CHAPTER 1

1.1 Introduction et Motivation

1.1.1 Omniprésence des bases de données

- Les données sont **omniprésentes** dans le monde moderne.
- Exemples d'omniprésence des données:
  - **Banque - Finance - Assurance**: *Banking system.* Gestion des comptes, transactions, prêts, assurances.
  - **Système d'Information Entreprise**: *Information System.* Gestion du personnel, clients, stocks.
  - **Gestion de réservations**: *Airline reservation system.* Transports, Hotels, Spectacles. Gestion des disponibilités et réservations.
  - **Commerce électronique**: *e-commerce.* Culture, Agro-alimentaire, Bricolage. Gestion des catalogues, commandes, paiements.

1.1.2 Motivation: Le Déluge de Données

- **Big Data**, **Data Deluge**, **Data Scientist**. Volume, variété et vitesse croissantes des données. Besoin d'experts pour extraire de la valeur.
- L'humanité produit **2,5 quintillons** (10^{30} bytes/j). Quantité phénoménale de données générées quotidiennement.
- **Data Never Sleeps 1.0 VS. 10.0**. Infographies illustrant l'augmentation exponentielle des données générées chaque minute sur Internet.

1.1.3 Rôle central des SGBD

- Les **SGBD** sont au coeur de la gestion du déluge de données. Ils permettent d'organiser, stocker et exploiter efficacement les informations.
- Cette révolution a commencé dans les années **1960** ! Premiers concepts et systèmes de gestion de bases de données.
- **Plus de 50 ans d'évolution !** Développement de systèmes sophistiqués et performants.

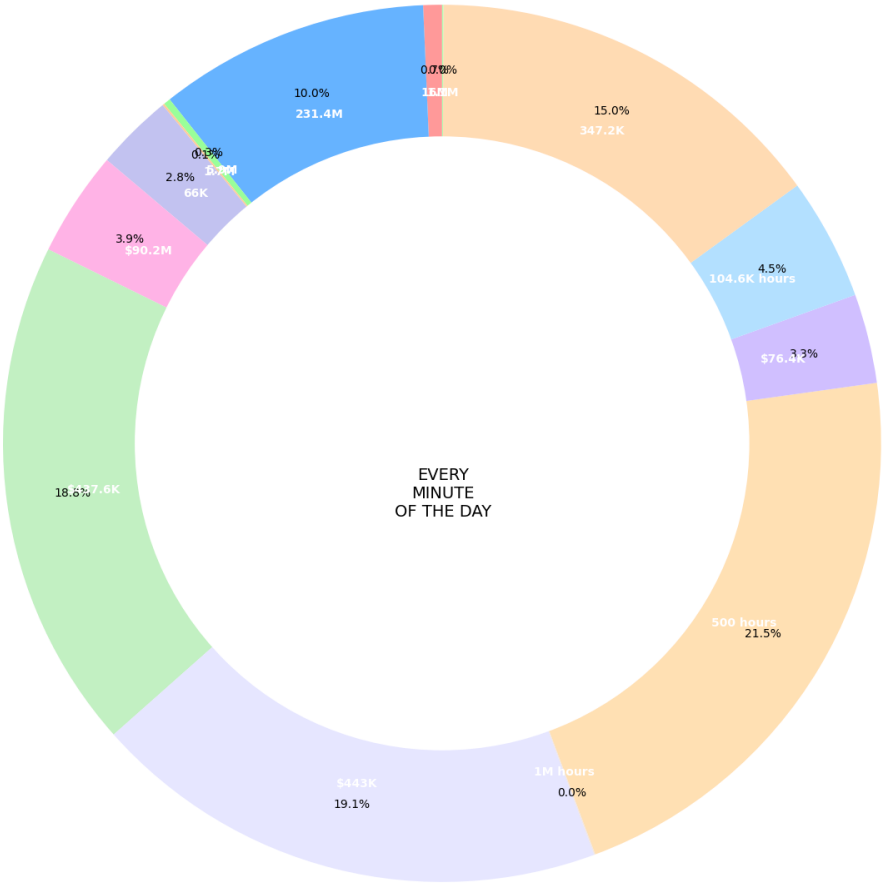

Figure 1.1: Data Never Sleeps

- **Des données, encore des données !** L'ère numérique est caractérisée par une croissance exponentielle des données. Les SGBD permettent de maîtriser et exploiter ce flux.

1.1.4 Que devez-vous connaître après ce cours ?

- Qu'est-ce qu'une **base de données** ? *Database*. Définition claire et concepts fondamentaux.
- **Notions élémentaires & vocabulaire**. Vocabulaire essentiel pour parler de bases de données (table, requête, etc.).
- **Fonctionnalités de base d'un SGBD**. Fonctionnalités essentielles : stockage, requête, mise à jour, intégrité, sécurité.
- Un peu d'**histoire**. Étapes clés de l'évolution des bases de données.
- Quelques **systèmes**. Exemples de SGBD populaires (Oracle, MySQL, PostgreSQL...).
- Quelques **métiers**. Métiers liés aux bases de données (concepteur, administrateur, développeur, data scientist).

