CHAPTER 1

1.1 Introduction et Motivation

1.1.1 Omniprésence des bases de données

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

1.1.2 Motivation: Le Déluge de Données

- **Big Data**, **Data Deluge**, **Data Scientist**.
- L'humanité produit **2,5 quintillons** (10^{30} bytes/j).
- **Data Never Sleeps 1.0 VS. 10.0**.

1.1.3 Rôle central des SGBD

- Les **Systèmes de Gestion de Bases de Données (SGBD)** sont au coeur de cette révolution.
- Cette révolution a commencé dans les années **1960** !
- **IL Y A plus de 50 ans !**
- **Des données, des données et des données !**

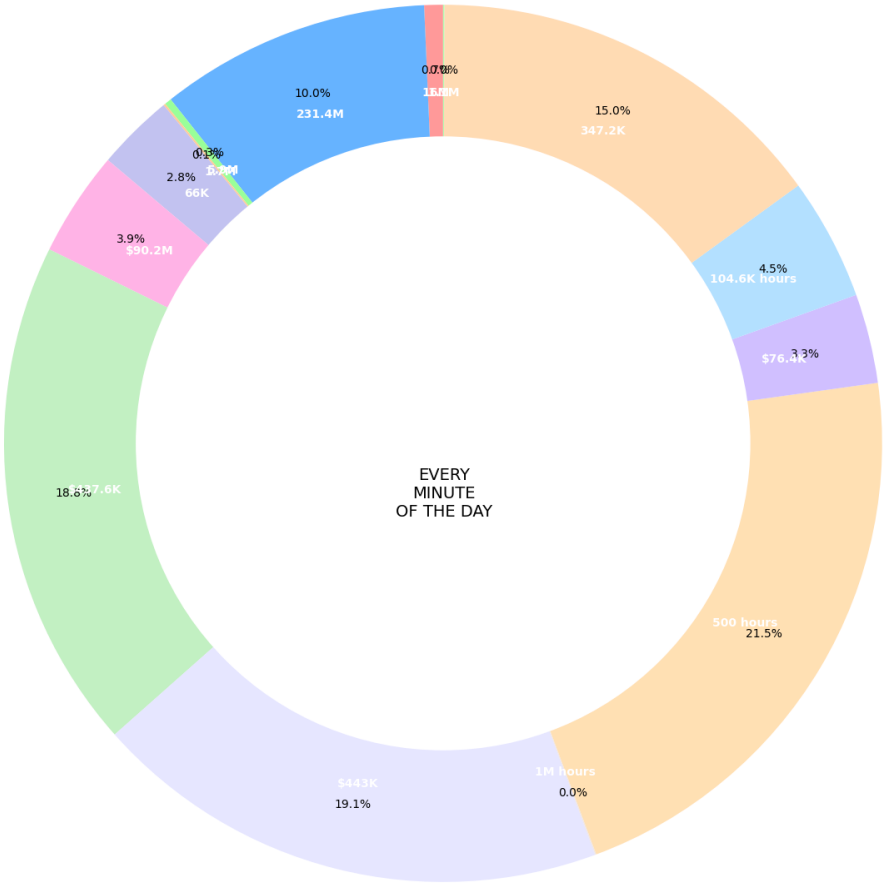

Figure 1.1: Data Never Sleeps

1.1.4 Que devez-vous connaître après ce cours ?

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

1.2 Systèmes de Gestion de Bases de Données (SGBD) : Fonctionnalités

1.2.1 Qu'est-ce qu'un Système de Gestion de Base de Données ?

Definition 1.1. Database Management System (SGBD) Un SGBD permet de:

- **Stocker les données**: *Information storage*
- **Répondre à des questions = requêtes**: *Query answering*
- **Mise à jour les données**: *Update*

1.2.2 Fonctionnalités Clés d'un SGBD

- 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*

1.2.3 Gestion de grandes masses de données

- **grandes masses** de données *massive data*
- Gigabytes (10^9 octets) c'est rien !
- Terabytes (10^{12} octets) / jour, Petabytes (10^{15} octets).
- $\rightsquigarrow$ données stockées en mémoire secondaire (disques).
- $\rightsquigarrow$ gestion des disques et buffer.
- Quintillions (10^{30} octets) *very massive data*
- $\rightsquigarrow$ besoin de nouveaux supports de stockage.
- $\rightsquigarrow$ de nouveaux mécanismes de stockage et d'accès aux données.

1.2.4 Efficacité

- **efficacité** *performance*
- Le système doit être capable de gérer plusieurs milliers de requêtes par seconde ! ces requêtes pouvant être complexes.
- $\rightsquigarrow$ optimisation - optimisation - optimisation

1.2.5 Persistance

- **persistance** *persistency*
- Consultation des palmarès à chaque demande: la réponse peut changer ... en fonction des mises à jour !
- $\rightsquigarrow$ Toute modification des données faite par un programme est enregistrée sur disque et persiste au delà de la seule exécution du programme.

1.2.6 Fiabilité

- **fiabilité** (pannes) *reliability*
- Résistance aux pannes: pannes matérielles, logicielles, électriques, ..., incendie, ...
- $\rightsquigarrow$ reprise sur panne: *failure recovery*
- $\rightsquigarrow$ journal: *logs, write ahead, checkpoints*

1.2.7 Sécurité

- **sécurité** *safety*
- Les données de la base peuvent être critiques pour l'entreprise, pour les clients, pour les états.
- $\rightsquigarrow$ protéger les données d'accès malveillants.
- $\rightsquigarrow$ contrôle d'accès, vues: *access grant*
- La protection de la vie privée (RGPD) est un autre sujet *privacy*.

1.2.8 Multi-utilisateur

- **multi-utilisateur** *multiuser*
- 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.
- $\rightsquigarrow$ contrôle de la concurrence: *concurrency control*

Example 1.2. M. et Mme Dupont ont 100 euros sur leur compte commun. Il retire 50 euros. Elle retire 20 euros. **En même temps**.

- Exécution 1:
  1. Lire_bd Cpte_Dupont $\rightarrow$
  2. := - 50
  3. Ecrire_bd AM $\rightarrow$ Cpte_Dupont
- Exécution 2:
  - i Lire_bd Cpte_Dupont $\rightarrow$ AMme
  - ii AMme := AMme - 20
  - iii Ecrire_bd AMme $\rightarrow$ Cpte_Dupont
- Exécution concurrente = 1, i, 2, ii, 3, iii

1.2.9 Fiabilité - qualité

- **fiabilité - qualité** *data quality*
- données $\neq$ informations
- données + propriétés = représentation d'informations
- $\rightsquigarrow$ contraintes d'intégrité
- $\rightsquigarrow$ vérification automatique de l'intégrité

Example 1.3.
- 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.

1.2.10 Simplicité d'utilisation

- **simplicité d'utilisation** *convenience*
- Développement d'applications / Ecriture de requêtes.
- Maintenance et optimisation.
- ⇝ Langage de haut niveau & déclaratif: **SQL**

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

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

1.3 Principes des SGBD

1.3.1 Schéma versus instance

- **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

- **Les Trois Niveaux d'Abstraction**
  - **Niveau externe (vues)**: description et manipulation des données dédiées à un groupe d'utilisateurs et applications
  - **Niveau logique**: description et manipulation des données "haut niveau"
  - **Niveau physique (interne)**: organisation et stockage des données en mémoire secondaire
- **Utilisateurs versus Niveaux**
  - Utilisateurs λ, Développeurs d'Applications $\Longrightarrow$ Vues, Niveau Logique
  - Développeurs BD, Administrateur BD $\Longrightarrow$ Niveaux Logique & Physique

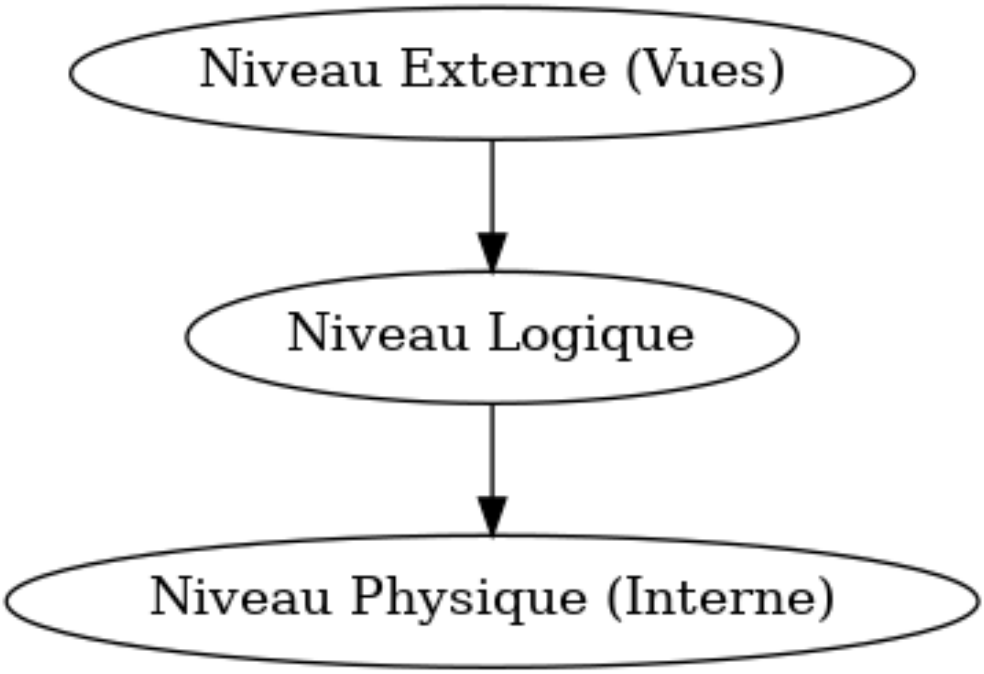

Figure 1.2: Architecture 3 niveaux

1.3.3 Indépendance physique

- **Indépendance physique**
Programmes d'application	**Invariants**
Schéma Physique	Modification

Example 1.4. Exemple:

- Intitialement = BD Palmarès festivals français
- $\rightsquigarrow$ schéma physique = fichier non trié, pas d'index *heapfile*

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

- Qcq années + tard = BD Palmarès festivals internationaux
- $\rightsquigarrow$ schéma physique = fichier indexé (Arbre B+) *index file*

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

1.4 Histoire des SGBD

1.4.1 Un peu d'histoire des SGBDs

- **60-70 SGBD Réseau**
  - IDS (General Electric), APL, DMS 1100, ..., ADABAS (Software AG)
- **60-70 SGBD Hiérarchique**
  - ISM, IBM (www.software.ibm.com)
  - gestion de pointeurs entre enregistrements
  - problème : pas d'**indépendance physique**
- **70- SGBD RELATIONNEL** *Relational DBMS*
  - 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
  - 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
- **NoSQL**
  - Adéquation aux besoins actuels
  - "SQL" = SGBDs relationnels classiques
  - NoSQL= Ne pas utiliser les SGBDs relationnels classiques
  - NoSQL $\neq$ Ne pas utiliser le langage SQL.
  - NoSQL = Ne pas utiliser **seulement** les SGBDs relationnels

1.4.2 Quelques 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, Microsoft www.microsoft.com
  - DB2, MaxDB, 4D, dBase, Informix, Sybase ...
- **Systèmes libres**
  - PostgreSQL www.enterprisedb.com
  - MariaDB (MySQL) www.mariadb.org
  - Firebird, Ingres, HSQLDB, Derby

1.5 Les différents acteurs

1.5.1 Acteurs autour des SGBD

- **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 (tunning) des applications

1.6 Objectifs du Cours

1.6.1 Objectif principal

- Un SGBD relationnel est un **système complexe** !
- **facile à utiliser - difficile à bien utiliser !**

1.7 Plan du cours

1.7.1 Bases de Données - Plan du cours

- **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

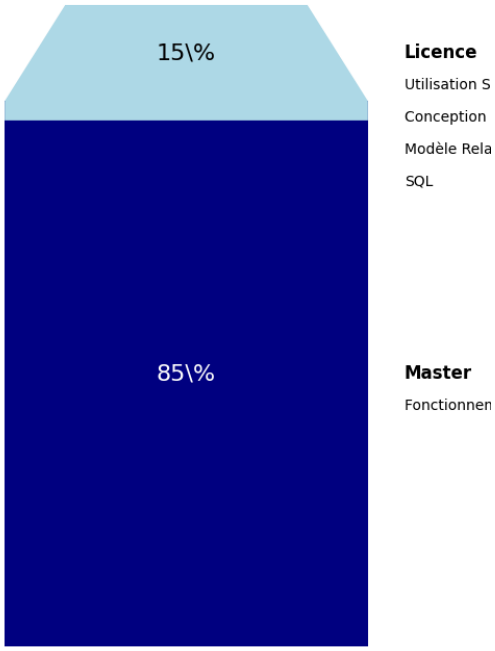

Figure 1.3: Iceberg des SGBD

  - SQL : Langage de manipulation (LMD)

- **Dépendances & Conception de schéma**
  - Dépendances fonctionnelles
  - Schéma sous forme normale

1.8 Bibliographie

1.8.1 Une Petite Bibliographie

- **Bases de données**, G. Gardarin, Editions Eyrolles.
- **Système de gestion des bases de données**, H. Korth et A. Silberschatz, McGraw-Hill.
- **A first course in Database System**, J. Ullman et J. Widom.
- **Fondements des bases de données**, Serge Abiteboul
- **Database System Concepts**, Korth, Silberschatz
- **Database Systems - Concepts, Languages and Architectures**, Atzeni *et al.* http://dbbook.dia.uniroma3.it/dbbook.pdf

CHAPTER 2

Conception de Bases de Données : Modèle Entité-Association

Introduction

Dans le monde actuel, les bases de données sont omniprésentes et essentielles au fonctionnement de nombreuses applications et systèmes d'information. Elles permettent de stocker, d'organiser et de gérer de grandes quantités de données de manière efficace et structurée. Une conception rigoureuse de ces bases de données est cruciale pour garantir la performance, la fiabilité et la maintenabilité des systèmes qui les utilisent.

