).

4.5 Dépendance de Clé

La notion de clé est fondamentale dans les bases de données relationnelles. Une clé permet d'identifier de manière unique chaque n-uplet (ligne) dans une instance de relation. Elle est définie à l'aide des dépendances fonctionnelles.

Definition 4.5.1 (Clé Candidate)**.** Soit $R(A_1, ..., A_n)$ un schéma de relation. Un sous-ensemble d'attributs $X \subseteq \{A_1, ..., A_n\}$ est une **clé candidate** (ou simplement une **clé**) de R si :

1. $X \to \{A_1, ..., A_n\}$ (c'est-à-dire, X détermine fonctionnellement tous les attributs de R).
2. X est **minimale** : aucun sous-ensemble propre de X ne détermine tous les attributs de R. (Si on enlève un attribut de X, la propriété 1 n'est plus vraie).

Sémantique : Si X est une clé de R, alors dans toute instance r de R qui satisfait les DFs définissant la clé :

- Si deux n-uplets $u, v \in r$ ont les mêmes valeurs pour les attributs de X ($u[X] = v[X]$), alors ces n-uplets doivent être identiques ($u = v$).
- Donner des valeurs pour les attributs de X permet de trouver *au plus* un n-uplet dans r.

Généralement, une relation a une ou plusieurs clés candidates. L'une d'elles est choisie comme **clé primaire**.

Remark 4.5.2. La condition 1 ($X \to \{A_1, ..., A_n\}$) est équivalente à dire que pour tout attribut $A \in Att(R) \setminus X$, on a $X \to A$.

4.6 Dépendances d'Inclusion (DI)

Les dépendances d'inclusion permettent d'exprimer des contraintes entre *plusieurs* relations, typiquement pour assurer l'intégrité référentielle (comme dans l'exemple 2, où un film projeté doit exister dans la table des films). Elles sont la base des clés étrangères.

Definition 4.6.1 (Dépendance d'Inclusion)**.** Soient $R(A_1, ..., A_n)$ et $S(B_1, ..., B_m)$ deux schémas de relation (R et S peuvent être identiques). Soient $X = (A_{\alpha_1}, ..., A_{\alpha_k})$ une séquence d'attributs de R et $Y = (B_{\beta_1}, ..., B_{\beta_k})$ une séquence d'attributs de S de même longueur k. Une **dépendance d'inclusion (DI)** est une assertion de la forme :

$$R[X] \subseteq S[Y]$$

ou plus explicitement

$$R[A_{\alpha_1}, ..., A_{\alpha_k}] \subseteq S[B_{\beta_1}, ..., B_{\beta_k}]$$

On lit : "Les valeurs de X dans R doivent être présentes comme valeurs de Y dans S".

Sémantique : Un couple d'instances (r, s) des relations R et S satisfait la DI $R[X] \subseteq S[Y]$ (noté $r, s \models R[X] \subseteq S[Y]$) si et seulement si : Pour tout n-uplet $u \in r$, il existe (au moins) un n-uplet $v \in s$ tel que :

$$u[A_{\alpha_j}] = v[B_{\beta_j}] \quad \text{pour tout } j = 1..k$$

Autrement dit, la projection de r sur les attributs X doit être un sous-ensemble de la projection de s sur les attributs Y.

$$\pi_X(r) \subseteq \pi_Y(s)$$

Condition importante : Pour que la DI ait un sens, les domaines des attributs correspondants doivent être **compatibles** (par exemple, $Dom(A_{\alpha_j})$ doit être comparable ou égal à $Dom(B_{\beta_j})$ pour tout j).

Remark 4.6.2. L'exemple $CINE$[Nom-Ciné] $\subseteq$ $CINE$[Téléphone] n'a pas de sens car les domaines (chaînes de caractères pour les noms vs. numéros de téléphone) ne sont pas sémantiquement compatibles pour cette comparaison.

Example 4.6.3 (Reprise de l'Exemple 2)**.** La contrainte "Les films projetés sont des films répertoriés" s'écrit avec une DI :

$$\text{PROG[Titre]} \subseteq \text{FILM[Titre]}$$

Cela signifie que pour chaque ligne u dans la table PROG, la valeur u[Titre] doit exister comme valeur v[Titre] pour au moins une ligne v dans la table FILM.

4.7 Clé étrangère ou Contrainte de référence

La combinaison d'une dépendance de clé et d'une dépendance d'inclusion forme le concept très important de clé étrangère, qui est le mécanisme standard pour relier les tables dans une base de données relationnelle tout en maintenant l'intégrité référentielle.

Definition 4.7.1 (Clé Étrangère)**.** Soient $R(..., A, B, C, ...)$ et $S(..., A', B', C', ...)$ deux schémas de relation (R et S peuvent être la même relation). On suppose que les attributs A, B, C sont compatibles avec A', B', C' respectivement. L'ensemble d'attributs $\{A', B', C'\}$ de S est une **clé étrangère** référençant la clé $\{A, B, C\}$ de R si les deux conditions suivantes sont satisfaites :

1. $\{A, B, C\}$ est une **clé candidate** (généralement la clé primaire) de R.
2. La dépendance d'inclusion suivante est satisfaite :

$$S[A', B', C'] \subseteq R[A, B, C]$$

Cela garantit que toute valeur (ou combinaison de valeurs) apparaissant dans les colonnes A', B', C' de la table S doit également exister dans les colonnes clés A, B, C de la table R.

Remark 4.7.2. Cette définition combine une contrainte de clé (sur R) et une contrainte d'inclusion (entre S et R). La notion de clé étrangère est directement supportée en SQL pour définir les liens entre tables.

Source: cours de Nicole Bidoit 2022-2023

Chapter 5

Algèbre relationnel

CHAP. 4 : LANGAGES de REQUÊTES

Plan du Chapitre

- L'algèbre relationnelle
- SQL : Langage de requête (mentionné, mais non détaillé ici)
- SQL : Langage de mise à jour (mentionné, mais non détaillé ici)

source : cours N. Bidoit 22-23

5.1 Langage & Base de Données

Les systèmes de gestion de bases de données fournissent des langages pour interagir avec les données. On distingue principalement trois types de langages :

- **Langage de définition de données** : Permet la spécification du schéma de la base de données (création des tables, définition des attributs et de leurs types, contraintes, etc.).
- **Langage d'interrogation** : Permet la spécification des requêtes pour extraire de l'information de la base de données.
- **Langage de mise à jour** : Permet la spécification des modifications à apporter aux données stockées (insertion, suppression, modification).

5.2 Langages d'interrogation

Interroger une base de données consiste à **déduire des informations** à partir de celles qui y sont stockées.

Exemples de requêtes

Requêtes simples :

- Dans quels films a joué I. Adjani ?
- Quels sont les films à l'affiche ?
- Quels sont les acteurs qui ont joué dans *l'Été meurtrier* ?

Requêtes un peu plus complexes :

- Qui a joué avec I. Adjani (dans un de ses films) ?
- Quels films de J. Becker sont à l'affiche ?
- Quel est le cinéma le plus proche qui programme un film avec I. Adjani ?
- Quel cinéma projette tous les films de J. Becker ?
- Quel est, en moyenne et par an sur les 5 dernières années, le nombre de prix gagnés par des acteurs français dans des festivals étrangers?

Approches des langages d'interrogation

Il existe deux principales approches pour les langages d'interrogation :

- **Langage déclaratif** : On spécifie **quoi** extraire. Le système détermine comment le faire.
  - Basé sur le calcul (à variable domaine ou n-uplet).
  - Exemple : SQL.
  - Analogie : "Donnez-moi une tartine de confiture".
- **Langage procédural** : On spécifie **quoi** extraire **et comment** le faire, étape par étape.
  - Basé sur l'algèbre.
  - Exemple : Algèbre relationnelle.
  - Analogie : "Coupez une tranche de pain, ouvrez le pot de confiture de fraises, étalez la confiture sur le pain, ...".

Pourquoi apprendre l'algèbre relationnelle ?

Bien que SQL (déclaratif) soit le langage le plus utilisé pour spécifier des requêtes, l'algèbre relationnelle (procédurale) est fondamentale pour comprendre comment ces requêtes sont **évaluées** par le système de gestion de base de données.

- Spécifier une requête $\rightarrow$ utiliser SQL (déclaratif)
- Évaluer une requête $\rightarrow$ utiliser l'algèbre relationnelle (procédural)

5.3 Algèbre relationnelle : introduction des opérateurs

L'algèbre relationnelle définit un ensemble d'opérateurs qui agissent sur des relations (tables) pour en produire de nouvelles.

Types d'opérateurs

Opérateurs pour requêtes simples :

- 3 opérateurs unaires (agissant sur une seule relation) :
  - **Extraction** : Sélectionner une table entière.
  - **Projection** (Π) : Sélectionner certaines colonnes.
  - **Sélection** (σ) : Sélectionner certaines lignes.

Opérateurs pour requêtes complexes :

- 3 opérateurs binaires (agissant sur deux relations) :
  - **Jointure** ($\bowtie$) : Combiner deux tables basées sur des colonnes communes ou une condition.
  - **Union** ($\cup$) : Combiner les lignes de deux tables compatibles.
  - **Différence** ($-$) : Obtenir les lignes d'une table qui ne sont pas dans une autre table compatible.
- Opérateurs dérivés (exprimables à partir des précédents) :
  - **Produit cartésien** ($\times$)
  - **Intersection** ($\cap$)
  - **Division** ($\div$) (utile pour les requêtes "pour tous")
  - **Renommage** (ρ)

5.3.1 Typage des opérateurs

Chaque opérateur de l'algèbre relationnelle définit une **application** de l'ensemble des instances d'un ou plusieurs **schémas source** dans l'ensemble des instances d'un **schéma cible**.

- **Schéma source** : Le(s) schéma(s) de(s) relation(s) en entrée de l'opérateur. Peut être constitué de plusieurs schémas de relation pour les opérateurs binaires.
- **Schéma cible** : Le schéma de la relation résultante (en sortie). Est toujours un schéma de relation unique.

Le schéma cible (le format du résultat) est déterminé par :

1. Le(s) schéma(s) source(s).
2. L'opérateur utilisé.

5.3.2 Exemple de base de données

Considérons une base de données simple sur des films, cinémas et leur programmation.

Example 5.3.1 (Schémas de la base de données). • FILM(¡u¿Titre¡/u¿, ¡u¿Metteur-en-scène¡/u¿, ¡u¿Acteur¡/u¿)

• CINE(¡u¿Nom-Ciné¡/u¿, Adresse, Téléphone)

• PROGR.(¡u¿Nom-Ciné¡/u¿, ¡u¿Titre¡/u¿, ¡u¿Horaire¡/u¿)

Note: Les clés primaires sont soulignées (ici, de manière simplifiée, on suppose des dépendances qui ne sont pas forcément réalistes, ex: plusieurs acteurs par film/metteur en scène créent des répétitions).

Example 5.3.2 (Instances des relations). **Relation FILM (instance 'film')**

Titre	Metteur-en-scène	Acteur
Speed 2	Jan de Bont	S. Bullock
Speed 2	Jan de Bont	J. Patric
Speed 2	Jan de Bont	W. Dafoe
Marion	M. Poirier	C. Tetard
Marion	M. Poirier	M-F Pisier

Relation CINE (instance 'ciné')

Nom-Ciné	Adresse	Téléphone
Français	9, rue Montesquieu	05 56 44 11 87
Gaumont	9, c. G-Clémenceau	05 56 52 03 54
Trianon	6, r. Franklin	05 56 44 35 17
UGC Ariel	20, r. Judaïque	05 56 44 31 17

Relation PROGR. (instance 'programmation')

Nom-Ciné	Titre	Horaire
Français	Speed 2	18h00
Français	Speed 2	20h00
Français	Speed 2	22h00
Français	Marion	16h00
Trianon	Marion	18h00

5.4 Les opérateurs unaires

5.4.1 L'opérateur d'extraction

Definition 5.4.1 (Opérateur d'extraction)**.** L'opérateur d'extraction permet simplement de sélectionner une table (relation) parmi celles de la base de données.

- **Schéma source** : Le schéma global de la base de données $BD = \{R_1, R_2, \ldots, R_n\}$, où R_i est un schéma de relation avec les attributs $Att(R_i) = W_i$.
- **Syntaxe** : $[R_i]$
- **Application** : Prend une instance $bd = (r_1, r_2, \ldots, r_n)$ de BD et retourne une instance r_i de R_i.
- **Schéma cible** : $Res(W_i)$, qui est identique à $R_i(W_i)$.
- **Sémantique** : $[R_i](bd) = r_i$.

Example 5.4.2 (Extraction de la programmation)**.** Question : Tout sur la programmation des films ? Expression : $[PROGR.]$

Résultat (sur l'instance 'bd' = ('film', 'ciné', 'programmation')) : L'instance 'programmation'.

Nom-Ciné	Titre	Horaire
Français	Speed 2	18h00
Français	Speed 2	20h00
Français	Speed 2	22h00
Français	Marion	16h00
Trianon	Marion	18h00

5.4.2 L'opérateur de projection (Π)

Cet opérateur effectue une opération "verticale" : il supprime des colonnes d'une table et élimine les doublons éventuels dans les lignes résultantes.

Definition 5.4.3 (Opérateur de projection)**.**

- **Opération** : Unaire, verticale.
- **Schéma source** : Une relation $R(V)$, où V est l'ensemble des attributs.
- **Paramètre** : Un sous-ensemble d'attributs $W \subseteq V$.

- **Syntaxe** : Π_W
- **Application** : Prend une instance r de $R(V)$ et retourne une instance de $Res(W)$.
- **Schéma cible** : $Res(W)$.
- **Sémantique** : Soit r une instance de R. $\Pi_W(r) = \{u[W] \mid u \in r\}$. Où $u[W]$ désigne le n-uplet u restreint aux attributs de W. L'ensemble résultant ne contient pas de doublons.

Example 5.4.4 (Projection sur Titre). Question : Les titres des films ? Expression : $\Pi_{Titre}([FILM])$
Appliqué à l'instance 'film' :

Titre	M-en-S	Acteur
Speed 2	Jan de Bont	S. Bullock
Speed 2	Jan de Bont	J. Patric
Speed 2	Jan de Bont	W. Dafoe
Marion	M. Poirier	C. Tetard
Marion	M. Poirier	M-F Pisier

$\downarrow \Pi_{Titre}$

Titre
Speed 2
Marion

Résultat sans redondance.

Example 5.4.5 (Projection sur Titre des films à l'affiche). Question : Les titres des films à l'affiche ? Expression : $\Pi_{Titre}([PROGR.])$
Appliqué à l'instance 'programmation' :

Nom-Ciné	Titre	Horaire
Français	Speed 2	18h00
Français	Speed 2	20h00
Français	Speed 2	22h00
Français	Marion	16h00
Trianon	Marion	18h00

$\downarrow \Pi_{Titre}$

Titre
Speed 2
Marion

Remark 5.4.6 (Cas limite : Projection sur l'ensemble vide d'attributs $\Pi_\emptyset$).
- Quel est le schéma cible ? $S(\emptyset)$. Il n'y a qu'un seul attribut : l'ensemble vide.
- Quelles sont les instances possibles de $S(\emptyset)$?
  - $s_1 = \{()\}$: l'ensemble contenant le 0-uplet (unique). Sémantiquement "Vrai".
  - $s_2 = \{\}$: l'ensemble vide. Sémantiquement "Faux".
- Quel est le résultat de $\Pi_\emptyset(r)$?
  - Si l'instance r de R est **non vide**, alors $\Pi_\emptyset(r) = \{()\}$ ("Vrai").

  - Si l'instance r de R est **vide**, alors $\Pi_\emptyset(r) = \{\}$ ("Faux").

- Utilité : Permet d'exprimer des requêtes OUI/NON.
- Exemple : Y a-t-il des films à l'affiche ? $\Pi_\emptyset([PROGR.])$ (Si le résultat est $\{()\}$, la réponse est oui, si c'est $\{\}$, la réponse est non).

5.4.3 L'opérateur de sélection (σ)

Cet opérateur effectue une opération "horizontale" : il garde seulement les lignes (n-uplets) qui satisfont une certaine condition (critère).

Definition 5.4.7 (Opérateur de sélection)**.**

- **Opération** : Unaire, horizontale.
- **Schéma source** : Une relation $R(V)$.
- **Paramètre** : Une condition C sur les attributs de V.
- **Syntaxe** : σ_C
- **Application** : Prend une instance r de $R(V)$ et retourne une instance de $Res(V)$.
- **Schéma cible** : $Res(V)$ (le schéma ne change pas).
- **Sémantique** : Soit r une instance de R. $\sigma_C(r) = \{u \mid u \in r \text{ et } u \text{ satisfait } C\}$.

Definition 5.4.8 (Conditions de sélection)**.** Soit V un ensemble d'attributs.

- Une **condition élémentaire** sur V est de la forme :
  - A comp a : Comparaison d'un attribut $A \in V$ avec une constante a du domaine de A ($a \in Dom(A)$).
  - A comp B : Comparaison de deux attributs $A, B \in V$ ayant le même domaine ($Dom(A) = Dom(B)$).
  - *comp* est un opérateur de comparaison ($=, \neq, <, \leq, >, \geq$).
- Une **condition** sur V est définie récursivement :
  1. Toute condition élémentaire est une condition.
  2. Si C_1 et C_2 sont des conditions sur V, alors $C_1 \wedge C_2$ (ET logique), $C_1 \vee C_2$ (OU logique), et $\neg C_1$ (NON logique) sont aussi des conditions sur V.

La satisfaction d'une condition C par un n-uplet u ("u satisfait C") est définie de façon intuitive.