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.* Mécanisme fiable et efficace pour stocker les données de manière persistante.
- **Répondre à des questions = requêtes**: *Query answering.* Permettre d'interroger les données stockées et d'extraire des informations spécifiques.
- **Mise à jour les données**: *Update.* Permettre de modifier, ajouter et supprimer des données de manière contrôlée et sécurisée.

1.2.2 Fonctionnalités Clés d'un SGBD

- Rôle d'un SGBD pour les applications : interface centrale pour la gestion des données.
- Il assure le **stockage**, la **mise à jour** et l'**accès** (requêtes) à de **grandes masses** de données *massive data*.
- En assurant :
  - **Efficacité**: *performance.* Traitement rapide des requêtes et mises à jour.
  - **Persistance**: *persistency.* Conservation durable des données.
  - **Fiabilité**: *reliability.* Résistance aux pannes et reprise après incident.
  - **Simplicité d'utilisation**: *convenience.* Interfaces conviviales et outils facilitant la gestion.
  - **Sécurité**: *safety.* Protection des données contre les accès non autorisés.
  - **Multi-utilisateur**: *multi-user.* Gestion des accès concurrents.

1.2.3 Gestion de grandes masses de données

- **grandes masses** de données *massive data.* Capacité à gérer de très grands volumes de données.
- Gigabytes (10^9 octets) : relativement simple à gérer.
- Terabytes (10^{12} octets) / jour, Petabytes (10^{15} octets) : défis considérables.
- $\rightsquigarrow$ données stockées en mémoire secondaire (disques).
- $\rightsquigarrow$ gestion des disques et buffer pour optimiser l'accès.
- Quintillions (10^{30} octets) *very massive data* : défis futurs.
- $\rightsquigarrow$ besoin de nouveaux supports de stockage et mécanismes d'accès.

1.2.4 Efficacité

- **efficacité** *performance.* Traitement rapide des requêtes et mises à jour.
- Le système doit être capable de gérer plusieurs milliers de requêtes par seconde, même complexes.
- $\rightsquigarrow$ optimisation à tous les niveaux (requêtes, stockage, accès).

1.2.5 Persistance

- **persistance** *persistency*. Conservation durable des données.
- Modifications des données prises en compte lors des consultations ultérieures.
- $\rightsquigarrow$ Modifications enregistrées sur disque et persistent au-delà de l'exécution du programme.

1.2.6 Fiabilité

- **fiabilité** (pannes) *reliability*. Fonctionnement correct et intégrité des données même en cas de pannes.
- Résistance aux pannes: matérielles, logicielles, électriques...
- $\rightsquigarrow$ reprise sur panne: *failure recovery*. Redémarrage et restauration cohérente après panne.
- $\rightsquigarrow$ journal: *logs, write ahead, checkpoints*. Utilisation d'un journal pour la reprise sur panne.

1.2.7 Sécurité

- **sécurité** *safety*. Protection contre les accès non autorisés et les modifications malveillantes.
- Données critiques pour les entreprises, clients, états.
- $\rightsquigarrow$ protéger les données d'accès malveillants.
- $\rightsquigarrow$ contrôle d'accès, vues: *access grant*. Mécanismes de contrôle d'accès.
- La protection de la vie privée (RGPD) est un autre sujet *privacy*. Conformité aux réglementations sur la protection des données.

1.2.8 Multi-utilisateur

- **multi-utilisateur** *multiuser*. Gestion des accès simultanés par plusieurs utilisateurs et applications.
- Accès partagé par de nombreux utilisateurs et programmes.
- Nécessité de contrôler l'accès (écriture/lecture) pour assurer la cohérence.
- $\rightsquigarrow$ contrôle de la concurrence: *concurrency control*. Mécanismes pour gérer les transactions concurrentes.

Example 1.2. M. et Mme Dupont ont 100 euros sur leur compte commun. Retraits simultanés de 50 et 20 euros.

- Exécution 1 (M. Dupont): Lecture, débit, écriture.
- Exécution 2 (Mme Dupont): Lecture, débit, écriture.
- Exécution concurrente : risque d'incohérence sans contrôle de concurrence.

1.2.9 Fiabilité - qualité

- **fiabilité - qualité** *data quality*. Qualité des données : exactitude, validité, cohérence, complétude.
- données $\neq$ informations. Données brutes vs. données interprétées.
- données + propriétés = informations. Nécessité de propriétés pour la qualité des données.
- $\rightsquigarrow$ contraintes d'intégrité. Règles pour garantir la validité et la cohérence des données.
- $\rightsquigarrow$ vérification automatique de l'intégrité. Rejet des opérations violant les contraintes.

Example 1.3.
- Le César de la meilleure actrice est attribué à une femme (contrainte d'intégrité).
- Un seul César de la meilleure actrice par an (contrainte d'intégrité).

1.2.10 Simplicité d'utilisation

- **simplicité d'utilisation** *convenience*. Facilité d'utilisation pour développeurs, administrateurs et utilisateurs finaux.
- Développement d'applications / Écriture de requêtes simplifiés.
- Maintenance et optimisation facilitées.
- $\rightsquigarrow$ Langage de haut niveau & déclaratif: **SQL**. Langage standard pour interagir avec les SGBD relationnels.