Le modèle Entité-Association (E/A) est un outil puissant et largement utilisé pour la conception conceptuelle des bases de données. Il offre une approche intuitive et graphique pour représenter les données et leurs relations, facilitant ainsi la communication entre les concepteurs, les développeurs et les utilisateurs finaux.

Ce manuel a pour objectif de vous guider à travers les concepts fondamentaux du modèle Entité-Association et de vous fournir les connaissances nécessaires pour concevoir vos propres modèles E/A. À travers ce cours, vous apprendrez les étapes clés de la conception d'une base de données, les composants du modèle E/A, et les principes essentiels pour une conception efficace. Avec les exemples et illustrations fournis, vous serez en mesure de comprendre et d'appliquer ce modèle dans vos projets de conception de bases de données.

2.1 Conception d'une Base de Données

2.1.1 Objectifs de la Conception d'une Base de Données

La conception d'une base de données répond à un besoin fondamental : celui de développer des applications qui manipulent et exploitent de manière intensive de grandes masses de données. Ces données doivent être persistantes, c'est-à-dire qu'elles doivent survivre à l'exécution des programmes et être disponibles à tout moment. Les bases de données sont donc au cœur des systèmes d'information modernes.

On retrouve des bases de données dans de nombreux domaines et applications, par exemple :

- La gestion des stocks et des employés dans une entreprise.
- Les agences de voyages pour la gestion des réservations et des clients.
- Les services hospitaliers pour le suivi des patients et des dossiers médicaux.
- Les systèmes de marketing pour l'analyse des comportements des consommateurs.

- Le traitement des infractions et la gestion des dossiers judiciaires.
- Les cinémas pour la gestion des films et des séances.
- Les agences spatiales pour le stockage et l'analyse des données spatiales.
- La scolarité à l'université pour la gestion des étudiants et des cours.
- Les laboratoires de recherche pour l'organisation et l'analyse des données expérimentales.
- ... et bien d'autres encore.

2.1.2 Place de la Conception de BD dans le Cycle de Vie d'un Logiciel

La conception d'une base de données est une étape cruciale dans le processus de développement d'un logiciel. Elle fait partie du génie logiciel, mais se distingue du développement de programmes classique. Alors que le génie logiciel classique se concentre sur le développement des programmes et des algorithmes, la conception de base de données se focalise sur la structure et l'organisation des données.

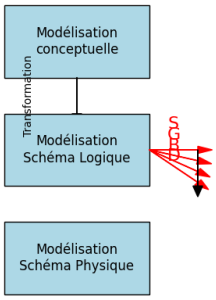

Figure 2.1: Processus de Conception d'une Base de Données

La conception d'une base de données est donc la partie du processus de génie logiciel qui produit le schéma de la base de données.

2.1.3 Les Trois Étapes de la Conception

La conception d'une base de données se déroule généralement en trois étapes principales, comme illustré à la Figure **??** :

1. **Modélisation Conceptuelle** : Cette première étape consiste à comprendre et à analyser le domaine d'application et les besoins des utilisateurs. L'objectif est d'identifier les concepts fondamentaux du domaine, leurs propriétés et les relations qui les unissent. Le résultat de cette étape est un modèle conceptuel, une représentation abstraite des données, indépendante du modèle de données et du Système de Gestion de Base de Données (SGBD) qui sera utilisé. Le modèle Entité-Association est typiquement utilisé à cette étape.
2. **Modélisation Logique** : Cette étape consiste à traduire le modèle conceptuel en un schéma logique, en utilisant un modèle de données spécifique, comme le modèle relationnel, le modèle orienté-objet ou le modèle semi-structuré. Le choix du modèle de données est souvent influencé par le SGBD cible. Le schéma logique reste indépendant du SGBD spécifique.

3. **Modélisation Physique** : La dernière étape consiste à définir la structure de stockage physique des données, en tenant compte des spécificités du SGBD choisi et des performances attendues. Cela inclut la définition des index, des paramètres de configuration du SGBD, et d'autres optimisations physiques.

2.1.4 Difficultés de la Conception

La conception d'une base de données est une tâche complexe et souvent difficile. Plusieurs facteurs contribuent à cette complexité :

- **Complexité du domaine d'application** : Le monde réel est complexe et riche en exceptions. Il est souvent difficile de modéliser fidèlement cette complexité dans un modèle de base de données.
- **Interaction avec les experts du domaine et les futurs utilisateurs** : La conception d'une base de données nécessite une collaboration étroite avec les experts du domaine d'application et les futurs utilisateurs du système. Il est essentiel de bien comprendre leurs besoins et leurs attentes. Cela implique d'analyser et de comprendre le domaine, les besoins métier et d'identifier les données nécessaires aux traitements, tant pour les besoins actuels que futurs.
- **Abstraction et simplification** : Coder le monde réel, avec sa complexité et ses exceptions, à l'aide d'un modèle informatique sans flexibilité oblige à abstraire et à simplifier. Il faut donc trouver un juste milieu entre la fidélité au réel et la simplicité du modèle.
- **Gestion de la complexité et du volume d'informations** : Il faut maîtriser la complexité et le nombre d'informations à représenter, qui peuvent être très importants.
- **Performances** : Il est crucial de ne jamais perdre de vue les performances du système. La base de données doit être conçue de manière à garantir des temps de réponse acceptables, même avec de grandes quantités de données et un nombre important d'utilisateurs.

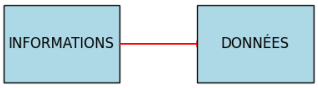

Figure 2.2: Passage des Informations aux Données

La Figure **??** illustre le passage des informations du monde réel aux données structurées dans une base de données. C'est ce passage complexe que la conception de base de données doit faciliter et optimiser.

2.2 Le Modèle Entité-Association (E/A)

2.2.1 Présentation du Modèle E/A

Le modèle Entité-Association (E/A) a été introduit par Peter Chen en 1976. Il propose un ensemble de concepts et de symboles graphiques pour modéliser les données d'une application de manière conceptuelle. L'objectif est de fournir une représentation claire et intuitive des données, indépendante de toute considération technique liée à l'implémentation.

Le modèle E/A repose sur trois concepts centraux :

- **Entité** : Représente un objet ou un concept du monde réel qui a une existence propre et qui est pertinent pour l'application.
- **Attribut** : Décrit une propriété ou une caractéristique d'une entité.
- **Association** : Représente une relation ou un lien entre deux ou plusieurs entités.

En plus de ces concepts fondamentaux, le modèle E/A inclut également des notions plus avancées comme :

- Les contraintes de cardinalité, qui précisent la nature et le nombre de liens entre les entités.
- Les clés et les entités faibles, qui permettent de gérer l'identification des entités.
- Quelques règles de conception pour garantir la qualité et la cohérence du modèle.

2.2.2 Concepts Fondamentaux

Entité (ou Type Entité)

Definition 2.1 (Entité)**.** Une **entité** (ou type entité) représente un ensemble d'objets qui sont centraux dans le domaine de l'application et pour lesquels des informations doivent être stockées. Une entité correspond à une catégorie d'objets ou de concepts du monde réel que l'on souhaite représenter dans la base de données.

Les entités peuvent représenter des objets concrets (comme les acteurs, les films, les clients, les produits, etc.) ou des concepts abstraits (comme les événements, les transactions, les catégories, etc.).

Example 2.2 (Exemples d'entités)**.** Dans le domaine du cinéma, on peut identifier les entités suivantes :

- **Acteur** : représente l'ensemble des acteurs de cinéma.
- **Film** : représente l'ensemble des films.
- **Tournage** : peut représenter l'ensemble des tournages de films (si l'on souhaite modéliser des informations spécifiques sur les tournages).

Graphiquement, une entité est représentée par un rectangle, contenant le nom de l'entité.

Figure 2.3: Entités Acteur et Film

Figure 2.4: Entité Tournage

Attribut

Definition 2.3 (Attribut)**.** Un **attribut** est une propriété ou une caractéristique descriptive d'une entité. Il sert à préciser et à qualifier une entité, en décrivant les informations que l'on souhaite stocker pour chaque instance de cette entité.

Chaque entité est décrite par un ensemble d'attributs. Un attribut est spécifié par un nom et un ensemble de valeurs possibles, appelé domaine de valeurs.

Example 2.4 (Exemples d'attributs pour l'entité Acteur)**.** Pour l'entité **Acteur**, on peut définir les attributs suivants :

- **Prénom** : le prénom de l'acteur (domaine de valeurs : chaîne de caractères).
- **Nom** : le nom de famille de l'acteur (domaine de valeurs : chaîne de caractères).
- **Date de naissance (Ddn)** : la date de naissance de l'acteur (domaine de valeurs : date).

Les attributs peuvent être de différents types :

- **Attribut simple** : un attribut dont la valeur est atomique et indivisible (ex : Nom, Prénom).
- **Attribut complexe** : un attribut dont la valeur est composée de plusieurs parties (ex : Date de naissance, qui peut être composée du jour, du mois et de l'année).

Dans un diagramme E/A, les attributs sont généralement listés à l'intérieur du rectangle représentant l'entité.

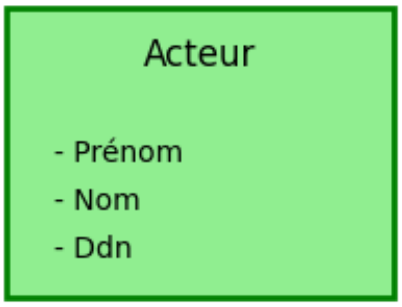

Figure 2.5: Attributs de l'entité Acteur

Association (ou Type d'Association)

Definition 2.5 (Association)**.** Une **association** (ou type d'association) représente un lien, une relation ou une interaction entre deux ou plusieurs entités. Elle permet de modéliser des liens concrets entre les instances de ces entités.

Les associations peuvent être binaires (entre deux entités), ternaires (entre trois entités), ou n-aires (entre n entités). Les associations binaires sont les plus fréquentes.

Example 2.6 (Exemple d'association binaire)**.** Dans le domaine du cinéma, on peut définir l'association **Joue** entre les entités **Acteur** et **Film**. Cette association représente le fait qu'un acteur joue dans un film.

Graphiquement, une association est représentée par un losange, relié par des traits aux entités qu'elle associe. Le nom de l'association est écrit à l'intérieur du losange.

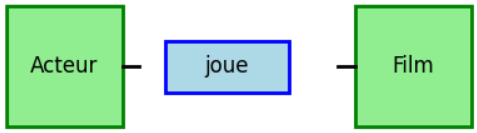

Figure 2.6: Association Joue entre Acteur et Film

Une association est spécifiée par un nom, les entités qu'elle associe et, éventuellement, des **rôles** et des attributs.

Rôles

Dans une association, chaque entité participante joue un rôle spécifique. Le rôle précise la fonction de l'entité dans le contexte de l'association. Les rôles sont particulièrement utiles pour clarifier le sens des associations, surtout lorsque plusieurs associations existent entre les mêmes entités ou lorsque l'association est réflexive (une entité est associée à elle-même).

Dans l'exemple de l'association **Joue**, on pourrait préciser les rôles : l'acteur joue le rôle de Acteur et le film a comme rôle Film dans lequel il joue. Cependant, dans cet exemple simple, les rôles sont implicites et ne sont pas obligatoires.

Dans certains cas, une association peut avoir des attributs propres. Par exemple, si l'on considère l'association **Joue** et que l'on souhaite stocker le rôle spécifique joué par un acteur dans un film, on peut ajouter un attribut **Rôle** à l'association elle-même.

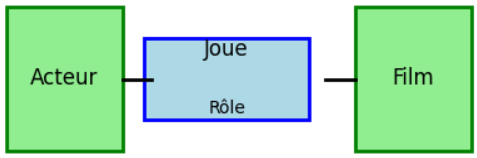

Figure 2.7: Association Joue avec l'attribut Rôle

2.2.3 Contraintes de Cardinalité

Les contraintes de cardinalité permettent de préciser la nature et le nombre de liens qui peuvent exister entre les instances des entités participant à une association. Elles apportent une précision importante au modèle E/A et permettent de refléter plus fidèlement les règles de gestion du domaine d'application.

La cardinalité s'exprime en termes de nombre minimal et maximal d'instances d'une entité qui peuvent être liées à une instance d'une autre entité via une association. On note généralement la cardinalité sous la forme **(min, max)**, où :

- **min** : le nombre minimum d'instances de l'entité cible auxquelles une instance de l'entité source doit être liée.

- **max** : le nombre maximum d'instances de l'entité cible auxquelles une instance de l'entité source peut être liée.