Example 5.4.9 (Sélection de films par titre)**.** Question : Tout sur les films dont le titre est "Speed2" ? Critère de sélection : $Titre$ = 'Speed2' Expression : $\sigma_{Titre=\text{'Speed2'}}([FILM])$

Appliqué à l'instance 'film' :

Titre	M-en-S	Acteur
Speed 2	Jan de Bont	S. Bullock
Speed 2	Jan de Bont	J. Patric
Speed 2	Jan de Bont	W. Dafoe
Marion	*M. Poirier*	*C. Tetard*
Marion	*M. Poirier*	*M-F Pisier*

$\downarrow \sigma_{Titre=\text{'Speed2'}}$

Titre	M-en-S	Acteur
Speed 2	Jan de Bont	S. Bullock
Speed 2	Jan de Bont	J. Patric
Speed 2	Jan de Bont	W. Dafoe

Example 5.4.10 (Sélection de films où metteur en scène = acteur)**.** Question : Qui a été son propre metteur en scène ? et dans quel film ? Critère de sélection : $MeS = Acteur$ Expression : $\sigma_{MeS=Acteur}([FILM])$

Appliqué à l'instance 'film' :

Titre	M-en-S	Acteur
Speed 2	*Jan de Bont*	*S. Bullock*
Speed 2	*Jan de Bont*	*J. Patric*
Speed 2	*Jan de Bont*	*W. Dafoe*
Marion	*M. Poirier*	*C. Tetard*
Marion	M. Poirier	M-F Pisier
Marion	*M. Poirier*	*M. Poirier*

$\downarrow \sigma_{MeS=Acteur}$

Titre	M-en-S	Acteur
Marion	M. Poirier	M. Poirier

(Note: Dans l'instance fournie initialement, ce résultat serait vide.)

Example 5.4.11 (Sélection dans la programmation)**.** Question : Tout sur la programmation après 20h00 du film "Marion". Critère de sélection : $Titre$ = 'Marion' $\wedge$ $Horaire \geq$ '20h00' Expression : $\sigma_{Titre='\text{Marion}'\wedge Horaire\geq'\text{20h00}'}([PROGR.])$

Appliqué à l'instance 'programmation' :

Nom-Ciné	Titre	Horaire
Français	*Speed 2*	*18h00*
Français	*Speed 2*	*20h00*
Français	Speed 2	22h00
Français	*Marion*	*16h00*
Trianon	*Marion*	*18h00*
Trianon	Marion	22h00

$\downarrow \sigma_{Titre='\text{Marion}'\wedge Horaire\geq'\text{20h00}'}$

Nom-Ciné	Titre	Horaire
Trianon	Marion	22h00

(Note: Dans l'instance fournie initialement, ce résultat serait vide.)

Example 5.4.12 (Sélection avec OU)**.** Question : La programmation des films dont le titre est "Marion" OU à l'affiche de l'UGC. Critère de sélection : $Titre$ = 'Marion' $\vee$ $Nom\text{-}Cine$ = 'UGC Ariel' (en supposant que 'UGC' signifie 'UGC Ariel') Expression : $\sigma_{Titre='\text{Marion}'\vee Nom\text{-}Cine='\text{UGC Ariel}'}([PROGR.])$

Appliqué à l'instance 'programmation' (avec ajout d'une ligne UGC pour l'exemple):

Nom-Ciné	Titre	Horaire
Français	Speed 2	18h00
Français	Speed 2	20h00
Français	Speed 2	22h00
Français	Marion	16h00
Trianon	Marion	18h00
Trianon	Marion	22h00
UGC Ariel	Speed 2	22h00

$\downarrow \sigma_{Titre=\text{'Marion'} \vee Nom\text{-}Cine=\text{'UGC Ariel'}}$

Nom-Ciné	Titre	Horaire
Français	Marion	16h00
Trianon	Marion	18h00
Trianon	Marion	22h00
UGC Ariel	Speed 2	22h00

5.4.4 Composition des opérateurs

Les opérateurs de l'algèbre relationnelle peuvent être composés, car la sortie d'un opérateur (une relation) peut servir d'entrée à un autre opérateur.

- Opérateur = application
- Composition d'opérateurs = composition d'applications

Definition 5.4.13 (Expression algébrique)**.** Une **expression algébrique** est définie récursivement :

- $[R]$ est une expression algébrique (extraction), où R est un schéma de relation avec $Att(R) = V$. Son schéma source est BD, son schéma cible est $RES_E(V) = R(V)$.
- Si E est une expression algébrique dont le schéma cible est $RES_E(V)$, alors :
  - $\Pi_W[E]$ est une expression algébrique (projection) si $W \subseteq V$. Son schéma source est $RES_E(V)$, son schéma cible est $RES_1(W)$.
  - $\sigma_C[E]$ est une expression algébrique (sélection) si C est une condition sur V. Son schéma source est $RES_E(V)$, son schéma cible est $RES_2(V)$.

(Cette définition sera complétée avec les opérateurs binaires).

Example 5.4.14 (Composition : Extraction, Sélection, Projection)**.** Question : Les noms des cinémas qui projettent le film Marion après 20h00, avec l'horaire exact.

Étapes : 1. **Extraction** de la table de programmation : $[PROGR.]$ Résultat : 'programmation' (table entière) Schéma : (Nom-Ciné, Titre, Horaire) 2. **Sélection** des lignes concernant Marion après 20h00 : $\sigma_{Titre=\text{'Marion'} \wedge Horaire \geq \text{'20h00'}}([PROGR.])$ Résultat intermédiaire ('RES-int') : Lignes de 'programmation' satisfaisant la condition. Schéma : (Nom-Ciné, Titre, Horaire) Instance (avec ajout hypothétique) :

Nom-Ciné	Titre	Horaire
Trianon	Marion	22h00

3. **Projection** sur les attributs désirés (Nom-Ciné, Horaire) : $\Pi_{Nom\text{-}Cine,Horaire}[\sigma_{Titre=\text{'Marion'} \wedge Horaire \geq \text{'20h00'}}([PROGR.])]$ Résultat final ('RES-final') : Schéma : (Nom-Ciné, Horaire) Instance (avec ajout hypothétique) :

Nom-Ciné	Horaire
Trianon	22h00

Expression algébrique complète :

$$\Pi_{Nom\text{-}Cine,Horaire}[\sigma_{Titre=\text{'Marion'}\wedge Horaire\geq\text{'20h00'}}([PROGR.])](bd)$$

Les opérateurs unaires (extraction, projection, sélection) permettent de répondre aux requêtes simples. C'est fait !

5.5 Les opérateurs binaires

Pour répondre à des requêtes plus complexes nécessitant de combiner des informations issues de plusieurs tables, nous avons besoin d'opérateurs binaires.

5.5.1 L'opérateur de jointure ($\bowtie$)

La jointure est l'opérateur principal pour combiner des informations de deux relations. Elle assemble les n-uplets des deux relations qui "correspondent" sur leurs attributs communs.

Definition 5.5.1 (Jointure naturelle).

- **Opération** : Binaire.
- **But** : Combiner le contenu de deux instances en se servant des valeurs dans les colonnes communes.
- **Schémas source** : Deux relations $R(V)$ et $S(W)$.
- **Syntaxe** : $[R] \bowtie [S]$ (ou simplement $R \bowtie S$ pour les instances r et s)
- **Application** : Prend un couple d'instances (r, s) de R et S et retourne une instance de $Res(V \cup W)$.
- **Schéma cible** : $Res(V \cup W)$. Le schéma résultant contient tous les attributs de R et tous ceux de S, en ne gardant qu'une seule copie des attributs communs $(V \cap W)$.
- **Sémantique** : $r \bowtie s = \{u \mid \exists u_r \in r, \exists u_s \in s$ tels que $u[V] = u_r$ et $u[W] = u_s\}$. Autrement dit, un n-uplet u est dans le résultat si sa partie correspondant aux attributs de R $(u[V])$ existe dans r ET sa partie correspondant aux attributs de S $(u[W])$ existe dans s. Cela implique que pour les attributs communs $(A \in V \cap W)$, les valeurs doivent être égales $(u_r[A] = u_s[A])$.

Example 5.5.2 (Jointure générique). Soient les instances r de $R(A, B, C)$ et s de $S(B, C, D)$. L'attribut commun est $\{B, C\}$. Le schéma résultat est (A, B, C, D).

Instance r:

A	B	C
a1	b1	c1
a2	b1	c1
a2	b2	c2
a3	b2	c3

Instance s:

B	C	D
b1	c1	d1
b2	c3	d3
b4	c2	d1

Calcul de $r \bowtie s$:

- Ligne 1 de r (a1, b1, c1) : Cherche dans s des lignes avec (b1, c1). Trouvé : Ligne 1 de s (b1, c1, d1). Combinaison : (a1, b1, c1, d1).
- Ligne 2 de r (a2, b1, c1) : Cherche dans s des lignes avec (b1, c1). Trouvé : Ligne 1 de s (b1, c1, d1). Combinaison : (a2, b1, c1, d1).
- Ligne 3 de r (a2, b2, c2) : Cherche dans s des lignes avec (b2, c2). Non trouvé.
- Ligne 4 de r (a3, b2, c3) : Cherche dans s des lignes avec (b2, c3). Trouvé : Ligne 2 de s (b2, c3, d3). Combinaison : (a3, b2, c3, d3).

Résultat $r \bowtie s$:

A	B	C	D
a1	b1	c1	d1
a2	b1	c1	d1
a3	b2	c3	d3

Example 5.5.3 (Jointure FILM et PROGR.)**.** Question : Les cinémas où on peut aller voir un film dans lequel joue M.F Pisier ; pour chaque cinéma donner le(s) titre(s) du(es) film(s) et l'horaire(s) et l'adresse du cinéma.

Analyse :

- Information sur les acteurs → table FILM.
- Information sur la programmation (cinéma, titre, horaire) → table PROGR.
- Information sur l'adresse du cinéma → table CINE.
- Il faut combiner ces informations.

Étapes algébriques : 1. Sélectionner les films où joue M.F Pisier : $E_1 = \sigma_{Acteur='\text{M-F Pisier}'}([FILM])$ Résultat e_1:

Titre	M-en-S	Acteur
Marion	M. Poirier	M-F Pisier

Schéma : (Titre, M-en-S, Acteur) 2. Faire la jointure entre ces films (e_1) et la programmation ($[PROGR.]$) sur l'attribut commun 'Titre'. $E_2 = E_1 \bowtie [PROGR.]$ Résultat e_2:

Titre	M-en-S	Acteur	Nom-Ciné	Horaire
Marion	M. Poirier	M-F Pisier	Français	16h00
Marion	M. Poirier	M-F Pisier	Trianon	18h00
Marion	M. Poirier	M-F Pisier	Trianon	22h00

(avec ligne ajoutée)

Schéma : (Titre, M-en-S, Acteur, Nom-Ciné, Horaire) 3. Faire la jointure du résultat précédent (e_2) avec les cinémas ($[CINE]$) sur l'attribut commun 'Nom-Ciné'. $E_3 = E_2 \bowtie [CINE]$ Résultat e_3:

Titre	M-en-S	...	Nom-Ciné	Horaire	Adresse	Téléphone
Marion	M. Poirier	...	Français	16h00	9, rue Montesquieu	...
Marion	M. Poirier	...	Trianon	18h00	6, r. Franklin	...
Marion	M. Poirier	...	Trianon	22h00	6, r. Franklin	...

Schéma : (Titre, M-en-S, Acteur, Nom-Ciné, Horaire, Adresse, Téléphone) 4. Projeter sur les attributs demandés : Titre, Horaire, Adresse. $E_4 = \Pi_{Titre,Horaire,Adresse}[E_3]$ Résultat final e_4:

Titre	Horaire	Adresse
Marion	16h00	9, rue Montesquieu
Marion	18h00	6, r. Franklin
Marion	22h00	6, r. Franklin

Schéma : (Titre, Horaire, Adresse)

Expression complète :

$$\Pi_{Titre,Horaire,Adresse}[(\sigma_{Acteur=\text{'M-F Pisier'}}([FILM])) \bowtie [PROGR.] \bowtie [CINE]]$$

Alternativement, en joignant d'abord PROGR et CINE:

$$\Pi_{Titre,Horaire,Adresse}[(\sigma_{Acteur=\text{'M-F Pisier'}}([FILM])) \bowtie ([PROGR.] \bowtie [CINE])]$$

Alternative (moins optimisée, jointure plus grosse au début) :

$$\Pi_{Titre,Horaire,Adresse}[\sigma_{Acteur=\text{'M-F Pisier'}}([FILM] \bowtie [PROGR.] \bowtie [CINE])]$$

Example 5.5.4 (Autre expression équivalente - exemple page 184)**.** Les cinémas qui projettent un film dans lequel joue M.F Pisier avec titres et horaires. Expression 1 (détaillée) :

1. Les films avec M-F Pisier : $(E_1) = \sigma_{Acteur=\text{'M-F.Pisier'}}[FILM]$ (Schéma: FILM(Titre, MeS, Acteur))
2. Projection sur Titre : $(E_{1'}) = \Pi_{Titre}[E_1]$ (Schéma: RES1(Titre))
3. Programmation de ces films : $E_2 = E_{1'} \bowtie [PROG.]$ (Schémas sources: RES1(Titre), PROG.(Nom-Ciné, Titre, Horaire)). Schéma cible : RES2(Nom-Ciné, Titre, Horaire).

Expression Algébrique : $[\Pi_{Titre}[\sigma_{Acteur=\text{'M-F.Pisier'}}[FILM]]] \bowtie [PROG.]$ (Note : On a perdu Nom-Ciné ici, il faut projeter à la fin). Correction pour garder Nom-Ciné, Titre, Horaire :

$$\Pi_{Nom\text{-}Cine,Titre,Horaire}[(\Pi_{Titre}[\sigma_{Acteur=\text{'M-F.Pisier'}}[FILM]]) \bowtie [PROG.]]$$

Expression 2 (équivalente mais potentiellement moins efficace) :

$$\Pi_{Nom\text{-}Cine,Titre,Horaire}[\sigma_{Acteur=\text{'M-F.Pisier'}}[FILM] \bowtie [PROG.]]$$

Ici on fait d'abord la jointure complète de FILM et PROG, puis on sélectionne, puis on projette.

Remark 5.5.5 (Jointure et auto-jointure - exemple page 185-188)**.** Question : Les acteurs qui ont joué dans les films avec M-F. Pisier.

Une approche naïve et **incorrecte** serait : $\Pi_{Acteur}([FILM] \bowtie \sigma_{Acteur=\text{'M-F.Pisier'}}[FILM])$ Pourquoi est-ce incorrect ? La jointure se fait sur TOUS les attributs communs : Titre, M-e-S, et Acteur. Donc $\sigma_{Acteur=\text{'M-F.Pisier'}}[FILM]$ ne contient que les lignes où Acteur='M-F.Pisier'. La jointure ne retournera que les lignes où Acteur est aussi 'M-F.Pisier'. Le résultat final après projection sera juste 'M-F.Pisier'.

Une approche correcte nécessite souvent de renommer les attributs ou de passer par une étape intermédiaire. Par exemple, trouver les titres des films où M-F. Pisier a joué, puis trouver tous les acteurs de ces films : 1. Titres des films avec M-F. Pisier : $T = \Pi_{Titre}(\sigma_{Acteur=\text{'M-F.Pisier'}}[FILM])$ 2. Jointure de ces titres avec la table FILM : $R = T \bowtie [FILM]$ 3. Projection sur les acteurs : $\Pi_{Acteur}(R)$

Expression algébrique correcte :

$$\Pi_{Acteur}((\Pi_{Titre}(\sigma_{Acteur=\text{'M-F.Pisier'}}[FILM])) \bowtie [FILM])$$

Cette expression trouve d'abord les films ('Marion'), puis rejoint avec FILM sur 'Titre' pour

obtenir toutes les lignes de 'Marion', et enfin projette les acteurs ('C. Tetard', 'M-F Pisier'). **NON !** $\Pi_{Acteur}([FILM] \bowtie [\sigma_{Acteur=\textbf{M-F.Pisier}}[FILM]])$ **NON ! NON !** $\Pi_{Acteur}([FILM] \bowtie [\Pi_{Titre}[\sigma_{Acteur=\textbf{M-F.Pisier}}[FILM]]])$ **NON !** (Celle-ci est aussi incorrecte car la jointure se fait sur Titre, mais le premier opérande a encore l'attribut Acteur, ce qui limite le résultat).

La version correcte est bien:

$$\Pi_{Acteur}((\Pi_{Titre}(\sigma_{Acteur='\text{M-F.Pisier}'}[FILM])) \bowtie [FILM])$$

5.5.2 Cas particuliers de la jointure

Intersection ($\cap$)

Si deux relations $R(V)$ et $S(W)$ ont exactement le même schéma (i.e., $V = W$), alors leur jointure naturelle est équivalente à leur intersection ensembliste. $[R] \bowtie [S] = [R] \cap [S]$ (si $V = W$) L'intersection retourne les n-uplets présents à la fois dans r et dans s.

Example 5.5.6 (Intersection). Question : Les titres des films dans lesquels joue M-F. Pisier ET qui sont à l'affiche. 1. Titres des films avec M-F. Pisier : $T_1 = \Pi_{Titre}[\sigma_{Acteur='\text{M-F.Pisier}'}[FILM]]$ Résultat t_1: {Marion} 2. Titres des films à l'affiche : $T_2 = \Pi_{Titre}[PROGR.]$ Résultat t_2: {Speed 2, Marion} 3. Intersection : $T_1 \cap T_2$ (possible car T_1 et T_2 ont le même schéma : (Titre)) Résultat : {Marion}

Expression : $[\Pi_{Titre}[\sigma_{Acteur='\text{M-F.Pisier}'}[FILM]]] \cap [\Pi_{Titre}[PROGR.]]$ (Note: On suppose ici que CINE est la table des cinémas et PROGR est la programmation comme vu avant. L'image p.189 mentionne [CINE] pour les films à l'affiche, ce qui est probablement une erreur de notation, il faut utiliser [PROGR.])