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

- $\rightsquigarrow$ Schéma & Instance. Schéma (structure) et Instance (données).
- $\rightsquigarrow$ 3 niveaux d'abstraction. Niveaux externe, logique et physique.
- $\rightsquigarrow$ Principe d'indépendance physique. Modification du niveau physique sans impacter les applications.

1.3 Principes des SGBD

1.3.1 Schéma versus instance

- **schéma** versus **instance**. Structure vs. contenu des données.
- **schéma** = description de la structuration des données. Définition des types, relations et contraintes.
- le schéma est une donnée. Stocké, modifiable et interrogeable.
- $\rightsquigarrow$ stocké + modifiable + interrogeable.
- **instance** (du schéma) = données organisées selon le schéma.

schéma = récipient instance = contenu

1.3.2 Les Trois Niveaux d'Abstraction

- **Les Trois Niveaux d'Abstraction**. Architecture en trois niveaux pour gérer la complexité et assurer l'indépendance des données.
  - **Niveau externe (vues)**: Vues personnalisées pour utilisateurs et applications.
  - **Niveau logique**: Description globale de la structure des données (modèle relationnel).
  - **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.

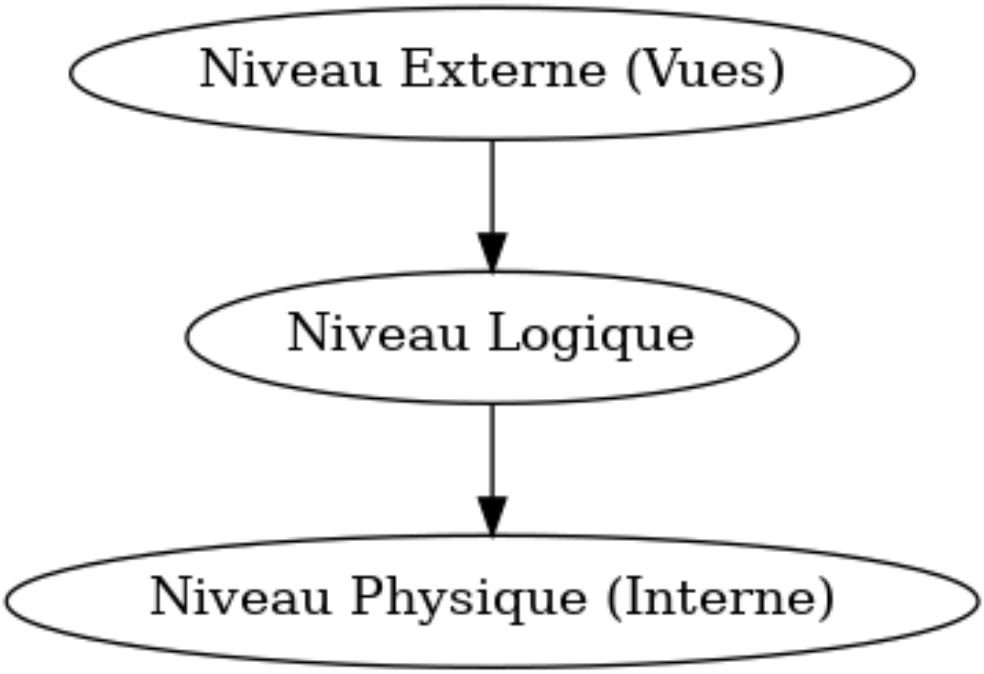

Figure 1.2: Architecture 3 niveaux

1.3.3 Indépendance physique

- **Indépendance physique**. Modification du niveau physique sans affecter les niveaux supérieurs.
- $\frac{\textbf{Programmes d'application} \mid \textbf{Invariants}}{\textbf{Schéma Physique} \mid \text{Modification}}$ Programmes et schéma logique invariants malgré modification du schéma physique.

Example 1.4. Exemple: Base de données des palmarès de festivals de cinéma.

- Initialement = 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. Augmentation du volume de données.
- $\rightsquigarrow$ schéma physique = fichier indexé (Arbre B+) *index file*. Amélioration des performances avec index.

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

> Requête SQL inchangée, applications fonctionnent sans modification grâce à l'indépendance physique.

1.4 Histoire des SGBD

1.4.1 Un peu d'histoire des SGBDs

- **60-70 SGBD Réseau**. Premières générations. Relations complexes, mais complexes à mettre en œuvre et manque d'indépendance.
  - Exemples: IDS, APL, DMS 1100, ADABAS.
- **60-70 SGBD Hiérarchique**. Organisation arborescente. Plus simples, moins flexibles.
  - Exemple: ISM (IBM).
  - Gestion de pointeurs entre enregistrements.
  - Problème : pas d'**indépendance physique**.
- **70- SGBD RELATIONNEL** *Relational DBMS*. Modèle révolutionnaire d'Edgar F. Codd.
  - Modèle logique simple : données = table.
  - Formalisation mathématique : théorie des ensembles ou logique.
  - Langage normalisé = **SQL**.
- **SGBD relationnel non normalisé**. Extensions pour données plus complexes.
  - Tables dans des tables.
- **SGBD orienté objet (OO)**. Concepts de l'orienté objet intégrés aux bases de données.
  - Modèle inspiré des langages objets.
  - Identité d'objet, héritage, méthodes.
- **Modèle semi-structuré et XML**. Gestion de données à structure souple (XML).
  - Information incomplète, schéma souple.
  - Motivé par l'échange de données par les services web.
- **Modèle de graphes et RDF**. Représentation des données sous forme de graphes.
  - Information incomplète, raisonnement possible, schéma souple et interopérable.
  - Motivé par la publication et le partage de données sur le web (Web sémantique).
- **NoSQL**. Catégorie de SGBD s'éloignant du modèle relationnel.
  - Adéquation aux besoins actuels (scalabilité, performance, flexibilité).
  - "SQL" = SGBDs relationnels classiques.
  - NoSQL= Ne pas utiliser les SGBDs relationnels classiques (seulement si nécessaire).
  - NoSQL $\neq$ Ne pas utiliser le langage SQL (pas toujours).
  - NoSQL = Ne pas utiliser **seulement** les SGBDs relationnels.