Les valeurs possibles pour **min** sont 0 ou 1 (ou parfois des valeurs plus grandes, mais rarement utilisées dans la modélisation conceptuelle). Les valeurs possibles pour **max** sont 1 ou **m** (pour plusieurs, many en anglais).

Example 2.7 (Exemple de contraintes de cardinalité pour l'association Joue)**.** Considérons l'association **Joue** entre **Acteur** et **Film**. On peut définir les contraintes de cardinalité suivantes :

- Un acteur peut jouer dans zéro, un ou plusieurs films. Donc, du côté de l'entité **Acteur**, la cardinalité est **(0, m)**.
- Un film peut avoir zéro, un ou plusieurs acteurs qui jouent dedans (par exemple, un film d'animation peut ne pas avoir d'acteurs réels). Donc, du côté de l'entité **Film**, la cardinalité est également **(0, m)**.

Graphiquement, les contraintes de cardinalité sont indiquées sur les traits reliant l'association aux entités, sous la forme **(min, max)**.

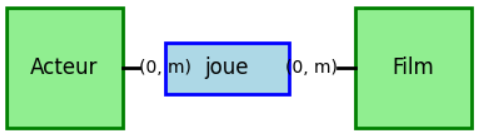

Figure 2.8: Contraintes de Cardinalité pour l'association Joue

2.2.4 Clés (ou Identifiants) d'une Entité

Definition 2.8 (Clé d'une entité)**.** Une **clé** (ou identifiant) d'une entité est un attribut (ou un ensemble d'attributs) qui permet d'identifier de manière unique chaque instance de cette entité. Elle garantit l'unicité des instances au sein d'une entité.

Il est essentiel de définir au moins une clé pour chaque entité afin de pouvoir distinguer et manipuler les instances de cette entité.

Example 2.9 (Clé pour l'entité Acteur)**.** Pour l'entité **Acteur**, on pourrait envisager plusieurs attributs comme clés potentielles :

- **Nom** : Le nom de famille seul n'est pas suffisant, car plusieurs acteurs peuvent avoir le même nom.
- **Prénom** : Le prénom seul n'est pas suffisant non plus pour la même raison.
- **Nom et Prénom** : La combinaison du nom et du prénom pourrait être une cléCandidate possible, mais il pourrait encore y avoir des homonymes.
- **Numéro d'identification unique (Num_A)** : Si l'on attribue un numéro unique à chaque acteur, cet attribut devient une cléCandidate idéale.

Dans un diagramme E/A, la clé primaire d'une entité est souvent soulignée ou indiquée d'une manière spécifique (par exemple, en la plaçant en premier dans la liste des attributs).

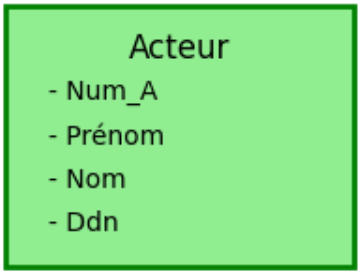

Figure 2.9: Clé de l'entité Acteur (Num_A)

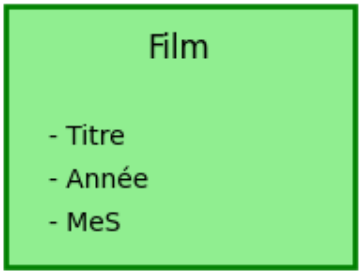

Figure 2.10: Clé de l'entité Film (Titre, Année)

2.2.5 Entités Faibles (Identification Relative)

Definition 2.10 (Entité Faible)**.** Une **entité faible** est une entité dont l'existence dépend de l'existence d'une autre entité, appelée entité forte ou entité propriétaire. Une entité faible ne peut pas être identifiée de manière unique par ses propres attributs, mais elle est identifiée relativement à l'entité forte dont elle dépend.

Les entités faibles sont souvent utilisées pour modéliser des informations qui sont des détails ou des compléments d'information relatifs à une autre entité.

Example 2.11 (Exemple d'entité faible : Salle de cinéma)**.** Considérons le domaine des cinémas. Une **Salle** de cinéma n'a pas d'existence indépendante d'un **Cinéma**. Une salle est toujours dans un cinéma. Pour identifier une salle de manière unique, il faut connaître le cinéma auquel elle appartient et son numéro de salle au sein de ce cinéma. Le numéro de salle est local au cinéma.

Dans ce cas, **Salle** est une entité faible, et **Cinéma** est l'entité forte ou propriétaire. L'identification d'une instance de **Salle** se fait de manière relative à une instance de **Cinéma**. La clé de **Salle** sera donc composée de la clé de **Cinéma** (par exemple, un identifiant unique de cinéma) et d'un attribut propre à **Salle** (par exemple, le numéro de salle).

Dans un diagramme E/A, une entité faible est souvent représentée par un rectangle avec un contour double. L'association entre l'entité faible et l'entité forte est généralement une association identifiante, souvent de cardinalité (1,1) du côté de l'entité faible.

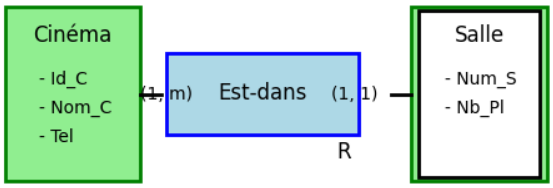

Figure 2.11: Entité Faible Salle et Entité Forte Cinéma

2.2.6 Héritage (Is-a)

Definition 2.12 (Héritage)**.** L'**héritage** (ou spécialisation/généralisation) est un concept qui permet d'organiser les entités en hiérarchies, en introduisant des relations de type est-un (is-a) entre les entités. L'héritage permet de modéliser des situations où certaines entités sont des cas particuliers ou des sous-types d'entités plus générales.

L'héritage est utile pour factoriser les attributs communs à plusieurs entités et pour représenter des hiérarchies de concepts.

Example 2.13 (Exemple d'héritage : Film, Documentaire, Animation)**.** Dans le domaine du cinéma, on peut considérer que les entités **Documentaire** et **Animation** sont des types particuliers de l'entité plus générale **Film**. On peut dire : un documentaire est un film et un film d'animation est un film. Il y a donc une relation d'héritage entre **Film** (entité super-classe) et **Documentaire** et **Animation** (entités sous-classes).

Les entités sous-classes héritent des attributs de l'entité super-classe et peuvent avoir des attributs spécifiques supplémentaires. Par exemple, l'entité **Film** peut avoir des attributs comme **Titre**, **Année**, **Metteur en scène**, tandis que **Documentaire** pourrait avoir un attribut spécifique comme **Sujet**, et **Animation** un attribut comme **Technique d'animation**.

Graphiquement, l'héritage est souvent représenté par un triangle pointant vers l'entité super-classe, avec la mention is-a ou un symbole similaire.

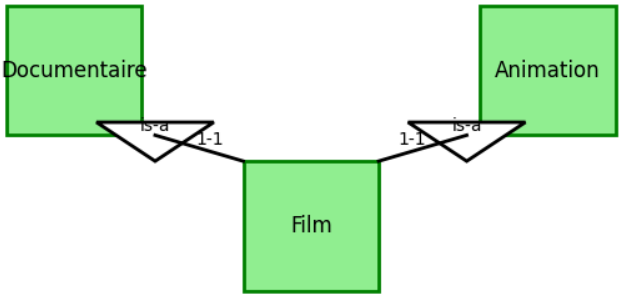

Figure 2.12: Héritage : Film, Documentaire et Animation

2.3 Principes de Conception

Une bonne conception de base de données repose sur le respect de certains principes fondamentaux. Voici quelques principes clés à suivre lors de la conception d'un modèle E/A :

2.3.1 Se Focaliser sur les Besoins de l'Application

Le principe le plus important est de toujours garder à l'esprit les besoins de l'application et des utilisateurs. Le modèle de données doit refléter la réalité du domaine d'application, mais il doit surtout être utile pour l'application pour laquelle il est conçu.

- **Entités et attributs pertinents** : Les entités et les attributs doivent être choisis en fonction de leur utilité pour l'application. Il est inutile d'introduire des entités ou des attributs qui ne seront pas utilisés ou qui n'apportent pas de valeur ajoutée. Par exemple, pour une application de gestion de cinémathèque, un attribut GENRE pour l'entité **Acteur** n'est probablement pas pertinent. De même, pour une application de type Amazon, stocker l'adresse de chaque acteur n'a aucun intérêt. En revanche, pour un studio de cinéma, stocker l'adresse et le portable de chaque acteur peut être important.
- **Associations significatives** : Les associations doivent refléter les relations importantes entre les entités, celles qui sont pertinentes pour l'application. Il faut s'interroger sur la nécessité de chaque association et sur sa cardinalité. Par exemple, est-il nécessaire d'avoir deux associations entre l'entité **Studio** et l'entité **Film**, l'une indiquant qu'un studio produit un film, et l'autre qu'un studio a participé au tournage ? Cela dépend des besoins de l'application.

2.3.2 Faire Simple

La simplicité est un principe essentiel en conception. Un modèle simple est plus facile à comprendre, à mettre en œuvre et à maintenir. Il est important de :

- **Eviter la complexité inutile** : Ne pas chercher à modéliser tous les détails et toutes les subtilités du domaine d'application. Se concentrer sur l'essentiel et simplifier au maximum.
- **Choisir des notations simples et précises** : Utiliser les notations du modèle E/A de manière rigoureuse et précise, mais sans les complexifier inutilement.

2.3.3 Eviter les Redondances

La redondance, c'est-à-dire la répétition de la même information sous différentes formes, est à éviter au maximum dans une base de données. La redondance entraîne des problèmes d'efficacité (gaspillage d'espace de stockage, temps d'accès plus longs) et de qualité des données (risques d'incohérences si les informations redondantes ne sont pas mises à jour de manière synchronisée).

Il faut donc s'efforcer de :

- **Identifier et éliminer les redondances** : Analyser attentivement le modèle E/A pour détecter les redondances potentielles et les supprimer. Cela peut passer par une restructuration du modèle, une décomposition des entités, ou l'introduction de nouvelles entités ou associations. Faire la chasse aux redondances est une activité importante de la conception.
- **Normalisation** : Les techniques de normalisation des bases de données, qui seront étudiées plus tard, permettent de réduire au maximum la redondance et d'améliorer la qualité des données.

2.3.4 Choisir Judicieusement les Entités, Associations et Attributs

Le choix des entités, des associations et des attributs est crucial pour la qualité du modèle E/A. Il faut se poser les bonnes questions et faire les bons compromis.

- **Attribut versus Entité** : Dans certains cas, on peut hésiter à modéliser une information comme un attribut ou comme une entité à part entière. Par exemple, dans l'exemple de l'association **Joue**, on pourrait considérer que le Rôle joué par un acteur dans un film est un simple attribut de l'association, ou bien le modéliser comme une entité **Rôle** liée à l'association **Joue**. Le choix dépendra de la complexité de l'information à représenter et des besoins de l'application. Si un acteur peut jouer plusieurs rôles dans un même film et si l'on souhaite décrire des informations spécifiques sur chaque rôle (comme un texte de rôle, un costume, etc.), il sera plus judicieux de modéliser **Rôle** comme une entité. Sinon, un simple attribut **Rôle** dans l'association **Joue** peut suffire.
- **Associations pertinentes** : S'assurer que chaque association modélise une relation significative et pertinente pour l'application. Eviter les associations inutiles ou ambiguës.
- **Attributs descriptifs** : Choisir des attributs qui décrivent de manière précise et complète les entités et les associations, en fonction des besoins de l'application.

2.4 Conclusion

Ce manuel a introduit les concepts fondamentaux de la conception de bases de données en utilisant le modèle Entité-Association (E/A). Nous avons vu que la conception d'une base de données est un processus complexe et itératif, qui nécessite une compréhension approfondie du domaine d'application, une collaboration étroite avec les experts du domaine, et le respect de principes de conception clés comme la simplicité, la pertinence et l'absence de redondance.

Le modèle E/A est un outil puissant et flexible pour la modélisation conceptuelle des données. Il permet de représenter de manière intuitive les entités, leurs attributs et les relations qui les unissent. Les diagrammes E/A facilitent la communication et la compréhension du modèle par tous les acteurs du projet (concepteurs, développeurs, utilisateurs finaux).

Bien que le modèle E/A soit largement utilisé et efficace, il existe d'autres approches pour la modélisation conceptuelle, comme le diagramme UML (Unified Modeling Language) ou les modèles sémantiques et les ontologies. Ces approches peuvent être plus adaptées à certains types d'applications ou de domaines. La recherche continue dans le domaine de la modélisation des données, comme en témoigne la conférence ER (Entity-Relationship) dédiée à ce sujet.

La conception de bases de données est un métier à part entière, qui requiert des compétences techniques, mais aussi des qualités d'analyse, de communication et de compromis. Les outils graphiques de modélisation, comme WinDesign, Looping, Visual Paradigm ou Lucidchart, peuvent faciliter la tâche du concepteur, mais ils ne remplacent pas la réflexion et l'expertise humaine.

L'étape de modélisation conceptuelle n'est que la première étape de la conception d'une base de données. Elle est suivie des étapes de modélisation logique et physique, qui consistent à traduire le modèle conceptuel en un schéma implémentable dans un SGBD spécifique. Le choix du modèle de données (relationnel, orienté-objet, semi-structuré) et du SGBD (Oracle, MySQL, PostgreSQL, etc.) dépendra des besoins de l'application, des contraintes techniques et des performances attendues.