Produit Cartésien ($\times$)

Si deux relations $R(V)$ et $S(W)$ ont des ensembles d'attributs disjoints (i.e., $V \cap W = \emptyset$), alors leur jointure naturelle est équivalente à leur produit cartésien. $[R] \bowtie [S] = [R] \times [S]$ (si $V \cap W = \emptyset$) Le produit cartésien retourne toutes les combinaisons possibles d'un n-uplet de r et d'un n-uplet de s. Le schéma résultat est $Res(V \cup W)$.

Example 5.5.7 (Produit Cartésien). Soient $R(A, B)$ et $S(A', B')$ avec $V = \{A, B\}$ et $W = \{A', B'\}$. $V \cap W = \emptyset$. Instance r:

A	B
a	b1
a	b2

Instance s:

A'	B'
a'1	b'1
a'2	b'2

Résultat $r \times s$:

A	B	A'	B'
a	b1	a'1	b'1
a	b1	a'2	b'2
a	b2	a'1	b'1
a	b2	a'2	b'2

5.5.3 L'opérateur de Renommage (ρ)

Parfois, il est nécessaire de changer le nom des attributs d'une relation, par exemple :

- Pour effectuer une jointure qui ne serait pas naturelle (jointure basée sur des attributs de noms différents mais sémantiquement équivalents).
- Pour effectuer une intersection ou une différence entre relations ayant des schémas sémantiquement équivalents mais avec des noms d'attributs différents.
- Pour effectuer une auto-jointure (joindre une table avec elle-même) en distinguant les attributs provenant de chaque "copie" de la table.

Definition 5.5.8 (Renommage). • **Opération** : Unaire.

- **But** : Changer les noms de certains attributs.
- **Schéma source** : Une relation $R(V)$.
- **Paramètre** : Une liste de renommages $A_1 \to B_1, A_2 \to B_2, \ldots, A_k \to B_k$, où $A_i \in V$ et B_i sont les nouveaux noms. Les B_i doivent être distincts entre eux et des attributs de V non renommés.
- **Syntaxe** : $\rho_{A_1 \to B_1, \ldots, A_k \to B_k}$
- **Application** : Prend une instance r de $R(V)$ et retourne une instance de $Res(W)$, où W est l'ensemble d'attributs V après renommage.
- **Schéma cible** : $Res(W)$, où $W = (V \setminus \{A_1, \ldots, A_k\}) \cup \{B_1, \ldots, B_k\}$.
- **Sémantique** : L'opérateur ne modifie pas les données, seulement les noms des colonnes dans le schéma. $\rho_{A \to B}(r)$ est l'instance r où la colonne nommée A est maintenant nommée B.

Example 5.5.9 (Trouver les Metteurs en Scène qui sont aussi Acteurs). Problème : On veut comparer les Metteurs en Scène (MeS) et les Acteurs de la table FILM. Une intersection directe $\Pi_{MeS}[FILM] \cap \Pi_{Acteur}[FILM]$ est impossible car les schémas sont différents ((MeS) vs (Acteur)).

Solution avec renommage : 1. Projeter sur MeS : $E_1 = \Pi_{MeS}[FILM]$ (Schéma: (MeS)) 2. Projeter sur Acteur : $E_2 = \Pi_{Acteur}[FILM]$ (Schéma: (Acteur)) 3. Renommer Acteur en MeS dans E_2 : $E_2' = \rho_{Acteur \to MeS}[E_2]$ (Schéma: (MeS)) 4. Faire l'intersection : $E_1 \cap E_2'$ (Possible car les schémas sont identiques)

Expression algébrique :

$$\Pi_{MeS}[FILM] \cap \rho_{Acteur \to MeS}[\Pi_{Acteur}[FILM]]$$

Alternative avec renommage et jointure : 1. Projeter FILM sur MeS et renommer MeS en Personne : $P_1 = \rho_{MeS \to Personne}[\Pi_{MeS}[FILM]]$ (Schéma: (Personne)) 2. Projeter FILM sur Acteur et renommer Acteur en Personne : $P_2 = \rho_{Acteur \to Personne}[\Pi_{Acteur}[FILM]]$ (Schéma: (Personne)) 3. La jointure $P_1 \bowtie P_2$ est équivalente à l'intersection $P_1 \cap P_2$.

Illustration des résultats intermédiaires (basé sur l'instance 'film') : E_1: {Jan de Bont, M. Poirier} E_2: {S. Bullock, J. Patric, W. Dafoe, C. Tetard, M-F Pisier} E_2': {S. Bullock, J. Patric, W. Dafoe, C. Tetard, M-F Pisier} (avec schéma (MeS)) $E_1 \cap E_2'$: {} (Personne n'est à la fois MeS et Acteur dans cet exemple)

Example 5.5.10 (Films dirigés par au moins deux metteurs en scènes - Exemple p. 194). (Note: la question semble mal formulée sur la slide, elle vise probablement à illustrer l'auto-jointure après renommage pour comparer des n-uplets au sein de la même table). Supposons qu'on veuille trouver les films (Titre) ayant au moins deux entrées différentes pour Metteur en Scène (MeS), ce qui indiquerait une erreur ou une co-direction mal représentée. Ou plus probablement: Trouver les paires de Metteurs en Scène (M1, M2) qui ont travaillé sur le même Titre. L'exemple de la slide vise sans doute à trouver les films ayant plusieurs enregistrements (ex: plusieurs acteurs pour le même film/MeS) puis à comparer les MeS sur

ces enregistrements.

Soit la requête: Trouver les paires de (Titre, M1, M2) où M1 et M2 sont des MeS différents pour le même Titre. 1. Projection de FILM sur Titre, MeS: $E_0 = \Pi_{Titre,MeS}[FILM]$ Résultat e_0:

Titre	MeS
Speed 2	Jan de Bont
Marion	M. Poirier

2. Première copie renommée : $E_1 = \rho_{MeS \to M1}[E_0]$ (Schéma: (Titre, M1)) Résultat e_1:

Titre	M1
Speed 2	Jan de Bont
Marion	M. Poirier

3. Deuxième copie renommée : $E_2 = \rho_{MeS \to M2}[E_0]$ (Schéma: (Titre, M2)) Résultat e_2:

Titre	M2
Speed 2	Jan de Bont
Marion	M. Poirier

4. Jointure des deux projections renommées sur Titre : $E_3 = E_1 \bowtie E_2$ Résultat e_3:

Titre	M1	M2
Speed 2	Jan de Bont	Jan de Bont
Marion	M. Poirier	M. Poirier

Schéma : (Titre, M1, M2) 5. Sélection pour garder uniquement les lignes où M1 $\neq$ M2 : $E_4 = \sigma_{M1 \neq M2}[E_3]$ Résultat e_4: {} (Vide dans ce cas, car chaque film n'a qu'un seul MeS listé). Schéma : (Titre, M1, M2) 6. Projection sur Titre (si on veut juste les titres) : $E_5 = \Pi_{Titre}[E_4]$ Résultat e_5: {}

Expression algébrique complète pour les titres :

$$\Pi_{Titre}[\sigma_{M1 \neq M2}[(\rho_{MeS \to M1}[\Pi_{Titre,MeS}[FILM]]) \bowtie (\rho_{MeS \to M2}[\Pi_{Titre,MeS}[FILM]])]]$$

5.5.4 L'opérateur d'Union ($\cup$)

L'union combine les n-uplets de deux relations **compatibles en union**. Deux relations sont compatibles en union si elles ont le même nombre d'attributs et que les domaines des attributs correspondants sont les mêmes. L'union ensembliste standard élimine les doublons.

Definition 5.5.11 (Union).

- **Opération** : Binaire.
- **Schémas source** : Deux relations $R(V)$ et $S(W)$ compatibles en union (même arité, mêmes domaines par position. Souvent, on requiert $V = W$).
- **Syntaxe** : $[R] \cup [S]$
- **Application** : Prend (r, s) et retourne une instance de $Res(V)$ (ou $Res(W)$).
- **Schéma cible** : $Res(V)$.
- **Sémantique** : $r \cup s = \{u \mid u \in r \text{ ou } u \in s\}$.

Example 5.5.12 (Union des personnes impliquées dans Marion). Question : Les personnes (acteurs et M-e-S) ayant travaillé sur le tournage du film Marion. 1. Sélectionner les tuples de FILM pour Marion : $E_1 = \sigma_{Titre='\text{Marion}'}[FILM]$ Résultat e_1:

Titre	M-en-S	Acteur
Marion	M. Poirier	C. Tetard
Marion	M. Poirier	M-F Pisier

2. Projeter les MeS et renommer en Personne : $E_2 = \Pi_{MeS}[E_1]$ $E_4 = \rho_{MeS \to Personne}[E_2]$ (Schéma: (Personne)). Résultat e_4: {M. Poirier} 3. Projeter les Acteurs et renommer en Personne : $E_3 = \Pi_{Acteur}[E_1]$ $E_5 = \rho_{Acteur \to Personne}[E_3]$ (Schéma: (Personne)). Résultat e_5: {C. Tetard, M-F Pisier} 4. Faire l'union : $E = E_4 \cup E_5$ Résultat e: {M. Poirier, C. Tetard, M-F Pisier}

Expression algébrique :

$$(\rho_{MeS \to Personne}[\Pi_{MeS}[\sigma_{Titre='\text{Marion}'}[FILM]]]) \cup (\rho_{Acteur \to Personne}[\Pi_{Acteur}[\sigma_{Titre='\text{Marion}'}[FILM]]])$$

5.5.5 La différence ensembliste (−)

La différence retourne les n-uplets qui sont dans la première relation mais pas dans la seconde. Les relations doivent être compatibles en union.

Definition 5.5.13 (Différence).

- **Opération** : Binaire.
- **Schémas source** : Deux relations $R(V)$ et $S(W)$ compatibles en union ($V = W$).
- **Syntaxe** : $[R] - [S]$
- **Application** : Prend (r, s) et retourne une instance de $Res(V)$.
- **Schéma cible** : $Res(V)$.
- **Sémantique** : $r - s = \{u \mid u \in r \text{ et } u \notin s\}$.

Example 5.5.14 (Acteurs qui ne sont pas Metteurs en Scène). Question : Les acteurs qui ne sont pas MeS. 1. Obtenir la liste des acteurs : $E_1 = \Pi_{Acteur}[FILM]$ (Schéma: (Acteur)) 2. Obtenir la liste des MeS : $E_2 = \Pi_{MeS}[FILM]$ (Schéma: (MeS)) 3. Renommer MeS en Acteur dans E_2 pour compatibilité : $E_3 = \rho_{MeS \to Acteur}[E_2]$ (Schéma: (Acteur)) 4. Calculer la différence : $E = E_1 - E_3$

Expression algébrique :

$$\Pi_{Acteur}[FILM] - \rho_{MeS \to Acteur}[\Pi_{MeS}[FILM]]$$

Résultat (basé sur l'instance 'film'): E_1: {S. Bullock, J. Patric, W. Dafoe, C. Tetard, M-F Pisier} E_2: {Jan de Bont, M. Poirier} E_3: {Jan de Bont, M. Poirier} $E = E_1 - E_3$: {S. Bullock, J. Patric, W. Dafoe, C. Tetard, M-F Pisier} (car aucun acteur n'est MeS dans cet exemple)

5.6 Exemple de requête difficile : La Division

La division n'est généralement pas un opérateur de base mais peut être exprimée à l'aide des autres. Elle est utile pour les requêtes de type "tous".

Question : Les cinémas qui projettent **tous** les films de Poirier.

Instances pour l'exemple : **instance 'film'**:

Titre	MeS	Acteur
Marion	Poirier	Tetard
Marion	Poirier	Pisier
Western	Poirier	Pisier
Speed2	J. Bont	Bullock
Speed2	J. Bont	Patric

instance 'prog':

Nom-Ciné	Titre	Horaire
Trianon	Marion	20h00
Trianon	Marion	22h00
Trianon	Speed2	18h00
UGC	Speed2	20h00
Français	Marion	18h00
Français	Western	20h00
Français	Marion	22h00
Français	Speed2	16h00

Stratégie : Trouver les paires (Cinéma, Film de Poirier) qui *manquent* dans la programmation, puis soustraire les cinémas correspondants de la liste de tous les cinémas.

Étapes : 1. Trouver les films dirigés par Poirier : $E_1 = \Pi_{Titre}[\sigma_{MeS='\text{Poirier}'}[film]]$ Résultat 'res1' :

Titre
Marion
Western

2. Trouver les paires (cinéma, film) qui sont effectivement programmées : $E_2 = \Pi_{Nom\text{-}Cine,Titre}[prog]$ Résultat 'res2' :

Nom-Cine	Titre
Trianon	Marion
Trianon	Speed2
UGC	Speed2
Français	Marion
Français	Western
Français	Speed2

(doublons éliminés)

3. Trouver toutes les paires possibles (cinéma, film de Poirier) : Produit cartésien entre tous les cinémas et les films de Poirier. $E_{cinemas} = \Pi_{Nom\text{-}Cine}[prog]$ (ou de la table CINE si disponible) Résultat 'res$_cinemas$' : $\{Trianon, UGC, Français\}E_3 = E_{cinemas} \times E_1$ Résultat 'res3' :

Nom-Cine	Titre
Trianon	Marion
Trianon	Western
UGC	Marion
UGC	Western
Français	Marion
Français	Western

4. Trouver les paires (cinéma, film de Poirier) qui **ne sont pas** programmées : Différence entre toutes les paires possibles et les paires programmées. $E_4 = E_3 - E_2$ Résultat 'res4' :

Nom-Cine	Titre
Trianon	Western
UGC	Marion
UGC	Western

Ces lignes représentent les cinémas qui ne projettent PAS au moins un des films de Poirier.

5. Trouver les cinémas qui apparaissent dans E_4 (ceux qui manquent au moins un film de Poirier) : $E_5 = \Pi_{Nom\text{-}Cine}[E_4]$ Résultat 'res5' : {Trianon, UGC}

6. Trouver les cinémas qui ne sont PAS dans E_5 (ceux qui ne manquent aucun film de Poirier) : Différence entre tous les cinémas et E_5. $E_{final} = E_{cinemas} - E_5$ Résultat 'res$_final$' : $\{Français\}$

Expression algébrique (Division $A \div B$ souvent notée $R(X,Y) \div S(Y)$) Soit $R = \Pi_{Nom\text{-}Cine,Titre}[prog]$ (les paires cinéma-film programmées, X=Nom-Ciné, Y=Titre) Soit $S = \Pi_{Titre}[\sigma_{MeS='\text{Poirier}'}[film]]$ (les films de

Poirier, Y=Titre) On cherche les X (cinémas) tels que l'ensemble des Y (films de Poirier) associés à X dans R contienne S. La division $R \div S$ peut s'exprimer comme : $R \div S = \Pi_X(R) - \Pi_X((\Pi_X(R) \times S) - R)$

Appliquons : $\Pi_{Nom\text{-}Cine}(prog) - \Pi_{Nom\text{-}Cine}((\Pi_{Nom\text{-}Cine}(prog) \times \Pi_{Titre}(\sigma_{MeS=\text{'Poirier'}}(film))) - \Pi_{Nom\text{-}Cine,Titre}(prog))$ Ce qui correspond exactement aux étapes 1 à 6.

Expression finale décomposée : $E_{cinemas} = \Pi_{Nom\text{-}Cine}[prog]$ $E_{poirier\_films} = \Pi_{Titre}[\sigma_{MeS=\text{'Poirier'}}[film]]$ $E_{all\_pairs} = E_{cinemas} \times E_{poirier\_films}$ $E_{programmed\_pairs} = \Pi_{Nom\text{-}Cine,Titre}[prog]$ $E_{missing\_pairs} = E_{all\_pairs} - E_{programmed\_pairs}$ $E_{cinemas\_missing} = \Pi_{Nom\text{-}Cine}[E_{missing\_pairs}]$ $E_{result} = E_{cinemas} - E_{cinemas\_missing}$

Le cinéma "Français" est le seul à projeter tous les films de Poirier (Marion et Western).

Chapter 6

Conception de schémas relationnelles

Chapitre 5 : Conception de Schémas Relationnels

6.1 Introduction

Ce chapitre aborde la conception de schémas relationnels pour les bases de données. Une bonne conception est cruciale pour garantir l'intégrité des données, éviter les redondances inutiles, et faciliter la maintenance et l'évolution de la base. Nous explorerons les motivations derrière une conception soignée, les problèmes courants liés à une mauvaise conception (anomalies de mise à jour), et les outils théoriques comme les dépendances fonctionnelles et les formes normales qui nous guident vers de meilleurs schémas.

Plan du Chapitre :

- Motivation : Pourquoi une bonne conception est-elle importante ? Quels sont les problèmes d'une mauvaise conception ?
- Dépendances fonctionnelles : Formaliser les contraintes entre les données.
- Formes normales : Des critères pour évaluer la qualité d'un schéma.
- Normalisation : Le processus de transformation d'un schéma pour atteindre une forme normale souhaitée (décomposition).
- Dénormalisation : Quand et pourquoi déroger aux formes normales.

Source: basé sur le cours de N. Bidoit 22-23.

6.2 Motivation et Anomalies de Mise à Jour

6.2.1 Qu'est-ce qu'un bon schéma ?

La conception d'un schéma relationnel peut découler de la transformation d'un schéma Entité-Association (EA) ou être le résultat direct d'une analyse des besoins. Une fois le schéma défini (ou même s'il est généré automatiquement par un outil), la question se pose : est-ce un bon schéma ?

Un "bon" schéma relationnel doit permettre de stocker l'information sans redondance excessive et sans risque d'incohérence lors des opérations de mise à jour (insertion, suppression, modification). Une mauvaise conception peut entraîner des anomalies.