1.4.2 Quelques systèmes relationnels

- **Systèmes historiques**.
  - System R, IBM-San José. Prototype du SGBD relationnel.
  - Ingres, Postgres. Autre SGBD historique, PostgreSQL descendant.
- **Systèmes propriétaires**.
  - Oracle Database. SGBD majeur pour grandes entreprises.
  - Microsoft SQL Server. Populaire dans l'environnement Windows.
  - DB2, MaxDB, 4D, dBase, Informix, Sybase ...
- **Systèmes libres**.
  - PostgreSQL. SGBD open source puissant et riche en fonctionnalités.
  - MariaDB (MySQL). Fork open source de MySQL, populaire pour le web.
  - Firebird, Ingres, HSQLDB, Derby. Alternatives open source.

1.5 Les différents acteurs

1.5.1 Acteurs autour des SGBD

- **Développeurs de SGBD**. Conçoivent et développent les SGBD.
  - Concevoir et développer les systèmes de gestion de bases de données.
- **Concepteur de bases de données**. Conçoivent le schéma logique.
  - Concevoir le schéma logique de la base de données.
- **Développeur d'applications base de données**. Développent les applications utilisant les BD.
  - Développer les programmes (requêtes, mises à jour) pour les applications.
- **Administrateur de la base de données**. Gèrent et maintiennent les SGBD en production (DBA).
  - Paramétrer, maintenance, tunning, sécurité.

1.6 Objectifs du Cours

1.6.1 Objectif principal

- Un SGBD relationnel est un **système complexe** ! Nombreuses fonctionnalités sophistiquées.
- **facile à utiliser - difficile à bien utiliser !** Facile à utiliser en surface, mais difficile à maîtriser pleinement (performance, sécurité, etc.).

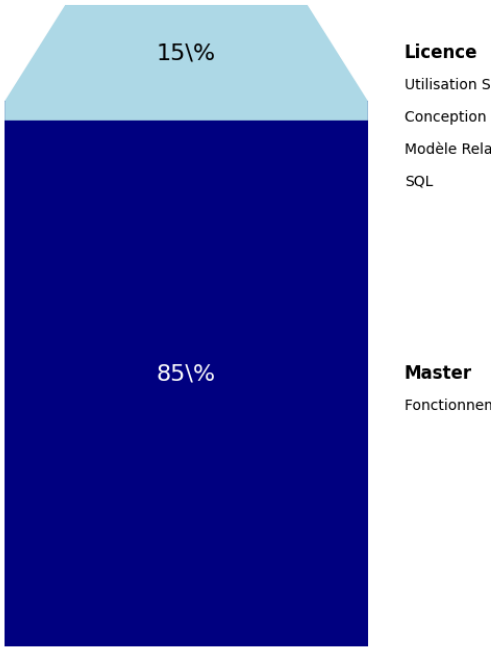

Figure 1.3: Iceberg des SGBD

1.7 Plan du cours

1.7.1 Bases de Données - Plan du cours

- **Conception d'une base de données**. Définition du schéma logique.
  - Processus de conception. Étapes de la conception.
  - Le modèle Entité-Association. Modélisation conceptuelle.
- **Modèle de données**. Modèle relationnel.
  - Le Modèle Relationnel. Concepts clés (tables, clés, relations).
  - Du Modèle E-A au modèle relationnel. Conversion modèle conceptuel → modèle logique.
  - SQL : Langage de définition (LDD). Création et modification du schéma.
- **Interrogation d'une base de données**. Extraction d'informations.
  - Algèbre relationnelle. Opérations fondamentales de requête.
  - SQL : Langage de manipulation (LMD). Requêtes et manipulation des données.
- **Dépendances & Conception de schéma**. Qualité des schémas relationnels.
  - Dépendances fonctionnelles. Contraintes sémantiques et redondances.
  - Schéma sous forme normale. Normalisation pour améliorer la qualité.

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

2.1 Introduction à la conception de bases de données

La conception de bases de données est une étape cruciale dans le développement d'applications informatiques, en particulier celles qui manipulent de grandes quantités de données persistantes. Elle consiste à structurer et organiser les données de manière efficace pour permettre un stockage, une récupération et une manipulation aisés. Une bonne conception est essentielle pour garantir la performance, la fiabilité et la maintenabilité d'une base de données.