La conception de bases de données est un domaine en constante évolution, en réponse aux nouveaux besoins des applications et aux progrès des technologies. Les bases de données NoSQL, les bases de données orientées graphes, les bases de données en mémoire, sont autant d'exemples de nouvelles approches et technologies qui viennent enrichir le paysage des bases de données. La maîtrise des principes de conception fondamentaux, comme ceux présentés dans ce manuel, reste cependant essentielle pour aborder sereinement les défis de la conception de bases de données, aujourd'hui et demain.

CHAPTER 3

3.1 Introduction

La conception d'une base de données est un processus crucial qui commence par la compréhension des besoins et la modélisation des données. Le modèle Entité-Association (EA) est un outil conceptuel puissant pour représenter la structure des données de manière abstraite. Cependant, pour implémenter une base de données, il est nécessaire de transformer ce modèle conceptuel en un modèle logique, le modèle relationnel, qui est directement compatible avec les Systèmes de Gestion de Bases de Données (SGBD) relationnels.

Cette transformation est essentielle pour passer d'une vision conceptuelle à une réalisation pratique. Elle implique de convertir les composants du modèle EA en structures relationnelles tout en préservant les informations et les contraintes.

Rappelons les concepts clés :

- **Modèle Entité-Association (EA) :** Modèle conceptuel de données utilisant les concepts suivants :
  - **Entité :** Représente un objet ou un concept du monde réel (ex: Acteur, Film).
  - **Association :** Représente une relation entre deux ou plusieurs entités (ex: Joue, Distribue).
  - **Clé :** Attribut ou ensemble d'attributs identifiant de manière unique une instance d'entité.
  - **Contraintes de cardinalité :** Spécifient le nombre minimum et maximum d'instances d'entités qui peuvent participer à une association.
- **Modèle Relationnel :** Modèle logique de données basé sur les concepts suivants :
  - **Relation :** Table composée de lignes (tuples) et de colonnes (attributs), représentant un ensemble d'entités ou d'associations.
  - **Dépendance fonctionnelle :** Contrainte entre attributs où la valeur d'un attribut détermine la valeur d'un autre attribut. Les clés sont basées sur les dépendances fonctionnelles.
  - **Dépendance d'inclusion (Clé étrangère) :** Contrainte assurant que les valeurs d'un attribut (clé étrangère) dans une relation existent comme valeurs d'une clé primaire dans une autre relation, établissant ainsi des liens entre les relations.

3.2 Règles de base de la transformation

La transformation du modèle EA au modèle relationnel suit des règles précises pour garantir la fidélité et l'intégrité des données. Les deux étapes fondamentales concernent la transformation des entités et des associations.

3.2.1 Transformation des entités

Règle 1 : Pour chaque entité, on crée un schéma de relation ayant le même nom et les mêmes attributs, et les mêmes clés.

Chaque entité du modèle EA est directement convertie en une relation dans le modèle relationnel.

- Le **nom** de la relation est identique au nom de l'entité.
- Les **attributs** de la relation sont les mêmes que ceux de l'entité (avec possibilité d'ajuster les noms pour la cohérence).
- La **clé primaire** de la relation est la clé (identifiant) de l'entité.

Example 3.1. Considérons les entités *Acteur*, *Film*, *Président* et *Studio* issues de notre modèle EA.

- **Entité Acteur** (Attributs: Num-A, Prénom, Nom, Ddn) devient la relation **Acteur** (Attributs: Num_A, Prénom, Nom, Ddn).
- **Entité Film** (Attributs: Num-F, Titre, Année) devient la relation **Film** (Attributs: Num_F, Titre, Année).
- **Entité Président** (Attributs: Num-P, Nom, An) devient la relation **Président** (Attributs: Num_P, Nom, An).
- **Entité Studio** (Attributs: Nom, Adr) devient la relation **Studio** (Attributs: Nom, Adr).

Note : Les clés primaires sont soulignées.

3.2.2 Transformation des associations

Règle 2 : Pour chaque association, on crée un schéma de relation ayant le même nom et ayant les attributs suivants :

- **Les attributs de l'association** (s'il y en a).
- **Les attributs clé des entités mises en jeux par l'association.** Ces attributs clés deviennent des clés étrangères dans la relation de l'association, référençant les relations des entités participantes.

Example 3.2. Considérons les associations *joue*, *distribue* et *dirige* de notre modèle EA.

- **Association joue** (avec attribut *Rôle*) reliant *Acteur* et *Film* devient la relation **Joue** (Attributs: Num_A, Num_F, Rôle).
  - **Clé primaire composée** de (Num_A, Num_F).
  - **Clé étrangère Num_A** référençant la relation *Acteur*(Num_A).
  - **Clé étrangère Num_F** référençant la relation *Film*(Num_F).
- **Association distribue** reliant *Film* et *Studio* devient la relation **Dist** (Attributs: Num_F, Nom).
  - **Clé primaire composée** de (Num_F, Nom).
  - **Clé étrangère Num_F** référençant la relation *Film*(Num_F).
  - **Clé étrangère Nom** référençant la relation *Studio*(Nom).
- **Association dirige** reliant *Président* et *Studio* devient la relation **Dirige** (Attributs: Num_P, Nom).
  - **Clé primaire composée** de (Num_P, Nom).

- **Clé étrangère Num_P** référençant la relation *Président*(Num_P).
- **Clé étrangère Nom** référençant la relation *Studio*(Nom).

3.3 Raffinement du schéma relationnel

Après l'application des règles de base, le schéma relationnel obtenu peut être raffiné pour optimiser la structure et gérer les contraintes supplémentaires, notamment les contraintes de cardinalité.

3.3.1 Analyse des contraintes de cardinalité

Les contraintes de cardinalité des associations jouent un rôle crucial dans le raffinement du schéma relationnel. Elles déterminent comment les relations d'association sont structurées et comment les clés étrangères sont gérées. En particulier les cardinalités maximales (1, N, M) influencent directement les décisions de fusion de schémas.

3.3.2 Fusion des schémas de relation

Dans certains cas, notamment lorsque la cardinalité maximale du côté d'une entité dans une association est 1 (relation 1:1 ou 1:N), il est possible de fusionner le schéma de relation de l'association avec le schéma de relation de l'entité du côté '1'. Cette fusion permet de simplifier le schéma et d'améliorer l'efficacité des requêtes.

3.4 Gestion des contraintes de cardinalité (Max1 : Max2)

Examinons différents cas de contraintes de cardinalité maximales (Max1 : Max2) et leurs implications sur la transformation.

3.4.1 CAS [1 : M]

Lorsque la cardinalité est de type [1:M] entre une entité E1 et une association A avec une entité E2 (E1 –(1,1)– A –(0,m)– E2), les schémas de relation associés à E1 et à l'association A peuvent être fusionnés en un seul schéma de relation, que nous appellerons AE1.

- Les **attributs** de AE1 seront : les attributs de A et les attributs de E1, ainsi que la clé étrangère de E2 (clé de E2).
- La **clé primaire** de AE1 reste la clé primaire de E1.
- La clé étrangère de E2 (K2) est ajoutée à AE1 pour établir le lien avec E2.

Example 3.3. Considérons l'association *distribue* (1,1) entre *Film* et *Studio* (1,n). Nous avons initialement :

- **Film**(Num_F, Titre, Année)
- **Studio**(Nom, Adr)
- **Dist**(Num_F, Nom) avec Dist(Num_F) ⊆ Film(Num_F) et Dist(Nom) ⊆ Studio(Nom)

En fusionnant *Film* et *Dist* en raison de la cardinalité [1:M] (ici 1,1 : 1,n), nous obtenons la relation **FilmDist** :

- **FilmDist**(Num_F, Titre, Année, Nom) *Nom non Null car cardinalité minimale 1 du côté de Film dans distribue.*

- **Studio**(Nom, Adr)
- Contraintes de dépendance d'inclusion : FilmDist(Nom) ⊆ Studio(Nom) et FilmDist(Num_F) ⊆ Film(Num_F).

3.4.2 CAS [1 : 1] – (0,1)—(0,1)

Dans le cas d'une cardinalité [1:1] – (0,1)—(0,1) entre deux entités E1 et E2 via une association A, où chaque entité peut participer au plus une fois à l'association, et au moins une entité doit participer (cardinalité minimale de 1 d'un côté), on peut fusionner les schémas de relation de E1 et de A.

- Les schémas de relation associés à l'entité E1 et à l'association A sont fusionnés en un seul schéma de relation AE1.
- Les attributs de AE1 sont: les attributs de A et les attributs de E1, et les attributs clé de E2.
- La clé de AE1 reste la clé de E1.
- La clé étrangère de E2 est ajoutée à AE1 et peut accepter les valeurs nulles si la cardinalité minimale du côté de E2 est 0.

Example 3.4. Considérons l'association *dirige* (0,1) – (0,1) entre *Président* et *Studio*. Nous avons initialement :

- **Président**(Num_P, Nom, An)
- **Studio**(NomS, Adr)
- **Dirige**(Num_P, NomS) avec Dirige(Num_P) ⊆ Président(Num_P) et Dirige(NomS) ⊆ Studio(NomS)

En fusionnant *Président* et *Dirige*, nous obtenons la relation **Presidirige** :

- **Presidirige**(Num_P, Nom, An, NomS) *Attention au renommage des attributs pour éviter les collisions de noms (Nom devient Nom et Nom du studio devient NomS).*
- **Studio**(NomS, Adr)
- Contraintes de dépendance d'inclusion : Presidirige(NomS) ⊆ Studio(NomS) et Presidirige(Num_P) ⊆ Président(Num_P).

3.4.3 CAS [M : M]

Pour une association de type [M:M] entre deux entités E1 et E2 via une association A, on ne fusionne pas les entités. L'association A devient une relation séparée.

- Les schémas de relation associés aux entités E1 et E2 restent séparés.
- L'association A est transformée en une relation R.
- Les attributs de R sont : les attributs de A (s'il y en a) et les clés primaires de E1 et E2.
- La clé primaire de R est généralement la combinaison des clés primaires de E1 et E2.
- On ajoute des contraintes d'inclusion pour assurer l'intégrité référentielle (clés étrangères).

Example 3.5. Considérons une association *assure* (1,1) – (1,1) entre *Acteur* et *Contrat*.

- **Acteur**(Num_A, Prénom, Nom, Adr)
- **Contrat**(Num_C, Type, Durée)
- **Assure**(Num_A, Num_C) avec Assure(Num_A) ⊆ Acteur(Num_A) et Assure(Num_C) ⊆ Contrat(Num_C).

Dans ce cas, on ne fusionne pas. On conserve les trois relations : **Acteur**, **Contrat** et **Assure**. La relation **Assurance** dans l'exemple des slides, qui fusionne, semble être une simplification ou une interprétation spécifique du contexte. En général pour M:M, on ne fusionne pas pour éviter la redondance. Si on devait fusionner pour un cas [1:1] – (1,1)—(1,1) on pourrait obtenir:

- **Assurance**(Num_A, Prénom, Nom, Adr, Num_C, Type, Durée)
- **Acteur**(Num_A, Prénom, Nom, Adr) *Relation Acteur devient Assurance.*
- **Contrat**(Num_C, Type, Durée) *Relation Contrat disparait.*
- Contraintes de dépendance d'inclusion : Assurance(Num_A) ⊆ Acteur(Num_A) et Assurance(Num_C) ⊆ Contrat(Num_C).

3.5 Entités Faibles

Les entités faibles sont des entités dont l'existence dépend d'une autre entité, dite entité forte. Lors de la transformation, le schéma de relation associé à une entité faible doit inclure la clé primaire de l'entité forte pour former sa propre clé primaire.

- Le schéma de relation RF associé à une entité faible F inclut les attributs clé K1 de l'entité forte E1, en plus des attributs propres de F.
- La clé primaire de RF est composée des attributs clé de E1 (K1) et des attributs clé de F (KF) qui permettent d'identifier F de manière unique relativement à E1.
- L'association AF entre E1 et F est traduite implicitement par l'inclusion de la clé étrangère (K1) dans RF.

Example 3.6. Considérons l'exemple *Cinéma*, *Salle* et *Projète* où *Salle* est une entité faible dépendante de *Cinéma*.

- **Cinéma**(Id_C, Nom_C, Tél)
- **Salle**(Id_C, Num_S, Nb_Pl)
  - Clé primaire composée de (Id_C, Num_S).
  - Clé étrangère Id_C référençant *Cinéma*(Id_C).
- **Film**(Num_F, Titre, Année)
- **Projète**(Id_C, Num_S, Num_F)
  - Clé primaire composée de (Id_C, Num_S, Num_F).
  - Clé étrangère (Id_C, Num_S) référençant *Salle*(Id_C, Num_S).
  - Clé étrangère Num_F référençant *Film*(Num_F).

3.6 Associations d'héritage

Les associations d'héritage (ISA) représentent une spécialisation d'entités. Chaque sous-entité hérite des attributs de la sur-entité et possède ses attributs spécifiques. Lors de la transformation, chaque sous-entité devient une relation dont la clé primaire est la clé primaire de la sur-entité.