6.2.2 Anomalies de mise à jour

Considérons une relation unique pour stocker des informations sur les projections de films dans des cinémas : `PROJECTIONS(Titre, M-e-S, Acteur, N-Ciné, Adr, Tél, Heure)`

Example 6.2.1 (Table d'exemple et Redondances)**.** Voici un exemple d'instance de cette relation :

Titre	M-e-S	Acteur	N-Ciné	Adr.	Tél	Heure
Speed	J. de B.	Bullock	Français	9, M.	..1187	18h
Speed	J. de B.	Bullock	Français	9, M.	..1187	20h
Speed	J. de B.	Patric	Français	9, M.	..1187	18h
Speed	J. de B.	Patric	Français	9, M.	..1187	20h
Speed	J. de B.	Bullock	Français	9, M.	..0445	18h
Speed	J. de B.	Bullock	Français	9, M.	..0445	20h
Speed	J. de B.	Patric	Français	9, M.	..0445	18h
Speed	J. de B.	Patric	Français	9, M.	..0445	20h
Marion	Poirier	Pisier	Trianon	6, F.	..3117	16h
Marion	Poirier	Lopez	Trianon	6, F.	..3117	16h
...	...	...	...	...	...	...

On observe immédiatement des **redondances** (l'adresse et le téléphone du cinéma 'Français' sont répétés de nombreuses fois) et des risques d'**incohérences** (deux numéros de téléphone différents, ..1187 et ..0445, sont associés au même cinéma 'Français' à la même adresse '9, M.').

Ces redondances entraînent plusieurs types d'anomalies :

- **Anomalie d'Insertion :** Supposons que l'on veuille ajouter un nouveau cinéma 'Le Rex' avec son adresse et son téléphone, mais qu'aucun film n'y soit encore programmé. Avec ce schéma, c'est impossible sans insérer des valeurs nulles pour Titre, M-e-S, Acteur, Heure, ce qui n'est pas idéal. On ne peut pas insérer une information partielle sur une entité (le cinéma) sans l'associer à une autre (la projection).
- **Anomalie de Suppression :** Si l'on supprime la dernière projection du film 'Marion' au 'Trianon' (par exemple, les deux lignes avec Pisier et Lopez), on perd également l'information sur le cinéma 'Trianon' lui-même (son adresse et son téléphone), car cette information n'existe que dans les lignes des projections. La suppression d'une information (la projection) entraîne la perte involontaire d'une autre information (le cinéma). La redondance rend ambiguë la suppression : si on veut démolir le cinéma 'Français', quelles lignes supprimer ? Supprimer toutes les lignes 'Speed' supprime aussi l'info sur J. de B., Bullock, Patric.
- **Anomalie de Modification :** Si le numéro de téléphone du cinéma 'Français' change, il faut mettre à jour toutes les lignes correspondant à ce cinéma. Si l'on oublie une seule ligne, la base devient incohérente (plusieurs numéros pour le même cinéma). La mise à jour est coûteuse et source d'erreurs. Quelle ligne modifier si on apprend le bon numéro ?

6.2.3 Synthèse et Objectif

Les anomalies de mise à jour (insertion difficile/impossible, suppression avec perte d'information, modification coûteuse et risquée) sont la conséquence directe de la redondance d'information dans le schéma.

Objectif : Éviter ces anomalies en concevant des schémas qui minimisent la redondance. Comment ? En utilisant les concepts de **dépendances fonctionnelles** (qui capturent les liens sémantiques entre les données et sont souvent la cause de la redondance si mal gérées) et de **formes normales** (qui définissent des niveaux de "bonne conception" par rapport aux dépendances). Le processus pour améliorer un schéma s'appelle la **normalisation**, qui implique souvent de décomposer une relation en plusieurs relations plus petites et mieux structurées.

6.3 Dépendances Fonctionnelles (Complément)

Les dépendances fonctionnelles (DFs) sont des contraintes d'intégrité qui expriment des liens entre les attributs d'une relation. Elles sont fondamentales pour comprendre la structure des données et pour la conception de schémas.

Definition 6.3.1 (Dépendance Fonctionnelle)**.** Soit un schéma de relation $R(A_1, ..., A_n)$ et soient X, Y des sous-ensembles des attributs de R ($X, Y \subseteq \{A_1, ..., A_n\}$). Une **dépendance fonctionnelle** (DF), notée $X \to Y$, exprime la contrainte suivante : Pour toute instance r de R, si deux tuples u et v de r coïncident sur les attributs de X, alors ils doivent aussi coïncider sur les attributs de Y. Formellement :

$$r \models X \to Y \quad \text{si} \quad \forall u, v \in r, (\forall A \in X, u[A] = v[A]) \implies (\forall A \in Y, u[A] = v[A])$$

On lit : "X détermine Y" ou "Y dépend fonctionnellement de X".

Les DFs représentent des règles métier ou des propriétés intrinsèques des données. Par exemple, dans une relation 'Employe(NumSS, Nom, DateNaissance, Adresse)', la DF 'NumSS -¿ Nom, DateNaissance, Adresse' signifie que le numéro de sécurité sociale détermine de manière unique le nom, la date de naissance et l'adresse d'un employé.

Definition 6.3.2 (Schéma avec DFs)**.** Un schéma de base de données avec dépendances fonctionnelles est une paire $(R, \mathcal{F})$ où R est un schéma de relation et $\mathcal{F}$ est un ensemble de DFs définies sur les attributs de R. Une instance d'un schéma $(R, \mathcal{F})$ est une instance de R qui satisfait **toutes** les DFs de $\mathcal{F}$.

6.3.1 Implication de Dépendances

Un ensemble de DFs $\mathcal{F}$ donné peut en impliquer d'autres logiquement. Par exemple, si nous savons que 'NumSS -¿ DateNaissance' et 'DateNaissance -¿ SigneAstrologique', nous pouvons en déduire que 'NumSS -¿ SigneAstrologique'.

Definition 6.3.3 (Implication Sémantique)**.** Soit $(R, \mathcal{F})$ un schéma avec DFs. On dit que $\mathcal{F}$ **implique** (sémantiquement) une DF $X \to Y$, noté $\mathcal{F} \models X \to Y$, si toute instance r qui satisfait toutes les DFs de $\mathcal{F}$ satisfait aussi la DF $X \to Y$.

Example 6.3.4 (Implication de DFs)**.**

- Soit FILM(Titre, M-e-S, Acteur) avec $\mathcal{F} = \{\text{Titre} \to \text{M-e-S}\}$. Toute instance r satisfaisant Titre → M-e-S satisfera aussi, par exemple, Titre, Acteur → M-e-S. Donc, $\mathcal{F} \models$ Titre, Acteur → M-e-S.
- Soit PROGbis(Titre, Résumé, Salle, Nom-Ciné) avec les DFs :
  - (df1) Salle, Nom-Ciné → Titre (Un cinéma projette un seul film dans une salle donnée)
  - (df2) Titre → Résumé (Un seul résumé est proposé pour chaque film)

  Soit $\mathcal{F} = \{\text{df1, df2}\}$. Toute instance r de PROGbis satisfaisant $\mathcal{F}$ satisfera aussi la DF : Salle, Nom-Ciné → Résumé. En effet, si deux tuples u, v ont même Salle et Nom-Ciné, alors par df1 ils ont le même Titre. Ayant le même Titre, par df2 ils ont le même Résumé. Donc $\mathcal{F} \models$ Salle, Nom-Ciné → Résumé.

La question clé est : comment trouver *toutes* les DFs impliquées par un ensemble $\mathcal{F}$ donné ? C'est important pour comprendre toutes les contraintes sur nos données et pour identifier les clés.

6.3.2 Axiomes d'Armstrong (Inférence Syntaxique)

Les axiomes d'Armstrong fournissent un ensemble de règles d'inférence *syntaxiques* pour dériver de nouvelles DFs à partir d'un ensemble $\mathcal{F}$. On note $\mathcal{F} \vdash X \to Y$ si la DF $X \to Y$ peut être dérivée de $\mathcal{F}$ en utilisant ces règles.

Theorem 6.3.5 (Axiomes d'Armstrong)**.** Soit $(R, \mathcal{F})$ un schéma et X, Y, Z des ensembles d'attributs de R.

- **Initialisation :** Si $X \to Y \in \mathcal{F}$, alors $\mathcal{F} \vdash X \to Y$.

- **Réflexivité :** Si $Y \subseteq X$, alors $\mathcal{F} \vdash X \to Y$.
- **Augmentation :** Si $\mathcal{F} \vdash X \to Y$, alors $\mathcal{F} \vdash XZ \to YZ$. (On peut ajouter les mêmes attributs des deux côtés).
- **Transitivité :** Si $\mathcal{F} \vdash X \to Y$ et $\mathcal{F} \vdash Y \to Z$, alors $\mathcal{F} \vdash X \to Z$.

Theorem 6.3.6 (Correction et Complétude). Le système d'inférence d'Armstrong est **correct** (toute DF dérivée $\mathcal{F} \vdash X \to Y$ est aussi sémantiquement impliquée $\mathcal{F} \models X \to Y$) et **complet** (toute DF sémantiquement impliquée $\mathcal{F} \models X \to Y$ est dérivable $\mathcal{F} \vdash X \to Y$). Autrement dit, $\mathcal{F} \vdash X \to Y \iff \mathcal{F} \models X \to Y$.

Cela signifie que nous pouvons trouver toutes les DFs impliquées en appliquant systématiquement les axiomes d'Armstrong.

Règles Additionnelles (Dérivées)

D'autres règles utiles peuvent être dérivées des axiomes de base :

- **Union :** Si $\mathcal{F} \vdash X \to Y$ et $\mathcal{F} \vdash X \to Z$, alors $\mathcal{F} \vdash X \to YZ$.
- **Décomposition :** Si $\mathcal{F} \vdash X \to YZ$, alors $\mathcal{F} \vdash X \to Y$ et $\mathcal{F} \vdash X \to Z$.
- **Pseudo-Transitivité :** Si $\mathcal{F} \vdash X \to Y$ et $\mathcal{F} \vdash YZ \to W$, alors $\mathcal{F} \vdash XZ \to W$.
- **Augmentation-gauche :** Si $\mathcal{F} \vdash X \to Y$, alors $\mathcal{F} \vdash XZ \to Y$. (Attention: cette règle telle qu'écrite sur la diapositive est inhabituelle, elle découle d'Augmentation $XZ \to YZ$ et Décomposition $YZ \to Y$).

6.3.3 Fermeture d'un ensemble d'attributs

Une tâche essentielle est de déterminer tous les attributs qui sont fonctionnellement déterminés par un ensemble d'attributs X.

Definition 6.3.7 (Fermeture X^+). Soit $(R, \mathcal{F})$ un schéma et X un sous-ensemble des attributs de R. La **fermeture** de X sous $\mathcal{F}$, notée X^+, est l'ensemble de tous les attributs A tels que $\mathcal{F} \models X \to A$.

$$X^+ = \{A \mid \mathcal{F} \models X \to A\}$$

On peut calculer X^+ avec l'algorithme suivant :
Algorithme de Calcul de X^+

1. **Initialisation :** $RES := X$.
2. **Itération :** Tant qu'il existe une DF $(Z \to A) \in \mathcal{F}$ telle que $Z \subseteq RES$ et $A \notin RES$:
   - Ajouter A à RES ($RES := RES \cup \{A\}$).
3. **Sortie :** $RES = X^+$.

Example 6.3.8 (Calcul de fermetures).
- CINE(Nom-Ciné, Adresse, Téléphone) avec $\mathcal{F} = \{\text{Nom-Ciné} \to \text{Adresse}, \text{Nom-Ciné} \to \text{Téléphone}\}$. Calculons $(\text{Nom-Ciné})^+$:
  1. $RES := \{\text{Nom-Ciné}\}$
  2. $Z = \{\text{Nom-Ciné}\} \subseteq RES$. (Nom-Ciné $\to$ Adresse) $\in \mathcal{F}$. $A = \text{Adresse} \notin RES$. $RES := \{\text{Nom-Ciné, Adresse}\}$.
  3. $Z = \{\text{Nom-Ciné}\} \subseteq RES$. (Nom-Ciné $\to$ Téléphone) $\in \mathcal{F}$. $A = \text{Téléphone} \notin RES$. $RES := \{\text{Nom-Ciné, Adresse, Téléphone}\}$.
  4. Plus de DFs applicables.

Donc, (Nom-Ciné)$^+$ = {Nom-Ciné, Adresse, Téléphone}.

- R(A, B, C, D) avec $\mathcal{F} = \{\text{A} \to \text{B}, \text{BC} \to \text{D}\}$.
  - $A^+ = \{A, B\}$ (on utilise A → B)
  - $B^+ = \{B\}$
  - $C^+ = \{C\}$
  - $AC^+ = \{A, C\}$ (init) → $\{A, B, C\}$ (utilise A → B) → $\{A, B, C, D\}$ (utilise BC → D car $B, C \subseteq \{A, B, C\}$). Donc $AC^+ = \{A, B, C, D\}$.

6.3.4 Couverture Minimale

Souvent, l'ensemble de DFs $\mathcal{F}$ initial contient des redondances (des DFs qui peuvent être dérivées des autres). Une **couverture minimale** (ou irréductible) $\mathcal{G}$ de $\mathcal{F}$ est un ensemble équivalent ($\mathcal{G}^+ = \mathcal{F}^+$) qui est "minimal" en trois sens : 1. Tous les côtés droits des DFs dans $\mathcal{G}$ sont des attributs uniques (singleton). 2. Aucun attribut ne peut être retiré du côté gauche d'une DF dans $\mathcal{G}$ sans changer la fermeture $\mathcal{G}^+$. 3. Aucune DF entière ne peut être retirée de $\mathcal{G}$ sans changer la fermeture $\mathcal{G}^+$.

Example 6.3.9 (Réductions).

- **Réduction à droite :** $AE \to BCD$ est équivalent à $\{AE \to B, AE \to C, AE \to D\}$.
- **Réduction à gauche :** Si $\mathcal{F} = \{A \to B, AC \to B\}$. Puisque $A \to B$ implique $AC \to B$ (par Augmentation), la DF $AC \to B$ est redondante. On peut la supprimer ou, plus précisément, réduire $AC \to B$ en $A \to B$.
- **Minimalisation (Suppression de DF) :** Si $\mathcal{F} = \{A \to B, B \to C, A \to C\}$. Puisque $A \to B$ et $B \to C$ impliquent $A \to C$ (par Transitivité), la DF $A \to C$ est redondante et peut être supprimée.

Algorithme de Calcul d'une Couverture Minimale $\mathcal{G}$ de $\mathcal{F}$

1. **Étape 1 (Réduction droite) :** Remplacer chaque DF $X \to \{A_1, ..., A_n\}$ dans $\mathcal{F}$ par n DFs $X \to A_i$. Soit G cet ensemble.
2. **Étape 2 (Réduction gauche) :** Pour chaque DF $X \to A$ dans G: Pour chaque attribut $B \in X$, vérifier si A est dans la fermeture de $(X - \{B\})$ calculée avec l'ensemble G courant. Si oui, remplacer X par $(X - \{B\})$ dans la DF $X \to A$. Répéter jusqu'à ce qu'aucun attribut ne puisse être retiré de X.
3. **Étape 3 (Réduction ensembliste) :** Pour chaque DF $X \to A$ restante dans G: Vérifier si A est dans la fermeture de X calculée avec l'ensemble $G - \{X \to A\}$ (c'est-à-dire, en utilisant toutes les autres DFs de G). Si oui, supprimer la DF $X \to A$ de G.
4. **Sortie :** L'ensemble final G est une couverture minimale de $\mathcal{F}$.

(Note: L'ordre des étapes 2 et 3 peut parfois importer, et l'étape 2 peut nécessiter des itérations).

6.3.5 Clés d'un schéma relationnel

Les DFs permettent de définir formellement les clés d'une relation.

Definition 6.3.10 (Super-clé, Clé Candidate, Clé Primaire). Soit $(R, \mathcal{F})$ un schéma avec DFs.

- Une **super-clé** est un ensemble d'attributs $X \subseteq Att(R)$ tel que X^+ contient tous les attributs de R ($X^+ = Att(R)$), c'est-à-dire $X \to Att(R)$.
- Une **clé candidate** est une super-clé X telle qu'aucun sous-ensemble propre de X n'est une super-clé. C'est une super-clé minimale.

- La **clé primaire** est l'une des clés candidates choisie par l'administrateur de la base de données pour identifier de manière unique les tuples.

Definition 6.3.11 (Attribut Prime)**.** Un attribut A est dit **prime** s'il fait partie d'au moins une clé candidate. Sinon, il est dit **non-prime**.

Example 6.3.12 (Identification de Clés)**.**

- CINE(Nom-Ciné, Adresse, Téléphone) avec $\mathcal{F} = \{\text{Nom-Ciné} \rightarrow \text{Adresse}, \text{Nom-Ciné} \rightarrow \text{Téléphone}\}$. On a vu $(\text{Nom-Ciné})^+ = \{\text{Nom-Ciné, Adresse, Téléphone}\} = Att(R)$. Donc {Nom-Ciné} est une super-clé. Comme c'est un singleton, aucun sous-ensemble propre n'existe, c'est donc une clé candidate (et la seule).
- R(A, B, C, D) avec $\mathcal{F} = \{\text{A} \rightarrow \text{B}, \text{BC} \rightarrow \text{D}\}$. On a vu $AC^+ = \{A, B, C, D\} = Att(R)$. Donc $\{A, C\}$ est une super-clé. Est-ce une clé candidate ? Regardons les sous-ensembles propres : $A^+ = \{A, B\} \neq Att(R)$ et $C^+ = \{C\} \neq Att(R)$. Donc, aucun sous-ensemble propre n'est une super-clé. $\{A, C\}$ est une clé candidate. Y en a-t-il d'autres ? (Il faudrait tester toutes les combinaisons d'attributs). Dans cet exemple, A, C sont des attributs primes. B, D sont des attributs non-primes.
- CINE(Nom-Ciné, Adresse, Téléphone) *sans* DFs. La seule DF triviale est $\emptyset \rightarrow \emptyset$. La seule super-clé est l'ensemble de tous les attributs $\{Nom - Ciné, Adresse, Téléphone\}$, qui est donc aussi la seule clé candidate.