2.1.1 Les étapes de la conception d'une base de données

Le processus de conception d'une base de données se déroule généralement en trois étapes principales :

1. **Modélisation conceptuelle** : Cette première étape vise à comprendre et à analyser le domaine d'application et les besoins des utilisateurs. Elle aboutit à la création d'un modèle conceptuel, une représentation abstraite des données et de leurs relations, indépendante de toute technologie de base de données spécifique. Le modèle Entité-Association (E/A) est un outil couramment utilisé pour cette étape.

2. **Modélisation logique (ou schéma logique)** : À partir du modèle conceptuel, on élabore un schéma logique. Ce schéma décrit la structure des données de manière plus formelle, en tenant compte du type de système de gestion de base de données (SGBD) qui sera utilisé (par exemple, relationnel, objet, semi-structuré). La transformation du modèle conceptuel en schéma logique implique des choix de représentation des données et des relations dans le modèle logique cible.

3. **Modélisation physique (ou schéma physique)** : La dernière étape consiste à définir le schéma physique, qui décrit comment les données seront physiquement stockées sur les supports de stockage. Cela inclut des aspects tels que l'organisation des fichiers, les index, le partitionnement des tables, et d'autres paramètres d'optimisation des performances, en fonction des spécificités du SGBD choisi.

2.1.2 Objectif de la modélisation conceptuelle

La modélisation conceptuelle a pour objectif principal de saisir et de représenter les concepts fondamentaux du domaine d'application. Il s'agit de :

- **Isoler les concepts fondamentaux** : Identifier les objets, les idées et les informations qui seront représentés dans la base de données. Quelles données devons-nous stocker ? Quels sont les concepts élémentaires du monde réel que nous devons modéliser ? Quels sont les sous-concepts pertinents ?

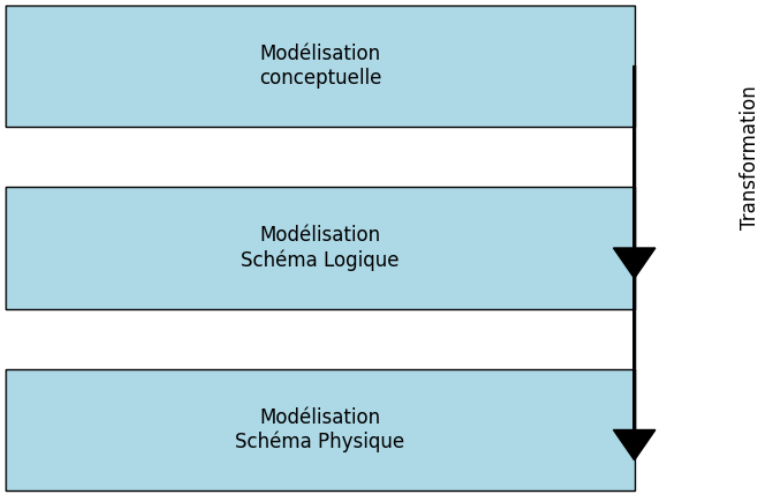

Figure 2.1: Étapes de la conception d'une base de données

- **Faciliter la visualisation du système** : Utiliser des diagrammes avec des notations simples et précises pour représenter le modèle conceptuel. L'objectif est de permettre une compréhension visuelle du système, au-delà de la simple description textuelle.

En résumé, la modélisation conceptuelle est une étape cruciale pour poser les bases d'une base de données bien conçue, en se concentrant sur la compréhension du domaine et la représentation claire des concepts importants.

2.2 Modèle Entité-Association (E/A)

Le modèle Entité-Association (E/A) est un modèle de données conceptuel de haut niveau utilisé pour la conception de bases de données. Il a été introduit par Peter Chen en 1976.

> *Chen, P., The Entity-Relationship Model – Towards a Unified View of Data, ACM TODS, Vol. 1, No. 1, 1976.*

Le modèle E/A offre :

- Un ensemble de **concepts** pour modéliser les données d'une application.
- Un ensemble de **symboles graphiques** pour représenter ces concepts sous forme de diagrammes, facilitant la communication et la compréhension du modèle.

2.2.1 Les trois concepts centraux du modèle E/A

Le modèle E/A repose sur trois concepts fondamentaux : l'entité, l'attribut et l'association.

Entité (ou type entité)

Definition 2.1 (Entité)**.** Une **entité** représente un type d'objet, de personne, de lieu, d'événement ou de concept qui existe dans le monde réel et qui est pertinent pour l'application à modéliser. Une entité correspond à un ensemble d'objets (appelés *instances* d'entité) qui partagent des propriétés communes.

Graphiquement, une entité est représentée par un rectangle. Le nom de l'entité, généralement un nom commun au singulier, est écrit à l'intérieur du rectangle.

Example 2.2. Exemples d'entités : *Acteur*, *Film*, *Tournage*, *Voyage*, *Client*, *Train*.

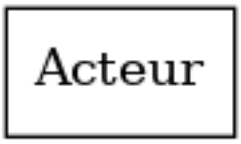

Figure 2.2: Entité Acteur

Figure 2.3: Entité Film

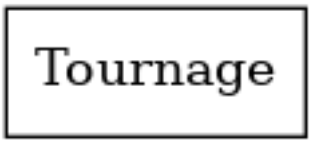

Figure 2.4: Entité Tournage

Attribut

Definition 2.3 (Attribut)**.** Un **attribut** est une propriété ou une caractéristique descriptive d'une entité. Il permet de préciser et de qualifier les instances d'une entité. Chaque attribut est associé à un *domaine de valeurs* possibles.