- Chaque sous-entité est transformée en un schéma de relation.
- Les attributs de la relation de la sous-entité sont les attributs spécifiques de la sous-entité ainsi que la clé primaire de la sur-entité (qui devient aussi la clé primaire de la sous-entité).
- Des contraintes d'inclusion sont utilisées pour lier les sous-entités à la sur-entité.

Example 3.7. Considérons l'exemple d'héritage avec *Film* comme sur-entité et *Documentaire* et *Animation* comme sous-entités.

- **Film**(Num_F, Titre, Année)
- **Documentaire**(Num_F, Sujet, Pays)
  - Clé primaire Num_F (clé étrangère référençant *Film*(Num_F)).
- **Animation**(Num_F, Style)
  - Clé primaire Num_F (clé étrangère référençant *Film*(Num_F)).
- Contraintes d'inclusion:
  - Documentaire(Num_F) $\subseteq$ Film(Num_F)
  - Animation(Num_F) $\subseteq$ Film(Num_F)

3.7 Exemple Complémentaire

Pour illustrer davantage, considérons un exemple plus complexe impliquant plusieurs entités et associations. Prenons le cas d'un modèle EA pour la gestion des départements, projets et employés dans une entreprise. (Voir diapositive 40 pour le diagramme original).

En suivant les règles de transformation, nous pourrions obtenir les relations suivantes (simplifiées) :

- **DEPARTEMENT**(Identifiant, Nom, Localisation)
- **PROJET**(Identifiant, Nom, Localisation_N°, Localisation_rue, Localisation_Ville, Localisation_CodePostal)
- **EMPLOYE**(N°_sécu, Salaire, Nom, Prénom)
- **CONTROLE**(Identifiant_Département, Identifiant_Projet) *Association M:N entre DEPARTEMENT et PROJET*
  - Clé étrangère Identifiant_Département référençant DEPARTEMENT(Identifiant)
  - Clé étrangère Identifiant_Projet référençant PROJET(Identifiant)
- **A_DIRIGE**(Identifiant_Département, Début) *Association 1:N entre DEPARTEMENT et PERIODE. La clé de PERIODE est incluse dans A_DIRIGE*
  - Clé étrangère Identifiant_Département référençant DEPARTEMENT(Identifiant)
- **TRAVAILLEPOUR**(Identifiant_Projet, N°_sécu) *Association M:N entre PROJET et EMPLOYE*

  - Clé étrangère Identifiant_Projet référençant PROJET(Identifiant)
  - Clé étrangère N°_sécu référençant EMPLOYE(N°_sécu)

- **TRAVAILLEDANS**(N°_sécu, Identifiant_Département) *Association M:N entre EMPLOYE et DEPARTEMENT*
  - Clé étrangère N°_sécu référençant EMPLOYE(N°_sécu)
  - Clé étrangère Identifiant_Département référençant DEPARTEMENT(Identifiant)

- **ESTCHEFDE**(N°_sécu, Identifiant_Département) *Association 0,n:0,1 entre EMPLOYE et DEPARTEMENT - peut être fusionnée avec EMPLOYE ou DEPARTEMENT selon le contexte et les cardinalités précises.*
  - Clé étrangère N°_sécu référençant EMPLOYE(N°_sécu)
  - Clé étrangère Identifiant_Département référençant DEPARTEMENT(Identifiant)

- **PERIODE**(Début, Fin) *Attributs de PERIODE deviennent attributs de A_DIRIGE avec Début comme clé partielle.*

Note: Les clés primaires sont soulignées. Les clés étrangères sont mentionnées pour chaque relation d'association.

Ce modèle relationnel, bien que plus complexe, illustre comment appliquer les principes de transformation à un schéma EA plus élaboré, en respectant les cardinalités et en gérant les relations entre les entités.

3.8 Glossaire

- **Association :** Relation entre deux ou plusieurs entités dans un modèle EA.
- **Cardinalité :** Contraintes spécifiant le nombre minimum et maximum d'instances d'entités participant à une association.
- **Clé (Modèle EA) :** Attribut ou ensemble d'attributs identifiant de manière unique une instance d'entité.
- **Clé étrangère (Modèle Relationnel) :** Attribut dans une relation qui référence la clé primaire d'une autre relation, établissant un lien entre elles.
- **Clé primaire (Modèle Relationnel) :** Attribut ou ensemble d'attributs qui identifie de manière unique chaque tuple (ligne) dans une relation.
- **Dépendance d'inclusion (Modèle Relationnel) :** Contrainte assurant que les valeurs d'une clé étrangère existent dans la relation référencée.
- **Dépendance fonctionnelle (Modèle Relationnel) :** Contrainte où la valeur d'un attribut détermine de manière unique la valeur d'un autre attribut.
- **Entité :** Représentation d'un objet ou concept du monde réel dans un modèle EA.
- **Modèle Entité-Association (EA) :** Modèle conceptuel de données utilisant des entités et des associations.
- **Modèle Relationnel :** Modèle logique de données basé sur des relations (tables).
- **Relation :** Table dans un modèle relationnel, composée de tuples (lignes) et d'attributs (colonnes).

CHAPTER 4

MODÈLE RELATIONNEL

4.1 Concepts Fondamentaux

Le **modèle relationnel** est un modèle de données qui utilise des tables pour représenter les données et les relations entre ces données. Ce modèle est basé sur des concepts mathématiques simples, ce qui le rend facile à comprendre et à utiliser.

Definition 4.1 (Schéma de relation)**.** Un **schéma de relation** R est défini par un nom de relation et un ensemble fini d'**attributs** $Att(R) = \{A, B, C, ...\}$. L'**arité** de R est le nombre d'attributs, c'est-à-dire la cardinalité de $Att(R)$.

Pour chaque attribut $A \in Att(R)$, on définit un **domaine de valeurs** $Dom(A)$, qui est l'ensemble des valeurs possibles que peut prendre l'attribut A.

Example 4.2. Considérons le schéma de relation **FILM** avec les attributs **Titre**, **Metteur-en-scène**, et **Acteur**. Nous pouvons le noter : FILM(Titre, Metteur-en-scène, Acteur).

Definition 4.3 (Relation (Instance de schéma de relation))**.** Une **relation** (ou **instance de schéma de relation**) r d'un schéma de relation $R(A_1, A_2, ..., A_n)$ est un ensemble de **n-uplets**. Un **n-uplet** u sur $(A_1, A_2, ..., A_n)$ est une séquence de n valeurs $(a_1, a_2, ..., a_n)$ telle que pour chaque $i \in \{1, ..., n\}$, $a_i \in Dom(A_i)$. On note $u[A_i] = a_i$ la composante de u correspondant à l'attribut A_i.

En termes plus simples, une relation est une table où chaque ligne est un n-uplet et chaque colonne correspond à un attribut.

Definition 4.4 (Instance de base de données relationnelle)**.** Une **instance de base de données relationnelle** I pour un schéma de base de données $BD = \{R_1, ..., R_m\}$ est un ensemble d'instances de relation $I = \{r_1, ..., r_m\}$, où chaque r_i est une instance du schéma de relation R_i.

Une instance de base de données est donc un ensemble de tables, chacune étant une instance de son schéma de relation.

4.2 Contraintes d'Intégrité

Les **contraintes d'intégrité** sont des règles qui assurent la qualité et la cohérence des données dans une base de données. Elles permettent de garantir que les données stockées respectent certaines propriétés. On distingue différents types de contraintes :

- **Contraintes structurelles (liées au modèle relationnel) :** Elles découlent directement du modèle relationnel. Par exemple, pour un n-uplet u et un attribut A_i, la valeur $u[A_i]$ doit appartenir au domaine $Dom(A_i)$.
- **Contraintes d'intégrité liées à l'application :** Elles sont spécifiques à l'application et expriment des règles métier. Elles doivent être satisfaites par chaque état valide de la base de données et vérifiées lors des mises à jour (insertions, modifications, suppressions).
- **Contraintes dynamiques :** Elles concernent l'évolution de la base de données au fil du temps et les transitions d'état autorisées.

Example 4.5 (Contraintes d'intégrité - Exemple 1)**.** Considérons une mini-application de gestion de cinémas où l'on souhaite imposer la contrainte suivante : "Un seul numéro de téléphone et une seule adresse par cinéma". Si nous avons une relation CINE(Nom-Ciné, Adresse, Téléphone), une instance de cette relation doit respecter cette contrainte pour être considérée comme **cohérente**.

L'instance suivante **n'est pas cohérente** car le cinéma 'Français' a deux adresses et deux numéros de téléphone différents :

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

En revanche, l'instance suivante **est cohérente** :

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

Example 4.6 (Contraintes d'intégrité - Exemple 2)**.** Autre exemple de contrainte : "Les films projetés dans les cinémas doivent être des films répertoriés dans la base de données des films". Si nous avons deux relations PROGRAMME(Nom-Ciné, Titre, Horaire) et FILM(Titre, Metteur-en-scène, Acteur), alors pour chaque tuple dans PROGRAMME, la valeur de l'attribut Titre doit exister comme valeur de l'attribut Titre dans la relation FILM.

L'instance suivante **n'est pas cohérente** car le film 'Western' projeté au cinéma 'Français' n'est pas répertorié dans la table FILM :

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

FILM		
Titre	**Metteur-en-scène**	**Acteur**
Speed 2	Jan de Bont	S. Bullock
Marion	M. Poirier	C. Tetard

En revanche, l'instance suivante **est cohérente** :

PROGRAMME		
Nom-Ciné	**Titre**	**Horaire**
Français	Speed 2	18h00
Français	Speed 2	20h00

FILM		
Titre	**Metteur-en-scène**	**Acteur**
Speed 2	Jan de Bont	S. Bullock
Marion	M. Poirier	C. Tetard

4.3 Dépendances Fonctionnelles

Les **dépendances fonctionnelles (DF)** sont un type important de contrainte d'intégrité qui spécifie une relation entre des ensembles d'attributs. Elles sont fondamentales pour la conception de bases de données relationnelles, notamment pour la normalisation des schémas.

Definition 4.7 (Dépendance Fonctionnelle)**.** Soient $R(A_1, ..., A_n)$ un schéma de relation, et $X, Y \subseteq \{A_1, ..., A_n\}$ deux ensembles d'attributs. On dit qu'il existe une **dépendance fonctionnelle** de X vers Y, notée $X \to Y$, si pour toute instance r de R et pour tout couple de n-uplets $u, v \in r$, si $u[X] = v[X]$ (c'est-à-dire si les valeurs des attributs de X sont identiques dans u et v), alors $u[Y] = v[Y]$ (alors les valeurs des attributs de Y sont aussi identiques dans u et v).

On dit que "X détermine fonctionnellement Y" ou "Y dépend fonctionnellement de X".

Example 4.8. Dans le schéma de relation CINE(Nom-Ciné, Adresse, Téléphone), si l'on impose la contrainte "un seul numéro de téléphone et une seule adresse par cinéma", alors on a les dépendances fonctionnelles suivantes :

- Nom-Ciné → Adresse
- Nom-Ciné → Téléphone

Remark 4.9 (Dépendances fonctionnelles triviales)**.** Une dépendance fonctionnelle $X \to Y$ est dite **triviale** si $Y \subseteq X$. Les dépendances triviales sont toujours satisfaites pour toute instance de relation. Par exemple, $A \to A$ ou $\{A, B\} \to \{A\}$ sont des dépendances triviales.

4.3.1 Exercices

Example 4.10 (Exercice)**.** Considérons la relation (instance) R suivante :

A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2
a2	b2	c2	d2
a2	b3	c2	d3
a3	b3	c2	d4

Vérifions si les dépendances fonctionnelles suivantes sont satisfaites par cette instance de relation R :

1. $A \to C$
2. $C \to A$
3. $\{A, B\} \to D$
4. $A \to A$

Solution.

1. $A \to C$ **: Vrai.** Comparons les n-uplets avec la même valeur pour l'attribut A.
   - Pour $A = a_1$, les n-uplets sont (a1, b1, c1, d1) et (a1, b2, c1, d2). Ils ont la même valeur pour C (c_1).
   - Pour $A = a_2$, les n-uplets sont (a2, b2, c2, d2) et (a2, b3, c2, d3). Ils ont la même valeur pour C (c_2).
   - Pour $A = a_3$, un seul n-uplet (a3, b3, c2, d4).

   Donc $A \to C$ est satisfaite.
2. $C \to A$ **: Faux.** Comparons les n-uplets avec la même valeur pour l'attribut C.
   - Pour $C = c_1$, les n-uplets sont (a1, b1, c1, d1) et (a1, b2, c1, d2). Ils ont la même valeur pour A (a_1).
   - Pour $C = c_2$, les n-uplets sont (a2, b2, c2, d2), (a2, b3, c2, d3) et (a3, b3, c2, d4). Ils n'ont **pas** tous la même valeur pour A (a_2 et a_3).

   Donc $C \to A$ n'est pas satisfaite.
3. $\{A, B\} \to D$ **: Vrai.** Comparons les n-uplets avec les mêmes valeurs pour les attributs A et B.
   - Pour $\{A = a_1, B = b_1\}$, un seul n-uplet (a1, b1, c1, d1).
   - Pour $\{A = a_1, B = b_2\}$, un seul n-uplet (a1, b2, c1, d2).
   - Pour $\{A = a_2, B = b_2\}$, un seul n-uplet (a2, b2, c2, d2).
   - Pour $\{A = a_2, B = b_3\}$, un seul n-uplet (a2, b3, c2, d3).
   - Pour $\{A = a_3, B = b_3\}$, un seul n-uplet (a3, b3, c2, d4).