6.4 Formes Normales

Les formes normales (FN) sont des propriétés souhaitables pour les schémas de relations, visant à réduire la redondance et les anomalies de mise à jour. Elles sont définies en fonction des dépendances fonctionnelles. Un schéma est dit être dans une certaine forme normale s'il satisfait les conditions de cette forme. Il existe une hiérarchie : BCNF $\implies$ 3NF $\implies$ 2NF.

Objectifs des formes normales :

- Éviter la redondance.
- Améliorer l'intégrité des données.
- Faciliter la maintenance et l'évolution de la base.

6.4.1 Deuxième Forme Normale (2NF)

La 2NF s'attaque aux *dépendances partielles* : un attribut non-prime ne doit pas dépendre d'une partie seulement d'une clé candidate.

Definition 6.4.1 (2ème Forme Normale - 2NF)**.** Un schéma de relation $(R, \mathcal{F})$ est en **2ème forme normale (2NF)** si tout attribut **non-prime** A est **pleinement dépendant** de toute clé candidate K. Cela signifie que pour toute clé candidate K et tout attribut non-prime A, il n'existe pas de sous-ensemble propre $X \subset K$ tel que $\mathcal{F} \models X \rightarrow A$. (Formulation équivalente de la diapo : pour tout A n'appartenant pas à une clé, si $X \rightarrow A$ est une df impliquée par $\mathcal{F}$, alors X n'est pas inclus strictement dans une clé.)

Example 6.4.2 (Relation non 2NF)**.** Soit Cachet(Acteur, Genre, Qualif, Titre, Salaire). Supposons que la seule clé candidate soit $K = \{\text{Acteur, Genre, Qualif, Titre}\}$. Supposons la DF : Qualif $\rightarrow$ Salaire. Ici, $A =$ Salaire est un attribut non-prime (il ne fait pas partie de la clé K). $X = \{\text{Qualif}\}$ est un sous-ensemble propre de la clé K ($X \subset K$). La DF $X \rightarrow A$ (Qualif $\rightarrow$ Salaire) existe. La condition 2NF est violée car l'attribut non-prime Salaire dépend d'une partie (Qualif) de la clé candidate K.

Conséquence : Redondance. Le salaire associé à une qualification (par ex., 'n-one' → 500) est répété pour chaque acteur/genre/titre ayant cette qualification.

Acteur	Genre	Qualif	Titre	Salaire
Dafoe	Comique	n-one	Speed2	500
Dafoe	Historique	i-one	Western	700
Pisier	Historique	n-one	Western	500
Pisier	Historique	i-one	Western	700
Pisier	Drame	n-one	Marion	500
...	...	...	...	...

Example 6.4.3 (Modélisation 2NF)**.** Pour résoudre le problème précédent, on décompose la relation :

- Grille(Qualif, Salaire) avec DF : Qualif → Salaire. (Clé: Qualif)
- Prestation(Acteur, Genre, Qualif, Titre) (Clé: Acteur, Genre, Qualif, Titre)

Dans Grille, Salaire dépend de la clé Qualif, c'est OK. Dans Prestation, il n'y a pas d'attributs non-primes, donc la 2NF est trivialement satisfaite. La redondance sur le salaire a disparu.

Grille

Qualif	Salaire
n-one	500
i-one	700

Prestation

Acteur	Genre	Qualif	Titre
Dafoe	Comique	n-one	Speed2
Dafoe	Historique	i-one	Western
Pisier	Historique	n-one	Western
Pisier	Historique	i-one	Western
Pisier	Drame	n-one	Marion
...	...	...	...

6.4.2 Troisième Forme Normale (3NF)

La 3NF s'attaque aux *dépendances transitives* : un attribut non-prime ne doit pas dépendre d'un autre attribut non-prime.

Definition 6.4.4 (3ème Forme Normale - 3NF)**.** Un schéma de relation $(R, \mathcal{F})$ est en **3ème forme normale (3NF)** si pour toute dépendance fonctionnelle **non triviale** $X \to A$ impliquée par $\mathcal{F}$ (où $A \notin X$), au moins l'une des conditions suivantes est vérifiée :

- X est une super-clé de R.
- A est un attribut prime (fait partie d'au moins une clé candidate).

En d'autres termes, toute dépendance $X \to A$ qui viole la 3NF est une dépendance où X n'est pas une super-clé et A est un attribut non-prime. Cela correspond souvent à une dépendance transitive : Clé → NonPrime1 → NonPrime2.

Example 6.4.5 (Relation non 3NF)**.** Soit Conf-Presse(Nom-Ciné, Salle, Horaire, Titre). DFs :

- K = {Nom-Ciné, Titre} est clé candidate (une conf par ciné/film).
- Nom-Ciné, Titre → Salle (La conf pour un film/ciné a lieu dans une salle unique).
- Nom-Ciné, Salle → Horaire (Une seule conf par jour/salle/ciné $\implies$ l'horaire dépend du ciné et de la salle).

Considérons la DF : Nom-Ciné, Salle → Horaire.

- X = {Nom-Ciné, Salle} n'est pas une super-clé (ne détermine pas Titre).

- A = Horaire est un attribut non-prime (ne fait pas partie de la clé K).

Aucune des conditions 3NF n'est vérifiée. La relation n'est pas en 3NF. La dépendance Nom-Ciné, Salle → Horaire est transitive via la clé : Nom-Ciné, Titre → Salle, et ensuite Nom-Ciné, Salle → Horaire.

Conséquence : Redondance. Si une salle d'un cinéma est utilisée pour plusieurs conférences (pour des films différents), son horaire de conférence (supposé unique par salle/jour) sera répété.

Nom-Ciné	Salle	Horaire	Titre
Trianon	-1-	20h	Western
Trianon	-1-	20h	Marion
Trianon	-2-	16h	Speed2
...	...	...	...

Example 6.4.6 (Modélisation 3NF)**.** On décompose pour éliminer la dépendance transitive :

- Conf-Presse-Film(Nom-Ciné, Salle, Titre) avec DF : Nom-Ciné, Titre → Salle. (Clé: Nom-Ciné, Titre). Cette relation est en 3NF (et même BCNF).
- Conf-Presse-Horaire(Nom-Ciné, Salle, Horaire) avec DF : Nom-Ciné, Salle → Horaire. (Clé: Nom-Ciné, Salle). Cette relation est en 3NF (et même BCNF).

La redondance sur l'horaire a disparu.

Conf-Presse-Film

Nom-Ciné	Salle	Titre
Trianon	-1-	Western
Trianon	-1-	Marion
Trianon	-2-	Speed2
...	...	...

Conf-Presse-Horaire

Nom-Ciné	Salle	Horaire
Trianon	-1-	20h
Trianon	-2-	16h
...	...	...

6.4.3 Forme Normale de Boyce-Codd (BCNF)

La BCNF est une forme normale plus stricte que la 3NF. Elle élimine certaines redondances que la 3NF peut laisser passer dans des cas plus rares (souvent liés à des clés candidates multiples et qui se chevauchent).

Definition 6.4.7 (Forme Normale de Boyce-Codd - BCNF)**.** Un schéma de relation $(R, \mathcal{F})$ est en **Forme Normale de Boyce-Codd (BCNF)** si pour toute dépendance fonctionnelle **non triviale** $X \to A$ impliquée par $\mathcal{F}$, X est une super-clé de R.

La seule différence avec la 3NF est que la BCNF ne permet pas l'exception "A est un attribut prime". Toute dépendance doit provenir d'une super-clé.

Example 6.4.8 (Relation 3NF mais non BCNF)**.** Soit FILM-débutant(Titre, M-e-S, Acteur). DFs (selon l'interprétation la plus plausible de la diapo) :

- Titre, Acteur → M-e-S (Un acteur ne joue qu'un M-e-S pour un titre donné ? Ou le metteur en scène est déterminé par le film et l'acteur principal ? Sémantique peu claire).
- M-e-S → Titre (Un metteur en scène ne fait qu'un seul film 'débutant' ? Ou un film 'débutant' n'a qu'un seul M-e-S ?)

Supposons ces DFs. Clés candidates : K_1 = {Titre, Acteur} (car K_1^+ = {Titre, Acteur, M-e-S} en

utilisant la 1ere DF). K_2 = {M-e-S, Acteur} (car K_2^+ = {M-e-S, Acteur, Titre} en utilisant la 2eme DF). Tous les attributs (Titre, M-e-S, Acteur) sont primes. Vérifions la 3NF :

- Pour Titre, Acteur → M-e-S : X = {Titre, Acteur} est une super-clé (K_1). OK.
- Pour M-e-S → Titre : A = Titre est un attribut prime. OK.

La relation est en 3NF. Vérifions la BCNF :

- Pour Titre, Acteur → M-e-S : X = {Titre, Acteur} est une super-clé. OK.
- Pour M-e-S → Titre : X = {M-e-S} n'est pas une super-clé (ne détermine pas Acteur). La condition BCNF (X doit être une super-clé) est violée.

La relation n'est pas en BCNF.

Conséquence : Redondance. Si un metteur en scène (M-e-S) est associé à un Titre, cette association est répétée pour chaque Acteur jouant dans ce film.

Titre	M-e-S	Acteur
Speed2	J. de Bont	S. Bullock
Speed2	J. de Bont	J. Patric
Speed2	J. de Bont	W. Dafoe
...	...	...

Example 6.4.9 (Modélisation BCNF)**.** On décompose pour satisfaire BCNF :

- FilmParMES(M-e-S, Titre) avec DF : M-e-S → Titre. (Clé: M-e-S).
- CastingMES(M-e-S, Acteur) (Clé: M-e-S, Acteur).

Cette décomposition est en BCNF et élimine la redondance.

FilmParMES

M-e-S	Titre
J. de Bont	Speed2
...	...

CastingMES

M-e-S	Acteur
J. de Bont	S. Bullock
J. de Bont	J. Patric
J. de Bont	W. Dafoe
...	...

6.4.4 Tableau Récapitulatif

Le tableau suivant résume quand une DF non triviale $X \to A$ respecte chaque forme normale, basé sur les propriétés de X et A.

Propriété de X	Propriété de A	2NF	3NF	BCNF
super-clé	$\notin$ clé	OK	OK	OK
super-clé	$\in$ clé	OK	OK	OK
$\subset$ clé	$\notin$ clé	NON	NON	NON
$\subset$ clé	$\in$ clé	OK	OK	NON
autre	$\notin$ clé	OK	NON	NON
autre	$\in$ clé	OK	OK	NON

(Note: 'autre' signifie que X n'est ni une super-clé, ni un sous-ensemble propre d'une clé).

On a bien BCNF $\implies$ 3NF $\implies$ 2NF.

6.5 Normalisation par Décomposition

Si un schéma de relation n'est pas dans la forme normale souhaitée (par exemple, BCNF ou 3NF), on peut le **normaliser** en le **décomposant** en plusieurs schémas de relations plus petits qui, eux, satisfont la forme normale.

Definition 6.5.1 (Décomposition)**.** Soit $(R(A_1, ..., A_n), \mathcal{F})$ un schéma. Une **décomposition** de R est un ensemble de schémas $(R_1(V_1), \mathcal{F}_1), ..., (R_k(V_k), \mathcal{F}_k)$ où :

- Chaque V_i est un sous-ensemble des attributs de R ($V_i \subseteq \{A_1, ..., A_n\}$).
- L'union de tous les V_i recouvre tous les attributs de R ($\bigcup_{i=1}^{k} V_i = \{A_1, ..., A_n\}$).
- Chaque $\mathcal{F}_i$ est un ensemble de DFs sur V_i, généralement obtenu par projection de $\mathcal{F}$ sur V_i (i.e., les DFs de $\mathcal{F}^+$ dont tous les attributs sont dans V_i).

L'objectif de la décomposition est d'obtenir des relations R_i qui sont dans une forme normale élevée (e.g., BCNF), éliminant ainsi les anomalies de mise à jour présentes dans R. Cependant, la décomposition doit idéalement posséder deux propriétés importantes.

6.5.1 Décomposition sans Perte d'Information (Lossless Join)

Une décomposition est **sans perte d'information** si, pour toute instance r de R, la jointure naturelle de ses projections sur les R_i nous redonne exactement r.

$$r = \pi_{V_1}(r) \bowtie \pi_{V_2}(r) \bowtie \ldots \bowtie \pi_{V_k}(r)$$

Si cette propriété n'est pas satisfaite, la décomposition est dite "avec perte d'information", car la jointure peut générer des tuples "parasites" qui n'existaient pas dans r, ou perdre des tuples originaux (bien que ce dernier cas soit moins courant avec la projection/jointure naturelle).

Example 6.5.2 (Décomposition avec Perte)**.** Soit R(Titre, Nom-Ciné, Tél) sans DFs. Instance r:

Titre	Nom-Ciné	Tél
Speed 2	Français	..1187
Speed 2	Français	..1188
Marion	Français	..1187

Décomposons en R_1(Titre, Nom-Ciné) et R_2(Nom-Ciné, Tél). Projections $r_1 = \pi_{R_1}(r)$ et $r_2 = \pi_{R_2}(r)$:

r_1:

Titre	Nom-Ciné
Speed 2	Français
Marion	Français

r_2:

Nom-Ciné	Tél
Français	..1187
Français	..1188

Jointure $r_1 \bowtie r_2$ (sur Nom-Ciné):

Titre	Nom-Ciné	Tél
Speed 2	Français	..1187
Speed 2	Français	..1188
Marion	Français	..1187
Marion	**Français**	**..1188**

La jointure contient le tuple (Marion, Français, ..1188) qui n'était pas dans r. C'est une décomposition avec perte d'information. On ne peut plus savoir quel était le téléphone associé à Marion au cinéma Français.

Theorem 6.5.3 (Décomposition Binaire Sans Perte (Théorème de Heath)). La décomposition d'une relation $R(XYZ)$ (où X, Y, Z sont des ensembles d'attributs) en deux relations $R_1(XY)$ et $R_2(XZ)$ est **sans perte d'information** si et seulement si l'une des DFs suivantes est impliquée par $\mathcal{F}$ (les DFs de R):

- $X \to Y$ (l'intersection X détermine tous les attributs de R_1)
- ou $X \to Z$ (l'intersection X détermine tous les attributs de R_2)

où $X = V_1 \cap V_2$, $Y = V_1 - V_2$, $Z = V_2 - V_1$.

Example 6.5.4 (Décomposition Sans Perte). Soit R(A, B, C, D) avec DF: AB $\to$ C. Décomposition en $R_1(ABC)$ et $R_2(ABD)$. Ici $V_1 = \{A, B, C\}$, $V_2 = \{A, B, D\}$. Intersection $X = V_1 \cap V_2 = \{A, B\}$. Attributs propres à R_1: $Y = V_1 - V_2 = \{C\}$. Attributs propres à R_2: $Z = V_2 - V_1 = \{D\}$. La condition du théorème est : est-ce que $X \to Y$ (i.e., $AB \to C$) ou $X \to Z$ (i.e., $AB \to D$) est impliquée par $\mathcal{F}$? Oui, $AB \to C$ est donnée dans $\mathcal{F}$. Donc, la décomposition est sans perte d'information.

6.5.2 Décomposition avec Préservation des Dépendances

Une décomposition **préserve les dépendances** si toutes les DFs de l'ensemble original $\mathcal{F}$ peuvent être vérifiées en examinant chaque relation décomposée R_i individuellement. Formellement, si l'union des projections $\mathcal{F}_i$ est une couverture de $\mathcal{F}$ original. $(\bigcup \mathcal{F}_i)^+ = \mathcal{F}^+$.

Si une décomposition ne préserve pas les dépendances, cela signifie que certaines contraintes (DFs) de la relation originale s'étendent sur plusieurs des nouvelles relations et ne peuvent être vérifiées qu'en effectuant une jointure coûteuse entre elles.

Example 6.5.5 (Perte de Dépendance). Soit R(Titre, Acteur, Nom-Ciné) avec $\mathcal{F}$ = {Nom-Ciné $\to$ Titre, Titre, Acteur $\to$ Nom-Ciné}. (Cette combinaison de DFs est un peu étrange, mais suivons l'exemple de la diapo). Décomposition en R_1(Nom-Ciné, Titre) avec $\mathcal{F}_1$ = {Nom-Ciné $\to$ Titre} et R_2(Nom-Ciné, Acteur) avec $\mathcal{F}_2 = \emptyset$. La DF Nom-Ciné $\to$ Titre est préservée car elle peut être vérifiée sur R_1. Cependant, la DF Titre, Acteur $\to$ Nom-Ciné n'est pas préservée. Ses attributs {Titre, Acteur, Nom-Ciné} ne sont présents ensemble dans aucune des relations décomposées (R_1 ou R_2). Pour vérifier cette contrainte, il faudrait joindre R_1 et R_2.

Si l'on insère (Trianon, Bullock) dans R_2, la jointure peut produire des tuples violant la DF perdue. Par exemple, si R_1 contient (Français, Speed2) et (Trianon, Speed2), et R_2 contient (Français, Bullock) et (Trianon, Bullock), la jointure contiendra (Speed2, Bullock, Français) et (Speed2, Bullock, Trianon), ce qui viole Titre, Acteur $\to$ Nom-Ciné car (Speed2, Bullock) détermine deux Nom-Ciné différents.

6.5.3 Objectifs de la Normalisation

Idéalement, on cherche une décomposition qui soit : 1. En BCNF (ou au moins 3NF). 2. Sans perte d'information. 3. Avec préservation des dépendances.