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

Example 2.4. Pour l'entité *Acteur*, on peut définir les attributs suivants : *Prénom*, *Nom*, *Date de naissance* (Ddn). Pour l'entité *Film*, on peut avoir les attributs *Titre*, *Année de sortie*, *Metteur en scène* (MeS).

Les attributs peuvent être :

- **Simples** (ou atomiques) : Ils ne sont pas divisibles en sous-parties significatives (ex : *Nom*).
- **Complexes** (ou composés) : Ils sont formés de plusieurs composants (ex : *Date de naissance* peut être composée de *jour*, *mois*, *an*).

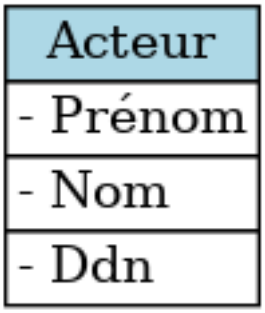

Figure 2.5: Entité Acteur et ses attributs

Association (ou type d'association)

Definition 2.5 (Association)**.** Une **association** représente une relation, un lien significatif entre deux ou plusieurs entités. Elle décrit comment les instances d'entités sont connectées ou interagissent entre elles. Une association peut également avoir ses propres attributs.

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

Example 2.6. L'association *joue* peut relier l'entité *Acteur* à l'entité *Film* pour exprimer le fait qu'un acteur joue dans un film. L'association *dirige* peut relier *Metteur en scène* (considérée ici comme une entité) et *Film*. L'association *est_remake* peut relier deux entités *Film* pour indiquer qu'un film est le remake d'un autre.

Figure 2.6: Association *joue* entre Acteur et Film

Remark 2.7. Une association peut avoir des **rôles** pour préciser la fonction de chaque entité participante dans la relation. Par exemple, dans l'association *est_remake* entre deux *Film*, on peut préciser le rôle *original* et le rôle *remake*. Une association peut également avoir des **attributs** propres, décrivant la relation elle-même (ex : le *rôle* joué par un acteur dans un film peut être un attribut de l'association *joue*).

2.3 Contraintes de Cardinalité

Les contraintes de cardinalité permettent de préciser la nature des liens entre les instances des entités participant à une association. Elles définissent le nombre minimum et maximum d'instances d'une entité qui peuvent être liées à une instance d'une autre entité via une association.

2.3.1 Cardinalité minimale et maximale

Pour une association entre deux entités A et B, on définit :

- **Cardinalité minimale** : Pour chaque instance de l'entité A, quel est le nombre minimum d'instances de l'entité B avec lesquelles elle doit être associée ? Les valeurs possibles sont 0 (l'instance de A peut ne pas être liée à B) ou 1 (l'instance de A doit être liée à au moins une instance de B).
- **Cardinalité maximale** : Pour chaque instance de l'entité A, quel est le nombre maximum d'instances de l'entité B avec lesquelles elle peut être associée ? Les valeurs possibles sont 1 (l'instance de A peut être liée à au plus une instance de B) ou m (ou n, *), signifiant "plusieurs" (l'instance de A peut être liée à un nombre quelconque d'instances de B).

La cardinalité est généralement notée sous la forme (min, max) sur chaque trait reliant l'association aux entités.

Example 2.8. Considérons l'association *joue* entre *Acteur* et *Film*.

- Un acteur peut ne jamais avoir joué dans un film (cardinalité minimale côté Film = 0).
- Un acteur peut jouer dans plusieurs films (cardinalité maximale côté Film = m).

- Un film peut être tourné sans acteur (cardinalité minimale côté Acteur = 0, pour les films d'animation ou documentaires).
- Un film met généralement en scène plusieurs acteurs (cardinalité maximale côté Acteur = m).

La cardinalité de l'association *joue* serait donc (0, m) côté *Acteur* et (0, m) côté *Film*.

Figure 2.7: Cardinalité de l'association *joue*

2.3.2 Différents cas d'associations

En fonction des cardinalités maximales, on distingue différents types d'associations binaires :

- **Association 1:1** (un-à-un) : Chaque instance de l'entité A est liée à au plus une instance de l'entité B, et vice versa. Cardinalités maximales (1, 1).
- **Association 1:m** (un-à-plusieurs) : Chaque instance de l'entité A est liée à plusieurs instances de l'entité B, mais chaque instance de l'entité B est liée à au plus une instance de l'entité A. Cardinalités maximales (1, m).
- **Association m:n** (plusieurs-à-plusieurs) : Chaque instance de l'entité A peut être liée à plusieurs instances de l'entité B, et chaque instance de l'entité B peut être liée à plusieurs instances de l'entité A. Cardinalités maximales (m, m).

2.4 Clés (ou Identifiants)

Definition 2.9 (Clé)**.** 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. Pour deux instances différentes d'une entité, les valeurs de la clé doivent être différentes.

