1 Introduction au Modèle Entité-Association

1.1 Qu'est-ce que le Modèle Entité-Association ?

Le Modèle Entité-Association (E/A) est un modèle de données conceptuel de haut niveau utilisé dans la conception de bases de données. Son objectif principal est de fournir une représentation graphique et intuitive de la structure des données d'une application. En se concentrant sur les entités importantes et les relations entre elles, le modèle E/A permet de :

- **Modéliser les données d'une application** : Identifier les concepts clés et leurs interrelations.
- **Faciliter la communication** : Offrir un langage commun entre les experts du domaine, les utilisateurs et les concepteurs de bases de données.
- **Servir de base à la conception logique** : Constituer un point de départ solide pour la création de schémas de bases de données relationnelles ou autres.

Comme son nom l'indique, le modèle E/A repose sur trois concepts principaux : les **entités**, les **attributs** qui les décrivent, et les **associations** (ou relations) qui les relient. Il utilise un ensemble de symboles graphiques pour représenter ces concepts et leurs interactions, rendant le modèle visuellement accessible et facile à comprendre.

1.2 Origines du Modèle E/A

Le modèle Entité-Association a été introduit par Peter Chen en 1976. Peter Chen était professeur à la Louisiana State University au moment de la publication de son article fondateur : "The Entity-Relationship Model – Towards a Unified View of Data". Cet article, paru dans ACM TODS, Vol. 1, No. 1, a posé les bases théoriques et pratiques du modèle E/A, qui est depuis devenu un standard dans le domaine de la conception de bases de données.

1.3 Importance du Modèle E/A

Le modèle E/A joue un rôle crucial dans la phase de conception des bases de données. Il permet de :

- **Isoler les concepts fondamentaux** : Déterminer quelles données doivent être représentées dans la base de données, découvrir les éléments essentiels du monde réel à modéliser et définir les sous-concepts pertinents.
- **Faciliter la visualisation du système** : Utiliser des diagrammes avec des notations simples et précises pour une compréhension visuelle et pas seulement intellectuelle du modèle.
- **Analyser et comprendre le domaine** : Permettre une interaction efficace avec les experts du domaine pour identifier les besoins "métier" et les données nécessaires aux traitements.
- **Identifier les données nécessaires** : Préciser les informations à stocker pour répondre aux exigences de l'application, tant pour les besoins actuels que futurs.
- **Coder le "monde réel"** : Transformer la complexité du monde réel et ses exceptions en un modèle informatique cohérent, en abstrayant et en simplifiant les informations pertinentes.
- **Maîtriser la complexité** : Gérer le volume d'informations à représenter de manière structurée et organisée.
- **Ne jamais perdre de vue les performances** : Concevoir un modèle qui, dès le départ, prend en compte les aspects de performance et d'efficacité de la future base de données.

2 Concepts Fondamentaux du Modèle E/A

Le modèle Entité-Association repose sur trois concepts centraux, qui permettent de structurer et de représenter les données : les entités, les attributs et les associations.

2.1 Entités

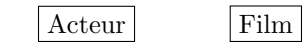
Definition 2.1 (Entité (ou Type Entité))**.** Une **entité** représente une catégorie d'objets, de personnes, de concepts ou d'événements du monde réel qui sont pertinents pour l'application et pour lesquels nous souhaitons stocker des informations. On parle également de **type entité** pour désigner cette catégorie.

Les entités sont les éléments centraux de notre modèle. Elles correspondent à des ensembles d'objets (ou instances) concrets ou abstraits. Chaque entité est spécifiée par un nom unique qui la distingue des autres entités.

Example 2.2. Dans le contexte d'une application de gestion de films et d'acteurs, nous pouvons identifier les entités suivantes :

- **Acteur** : Représente les acteurs et actrices.
- **Film** : Représente les films.
- **Tournage** : Pourrait représenter les tournages de films (si l'on souhaite stocker des informations spécifiques sur les tournages).

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



Entité Acteur Entité Film

2.2 Attributs

Definition 2.3 (Attribut (d'une entité))**.** 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 spécifié par :

- **Un nom** : Identifiant unique de l'attribut au sein d'une entité.
- **Un domaine de valeurs** : L'ensemble des valeurs possibles que peut prendre cet attribut.

Example 2.4. Pour l'entité **Acteur**, nous pouvons définir les attributs suivants :

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

Les attributs peuvent être de différents types :

- **Attributs simples** : Atomiques, indivisibles (ex: Nom).
- **Attributs complexes (ou composés)** : Structurés en plusieurs parties (ex: Ddn pourrait être décomposé en [jour, mois, an]).