Il est toujours possible d'obtenir une décomposition en BCNF et sans perte d'information. Il est toujours possible d'obtenir une décomposition en 3NF, sans perte d'information, et avec préservation des dépendances. Il n'est **pas toujours possible** d'obtenir une décomposition en BCNF qui soit à la fois sans perte et avec préservation des dépendances (l'exemple précédent en est une illustration si la décomposition était en BCNF). Dans ce cas, un compromis doit être fait : soit accepter la 3NF pour préserver les dépendances, soit accepter la BCNF et gérer la vérification des dépendances perdues au niveau applicatif ou par triggers.

6.6 Normalisation vs Dénormalisation

6.6.1 Normalisation : Avantages et Inconvénients

- **Avantages :**

  - Réduit la redondance.
  - Évite les anomalies de mise à jour (insertion, suppression, modification).
  - Améliore l'intégrité et la cohérence des données.
  - Facilite la maintenance.

- **Inconvénients :**
  - Peut nécessiter plus de jointures pour répondre à certaines requêtes, ce qui peut dégrader les performances.
  - Peut rendre certaines requêtes plus complexes à écrire.

6.6.2 Dénormalisation : Quand et Pourquoi ?

La **dénormalisation** est le processus inverse : combiner des relations qui étaient séparées (souvent après une normalisation) ou introduire de la redondance calculée pour améliorer les performances de certaines requêtes critiques.

On peut envisager la dénormalisation si :

- Une requête très fréquente et critique en termes de performance nécessite la jointure de plusieurs tables fortement normalisées.
- Les données sont rarement mises à jour, minimisant les risques liés aux anomalies.

La dénormalisation est un compromis : on échange la "pureté" du modèle et la facilité de mise à jour contre de meilleures performances en lecture pour des cas spécifiques. Elle doit être utilisée avec précaution et de manière ciblée.

Example 6.6.1 (Composition/Dénormalisation)**.** Si les requêtes demandant le Titre, M-e-S, Acteur, Nom-Ciné, Adr, Tél pour une projection sont très fréquentes, et si la base de données BD-Cine décomposée en FILM, PROG, CINE (comme vu page 16) est trop lente à cause des jointures, on pourrait envisager de revenir à la relation unique BD-Cine, en acceptant la redondance et les risques d'anomalies, mais en optimisant les requêtes de lecture.

6.7 Conclusion

La conception de schémas relationnels est une étape fondamentale dans la création d'une base de données robuste et efficace. La théorie des dépendances fonctionnelles et des formes normales fournit un cadre rigoureux pour analyser la qualité d'un schéma et le guider vers une structure minimisant la redondance et les anomalies de mise à jour. La normalisation, via la décomposition, est le processus standard pour atteindre ces objectifs, en visant idéalement la BCNF ou la 3NF, tout en préservant l'information (lossless join) et si possible les dépendances. La dénormalisation reste une option pour des optimisations de performance ciblées, mais doit être considérée avec prudence en raison de ses inconvénients potentiels sur l'intégrité et la maintenance des données.

Chapter 7

SQL

CHAPITRE 3 : MODÈLE de DONNÉES

INTRODUCTION AUX BASES de DONNÉES

Plan du Chapitre

- Le Modèle Relationnel
- SQL : Langage de définition (LDD)
- Du Modèle E-A au modèle relationnel

7.1 Introduction à SQL

SQL signifie **Structured Query Language**. Sa première version a été développée par IBM en 1970 par Donald Chamberlain & Raymond Boyce, initialement sous le nom de SEQUEL (Structured English as a QUEry Language).

SQL est standardisé par des normes telles que :

- norme ANSI (American National Standards Institute)
- standard ISO (International Organisation for Standardization)

Avantages

- SQL est implanté, plus ou moins complètement, sur les principaux Systèmes de Gestion de Bases de Données (SGBDs).
- Il offre une portabilité des applications et une interopérabilité entre différents systèmes.

Inconvénients

- Le langage évolue par des "extensions" successives, ce qui peut complexifier sa maîtrise et sa standardisation réelle. Les versions majeures incluent SQL1 (1989), SQL2 (1992), SQL3 (1999), SQL:2003, SQL:2008, SQL:2011, SQL:2016.
- Cette évolution par extensions peut freiner l'émergence de nouveaux langages potentiellement plus modernes ou adaptés.

Composants et Modes d'Utilisation de SQL

SQL est composé de plusieurs sous-langages :

- **Langage de définition de données (LDD)** : Utilisé pour la création des schémas ('CREATE TABLE', 'ALTER TABLE', 'DROP TABLE') et la déclaration des contraintes d'intégrité.
- **Langage de requêtes (LMD)** : Permet d'interroger la base de données. Inclut les requêtes simples ('SELECT ... FROM ... WHERE'), les requêtes imbriquées, et les agrégats ('COUNT', 'GROUP BY', 'HAVING').
- **Langage de mise à jour (LMD)** : Utilisé pour l'insertion ('INSERT'), la suppression ('DELETE'), et la modification ('UPDATE') des données.
- **Autres aspects** : Incluent la définition de vues ('CREATE VIEW') et de déclencheurs ('CREATE TRIGGER').

SQL peut être utilisé de différentes manières :

- À la console, en mode interactif.
- En association avec des interfaces graphiques.
- En association avec des langages de programmation (C, C++, JAVA, PHP, ...).

L'expérimentation en SQL se fait souvent :

- En Travaux Pratiques (TP).
- Via de nombreux sites web offrant des interfaces SQL interactives.

7.2 Langage de Définition de Données (LDD)

7.2.1 Définition de Schéma de Relation

Commande simple

La commande de base pour créer une table est 'CREATE TABLE'.

```
CREATE TABLE Nom_Relation (
    Attribut_1 Type_1,
    Attribut_2 Type_2,
    ...
    Attribut_n Type_n
);
```

Types de données

SQL supporte divers types de données de base, et d'autres plus spécifiques :

- Numériques : 'INT', 'SMALLINT', 'BIGINT', 'FLOAT', ...
- Chaînes de caractères : 'CHAR(M)' (fixe), 'VARCHAR(M)' (variable), ...
- Objets larges : 'BLOB', ...
- Dates et heures : 'DATE', 'TIME', 'DATETIME', 'YEAR', ...

Il est important de consulter la norme SQL et le manuel du SGBD utilisé (par exemple, PostgreSQL en TP) pour connaître les types exacts et leurs spécificités.

Exemple de création de tables

Attention : L'exemple suivant est modifié par rapport à une version précédente pour ajouter l'attribut 'Num$_F$'$danslatable$'$Film$'.

```
CREATE TABLE Film (
    NumF INT,
    Titre CHAR(20),
    Annee DATE,
    Duree TIME
);

CREATE TABLE Prog (
    Nom_Cine CHAR(20),
    NumF INT,
    Salle INT,
    Horaire TIME
);

CREATE TABLE Cine (
    Nom_Cine CHAR(20),
    Adresse VARCHAR(60),
    Telephone CHAR(8) NOT NULL
);
```

7.2.2 Définition des Contraintes d'Intégrité

Les contraintes assurent la cohérence et la validité des données.

Exemple avec contraintes

```
CREATE TABLE Film (
    NumF INT PRIMARY KEY,
    Titre CHAR(20) NOT NULL,
    Annee DATE,
    Duree TIME
);

CREATE TABLE Prog (
    Nom_Cine CHAR(20),
    NumF INT,
    Salle INT,
    Horaire TIME,
    UNIQUE (Nom_Cine, NumF, Horaire)
);

CREATE TABLE Cine (
    Nom_Cine CHAR(20),
    Adresse VARCHAR(60),
    Telephone CHAR(8) NOT NULL,
    PRIMARY KEY (Nom_Cine)
);
```

Valeurs nulles et par défaut

- 'NOT NULL' : Empêche l'insertion d'une valeur nulle pour cet attribut.

- 'DEFAULT valeur' : Assigne une valeur par défaut si aucune valeur n'est fournie lors de l'insertion.

Exemple :

```
CREATE TABLE Nom_Relation (
    Attribut_1 Type_1,
    Attribut_2 Type_2 NOT NULL,
    ...
    Attribut_n Type_n DEFAULT 'Hello'
);
```

Clé primaire

- 'PRIMARY KEY ($K_1, ..., K_n$)' : $Définit la clé primaire de la table, composée d'un ou plusieurs attributs. Les attributs de la cl$

- Assure l'unicité de chaque tuple dans la table.

Autre Clé (Unique)

- 'UNIQUE ($K_1, ..., K_n$)' : $Assure l'unicité des valeurs pour la combinaison d'attributs spécifiée. Contrairement à `PRIMARY K$
  $Plusieurs tuples peuvent avoir `NULL` dans une colonne `UNIQUE`, car `NULL` n'est pas considéré comme égal à `NULL`.$
  Attention : 'PRIMARY KEY' $\neq$ 'UNIQUE' pour les valeurs nulles. Une table ne peut avoir qu'une seule 'PRIMARY KEY', mais plusieurs contraintes 'UNIQUE'.

7.2.3 Déclaration de Contraintes (par l'exemple)

Contraintes de référence (Clé étrangère)

Une clé étrangère établit un lien entre deux tables, assurant l'intégrité référentielle.

Syntaxe 1 : Inline La contrainte est définie directement lors de la définition de l'attribut.

```
CREATE TABLE Prog (
    Nom_Cine CHAR(20),
    NumF INT REFERENCES Film(NumF), -- Reference Film via NumF
    Salle INT,
    Horaire TIME
);

CREATE TABLE Film (
    NumF INT PRIMARY KEY,
    Titre CHAR(20) NOT NULL,
    Annee DATE,
    Duree TIME
);
```

Syntaxe 2 : Séparée La contrainte est définie après la liste des attributs.

```
CREATE TABLE Nom_Relation (
    Attribut_1 Type_1,
    Attribut_2 Type_2, ...
    Attribut_n Type_n,
    FOREIGN KEY (A_1, ..., A_n) REFERENCES Table1(K_1, ..., K_n)
);
```

Syntaxe 3 : ALTER TABLE La contrainte est ajoutée à une table existante.

```
ALTER TABLE Prog
ADD CONSTRAINT consKEprog
FOREIGN KEY (NumF) REFERENCES Film(NumF);
```

Actions en cas de violation de contrainte de référence

Que se passe-t-il quand une mise à jour (suppression ou modification dans la table référencée, insertion ou modification dans la table référençante) vient invalider une contrainte de référence ?

Cas : insertion / modification initiale dans Prog. Si l'on tente d'insérer dans 'Prog' un 'NumF' qui n'existe pas dans 'Film', l'opération est rejetée (**rejet**).

Cas : suppression / modification initiale dans Film. Si l'on supprime ou modifie un 'NumF' dans 'Film' qui est référencé dans 'Prog', plusieurs stratégies sont possibles :

- **Rejet (par défaut)** : L'opération sur 'Film' est refusée.
- **Cascade** : L'opération est répercutée sur 'Prog'. Si un film est supprimé, les programmations correspondantes sont aussi supprimées. Si le 'NumF' est modifié, il est modifié aussi dans 'Prog'.
- **Valeur nulle (SET NULL)** : Le 'NumF' dans les tuples correspondants de 'Prog' est mis à 'NULL' (si l'attribut 'NumF' dans 'Prog' autorise les 'NULL').

Exemples illustrés Considérons les données suivantes :

Table 7.1: Table FILM

NumF	Titre	Année	Durée
f1	T1	A1	D1
f2	T2	A1	D2
f3	T3	A2	D3
f4	T4	A1	D4

Table 7.2: Table PROG

Nom-Ciné	NumF	Salle	Horaire
NC1	f2	S1	16h00
NC1	f2	S1	20h00
NC1	f3	S2	18h00
NC2	f3	S5	18h00
NC2	f3	S5	20h00

- **Suppression du film f3**
  - *Stratégie Rejet (défaut)* : Suppression impossible car f3 est référencé dans PROG.
  - *Stratégie Cascade* : Les 3 lignes de PROG référençant f3 sont supprimées. La table PROG devient :

Table 7.3: Table PROG après suppression f3 (Cascade)

Nom-Ciné	NumF	Salle	Horaire
NC1	f2	S1	16h00
NC1	f2	S1	20h00

– *Stratégie Valeur Nulle (SET NULL)* : Le NumF des 3 lignes de PROG référençant f3 est mis à NULL. La table PROG devient :

Table 7.4: Table PROG après suppression f3 (SET NULL)

Nom-Ciné	NumF	Salle	Horaire
NC1	f2	S1	16h00
NC1	f2	S1	20h00
NC1	null	S2	18h00
NC2	null	S5	18h00
NC2	null	S5	20h00

- **Modification de la valeur f3 en f5 dans FILM**
  - *Stratégie Rejet (défaut)* : Modification impossible.
  - *Stratégie Cascade* : Le NumF des 3 lignes de PROG référençant f3 est mis à f5. La table PROG devient :

Table 7.5: Table PROG après modification f3-¿f5 (Cascade)

Nom-Ciné	NumF	Salle	Horaire
NC1	f2	S1	16h00
NC1	f2	S1	20h00
NC1	f5	S2	18h00
NC2	f5	S5	18h00
NC2	f5	S5	20h00

    La table FILM contiendrait f5 au lieu de f3.
  - *Stratégie Valeur Nulle (SET NULL)* : Le NumF des 3 lignes de PROG référençant f3 est mis à NULL (comme pour la suppression).

Choix d'une stratégie La stratégie est définie lors de la création de la contrainte de clé étrangère.

```
CREATE TABLE Prog (
    Nom_Cine CHAR(20),
    NumF INT,
    Salle INT,
    Horaire TIME,
    FOREIGN KEY (NumF) REFERENCES Film(NumF)
        ON DELETE SET NULL
        ON UPDATE CASCADE
);
```

7.2.4 Contraintes CHECK

Permet de définir des conditions de validation plus générales.

Contrainte sur un attribut

La condition est vérifiée pour toute valeur insérée ou modifiée de cet attribut. La vérification a lieu lors de l'insertion ou de la modification.

```
CREATE TABLE Prog (
    Nom_Cine CHAR(20),
```

```
    -- Assure que NumF existe dans la table Film
    -- Attention: Ceci est DIFFERENT d'une contrainte de reference!
    -- Ne gere pas les suppressions dans Film.
    NumF INT CHECK (NumF IN (SELECT NumF FROM Film)),
    Salle INT,
    Horaire TIME
);
```

Attention : Ceci est $\neq$ contrainte de référence !!! (ne gère pas la suppression dans Film).

Contrainte sur un n-uplet

La condition porte sur plusieurs attributs du tuple et est vérifiée à chaque insertion ou modification du tuple.

```
CREATE TABLE Prog (
    Nom_Cine CHAR(20),
    NumF INT,
    Salle INT,
    Horaire TIME,
    -- Verifie que seules les salles 'Artessai' programment
    -- des films apres 22:30:00
    CHECK (Salle = 'Artessai' OR Horaire < '22:30:00')
);
```

Explication : Seules les salles "Artessai" programment des films après 22:30.

7.2.5 Assertion

Une assertion définit une contrainte globale sur la base de données, pouvant impliquer plusieurs tables. Elle est vérifiée lors de toute mise à jour susceptible de la violer (sur les relations Prog et Film dans l'exemple).

Contrainte exemple : Le nombre de salles dans lesquelles est programmé un film de Poirier doit être inférieur à 10 fois le nombre de films de ce cinéaste. (Vérification grossière / approximative).

```
CREATE ASSERTION film_cine
CHECK (
    -- Nombre de salles programmant un film de Poirier
    (SELECT COUNT(Salle) FROM Prog
     WHERE NumF IN (SELECT NumF FROM Film WHERE MeS = 'Poirier'))
    <
    -- 10 fois le nombre de films de Poirier
    (SELECT COUNT(DISTINCT Titre) FROM Film WHERE MeS = 'Poirier') * 10
);
```

Note : La syntaxe exacte de la condition dans l'image originale ('HAVING Titre IN SELECT Titre ... WHERE MeS = 'Poirier'') semble incorrecte ou spécifique à un dialecte SQL. La version ci-dessus est une interprétation plus standard de l'intention.

7.3 Langage de Manipulation de Données (LMD) - Requêtes

7.3.1 Requête Simple / Mono-relation

La structure de base d'une requête SQL est :

```
SELECT <liste d'attributs>
FROM␣<liste␣de␣relations>
WHERE␣<condition>;
```

- Clause **FROM** : Spécifie la ou les relations source(s) utiles à la requête (extraction).
- Clause **WHERE** : Filtre les n-uplets selon des conditions (sélection et jointure).
- Clause **SELECT** : Définit les attributs du schéma cible (projection).

Exemples mono-relation

- Lister tous les attributs de tous les films :

```
SELECT * FROM FILM;
```

  Algèbre : $[FILM]$

- Lister tous les attributs des films où l'acteur est 'Adjani' :

```
SELECT * FROM FILM WHERE Acteur='Adjani';
```

  Algèbre : $\sigma_{\text{Acteur='Adjani'}}[\text{FILM}]$

- Lister les titres des films où l'acteur est 'Adjani' (avec doublons potentiels) :

```
SELECT Titre FROM FILM WHERE Acteur='Adjani';
```

  Algèbre : $\Pi_{\text{Titre}}(\sigma_{\text{Acteur='Adjani'}}[\text{FILM}])$ (projection classique élimine doublons, SQL non)

- Lister les titres uniques des films où l'acteur est 'Adjani' :

```
SELECT DISTINCT Titre FROM FILM WHERE Acteur='Adjani';
```

  Algèbre : $\Pi_{\text{Titre}}(\sigma_{\text{Acteur='Adjani'}}[\text{FILM}])$

Attention : Par défaut, SQL n'élimine pas les dupliqués / doublons dans le résultat d'un 'SELECT'. Utilisez 'DISTINCT' pour les supprimer.