2.4.1 Clé simple et clé composée

- **Clé simple** : La clé est constituée d'un seul attribut. Pour être une clé, cet attribut doit avoir des valeurs uniques pour chaque instance de l'entité.
- **Clé composée** : La clé est constituée de plusieurs attributs. La combinaison des valeurs de ces attributs doit être unique pour chaque instance de l'entité.

Example 2.10. Pour l'entité *Acteur*, un numéro d'identification unique (*Num_A*) pourrait servir de clé simple. Si un tel numéro n'existe pas, on pourrait considérer la combinaison des attributs (*Prénom*, *Nom*, *Ddn*) comme une clé composée, à condition que cette combinaison soit unique pour chaque acteur. Pour l'entité *Film*, le *Titre* pourrait être une clé si les titres de films sont uniques. Sinon, on pourrait utiliser la combinaison (*Titre*, *Année de sortie*) comme clé composée.

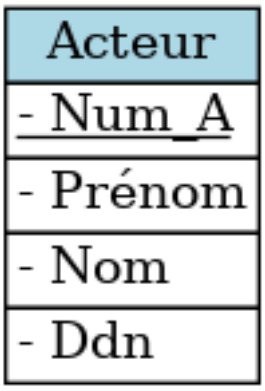

Figure 2.8: Clé de l'entité Acteur (*Num_A* souligné)

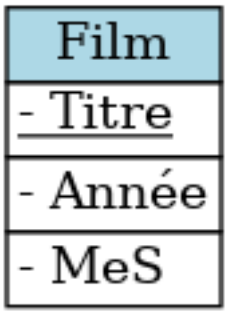

Figure 2.9: Clé de l'entité Film (*Titre* souligné)

Remark 2.11. Si plusieurs attributs peuvent servir de clé, on parle de **clés candidates**. On choisit généralement une de ces clés comme **clé primaire** pour identifier les instances de l'entité. Les clés primaires sont souvent soulignées dans les diagrammes E/A.

2.5 Entités Faibles

Definition 2.12 (Entité Faible)**.** Une **entité faible** est une entité dont l'existence dépend d'une autre entité, appelée **entité forte** (ou entité parent). Une entité faible ne peut pas être identifiée de manière unique par ses propres attributs ; son identifiant dépend de la clé de l'entité forte à laquelle elle est liée.

L'entité faible est souvent utilisée pour représenter des informations qui sont des compléments ou des détails relatifs à une entité forte.

Example 2.13. Considérons les entités *Cinéma* et *Salle*. Une *Salle* de cinéma est identifiée relativement à un *Cinéma*. On la numérote "localement" dans chaque cinéma (Salle 1, Salle 2, ... du cinéma X). L'entité *Salle* est une entité faible, dépendante de l'entité forte *Cinéma*. La clé de *Salle* est composée de la clé de *Cinéma* (par exemple, un identifiant de cinéma *Id_C*) et d'un numéro de salle (*Num_S*) unique localement dans le cinéma.

2.5.1 Association Identifiante

La relation entre une entité faible et son entité forte est souvent représentée par une **association identifiante**. Dans un diagramme E/A, les entités faibles et les associations identifiantes sont parfois représentées avec des traits doubles ou des losanges doubles pour les associations. La cardinalité de l'association identifiante est généralement (1,1) côté entité faible, indiquant qu'une instance d'entité faible est toujours liée à exactement une instance de l'entité forte.

Figure 2.10: Entité faible *Salle* et association identifiante *est-dans*

2.6 Héritage (is-a)

Le concept d'héritage (ou spécialisation/généralisation) permet de représenter des hiérarchies de types d'entités. Il est utile lorsque certaines entités partagent des propriétés communes mais ont aussi des caractéristiques spécifiques.

2.6.1 Sur-entité et Sous-entité

On définit une **sur-entité** (ou superclasse) qui représente les caractéristiques générales communes à un ensemble d'entités. Des **sous-entités** (ou sous-classes) sont ensuite définies, héritant des propriétés de la sur-entité et ajoutant des attributs ou des associations spécifiques. La relation d'héritage est souvent appelée relation "is-a" (est-un).

Example 2.14. Considérons l'entité *Film*. On peut distinguer des sous-types de films, comme les *Films d'animation* et les *Documentaires*. *Film* est la sur-entité, et *Film d'animation* et *Documentaire* sont des sous-entités. Un *Film d'animation est un Film*, et un *Documentaire est un Film*. Les sous-entités héritent des attributs de *Film* (Titre, Année, MeS) et peuvent avoir leurs propres attributs spécifiques (par exemple, catégorie d'animation pour *Film d'animation*, sujet pour *Documentaire*).

Dans un diagramme E/A, l'héritage est souvent représenté par un triangle pointant vers la sur-entité, relié aux sous-entités. La cardinalité de la relation d'héritage est généralement (1,1), indiquant qu'une instance de sous-entité correspond à exactement une instance de sur-entité.