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

2.3 Associations

Definition 2.5 (Association (ou Type d'Association))**.** Une **association** représente un lien, une relation ou une interaction entre deux ou plusieurs entités (souvent deux, on parle alors d'association binaire). Elle décrit une connexion significative entre les instances de ces entités. On parle également de **type d'association** pour désigner la catégorie de relation.

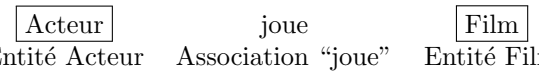
Les associations permettent de modéliser les interactions entre les entités du monde réel. Chaque association est spécifiée par :

- **Un nom** : Décrivant la nature de la relation (ex: "joue", "dirige", "est_remake").
- **Les entités associées** : Les entités qui participent à la relation.
- **Éventuellement des rôles** : Précisant la fonction de chaque entité dans l'association.
- **Éventuellement des attributs** : Décrivant la relation elle-même, plutôt que les entités liées.

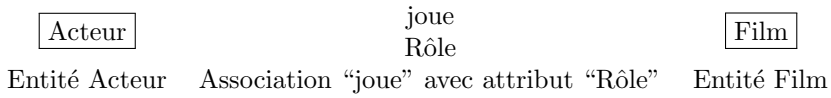
Example 2.6. Dans notre exemple, nous pouvons définir les associations suivantes :

- **Jouer** : Une association entre l'entité **Acteur** et l'entité **Film**. Un acteur "joue" dans un film.
- **Diriger** : Une association entre l'entité **Film** et une autre entité (par exemple, **MetteurEnScène**, si nous considérons le metteur en scène comme une entité à part entière). Un metteur en scène "dirige" un film.
- **Est_remake** : Une association entre deux instances de l'entité **Film**. Un film peut être le "remake" d'un autre film.

Graphiquement, une association est représentée par un losange, relié par des traits aux entités qu'elle associe. Le nom de l'association est inscrit dans le losange.



On peut également ajouter des **rôles** pour clarifier la signification de la relation, et des **attributs** à l'association elle-même. Par exemple, pour l'association "joue", on pourrait ajouter un attribut "Rôle" pour préciser le rôle joué par l'acteur dans le film.



3 Contraintes de Cardinalité

3.1 Définition et Importance

Les **contraintes de cardinalité** précisent le nombre d'instances d'une entité qui peuvent être liées à une instance d'une autre entité via une association. Elles apportent de la précision sur la nature des liens entre les entités associées et reflètent les règles du monde réel que nous modélisons.

La cardinalité est exprimée en termes de nombre minimal et maximal de liens sortant d'une instance d'entité.

3.2 Types de Cardinalités

On distingue principalement trois types de cardinalités binaires, en considérant une association entre une entité A et une entité B :

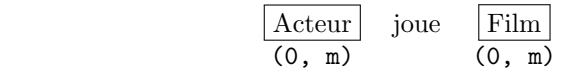
- **Un-à-un (1:1)** : Une instance de A est liée à au plus une instance de B, et réciproquement.
- **Un-à-plusieurs (1:m)** : Une instance de A est liée à zéro, une ou plusieurs instances de B, mais une instance de B est liée à au plus une instance de A.
- **Plusieurs-à-plusieurs (m:m)** : Une instance de A est liée à zéro, une ou plusieurs instances de B, et réciproquement.

On utilise les notations suivantes pour exprimer les cardinalités minimale et maximale :

- **0** : Zéro (cardinalité minimale nulle).
- **1** : Un (cardinalité minimale ou maximale égale à un).
- **m** : Plusieurs (cardinalité maximale indéterminée, souvent interprété comme "plusieurs").

3.3 Représentation Graphique

Les contraintes de cardinalité sont indiquées sur les traits reliant l'association aux entités. Pour une association entre une entité A et une entité B, on note généralement la cardinalité sous la forme `(min_A, max_A)` du côté de l'entité A et `(min_B, max_B)` du côté de l'entité B.



Dans l'exemple ci-dessus, la cardinalité `(0, m)` du côté de **Acteur** signifie qu'un film peut être joué par zéro ou plusieurs acteurs. La cardinalité `(0, m)` du côté de **Film** signifie qu'un acteur peut jouer dans zéro ou plusieurs films.