Sémantique formelle (cas mono-relation)

- 'SELECT B1, ..., Bk FROM R WHERE C' correspond à la projection "multi-ensembliste" $\Pi^*_{B_1 \ldots B_k} \sigma_C[R]$.
- 'SELECT DISTINCT B1, ..., Bk FROM R WHERE C' correspond à la projection ensembliste (classique) $\Pi_{B_1 \ldots B_k} \sigma_C[R]$.

Discussion sur DISTINCT

- 'DISTINCT' permet de ne garder (à l'affichage) qu'une seule ligne résultat parmi plusieurs totalement identiques.
- **Coût :** 'DISTINCT' implique généralement un tri externe, ce qui peut être coûteux en performance.
- **Inutilité :** Ne pas mettre 'DISTINCT' si c'est inutile, par exemple si la clé primaire de la table est incluse dans la liste 'SELECT'.

```
-- Table: FILM-Bis(Num_f PRIMARY KEY, Titre, Duree)
-- Inutile: Num_f est une cle de FILM-Bis
SELECT DISTINCT Num_f, Titre FROM FILM-Bis WHERE duree = 2;
```

- **Utilité :** Mettre 'DISTINCT' si des doublons sont possibles et non désirés.

```
-- Utile: Plusieurs films peuvent avoir la meme duree
SELECT DISTINCT Titre FROM FILM-Bis WHERE duree = 2;
```

On reprendra la discussion plus tard (agrégat).

Fonctionnalités additionnelles du SELECT

- **Renommage des attributs** avec 'AS' :

```
SELECT Titre AS Adjani_s_movies FROM FILM WHERE Acteur='Adjani';
```

- **Valeur calculée** (expression arithmétique) :

```
-- Ajout d'un attribut Duree en minutes
-- Table: FILM-duree(..., Duree, ...)
SELECT Titre, Duree * 0.016667 AS duree_en_heure FROM FILM-duree;
```

- **Constante** dans une colonne :

```
SELECT Titre, Duree * 0.016667 AS duree_en_heure, 'heure' AS Unite
FROM FILM-duree;
```

Attention : Chaînes de caractères et Divers

- **Chaînes de caractères :** Longueur fixe ('CHAR') vs. variable ('VARCHAR'). Les chaînes de longueur fixe peuvent être complétées par des blancs. La gestion des majuscules/minuscules dépend du SGBD et de la configuration :
  - Mots clés SQL : généralement insensibles à la casse.
  - Objets de la base (tables, attributs) : dépend du SGBD (souvent insensible ou converti).
  - Valeurs des conditions (données) : généralement sensibles à la casse.
- **Divers :**
  - La table résultat est une table temporaire.
  - L'ordre des attributs dans le résultat correspond à l'ordre d'apparition dans le 'SELECT' (ou l'ordre de la table pour 'SELECT *').

7.3.2 Condition de la Clause WHERE

La clause 'WHERE' filtre les tuples en fonction d'une condition.

- **Comparateurs "habituels"** : '¡', '¿', '=', '!=' (ou '¡¿'), '¡=', '¿='.
- **Comparateurs spécifiques** :
  - 'LIKE' : Recherche de motifs dans les chaînes.
  - 'BETWEEN' : Vérifie si une valeur est dans un intervalle.
  - 'IN' : Vérifie si une valeur appartient à une liste ou au résultat d'une sous-requête.
  - 'IS [NOT] NULL' : Teste si une valeur est nulle.
- **Expressions** : Arithmétique, concaténation de chaînes ('——'), ...
- **Connecteurs logiques** : 'OR', 'AND', 'NOT'.

Exemples de conditions

```
-- Films avec Adjani OU realises par Poirier
SELECT Titre FROM FILM WHERE Acteur='Adjani' OR MeS='Poirier';

-- Films avec Adjani OU Depardieu (equivalent a OR)
SELECT Titre FROM FILM WHERE Acteur IN ('Adjani', 'Depardieu');

-- Films avec Adjani ET Depardieu (attention: ceci concerne differents
   ↪ tuples
-- pour le meme film si la structure le permet, ou necessite une jointure
-- si Acteur est mono-value. L'exemple simple serait:)
SELECT Titre FROM FILM WHERE Acteur='Adjani' AND Acteur='Depardieu';
-- Cette derniere condition est toujours fausse si un film n'a qu'un
   ↪ acteur.
-- Une requete plus realiste impliquerait une jointure ou une sous-requete
   ↪ .
-- Exemple correct avec OR (equivalent a IN) :
SELECT Titre FROM FILM WHERE Acteur='Adjani' OR Acteur='Depardieu';
```

Motifs (LIKE)

- ‘

- ‘ : *remplaceununiquecaractère*.

```
-- Titres commencant par 'Star ' suivi de 4 caracteres
-- ex: Star Wars, Star Trek, ...
SELECT Titre FROM FILM WHERE Titre LIKE 'Star␣____';

-- Titres contenant le mot 'retour'
-- ex: Le retour a Sarajevo, Aliens le retour, ...
SELECT Titre FROM FILM WHERE Titre LIKE '%retour%';
```

Manipulation de dates

Le format standard est souvent 'AAAA-MM-JJ'.

```
-- Date specifique
... WHERE DateSortie = DATE '2003-11-06';

-- Intervalle de dates
... WHERE DateSortie BETWEEN DATE '2003-09-25' AND DATE '2004-02-15';
```

Valeurs nulles

- Une valeur ‘NULL‘ représente une valeur inconnue, inappropriée ou incertaine. Elle est différente de 0 ou d'une chaîne vide.
- **Comparaison :** La comparaison avec ‘NULL‘ (ex: ‘attribut = NULL‘) produit toujours le résultat ‘inconnu‘ (ni vrai, ni faux) selon la logique ternaire (vrai=1, faux=0, inconnu=1/2).
  - ‘x AND y = min(x, y)‘
  - ‘x OR y = max(x, y)‘
  - ‘NOT x = 1 - x‘
- **Conditions :** Pour tester la nullité, utiliser ‘IS NULL‘ ou ‘IS NOT NULL‘.

- **ATTENTION :** La clause ‘WHERE‘ ne sélectionne que les n-uplets pour lesquels la condition est évaluée à ‘VRAI‘. Si la condition est ‘FAUX‘ ou ‘INCONNU‘, le n-uplet n’est pas sélectionné. Donc, ‘WHERE attribut = NULL‘ ne sélectionnera **AUCUN** n-uplet.

7.3.3 Ordonnancement (ORDER BY)

La clause ‘ORDER BY‘ trie les lignes du résultat final.

- Syntaxe : ‘ORDER BY colonne1 [ASC—DESC], colonne2 [ASC—DESC], ...‘
- ‘ASC‘ (ascendant) est la valeur par défaut. ‘DESC‘ (descendant).
- Le tri est lexicographique sur les colonnes spécifiées.

```
-- Table: FILM-Bis(Num_f, Titre, Duree)
-- Selectionner les titres des films durant 2 heures ou plus, tries par
   ↪ duree
SELECT Titre
FROM FILM-Bis
WHERE Duree >= 2 -- Supposant Duree en heures ou une unite comparable
ORDER BY Duree; -- Tri ascendant par defaut
```

7.3.4 Requête Multi-relation

Implique plusieurs tables dans la clause ‘FROM‘.

Nom des attributs

Quand un nom d’attribut est ambigu (présent dans plusieurs tables de ‘FROM‘), il faut le préfixer par le nom de la table (ou son alias) : ‘NomTable.NomAttribut‘ ou ‘Alias.NomAttribut‘. Exemple : ‘R(A1, ..., An)‘, ‘S(B1, ..., Bm)‘. Si ‘R‘ et ‘S‘ ont un attribut en commun (ex: ‘Id‘), on utiliserait ‘R.Id‘ et ‘S.Id‘.

Jointure

La jointure combine des tuples de plusieurs tables basées sur une condition.

- **Modèle ”formel” (algèbre)** : Jointure naturelle $R \bowtie S$ ou thêta-jointure $R \bowtie_\theta S$.
- **SQL (implicite)** : Se fait via la clause ‘WHERE‘. Pour joindre ‘R(A, B)‘ et ‘S(B, C)‘ sur l’attribut commun ‘B‘:

  ```
  SELECT R.A, R.B, S.C -- Ou autres attributs necessaires
  FROM R, S
  WHERE R.B = S.B; -- Condition de jointure
  ```

  Ceci correspond en algèbre (avec renommage implicite des attributs par SQL) à $\Pi_{R.A,R.B,S.C}(\sigma_{R.B=S.B}(R \times S))$.

Sémantique formelle (cas multi-relation)

‘SELECT [DISTINCT] B1...Bk FROM R1 ... Rp WHERE C‘ Correspond à $\Pi^*_{B_1 \dots B_k}(\sigma_C(R_1 \times \cdots \times R_p))$ (projection multi-ensembliste). Les attributs ‘B1...Bk‘ doivent être non ambigus (ou préfixés). Le produit cartésien $R_1 \times \cdots \times R_p$ est filtré par la condition C (qui inclut typiquement les conditions de jointure), puis projeté.

Exemple de jointure

Quels sont les cinémas qui projettent un film dans lequel M.F. Pisier joue ? (Pour chaque cinéma, donner le(s) titre(s) du(es) film(s) et le(s) horaire(s)).

Analyse :

1. Relations impliquées : 'FILM' (pour trouver les films avec M.F. Pisier) et 'PROG' (pour trouver les cinémas et horaires de ces films).
2. Attributs de jointure : L'attribut qui lie 'FILM' et 'PROG' est le numéro ou le titre du film. Supposons que c'est 'Titre'. Donc, 'FILM.Titre = PROG.Titre'.
3. Condition de sélection : 'Acteur = 'M-F. Pisier'' sur la table 'FILM'.
4. Attributs à afficher : 'Nom-Cine', 'FILM.Titre', 'Horaire'.

```
SELECT DISTINCT P.Nom_Cine, F.Titre, P.Horaire
FROM FILM F, PROG P
WHERE F.Titre = P.Titre         -- Condition de jointure
  AND F.Acteur = 'M-F.␣Pisier'; -- Condition de selection
```

Note : Utilisation d'alias 'F' et 'P' pour plus de clarté et de concision.

Introduction de variables (Alias)

Il est parfois nécessaire d'utiliser la même table plusieurs fois dans une requête, comme si on en avait plusieurs copies. On utilise alors des alias.

Exemple : Trouver les films (Titre, MeS, Acteur) qui ont le même titre qu'un film dans lequel joue M-F. Pisier. (En pratique, trouver les metteurs en scène et acteurs des films où joue M-F. Pisier).

```
SELECT F2.Titre, F2.MeS, F2.Acteur
FROM FILM F1, FILM F2
WHERE F1.Titre = F2.Titre -- Jointure sur le titre
  AND F1.Acteur = 'M-F.␣Pisier'; -- Selection sur la "copie" F1
```

Interprétation alternative / Sémantique "itérative" : 'F1' et 'F2' sont des variables (alias) utilisées pour désigner n'importe quel couple de n-uplets de 'FILM'. L'algorithme implicite serait :

```
Pour tous les n-uplets f1 de FILM faire
  Pour tous les n-uplets f2 de FILM faire
    si (f1.Titre = f2.Titre ET f1.Acteur = 'M-F. Pisier') est vraie alors
      extraire f2.Titre, f2.MeS, f2.Acteur;
```

Note : Cet algorithme n'est qu'un squelette conceptuel, l'optimiseur SQL utilisera des stratégies bien plus efficaces.

Le Point : Structure Générale d'une Requête

```
SELECT [DISTINCT] exp1 [AS label1], exp2 [AS label2], ...
FROM relation1 [alias1], relation2 [alias2], ...
[WHERE condition]
[GROUP BY colonne1, colonne2, ...]
[HAVING condition_groupe]
[ORDER BY exp1 [DESC], exp2 [DESC], ...]
```

Où :

- 'exp1', 'exp2'... peuvent être des noms de colonnes, des constantes, des expressions (opérateurs binaires, fonctions, agrégats...).
- 'condition' dans 'WHERE' peut utiliser des opérateurs logiques, des comparaisons, des sous-requêtes...

- ‘GROUP BY‘ et ‘HAVING‘ sont utilisés pour les agrégats.
- ‘ORDER BY‘ trie le résultat final.

7.3.5 Ensembles et Multi-ensembles

- Un **multi-ensemble** est une collection d’éléments où les répétitions sont permises (ex: ‘1, 2, 1, 3‘). Un ensemble n’autorise pas les répétitions (ex: ‘1, 2, 3‘).
- Par défaut, SQL travaille avec des multi-ensembles (il conserve les doublons).
- ‘DISTINCT‘ force une sémantique d’ensemble pour le ‘SELECT‘.
- Les opérations ensemblistes (‘UNION‘, ‘EXCEPT‘, ‘INTERSECT‘) ont deux variantes :
  - Par défaut (‘UNION‘, ‘EXCEPT‘, ‘INTERSECT‘) : Éliminent les doublons (sémantique d’ensemble).
  - Avec ‘ALL‘ (‘UNION ALL‘, ‘EXCEPT ALL‘, ‘INTERSECT ALL‘) : Conservent les doublons (sémantique de multi-ensemble). ‘EXCEPT ALL‘ et ‘INTERSECT ALL‘ sont moins courants.

Table 7.6: Opérations et modes

Opération	**Mode ensembliste**	**Mode multi-ensemble**
select-from-where	‘DISTINCT‘	par défaut (‘ALL‘)
Union, Except, Intersect	par défaut	‘ALL‘

Exemples

- Élimination des dupliqués :

```
SELECT DISTINCT Titre FROM FILM;
```

- ”Conservation” des dupliqués avec ‘UNION ALL‘ :

```
-- Films avec R.A > 10 OU R.B = 5, en conservant les doublons
-- si un film satisfait les deux conditions
SELECT * FROM R WHERE R.A > 10
UNION ALL
SELECT * FROM R WHERE R.B = 5;
```

7.3.6 Sous-Requêtes et Imbrication

Une sous-requête est une requête ‘SELECT‘ imbriquée à l’intérieur d’une autre requête (dans les clauses ‘WHERE‘ ou ‘HAVING‘ principalement, parfois ‘SELECT‘ ou ‘FROM‘).

Idée de base

Une condition simple compare un attribut à une valeur : ‘A comp val‘. La généralisation compare un attribut au résultat d’une sous-requête : ‘A comp (SELECT ...)‘ Le comportement de ‘comp‘ dépend du type de résultat retourné par la sous-requête (une valeur, un ensemble de valeurs, etc.). **Attention :** Le plus souvent, ‘RESQ‘ (le résultat de la sous-requête) est un ensemble de n-uplets ! ‘comp‘ devient alors un test d’appartenance (‘IN‘) ou d’existence (‘EXISTS‘), etc.

Utilisation du résultat d'un Select-From-Where

Cas 1 : Sous-requête retournant une seule valeur Si la sous-requête garantit de retourner au plus une ligne et une colonne, on peut utiliser les comparateurs scalaires ('=', '¡', '¿', etc.).

Exemple : Trouver les acteurs du premier film (supposons une table 'FILM-DEB(Titre, Acteur)' qui stocke LE titre du premier film de chaque acteur) joué par M-F. Pisier.

```
SELECT Acteur
FROM FILM -- Table principale des films
WHERE Titre = ( SELECT Titre -- Sous-requete retournant 1 seul titre
                FROM FILM-DEB
                WHERE Acteur = 'M-F. Pisier' );
```

Condition : Le résultat de la sous-requête est une valeur, une seule valeur !

Cas 2 : Sous-requête retournant un ensemble de valeurs (une colonne) L'opérateur 'IN' teste l'appartenance d'une valeur à l'ensemble retourné par la sous-requête.

Exemple : Trouver les titres des films dont un des MeS (metteur en scène) est aussi un acteur (pas forcément dans le même film).

```
SELECT Titre
FROM FILM
WHERE MeS IN ( SELECT Acteur -- Sous-requete retournant un ensemble d'
    ↪ acteurs
               FROM FILM );
-- Renommage AS MeS n'est pas necessaire si la sous-requete
-- selectionne directement la colonne qui a le bon "sens".
-- Version equivalente de l'exemple image page 225:
SELECT Titre FROM FILM WHERE MeS IN (SELECT Acteur FROM FILM);
```

Le résultat de la sous-requête est un ensemble de n-uplets sur MeS (ou Acteur).

Cas 3 : Sous-requête testant l'existence (EXISTS) 'EXISTS (sous-requête)' est vrai si la sous-requête retourne au moins une ligne, faux sinon. 'NOT EXISTS' est l'inverse. La sous-requête est souvent corrélée à la requête externe (elle utilise des valeurs de la requête externe).

Exemple 1 : Trouver les films dirigés par au moins deux metteurs en scène (suppose qu'un film peut avoir plusieurs lignes avec différents MeS).

```
SELECT DISTINCT F1.Titre
FROM FILM F1
WHERE EXISTS ( SELECT F2.MeS -- On cherche s'il existe un autre MeS
               FROM FILM F2
               WHERE F1.Titre = F2.Titre -- pour le meme film
                 AND F1.MeS <> F2.MeS ); -- mais different de F1.MeS
```

Note : L'exemple de l'image p226 ('NOT F1.MeS=F2.MeS') est ambigu. '¡¿' ou '!=' est préférable.

Exemple 2 : Trouver les films dont au moins un acteur a joué dans un autre film.

```
SELECT DISTINCT F1.Titre
FROM FILM F1
WHERE EXISTS ( -- Existe-t-il un acteur F2 dans le film F1...
    SELECT F2.Acteur
    FROM FILM F2
    WHERE F1.Titre = F2.Titre
      AND EXISTS ( -- ...tel que cet acteur F2 a joue dans un autre film
          ↪ F3?
          SELECT F3.Titre
          FROM FILM F3
```

```
            WHERE F2.Acteur = F3.Acteur -- Meme acteur
              AND F1.Titre <> F3.Titre -- Mais film different de F1
      )
);
```

Note : L'exemple de l'image p227 utilise 'Not (F3.Titre=F1.Titre)', ce qui est équivalent à '¡¿'.