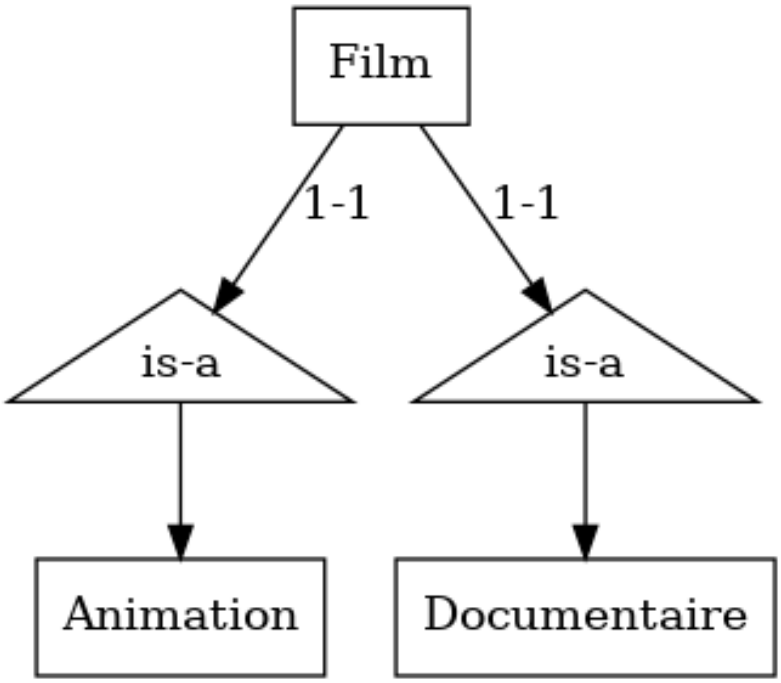

Figure 2.11: Héritage dans le modèle E/A : Sur-entité *Film* et sous-entités *Animation* et *Documentaire*

2.7 Principes de Conception

Une bonne conception de modèle E/A repose sur quelques principes fondamentaux :

- **Se concentrer sur les besoins de l'application et des utilisateurs** : Le modèle doit refléter fidèlement les besoins fonctionnels et les données manipulées par l'application. Les entités et attributs doivent être pertinents et utiles pour l'application. Par exemple, pour une application de gestion de cinéma, l'attribut *genre* d'un film peut être important, tandis que pour une application de gestion de stocks, il le sera moins.

- **Faire simple** : La simplicité est essentielle. Un modèle trop complexe sera difficile à comprendre, à maintenir et à implémenter. Il faut viser un modèle clair, concis et facile à communiquer.

- **Éviter les redondances** : La redondance, c'est-à-dire la répétition de la même information sous différentes formes, doit être évitée. Elle peut entraîner des problèmes d'efficacité (espace de stockage gaspillé, temps de traitement plus longs) et de qualité des données (données inconsistantes si les copies ne sont pas mises à jour de manière synchronisée). Par exemple, au lieu de stocker l'adresse d'un acteur à la fois dans l'entité *Acteur* et dans l'entité *Film* (si on voulait associer une adresse au film), il serait préférable de stocker l'adresse uniquement dans *Acteur* si c'est l'adresse de l'acteur qui est pertinente.

- **Choisir judicieusement les entités, associations et attributs** : Il est important de bien distinguer ce qui doit être représenté comme une entité, un attribut ou une association. Par exemple, faut-il représenter le metteur en scène comme un attribut de *Film* ou comme une entité à part entière ? Le choix dépend des besoins de l'application. Si on a besoin de stocker des informations spécifiques sur les metteurs en scène (nom, biographie, etc.) ou de les relier à d'autres entités, il est préférable de les modéliser comme une entité. De même, il faut choisir si un concept est un attribut d'une entité ou une entité liée par une association. Par exemple, le *rôle* joué par un acteur dans un film peut être un attribut de l'association *joue* si on a juste besoin du nom du rôle. Si on veut décrire le rôle plus en détail (texte du rôle, costume, etc.), il pourrait être préférable de créer une entité *Rôle* liée à l'association *joue*.

2.8 Conclusion

Le modèle Entité-Association (E/A) est un outil puissant et largement utilisé pour la modélisation conceptuelle de bases de données. Il permet de représenter de manière claire et intuitive les données et leurs relations, facilitant la communication entre les différents acteurs d'un projet de développement de base de données (concepteurs, développeurs, utilisateurs finaux).

2.8.1 Autres approches pour la modélisation conceptuelle

Bien que le modèle E/A soit très répandu, d'autres approches existent pour la modélisation conceptuelle, notamment :

- **Diagramme UML (Unified Modeling Language)** : UML est un langage de modélisation plus généraliste, utilisé dans le génie logiciel. Les diagrammes de classes UML peuvent également être utilisés pour la modélisation conceptuelle de données, offrant des fonctionnalités similaires au modèle E/A, mais avec une syntaxe et des concepts différents.
- **Modèles sémantiques et Ontologies** : Ces approches, issues du domaine de la représentation des connaissances, visent à modéliser la signification des données et les relations sémantiques entre les concepts. Elles sont particulièrement adaptées aux applications nécessitant une représentation riche et structurée de la connaissance.

2.8.2 Outils graphiques utiles

De nombreux outils graphiques facilitent la création de diagrammes E/A et d'autres modèles conceptuels. Parmi les outils populaires, on peut citer :

- **Lucidchart**
- **Visual Paradigm**
- **dbdiagram.io**
- **Draw.io**

Ces outils offrent des interfaces conviviales pour dessiner les diagrammes, valider les modèles et souvent générer automatiquement des schémas logiques ou des scripts de création de bases de données.

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.