   Dans tous les cas où nous avons une combinaison de valeurs de A et B, il n'y a qu'un seul n-uplet correspondant, donc la condition de la DF est trivialement satisfaite. (En fait, on devrait comparer les n-uplets ayant les mêmes valeurs pour A et B; ici il n'y en a jamais deux différents avec les mêmes valeurs pour A et B.)
4. $A \to A$ **: Vrai.** C'est une dépendance triviale, donc toujours satisfaite.

□

4.4 Dépendances d'Inclusion

Les **dépendances d'inclusion** expriment des contraintes entre les valeurs des attributs de différentes relations. Elles sont particulièrement utilisées pour représenter les **clés étrangères** et les relations de type ISA (est-un).

Definition 4.11 (Dépendance d'inclusion)**.** Soient $R(A_1, ..., A_n)$ et $S(B_1, ..., B_m)$ deux schémas de relation. Soient $\{A_{\alpha_1}, ..., A_{\alpha_k}\} \subseteq \{A_1, ..., A_n\}$ et $\{B_{\beta_1}, ..., B_{\beta_k}\} \subseteq \{B_1, ..., B_m\}$ deux ensembles d'attributs de même taille k. Il existe une **dépendance d'inclusion** de $\{A_{\alpha_1}, ..., A_{\alpha_k}\}$ dans R vers $\{B_{\beta_1}, ..., B_{\beta_k}\}$ dans S, notée :

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

si pour toute instance r de R et toute instance s de S, la projection des n-uplets de r sur les attributs $\{A_{\alpha_1}, ..., A_{\alpha_k}\}$ est un sous-ensemble de la projection des n-uplets de s sur les attributs $\{B_{\beta_1}, ..., B_{\beta_k}\}$.

En termes plus simples, pour chaque tuple u dans r, il doit exister un tuple v dans s tel que $u[A_{\alpha_j}] = v[B_{\beta_j}]$ pour tout $j = 1, ..., k$.

Example 4.12 (Dépendance d'inclusion - Exemple 2 (Reprise))**.** Reprenons l'exemple où "Les films projetés sont des films répertoriés". Avec les schémas de relation PROGRAMME(Nom-Ciné, Titre, Horaire) et FILM(Titre, Metteur-en-scène, Acteur), la contrainte "Les films projetés sont des films répertoriés" se traduit par la dépendance d'inclusion :

PROGRAMME[Titre] $\subseteq$ FILM[Titre]

Cela signifie que l'ensemble des titres de films dans la relation PROGRAMME doit être un sous-ensemble de l'ensemble des titres de films dans la relation FILM.

4.4.1 Clé étrangère ou Contrainte de référence

La notion de **clé étrangère** est une application importante des dépendances d'inclusion. Elle permet de relier deux relations et de maintenir l'intégrité référentielle entre elles.

Definition 4.13 (Clé étrangère)**.** Soient $R(..., A, B, C, ...)$ et $S(..., A', B', C', ...)$ deux schémas de relation. On dit que $\{A', B', C'\}$ est une **clé étrangère** de S référençant $\{A, B, C\}$ dans R si :

1. $\{A, B, C\}$ est une **clé** de R (c'est-à-dire, $\{A, B, C\}$ détermine tous les autres attributs de R).
2. Il existe une dépendance d'inclusion $S[A', B', C'] \subseteq R[A, B, C]$.
3. Les domaines de A' et A, B' et B, C' et C doivent être compatibles.

En résumé, une clé étrangère dans une relation S est un ensemble d'attributs dont les valeurs doivent correspondre à des valeurs existantes d'une clé candidate dans une autre relation R. Cela assure que les références entre les relations sont valides.

SQL - LANGAGE DE DÉFINITION DE DONNÉES (LDD)

4.5 Introduction à SQL

SQL (Structured Query Language) est un langage standardisé conçu pour la gestion et la manipulation de données dans les systèmes de gestion de bases de données relationnelles (SGBDR). Développé par IBM dans les années 1970 (première version en 1970 par Donald Chamberlain et Raymond Boyce, nommé SEQUEL à l'origine, pour Structured English as a Query Language), SQL est devenu la norme ANSI et ISO pour les bases de données relationnelles.

Avantages de SQL :

- **Standardisation :** SQL est un langage standard, ce qui assure une certaine portabilité des applications entre différents SGBDR.
- **Large diffusion :** SQL est implanté (complètement ou en partie) sur la plupart des SGBDR commerciaux et open source.
- **Interopérabilité et portabilité des applications :** Le standard SQL facilite le développement d'applications portables entre différents systèmes de bases de données.

Inconvénients de SQL :

- **Évolution par extensions :** L'évolution du standard SQL s'est faite par ajout d'"extensions" (SQL1: 1989, SQL2: 1992, SQL3: 1999, SQL:2003, SQL:2008, SQL:2011, SQL:2016), ce qui peut entraîner des problèmes de compatibilité entre les différentes versions et implémentations.
- **Frein à l'émergence de nouveaux langages :** La dominance de SQL peut freiner l'adoption de nouveaux paradigmes et langages pour la gestion de données.

SQL est un langage multi-facettes, utilisé pour :

- **Langage de définition de données (LDD) :** Création et modification de la structure de la base de données (schémas, tables, vues, index, etc.). Commandes principales : 'CREATE TABLE', 'ALTER TABLE', 'DROP TABLE'. Déclaration des contraintes d'intégrité.
- **Langage de manipulation de données (LMD) :** Requêtes pour extraire des informations (requêtes simples, requêtes imbriquées, agrégats) et mises à jour (insertion, suppression, modification) des données. Commandes principales pour les requêtes : 'SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ...'. Commandes principales pour les mises à jour : 'UPDATE', 'INSERT', 'DELETE'.

SQL peut être utilisé de différentes manières :

- **En mode interactif :** Via une console ou des interfaces graphiques (clients SQL).
- **Incorporé dans des langages de programmation :** Pour accéder aux bases de données depuis des applications (C, C++, Java, PHP, Python, etc.).

Pour les exercices pratiques (TP), nous utiliserons le SGBD PostgreSQL.

4.6 Définition de Schémas

La commande principale pour définir le schéma d'une relation en SQL est 'CREATE TABLE'.

Listing 4.1: Création de la table Film

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

Listing 4.2: Création de la table Prog

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

Listing 4.3: Création de la table Cine

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

La syntaxe générale de la commande ‘CREATE TABLE‘ est :

Listing 4.4: Syntaxe générale de CREATE TABLE

```
CREATE TABLE Nom\_Relation (
    Attribut\_1 Type\_1,
    Attribut\_2 Type\_2,
    ...
    Attribut\_n Type\_n
);
```

Les types de données de base en SQL incluent :

- Numériques : ‘INT‘, ‘SMALLINT‘, ‘BIGINT‘, ‘FLOAT‘, ‘REAL‘, ‘DOUBLE PRECISION‘
- Chaînes de caractères : ‘CHAR(M)‘ (longueur fixe), ‘VARCHAR(M)‘ (longueur variable)
- Données binaires : ‘BLOB‘, ‘BINARY‘, ‘VARBINARY‘
- Dates et heures : ‘DATE‘, ‘TIME‘, ‘DATETIME‘, ‘TIMESTAMP‘, ‘YEAR‘

Pour modifier la structure d'une table existante, on utilise la commande ‘ALTER TABLE‘. Pour supprimer une table, on utilise ‘DROP TABLE‘.

4.7 Contraintes d'Intégrité en SQL

SQL permet de définir différentes contraintes d'intégrité lors de la création ou de la modification des tables.

4.7.1 Valeurs nulles et valeurs par défaut

Par défaut, les attributs peuvent accepter la valeur ‘NULL‘ (absence de valeur). On peut spécifier explicitement qu'un attribut ne doit pas accepter la valeur ‘NULL‘ avec la contrainte ‘NOT NULL‘. On peut aussi définir une valeur par défaut pour un attribut avec ‘DEFAULT valeur‘.

4.7.2 Clé primaire

Pour définir une **clé primaire** (qui identifie de manière unique chaque n-uplet d'une table), on utilise la contrainte ‘PRIMARY KEY‘. Une clé primaire implique automatiquement la contrainte ‘NOT NULL‘.

Listing 4.5: Définition d'une clé primaire

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

4.7.3 Clé unique

Pour définir une **clé unique** (qui assure l'unicité des valeurs d'un ou plusieurs attributs, mais autorise la valeur NULL), on utilise la contrainte ‘UNIQUE‘.

Listing 4.6: Définition d'une clé unique

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

4.7.4 Clé étrangère (Contrainte de référence)

Pour définir une **clé étrangère**, on utilise la contrainte 'FOREIGN KEY ... REFERENCES ...'. Elle permet d'établir un lien entre deux tables et d'assurer l'intégrité référentielle.

Listing 4.7: Définition d'une clé étrangère

```
CREATE TABLE Prog (
    Nom_Cine CHAR(20),
    NumF INT,
    Salle INT,
    Horaire TIME,
    FOREIGN KEY (NumF) REFERENCES Film(NumF)
);
```

Ou en modifiant une table existante :

Listing 4.8: Ajout d'une clé étrangère avec ALTER TABLE

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

4.8 Actions sur Violations de Contraintes de Référence

Lorsqu'une opération de mise à jour (insertion, suppression, modification) viole une contrainte de référence (clé étrangère), SQL propose différentes actions pour gérer cette violation :

- **REJECT (RESTRIC ou NO ACTION) :** L'opération est rejetée et la base de données reste dans son état précédent. C'est l'action par défaut.
- **CASCADE :** Si une ligne référencée est supprimée (ou la valeur de la clé référencée est modifiée), les lignes référençantes sont aussi supprimées (ou la valeur de la clé étrangère est mise à jour en cascade).
- **SET NULL :** Si une ligne référencée est supprimée (ou la valeur de la clé référencée est modifiée), la valeur de la clé étrangère dans les lignes référençantes est mise à 'NULL'. Cette option n'est possible que si l'attribut clé étrangère accepte la valeur 'NULL' (pas de contrainte 'NOT NULL').
- **SET DEFAULT :** Similaire à 'SET NULL', mais la clé étrangère est mise à une valeur par défaut spécifiée.

Ces options se spécifient lors de la définition de la clé étrangère, avec les clauses 'ON DELETE' et 'ON UPDATE'.

Listing 4.9: Définition d'actions sur violations de contraintes de référence

```
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
);
```

Exemples d'actions sur violations de contraintes de référence :
Supposons les tables FILM et PROG avec la clé étrangère 'NumF' dans PROG référençant 'NumF' dans FILM.

- **REJECT (par défaut) :** Si on essaie de supprimer un film de FILM qui est encore référencé dans PROG, l'opération de suppression est rejetée.
- **CASCADE :** Si on supprime le film f3 de FILM, toutes les projections de ce film dans PROG (NC1, NC2 pour f3) sont automatiquement supprimées en cascade.
- **SET NULL :** Si on supprime le film f3 de FILM, la valeur de 'NumF' pour les projections de ce film dans PROG (NC1, NC2 pour f3) est mise à 'NULL'.

4.8.1 Contraintes CHECK et ASSERTION

Contrainte CHECK : Permet de définir une condition qui doit être vérifiée par toute valeur d'un attribut ou par tout n-uplet lors des opérations d'insertion ou de modification.

Listing 4.10: Contrainte CHECK sur un attribut

```
CREATE TABLE Prog (
    Nom_Cine CHAR(20),
    NumF INT CHECK (NumF IN (SELECT NumF FROM Film)),
    Salle INT,
    Horaire TIME
);
```

Listing 4.11: Contrainte CHECK sur un n-uplet

```
CREATE TABLE Prog (
    Nom_Cine CHAR(20),
    NumF INT,
    Salle INT,
    Horaire TIME,
    CHECK (Salle='Artessai' OR Horaire < '22:30:00')
);
```

Contrainte ASSERTION : Permet de définir des contraintes plus complexes impliquant potentiellement plusieurs tables. Les assertions sont vérifiées lors de chaque mise à jour des relations impliquées.

Listing 4.12: Contrainte ASSERTION

```
CREATE ASSERTION film\_cine
CHECK (
    (SELECT COUNT(Salle) From Prog
     WHERE Titre IN (SELECT Titre FROM Film WHERE Metteur\_en\_scene = 'Poirier'))
    <=
    (SELECT COUNT(DISTINCT Titre) FROM Film WHERE Metteur\_en\_scene = 'Poirier') * 10
);
```

Cette assertion vérifie que le nombre de salles programmant un film de Poirier est inférieur à 10 fois le nombre de films réalisés par Poirier.

4.8.2 Différence entre Contraintes et Triggers

Les **contraintes** sont des mécanismes déclaratifs pour imposer des règles d'intégrité dans une base de données. Elles sont définies au niveau du schéma et sont automatiquement vérifiées par le SGBD lors des opérations de mise à jour. Les **triggers** (déclencheurs) sont des procédures stockées qui sont automatiquement exécutées en réponse à certains événements de base de données (insertion, suppression, modification). Les triggers offrent plus de flexibilité que les contraintes et permettent d'implémenter des règles d'intégrité plus complexes ou des actions spécifiques en réaction à des événements. Cependant, les contraintes sont généralement plus performantes et plus faciles à maintenir pour les règles d'intégrité standard. Les triggers relèvent de la manipulation de données (LMD) et non de la définition de données (LDD).