Comparaison IN vs EXISTS Plusieurs façons d'exprimer la même requête, notamment pour la jointure. Soient 'R(A,B,C)' et 'S(B,C,D)'. Pour exprimer $\Pi_A(R \bowtie S)$:

1. Jointure classique :

   ```
   SELECT DISTINCT R.A FROM R, S WHERE R.B=S.B AND R.C=S.C;
   ```

2. Avec 'IN' (sur tuple) :

   ```
   SELECT DISTINCT A FROM R WHERE (R.B, R.C) IN (SELECT S.B, S.C FROM S);
   ```

3. Avec 'EXISTS' (corrélée) :

   ```
   SELECT DISTINCT A FROM R R1 WHERE EXISTS (SELECT S.B, S.C FROM S WHERE
      ↪  R1.B = S.B AND R1.C = S.C);
   ```

Attention : ÉVITEZ l'usage de sous-requêtes (surtout corrélées comme avec 'EXISTS') lorsque la même logique peut être exprimée par une jointure directe, car les SGBD optimisent généralement mieux les jointures explicites.

Cas 4 : Constructions comp ALL / comp ANY Ces opérateurs comparent une valeur scalaire à un ensemble de valeurs retourné par une sous-requête (une colonne).

- 'valeur comp ANY (sous-requête)' : Vrai si la comparaison 'comp' est vraie pour **au moins une** des valeurs retournées par la sous-requête. ('= ANY' est équivalent à 'IN').
- 'valeur comp ALL (sous-requête)' : Vrai si la comparaison 'comp' est vraie pour **toutes** les valeurs retournées par la sous-requête (y compris si la sous-requête ne retourne aucune ligne).

Exemple '¿ ALL' : Trouver les films projetés à l'UGC plus tard que **tous** les films projetés au Trianon.

```
SELECT Titre
FROM PROG
WHERE Nom_Cine = 'UGC'
  AND Horaire > ALL ( SELECT Horaire FROM PROG WHERE Nom_Cine = 'Trianon'
     ↪ );
```

Teste si la valeur de 'Horaire' (pour un film UGC) est supérieure à **tous** les éléments du résultat de la sous-requête.

Exemple '¡ ANY' : Trouver le téléphone des cinémas qui proposent une programmation après 23h.

```
SELECT Telephone
FROM CINE C1
WHERE 23 < ANY ( SELECT Horaire -- Convertir Horaire en nombre si besoin
                 FROM PROG P1
                 WHERE C1.Nom_Cine = P1.Nom_Cine );
```

Teste si la valeur 23 est inférieure à **UN** des éléments ('Horaire') du résultat de la sous-requête pour ce cinéma.

Résumé Conditions Complexes et Sous-requêtes

- 'exp comp sous-requête' (où 'comp' est '¡', '¿', '¡=', '¿=', '=', '¡¿') : La sous-requête doit renvoyer **au plus 1 n-uplet** (1 ligne, 1 colonne).
- 'exp [NOT] IN sous-requête' : La sous-requête doit renvoyer **1 ensemble** de valeurs (plusieurs lignes, 1 colonne).
- '[NOT] EXISTS sous-requête' : La sous-requête peut renvoyer n'importe quel nombre de lignes/-colonnes. Seule l'existence d'au moins une ligne compte.
- 'exp comp [ANY—ALL] sous-requête' : La sous-requête doit renvoyer **1 ensemble** de valeurs (plusieurs lignes, 1 colonne).

7.3.7 Agrégats

Les fonctions d'agrégat calculent une valeur unique à partir d'un ensemble de lignes.

- 'SUM(colonne)' : somme
- 'AVG(colonne)' : moyenne
- 'MIN(colonne)' : minimum
- 'MAX(colonne)' : maximum
- 'COUNT(colonne)' : compte le nombre de valeurs non nulles.
- 'COUNT(DISTINCT colonne)' : compte le nombre de valeurs distinctes non nulles.
- 'COUNT(*)' : compte le nombre total de lignes.

Ces fonctions opèrent sur un multi-ensemble de valeurs.

Exemple : Le nombre de films (distincts) dirigés par Bergman.

```
SELECT COUNT(DISTINCT Titre)
FROM FILM
WHERE MeS = 'Bergman';
```

7.3.8 Groupement (GROUP BY)

La clause 'GROUP BY' regroupe les lignes ayant les mêmes valeurs dans une ou plusieurs colonnes, afin d'appliquer des fonctions d'agrégat à chaque groupe.

Exemple : Nombre d'acteurs (distincts) par film (...ou presque, compte les lignes par titre).

```
SELECT Titre, COUNT(Acteur) -- Compte les acteurs pour chaque Titre
FROM FILM
GROUP BY Titre;
```

Processus du GROUP BY : 1. **Extraction (FROM/WHERE)** : Les tables sont jointes et filtrées. 2. **Projection** : Les colonnes nécessaires pour le 'GROUP BY', les agrégats et le 'SELECT' final sont conservées. 3. **Regroupement (GROUP BY)** : Les lignes sont partitionnées en groupes basés sur les valeurs des colonnes du 'GROUP BY'. 4. **Agrégat** : Les fonctions d'agrégat sont calculées pour chaque groupe. 5. **Projection finale (SELECT)** : Le résultat final est construit.

Illustration : Soit la table initiale (simplifiée) :

T	M	A
t1	m1	a1
t1	m1	a2
t1	m1	a3
t1	m2	a1
t1	m2	a2
t1	m2	a3
t2	m4	a2

Pour ‘SELECT Titre, COUNT(DISTINCT Acteur) FROM table GROUP BY Titre‘: 1. Extraction/Projection : On garde T et A.

T	A
t1	a1
t1	a2
t1	a3
t1	a1
t1	a2
t1	a3
t2	a2

2. Regroupement par T: Groupe t1: (t1,a1), (t1,a2), (t1,a3), (t1,a1), (t1,a2), (t1,a3) Groupe t2: (t2,a2) 3. Agrégat ‘COUNT(DISTINCT A)‘ par groupe: Groupe t1: Acteurs distincts a1, a2, a3. Count = 3. Groupe t2: Acteurs distincts a2. Count = 1. 4. Résultat final:

T	COUNT
t1	3
t2	1

Note : L’exemple de l’image p234 donne (t1, 6), (t2, 1), ce qui correspondrait à ‘COUNT(*)‘ ou ‘COUNT(A)‘ (sans ‘DISTINCT‘) si la table initiale était bien celle donnée.

Clause SELECT en présence d’agrégat : Contrainte : Si la requête contient une clause ‘GROUP BY‘, alors toute colonne dans la clause ‘SELECT‘ doit être :

- soit une colonne présente dans la clause ‘GROUP BY‘.
- soit le résultat d’une fonction d’agrégat.

Exemple : ‘SELECT listel, agg(liste2) FROM ... GROUP BY listel-bis‘ Contrainte : ‘listel‘ doit être incluse (ou fonctionnellement dépendante) dans ‘listel-bis‘.

Exemple : Nombre de films distincts par Nom de Cinéphile (jointure et groupement).

```
-- Schema: Cinephile(Nom, Nom-Cine), PROG(Nom-Cine, Titre, ...)
SELECT C.Nom, COUNT(DISTINCT P.Titre)
FROM Cinephile C, PROG P
WHERE C.Nom_Cine = P.Nom_Cine -- Jointure
GROUP BY C.Nom; -- Groupement par Nom du cinephile
```

Ici, ‘C.Nom‘ est dans le ‘SELECT‘ et dans le ‘GROUP BY‘. ‘COUNT(DISTINCT P.Titre)‘ est un agrégat.

7.3.9 Clause HAVING

La clause ‘HAVING‘ filtre les **groupes** formés par ‘GROUP BY‘, de manière similaire à ‘WHERE‘ qui filtre les **lignes**. La condition ‘HAVING‘ porte généralement sur des fonctions d’agrégat.

Exemple 1 : Lister les films et leur nombre d’acteurs, mais seulement pour les films ayant 3 acteurs ou plus.

```
SELECT Titre, COUNT(DISTINCT Acteur) AS NbActeurs
FROM FILM
GROUP BY Titre
HAVING COUNT(DISTINCT Acteur) >= 3; -- Filtre sur le resultat de l'agregat
-- Elimination des groupes ne satisfaisant pas la condition
```

Exemple 2 : Lister les titres des films dirigés par plus d'un metteur en scène.

```
SELECT Titre
FROM FILM
GROUP BY Titre
HAVING COUNT(MeS) > 1; -- Ou COUNT(DISTINCT MeS) > 1 selon interpretation
```

7.3.10 Jointure Externe (OUTER JOIN)

La jointure externe permet de conserver les lignes d'une ou des deux tables qui n'ont pas de correspondance dans l'autre table, en complétant les colonnes manquantes avec 'NULL'.

Soit R et S:

R

A	B
1	2
3	4

S

B	C
2	5
2	6
7	8

La jointure externe complète $R \bowtie S$ (Full Outer Join) est obtenue en calculant la jointure interne ($R \bowtie S$) puis en y ajoutant :

- # les n-uplets de R non joignables avec un n-uplet de S, complétés avec des valeurs nulles pour les colonnes de S. (Ici, (3, 4) de R n'a pas de correspondance).
- $ les n-uplets de S non joignables avec un n-uplet de R, complétés avec des valeurs nulles pour les colonnes de R. (Ici, (7, 8) de S n'a pas de correspondance).

Résultat R OUTER JOIN S:

A	B	C
1	2	5
1	2	6
3	4	null
null	7	8

SQL2 propose une variété de formes de jointure :

- 'R NATURAL JOIN S' : Jointure naturelle (sur colonnes de même nom), correspond à la jointure algébrique.
- 'R CROSS JOIN S' : Produit cartésien, équivaut à 'SELECT * FROM R, S'.
- 'R JOIN S ON R.B = S.B' : Thêta-jointure (jointure interne), équivaut à 'SELECT * FROM R, S WHERE R.B = S.B'.
- 'R FULL OUTER JOIN S ON condition' : Jointure externe complète (conserve # et $).
- 'R LEFT OUTER JOIN S ON condition' : Jointure externe gauche (conserve seulement #, les lignes de R non correspondantes).
- 'R RIGHT OUTER JOIN S ON condition' : Jointure externe droite (conserve seulement $, les lignes de S non correspondantes).

7.4 Langage de Manipulation de Données (LMD) - Mises à Jour

7.4.1 Insertion (INSERT)

Insertion d'un n-uplet

```
INSERT INTO NomTable (Colonne1, Colonne2, ...)
VALUES (Valeur1, Valeur2, ...);
```

Si la liste des colonnes est omise, les valeurs doivent être fournies pour toutes les colonnes dans l'ordre de la table.

```
-- Insere un film avec seulement le titre, les autres colonnes
-- (MeS, Acteur si elles existent) seront NULL si autorise.
INSERT INTO FILM (Titre) VALUES ('Tche');
```

La commande effectue l'insertion d'un n-uplet avec valeurs nulles sur les attributs non spécifiés (si c'est autorisé par les contraintes 'NOT NULL').

Insertion d'un ensemble de n-uplets

Permet d'insérer le résultat d'une requête 'SELECT'.

```
INSERT INTO NomTable (Colonne1, ...)
<sous-requête SELECT ...>;
```

7.4.2 Suppression (DELETE)

Supprime des lignes d'une table basées sur une condition.

```
DELETE FROM NomTable
WHERE <condition>;
```

La commande supprime de l'instance de 'NomTable' les n-uplets qui sont résultats de la requête 'SELECT * FROM NomTable WHERE ¡condition¿'. Si 'WHERE' est omis, toutes les lignes sont supprimées.

7.4.3 Modification (UPDATE)

Modifie les valeurs des colonnes pour les lignes satisfaisant une condition.

```
UPDATE NomTable
SET Colonne1 = Valeur1, Colonne2 = Valeur2, ...
WHERE <condition>;
```

La commande extrait tous les n-uplets 'u' résultats de 'SELECT * FROM NomTable WHERE ¡condition¿' et modifie pour ceux-ci la valeur de 'u[Colonne1]' en 'Valeur1', etc.

Exemple :

```
-- Met a jour le telephone pour le cinema a l'adresse specifiee
UPDATE CINE
SET Telephone = '05␣56␣44␣11␣87'
WHERE Adresse = '9,␣rue␣Montesquieu';
```

7.5 Autres Aspects

7.5.1 Définition de Vues (CREATE VIEW)

Une vue est une table virtuelle définie par une requête SQL. Elle n'est généralement pas matérialisée (stockée physiquement), mais sa définition est enregistrée.

```
CREATE VIEW NomVue AS
<requête SELECT ...>;
```

Exemple : Créer une vue des cinémas parisiens.

```
CREATE VIEW Cine_paris AS
SELECT *
FROM CINE
WHERE Adresse LIKE '%Paris%';
```

Une vue peut ensuite être interrogée comme une table normale.

Exemple : Requête posée à partir d'une vue : Les cinémas parisiens qui programment les films de Poirier.

```
SELECT Cp.Nom_Cine
FROM Cine_paris Cp, FILM F
WHERE F.MeS = 'Poirier'
  AND F.Titre = Cp.Titre; -- Suppose que Cine_paris a Titre, jointure
     ↪ necessaire
-- Note: L'exemple original jointait Cine_paris et FILM sur Titre,
-- ce qui suppose que la vue Cine_paris contient Titre, ce qui n'est
-- pas le cas dans sa definition precedente.
-- Une requete plus plausible joindrait Cine_paris et PROG:
SELECT DISTINCT Cp.Nom_Cine
FROM Cine_paris Cp, PROG P, FILM F
WHERE Cp.Nom_Cine = P.Nom_Cine -- Jointure Vue et Prog
  AND P.NumF = F.NumF          -- Jointure Prog et Film
  AND F.MeS = 'Poirier';       -- Selection Film
```

7.5.2 Déclencheurs – Triggers (CREATE TRIGGER)

Un trigger est une procédure stockée qui est exécutée automatiquement en réponse à certains événements (INSERT, DELETE, UPDATE) sur une table spécifique.

Règle ECA

Un trigger suit généralement la règle ECA :

- **Evénement** : Classe de mises à jour (ex: 'AFTER INSERT ON Prog').
- **Condition** : Test similaire à celui du 'WHERE' (clause 'WHEN'), évalué pour chaque ligne affectée (si 'FOR EACH ROW').
- **Action** : Instructions SQL à exécuter si la condition est vraie (bloc 'BEGIN ... END;').

Syntaxe

La syntaxe varie légèrement selon les SGBD. Une forme générale :

```
CREATE [OR REPLACE] TRIGGER nom_trigger
{BEFORE | AFTER | INSTEAD OF} -- Moment d'execution
{INSERT | DELETE | UPDATE [OF Colonne1, ...]} -- Evenement declencheur
ON nom_table
[FOR EACH ROW] -- Trigger de ligne (acces a :old/:new) ou d'instruction
[WHEN (condition)] -- Condition supplementaire
BEGIN
    -- Action(s) SQL;
END;
/ -- Ou autre terminateur selon le SGBD
```

Options :

- 'OR REPLACE' : Remplace un trigger existant de même nom.
- 'BEFORE'/'AFTER'/'INSTEAD OF' : Quand exécuter l'action par rapport à l'événement.
- 'FOR EACH ROW' : Le trigger s'exécute une fois par ligne affectée. Sans ceci, il s'exécute une fois par instruction.
- Variables spéciales (dans 'FOR EACH ROW') : ':old' (valeurs avant l'opération pour DELETE/UPDATE), ':new' (valeurs après l'opération pour INSERT/UPDATE). La syntaxe exacte (':' ou autre) dépend du SGBD.

Exemple 1 : Insertion automatique dans Film

Si un film est inséré dans 'Prog' mais n'existe pas dans 'Film', l'insérer automatiquement dans 'Film'.

```
CREATE TRIGGER Prog_trigger
AFTER INSERT ON Prog -- Evenement: apres insertion dans Prog
FOR EACH ROW -- Pour chaque ligne inseree
WHEN (:new.NumF NOT IN (SELECT NumF FROM Film)) -- Condition: NumF n'
   ↪ existe pas
BEGIN
    INSERT INTO Film (NumF, Titre) -- Action: Inserer dans Film
    VALUES (:new.NumF, 'Titre␣Inconnu'); -- Utilise la nouvelle valeur
       ↪ NumF
    -- On pourrait aussi inserer le titre si disponible: VALUES(:new.NumF,
       ↪  :new.Titre)
    -- Le Titre n'etant pas dans Prog, il faudrait une logique plus
       ↪ complexe
    -- ou accepter un titre par defaut. L'exemple original insere juste
       ↪ Titre,
    -- ce qui est probablement une simplification.
    -- Version de l'image originale (simplifiee/potentiellement incorrecte
       ↪ ):
    -- WHEN(new.Titre NOT IN (SELECT Titre FROM Film))
    -- BEGIN INSERT INTO Film(Titre) VALUES(:new.Titre); END;
END;
/
```

Exemple 2 : Maintenir une liste

Maintenir une table 'Liste$_{C}ine_{P}oirier$'$contenant les cinémas qui ont programmé (après une mise à jour de 'Prog') un film de Poiri$

```
CREATE TRIGGER Poiriertrigg
AFTER UPDATE OF Titre ON Prog -- Evenement: apres MAJ de Titre dans Prog
FOR EACH ROW
WHEN (:new.Titre IN (SELECT Titre FROM Film WHERE MeS = 'Poirier')) --
   ↪ Condition
BEGIN
    -- Action: Inserer le nom du cinema dans la table liste
    -- Suppose l'existence de Liste_Cine_Poirier(Nom_Cine)
    INSERT INTO Liste_Cine_Poirier (Nom_Cine)
    VALUES (:new.Nom_Cine); -- Utilise le nom du cinema de la ligne
       ↪ modifiee
END;
/
```