CHAPTER 5

5.1 Algèbre Relationnelle

5.1.1 Langages d'interrogation : Déclaratif vs. Procédural

- **Interroger** = déduire des informations à partir de celles stockées dans la base de données.

On distingue deux types de langages d'interrogation :

- **Langage déclaratif** : Spécifie *quoi* demander.
  - Calcul à variable domaine ou n-uplet.
  - **SQL** est un langage déclaratif.
  - Exemple : *Donnez-moi une tartine de confiture.*
- **Langage procédural** : Spécifie *quoi* et *comment* obtenir le résultat.
  - **Algèbre relationnelle** est un langage procédural.
  - Exemple : *Coupez une tranche de pain, ouvrez le pot de confiture de fraises, ...*

5.1.2 Pourquoi apprendre l'algèbre relationnelle ?

- Spécifier une requête = utiliser SQL (déclaratif).
- Évaluer une requête = utiliser l'algèbre relationnelle (procédural).

5.1.3 Introduction aux opérateurs de l'algèbre relationnelle

L'algèbre relationnelle est basée sur des opérateurs qui prennent en entrée des relations et produisent une nouvelle relation en sortie. On distingue :

- **Opérateurs unaires** (agissant sur une seule relation) :
  - Projection
  - Sélection
  - Extraction
- **Opérateurs binaires** (agissant sur deux relations) :
  - Jointure

  - Union
  - Différence

- **Opérateurs dérivés** :
  - Produit cartésien
  - Intersection
  - Division
  - ...

5.1.4 Typage des Opérateurs

- Chaque opérateur définit une **application** de l'ensemble des instances d'un **schéma source** dans l'ensemble des instances d'un **schéma cible**.
  - Schéma source = plusieurs schémas de relation.
  - Schéma cible = 1 schéma de relation.
- Le **schéma cible** (format du résultat) est déterminé par :
  1. Le schéma source
  2. L'opérateur

5.1.5 L'opérateur d'extraction

Définition

L'opérateur d'extraction permet d'**extraire une table parmi celles de la base de données**.

Definition 5.1 (Opérateur d'extraction)**.** L'opérateur d'extraction permet d'**extraire une table parmi celles de la base de données**.

Syntaxe et Sémantique

Soit un schéma source de base de données $BD = \{R_1, R_2, ..., R_n\}$ où $W_i = Att(R_i)$ sont les attributs de R_i. L'opérateur d'extraction $[R_i]$ est une application de l'ensemble des instances du schéma source BD dans l'ensemble des instances du schéma cible $Res(W_i)$.

Sémantiquement, pour une instance $bd = (r_1, r_2, ..., r_n)$ de BD, on a :

$$[R_i](bd) = r_i$$

Exemple

Example 5.2 (Opérateur d'extraction)**.** Considérons les tables `film`, `programmation` et `ciné` :

```
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

PROGR.    Nom-Ciné   Titre      Horaire
        Français    Speed 2     18h00
        Français    Speed 2     20h00
        Français    Speed 2     22h00
```

```
        Français    Marion      16h00
        Trianon     Marion      18h00

CINE      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'opération d'extraction $[PROG]$ appliquée à la base de données retourne la table `programmation`:

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

5.1.6 L'opérateur de projection (Π)

Définition

L'opérateur de projection est un **opérateur unaire** et **vertical** qui permet de **supprimer des colonnes d'une table**.

Definition 5.3 (Opérateur de projection)**.** L'opérateur de projection est un **opérateur unaire** et **vertical** qui permet de **supprimer des colonnes d'une table**.

Syntaxe et Sémantique

Soit un schéma source $R(V)$ et W un ensemble d'attributs tel que $W \subseteq V$. L'opérateur de projection Π_W est une application de l'ensemble des instances du schéma source $R(V)$ dans l'ensemble des instances du schéma cible $Res(W)$.

Sémantiquement, soit r une instance de R.

$$\Pi_W(r) = \{u[W] \mid u \in r\}$$

où $u[W]$ représente la restriction du tuple u aux attributs de W. La projection élimine les doublons dans les tuples résultants.

Exemples

Exemple 1 : Les titres des films ?

Appliquons l'opérateur de projection Π_{Titre} sur la relation `film`.

Example 5.4 (Projection - Exemple 1)**.** Table `film` source :

```
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
```

Résultat de la projection Π_{Titre}(`film`) :

```
Titre
```

```
Speed 2
Marion
```

On remarque l'élimination des doublons.

Exemple 2 : Les titres des films à l'affiche ?
Appliquons l'opérateur de projection Π_{Titre} sur la relation `programme`.

Example 5.5 (Projection - Exemple 2)**.** Table `programme` source :

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

Résultat de la projection Π_{Titre}(`programme`) :

```
Titre
Speed2
Marion
```

Cas limite

Projection sur aucun attribut : $\Pi_{\emptyset}[r]$?

- Instances du schéma de relation $S(\emptyset)$?
  - Un seul 0-uplet : ().
  - Deux instances possibles : $S_1 = \{()\}$ (vrai) et $S_2 = \{\}$ (faux).
- Résultat de la projection sur un ensemble vide d'attributs :
  - Si l'instance r de R est non vide alors $\Pi_{\emptyset}(r) = \{()\}$.
  - Si l'instance r de R est vide alors $\Pi_{\emptyset}(r) = \{\}$.
- Utilité ? Exprimer des requêtes OUI/NON.
- Exemple : Y-a-t-il des films à l'affiche ? $\Pi_{\emptyset}[PROG]$.

5.1.7 L'opérateur de sélection (σ)

Définition

L'opérateur de sélection est un **opérateur unaire** et **horizontal** qui permet de **garder les lignes satisfaisant certains critères**.

Definition 5.6 (Opérateur de sélection)**.** L'opérateur de sélection est un **opérateur unaire** et **horizontal** qui permet de **garder les lignes satisfaisant certains critères**.

Définitions des Conditions

Soit V un ensemble d'attributs.

- Une **condition élémentaire** sur V est de la forme :

$$A \text{ comp } a \quad \text{ou} \quad A \text{ comp } B$$

avec $A, B \in V$, $a \in Dom(A)$, $Dom(A) = Dom(B)$, et *comp* est un comparateur $(=, <, \leq, ...)$.

- Une **condition** sur V est :
  1. Une condition élémentaire.
  2. Si C_1 et C_2 sont des conditions sur V, alors $C_1 \wedge C_2$, $C_1 \vee C_2$, $\neg C_1$ sont des conditions sur V.

We can define condition and condition elementaire more formally.

Definition 5.7 (Condition élémentaire)**.** Une **condition élémentaire** sur V est de la forme :

$$A \text{ comp } a \quad \text{ou} \quad A \text{ comp } B$$

avec $A, B \in V$, $a \in Dom(A)$, $Dom(A) = Dom(B)$, et *comp* est un comparateur $(=, <, \leq, ...)$.

Definition 5.8 (Condition)**.** Une **condition** sur V est définie recursively as:

1. Une condition élémentaire.
2. Si C_1 et C_2 sont des conditions sur V, alors $C_1 \wedge C_2$, $C_1 \vee C_2$, $\neg C_1$ sont des conditions sur V.

Syntaxe et Sémantique

Soit un schéma source $R(V)$ et C une condition sur V. L'opérateur de sélection σ_C est une application de l'ensemble des instances du schéma source $R(V)$ dans l'ensemble des instances du schéma cible $Res(V)$.

Sémantiquement :

$$\sigma_C(r) = \{u \mid u \in r \text{ et } u \text{ satisfait } C\}$$

où "u satisfait C" est défini de manière intuitive.

Exemples

Exemple 1 : Films dont le titre est "Speed2"?

Appliquons l'opérateur de sélection $\sigma_{Titre="Speed2"}$ sur la relation `film`.

Example 5.9 (Sélection - Exemple 1)**.** Table `film` source :

```
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
\verbatim}

Résultat de la sélection $\sigma_{Titre="Speed 2"}(\texttt{film})$ :
\begin{verbatim}
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
```

Exemple 2 : Qui a son propre metteur en scène et dans quel film ?

Appliquons l'opérateur de sélection $\sigma_{MeS=Acteur}$ sur la relation `film`.

Example 5.10 (Sélection - Exemple 2)**.** Table `film` source :

```
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
```

Résultat de la sélection $\sigma_{MeS=Acteur}(\texttt{film})$:

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

Exemple 3 : Programmation après 20h00 du film "Marion"?
Appliquons l'opérateur de sélection $\sigma_{Titre="Marion" \wedge Horaire \geq 20h00}$ sur la relation `programme`.

Example 5.11 (Sélection - Exemple 3)**.** Table `programme` source :

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

Résultat de la sélection $\sigma_{Titre="Marion" \wedge Horaire \geq 20h00}(\texttt{programme})$:

```
Nom-Ciné   Titre      Horaire
Trianon     Marion      22h00
```

Exemple 4 : Programmation des films dont le titre est "Marion" ou à l'affiche de l'UGC ?
Appliquons l'opérateur de sélection $\sigma_{(Titre="Marion") \vee (Nom\_Cine="UGC")}$ sur la relation `programme`.

Example 5.12 (Sélection - Exemple 4)**.** Table `programme` source :

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

Résultat de la sélection $\sigma_{(Titre="Marion") \vee (Nom\_Cine="UGC")}(\texttt{programme})$:

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

5.1.8 Composition des opérateurs : extraction, projection et sélection

Les opérateurs peuvent être composés pour former des expressions plus complexes.

Exemple : Noms des cinémas qui projettent le film Marion après 20h00 avec l'horaire exact de projection.

Cette requête peut être décomposée en plusieurs étapes :

1. Extraction de la relation `PROG` : $[PROG]$

2. Sélection des tuples pour le film "Marion" après 20h00 : $\sigma_{Titre="Marion" \wedge Horaire \geq 20h00}([PROG])$
3. Projection sur les attributs Nom-Ciné et Horaire : $\Pi_{Nom\_Cine,Horaire}(\sigma_{Titre="Marion" \wedge Horaire \geq 20h00}([PROG]))$

Schéma des opérations :

```
db
|
extraction: [PROG]
↓
programme
|
sélection : _Titre=MarionHoraire20h00
↓
RES-int   Nom-Ciné    Titre   Horaire
          Trianon     Marion  22h00
|
projection: _Nom-cine, Horaire
↓
RES-final Nom-Ciné    Horaire
          Trianon     22h00
```

Expression algébrique complète :

$$\Pi_{Nom\_cine,Horaire}[\sigma_{Titre=Marion \wedge Horaire \geq 20h00}[PROG]](bd)$$

5.1.9 Expression Algébrique

- Un opérateur est une **application**.
- La composition d'opérateurs est la **composition d'applications**.
- Une **expression algébrique** est :
  - $[R]$ où R est un schéma de relation, $Att(R) = V$ (**extraction**).
    - Schéma source : $BD = ..., R, ...$
    - Schéma cible : $RES_E(V)$
  - Si E est une expression algébrique dont le schéma cible est $RES_E(V)$, alors :
    - $\Pi_W[E]$ est une expression si $W \subseteq V$ (**projection**).
      - Schéma source : $RES_E(V)$
      - Schéma cible : $RES_1(W)$
    - $\sigma_C[E]$ est une expression si C est une condition sur V (**sélection**).
      - Schéma source : $RES_E(V)$
      - Schéma cible : $RES_2(V)$

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

Introduction

La jointure est un **opérateur binaire** qui permet de **combiner le contenu de deux instances** en se servant des valeurs dans les colonnes communes.

Definition 5.13 (Opérateur de jointure)**.** La jointure est un **opérateur binaire** qui permet de **combiner le contenu de deux instances** en se servant des valeurs dans les colonnes communes.

Syntaxe et Sémantique

L'opérateur de jointure $[R] \bowtie [S]$ est une application de l'ensemble des couples d'instances de R et S dans l'ensemble des instances de $Res(V \cup W)$. Sémantiquement :

$$[R] \bowtie [S](r, s) = \{u \mid u[V] \in r \text{ et } u[W] \in s\}$$

Exemple

Considérons deux instances r de $R(A, B, C)$ et s de $S(B, C, D)$:

Example 5.14 (Jointure - Exemple). `r`

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

`s`

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

La jointure $[R] \bowtie [S]$ appliquée à (r, s) donne :

[R][S]	A	B	C	D
	a1	b1	c1	d1
	a2	b1	c1	d1
	a2	b2	c2	d1

Exemple concret

Requête : 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 et l'horaire(s) et l'adresse du cinéma !

Pour répondre à cette requête, on doit combiner les informations des tables `FILM`, `PROG` et `CINE`. L'expression algébrique correspondante est :

$$\Pi_{Titre,Horaire,Adresse}(\sigma_{Acteur=Pisier}[FILM] \bowtie [PROG.] \bowtie [CINE])$$

5.1.11 Cas particuliers de jointure

Intersection ($\cap$)

Si $R(V)$ et $S(W)$ sont deux schémas tels que $V = W$, alors la jointure devient l'intersection :

$$[R] \bowtie [S] = [R] \cap [S]$$

Remark 5.15 (Intersection et Jointure). Si $R(V)$ et $S(W)$ sont deux schémas tels que $V = W$, alors la jointure devient l'intersection :

$$[R] \bowtie [S] = [R] \cap [S]$$

Produit cartésien ($\times$)

Si $R(V)$ et $S(W)$ sont deux schémas tels que $V \cap W = \emptyset$, alors la jointure devient le produit cartésien :

$$[R] \bowtie [S] = [R] \times [S]$$

Remark 5.16 (Produit cartésien et Jointure). Si $R(V)$ et $S(W)$ sont deux schémas tels que $V \cap W = \emptyset$, alors la jointure devient le produit cartésien :

$$[R] \bowtie [S] = [R] \times [S]$$

5.1.12 Renomage (ρ)

Le renomage permet de changer le nom des attributs d'une relation. Il est utile dans plusieurs situations, notamment pour :

- Donner le même nom à des attributs distincts (pour l'union ou l'intersection).
- Donner des noms distincts à des occurrences distinctes d'un attribut (pour la jointure d'une relation avec elle-même).

Syntaxe : $\rho_{B \to K, C \to L}(R)$.

Exemple : Les metteurs en scène qui sont aussi acteurs. Pour trouver les metteurs en scène qui sont aussi acteurs, on peut utiliser le renomage et l'intersection :

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

5.1.13 Opérations ensemblistes : Union ($\cup$), Différence ($-$)

Union ($\cup$)

L'union de deux relations R et S (avec le même schéma) combine tous les tuples de R et S, en éliminant les doublons. **Exemple : Les personnes (acteurs et metteurs en scène) ayant travaillé sur le tournage du film Marion.**

On peut obtenir cette information en combinant les acteurs et les metteurs en scène du film "Marion" en utilisant l'union après projection et renommage.

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

Différence ($-$)

La différence de deux relations R et S (avec le même schéma) retourne les tuples de R qui ne sont pas dans S. **Exemple : Les acteurs qui ne sont pas metteurs en scène.**

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

5.2 Langage SQL

5.2.1 Introduction à SQL

SQL (Structured Query Language) est le langage standard pour la gestion et l'interrogation des bases de données relationnelles.

5.2.2 Langage de Définition de Données (LDD)

Le LDD permet de définir la structure de la base de données, c'est-à-dire les schémas des relations, les types de données, et les contraintes d'intégrité. Les principales commandes LDD sont :

- `CREATE TABLE` : Pour créer une nouvelle table.
- `ALTER TABLE` : Pour modifier la structure d'une table existante.
- `DROP TABLE` : Pour supprimer une table.

5.2.3 Langage de Manipulation de Données (LMD)

Le LMD permet de manipuler les données contenues dans la base, notamment pour :

- **Requêtes** (interrogation) : Extraire des informations de la base.
- **Mises à jour** : Modifier les données existantes (insertion, suppression, modification).

Requêtes Simples (SELECT, FROM, WHERE)

La structure de base d'une requête SQL simple est :

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

- `SELECT` : Spécifie les attributs à projeter (schéma cible).
- `FROM` : Spécifie les relations sources (extraction).
- `WHERE` : Spécifie les conditions de sélection (sélection et jointure).

Exemples :

- `SELECT * FROM FILM` : Sélectionne tous les attributs et tous les tuples de la relation `FILM`. Équivalent à l'opérateur d'extraction $[FILM]$ suivi d'une projection Π^*.
- `SELECT * FROM FILM WHERE Acteur='Adjani'` : Sélectionne tous les films où l'acteur est 'Adjani'. Équivalent à $\sigma_{Acteur='Adjani'}[FILM]$ suivi de Π^*.
- `SELECT DISTINCT Titre FROM FILM WHERE Acteur='Adjani'` : Sélectionne les titres uniques des films où l'acteur est 'Adjani'. Équivalent à $\Pi_{Titre}(\sigma_{Acteur='Adjani'}[FILM])$. `DISTINCT` permet d'éliminer les doublons.

Attention : Sans `DISTINCT`, SQL effectue une projection "multi-ensembliste" (avec doublons).

Clause DISTINCT

`DISTINCT` permet de ne conserver qu'une seule ligne parmi plusieurs lignes de résultat totalement identiques.
Coût : Opération coûteuse (tri externe). Ne pas utiliser si inutile.
Exemple : Relation `FILM-Bis(Num_f, Titre, Durée)`.

- `SELECT DISTINCT Num_f, Titre FROM FILM-Bis WHERE durée = 2` : Inutile car `Num_f` est clé de `FILM-Bis`.
- `SELECT DISTINCT Titre FROM FILM-Bis WHERE durée = 2` : Utile pour obtenir les titres uniques des films durant 2 unités de temps.

Renomage d'attributs (AS)

La clause `AS` permet de renommer les attributs dans le résultat de la requête. **Exemple :** `SELECT Titre AS Adjani_movies FROM FILM WHERE Acteur='Adjani'`.

Expressions arithmétiques et constantes

SQL permet d'utiliser des expressions arithmétiques et d'introduire des constantes dans la clause `SELECT`.
Exemples :

- `SELECT Titre, Durée*0.016667 AS durée-en-heure FROM FILM-durée` : Calcule la durée en heures à partir d'un attribut `Durée`.
- `SELECT Titre, Durée*0.016667 AS durée-en-heure, 'heure' AS Unité FROM FILM-durée` : Ajoute une colonne constante 'heure'.

Conditions complexes (WHERE)

La clause `WHERE` permet de spécifier des conditions complexes en utilisant :

- Comparateurs habituels : $<, >, =, !=, <=, >=$.
- Comparateurs spécifiques : `LIKE`, `BETWEEN`, `IN`, `IS [NOT] NULL`.
- Expressions arithmétiques, concaténation (——).
- Connecteurs logiques : `OR`, `AND`, `NOT`.

Exemples :

- `SELECT Titre FROM FILM WHERE Acteur='Adjani' OR MeS='Poirier'`.
- `SELECT Titre FROM FILM WHERE Acteur IN ('Adjani', 'Depardieu')`.
- `SELECT Titre FROM FILM WHERE Titre LIKE 'Star ____'`. (Motifs de recherche, `%` pour chaîne quelconque, `_` pour un caractère).
- Manipulation de dates avec `DATE 'aaaa-mm-jj'` et `BETWEEN DATE 'date1' AND DATE 'date2'`.

Valeurs Nulles

Une valeur nulle représente une valeur inconnue, un attribut inapproprié ou une valeur incertaine.

- Comparaison avec une valeur nulle : Résultat `inconnu`.
- Conditions : `IS NULL`, `IS NOT NULL`.
- **Attention** : `WHERE attribut = NULL` est toujours évalué à "inconnu", donc aucun tuple n'est sélectionné.

Ordonnancement (ORDER BY)

La clause `ORDER BY` permet d'ordonner les résultats selon un ou plusieurs attributs. Par défaut, l'ordre est ascendant (`ASC`), on peut spécifier descendant avec `DESC`. **Exemple :** `SELECT Titre FROM FILM-Bis WHERE Durée>=2 ORDER BY Durée`.

Jointures Multi-relations

Pour combiner des informations de plusieurs relations, on utilise la jointure en SQL.

Syntaxe de base pour la jointure :

```
SELECT <attributs des relations R et S>
FROM R, S
WHERE R.attribut_commun = S.attribut_commun
```

Équivalent algébrique (jointure naturelle) : $[R] \bowtie [S]$.

Exemple : Les cinémas qui projettent un film dans lequel joue M.F. Pisier.

```
SELECT DISTINCT Nom-Cine, FILM.Titre, Horaire
FROM FILM, PROG
WHERE FILM.titre = PROG.titre
  AND Acteur = 'M-F. Pisier'
```

Jointure Externe

SQL propose différentes formes de jointure externe :

- `NATURAL JOIN` : Jointure algébrique.
- `CROSS JOIN` : Produit cartésien.
- `JOIN ... ON` : Jointure avec condition explicite.
- `OUTER JOIN` : Jointure externe complète (gauche, droite, complète).
- `LEFT OUTER JOIN` : Jointure externe gauche.
- `RIGHT OUTER JOIN` : Jointure externe droite.

La jointure externe permet de conserver tous les tuples d'une ou des deux relations, même s'il n'y a pas de correspondance dans l'autre relation, en complétant avec des valeurs nulles.

Sous-Requêtes

Une sous-requête est une requête imbriquée dans une autre requête.

Sous-requêtes scalaires : Retournent une seule valeur. Utilisables dans les conditions avec des opérateurs de comparaison. **Exemple : Les acteurs du premier film joué par M-F. Pisier.**

```
SELECT Acteur FROM FILM
WHERE Titre = (SELECT Titre FROM FILM-DEB
               WHERE Acteur = 'M-F. Pisier')
```

Sous-requêtes retournant un ensemble de valeurs : Utilisables avec les opérateurs `IN`, `EXISTS`, `ALL`, `ANY`.

- `IN` : Teste si une valeur appartient à l'ensemble résultant de la sous-requête. **Exemple : Les titres des films dont un des metteurs en scène est acteur.**

  ```
  SELECT Titre FROM FILM
  WHERE MeS IN (SELECT Acteur AS MeS FROM FILM)
  ```

- `EXISTS` : Teste si la sous-requête retourne au moins un tuple. **Exemple : Les films dirigés par au moins deux metteurs en scène.**

  ```
  SELECT F1.Titre FROM FILM F1
  WHERE EXISTS (SELECT F2.MeS FROM FILM F2
                WHERE F1.Titre = F2.titre
                  AND NOT F1.MeS = F2.MeS)
  ```

- `ALL`, `ANY` : Comparaisons avec tous ou au moins un des éléments de l'ensemble retourné par la sous-requête. **Exemple avec `ALL` : 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')
  ```

Exemple avec ANY : Le téléphone des cinémas qui proposent une programmation après 23h.

```
SELECT Telephone FROM CINE AS C1
WHERE 23 < ANY (SELECT Horaire FROM PROG
                WHERE C1.Nom-Cine = PROG.Nom-Cine)
```

Attention : Optimisation SQL : Évitez l'usage excessif de sous-requêtes pour des raisons de performance.

Agrégats

Les fonctions d'agrégat permettent de calculer des valeurs synthétiques sur des groupes de tuples :

- `SUM()` : Somme.
- `AVG()` : Moyenne.
- `MIN()` : Minimum.
- `MAX()` : Maximum.
- `COUNT()` : Cardinalité (nombre de tuples).

Exemple : Nombre de films dirigés par Bergman. `SELECT COUNT (DISTINCT Titre) FROM FILM WHERE MeS = Bergman`.

Groupement (GROUP BY)

La clause `GROUP BY` permet de regrouper les tuples ayant les mêmes valeurs pour un ou plusieurs attributs, et d'appliquer des fonctions d'agrégat à chaque groupe. **Exemple : Nombre d'acteurs par film.** `SELECT Titre, COUNT (Acteur) FROM FILM GROUP BY Titre`.

Clause HAVING

La clause `HAVING` permet de filtrer les groupes résultant de la clause `GROUP BY` en fonction d'une condition sur les valeurs agrégées. **Exemple : Les films et le nombre d'acteurs de ces films à condition qu'il y ait plus de 3 acteurs.**

```
SELECT Titre, COUNT (DISTINCT Acteur) FROM FILM
GROUP BY Titre
HAVING COUNT(*) >= 3
```

Mises à Jour (INSERT, UPDATE, DELETE)

- `INSERT INTO` : Pour insérer de nouveaux tuples dans une relation. **Exemple :** `INSERT INTO FILM (Titre) VALUES ('Tche')`.
- `UPDATE` : Pour modifier des tuples existants. **Exemple :** `UPDATE FILM SET Téléphone = '05 56 44 11 87' WHERE Adresse = '9, rue Montesquieu'`.
- `DELETE FROM` : Pour supprimer des tuples. **Exemple :** `DELETE FROM R WHERE <condition>`.

Vues

Une vue est une table virtuelle, définie par une requête SQL, qui n'est pas matérialisée (les données ne sont pas stockées physiquement). **Création d'une vue :** `CREATE VIEW Nom_vue AS SELECT ... FROM ... WHERE ...`. **Exemple : Vue des cinémas parisiens.**

```
CREATE VIEW Cine-paris AS
SELECT * FROM CINE
WHERE Adresse LIKE '%Paris%'
```

Déclencheurs (Triggers)

Un déclencheur (trigger) est une action automatiquement exécutée en réponse à un événement sur la base de données (insertion, suppression, modification). Un trigger est défini par une règle ECA : Événement / Condition / Action. **Exemple de création de trigger :**

```
CREATE TRIGGER Prog-trigger
AFTER INSERT ON Prog
FOR EACH ROW
WHEN (new.Titre NOT IN (SELECT Titre FROM Film))
BEGIN
  INSERT INTO Film (Titre) VALUES (:new.Titre);
END;
```

5.3 Conclusion

Ce manuel a présenté une introduction à l'algèbre relationnelle et au langage SQL, en mettant l'accent sur les concepts fondamentaux et les opérations de manipulation de données. La compréhension de ces langages est essentielle pour interagir efficacement avec les bases de données relationnelles et extraire des informations pertinentes. L'optimisation des requêtes SQL, notamment en évitant les sous-requêtes complexes lorsque cela est possible, est un aspect important pour garantir la performance des applications bases de données.