Example 3.1. Considérons les règles suivantes :

- (a) Un acteur peut ne jamais avoir joué dans un film.
- (b) Un acteur peut jouer dans plusieurs films.
- (c) Certains films sont tournés sans acteur (ex: film d'animation, documentaire).
- (d) La distribution d'un film est constituée de plusieurs acteurs.

Ces règles justifient la cardinalité (0, m) - (0, m) pour l'association "joue" entre les entités Acteur et Film.

4 Clés (Identifiants) d'une Entité

4.1 Définition et Rôle

Une **clé** (ou **identifiant**) d'une entité est un attribut (ou un ensemble d'attributs) qui permet d'identifier de manière unique chaque instance d'une entité. Il est crucial de pouvoir distinguer et manipuler chaque instance d'une entité, et la clé remplit ce rôle.

4.2 Clé Simple vs. Clé Composée

- **Clé simple** : Une clé simple est constituée d'un seul attribut. Pour une entité E et une clé K (attribut de E), pour deux instances e et e' de E, si e est différent de e', alors la valeur de la clé K pour e doit être différente de la valeur de la clé K pour e' ($K(e) \neq K(e')$).
- **Clé composée** : Une clé composée est constituée de plusieurs attributs. Une clé K d'une entité E est composée de plusieurs attributs A_1, A_2, ..., A_n de E tel que pour 2 instances de E quelconques e et e' on a : si e est différent de e', alors au moins un des attributs composant la clé doit avoir une valeur différente pour e et e' ($A_1(e) \neq A_1(e')$ ou $A_2(e) \neq A_2(e')$ ... ou $A_n(e) \neq A_n(e')$).

Il est important de noter que si un ensemble d'attributs constitue une clé, alors tout sur-ensemble de cet ensemble d'attributs est également une clé. Cependant, on recherche généralement les clés minimales, c'est-à-dire celles qui sont composées d'un sous-ensemble d'attributs garantissant l'unicité des instances.

4.3 Identification d'une Clé

Pour identifier une clé pour une entité donnée, il faut analyser les attributs disponibles et déterminer quel(s) attribut(s) garantissent l'unicité des instances.

Example 4.1. Pour l'entité **Acteur**, on pourrait envisager les clés suivantes :

- **Num_A** (Numéro d'acteur) : Si chaque acteur se voit attribuer un numéro unique, cet attribut pourrait servir de clé simple.
- **Nom** et **Prénom** : La combinaison du nom et du prénom pourrait également identifier un acteur de manière unique, constituant une clé composée. Il faut cependant s'assurer qu'il n'y aura jamais deux acteurs avec le même nom et prénom.

Pour l'entité **Film**, l'attribut **Titre** pourrait être envisagé comme clé simple. Cependant, si plusieurs films peuvent avoir le même titre (remakes, etc.), il faudrait envisager une clé composée, par exemple **Titre** et **Année**.

5 Entités Faibles

5.1 Définition et Caractéristiques

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

Les entités faibles sont souvent utilisées pour représenter des informations qui sont des compléments ou des détails relatifs à une entité existante.

5.2 Identification Relative

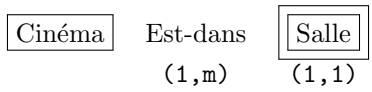
L'identification d'une entité faible est dite **relative** car elle dépend de l'entité forte. La clé d'une entité faible est généralement composée : elle inclut une partie de la clé de l'entité forte et un ou plusieurs attributs propres à l'entité faible, qui permettent de la distinguer *au sein* des instances liées à la même instance de l'entité forte.

Example 5.1. Considérons l'exemple d'une salle de cinéma. Une salle de cinéma (**Salle**) ne peut exister sans un cinéma (**Cinéma**). L'entité **Salle** est donc une entité faible, dépendante de l'entité forte **Cinéma**. Pour identifier une salle de cinéma, on utilise généralement un numéro de salle (**Num_S**) qui est unique *au sein d'un même cinéma*, mais pas forcément unique au niveau global. La clé de l'entité

Salle sera alors composée de la clé de **Cinéma** (par exemple, **Id_C**, identifiant unique du cinéma) et de **Num_S**.

5.3 Représentation Graphique

Graphiquement, une entité faible est souvent représentée par un rectangle à double trait. L'association reliant l'entité faible à l'entité forte est également représentée avec un losange à double trait et la cardinalité du côté de l'entité faible est souvent (1,1) indiquant une dépendance d'existence.



6 Héritage (is-a)

6.1 Définition et Utilité

L'**héritage** (ou relation **is-a**) permet de représenter une hiérarchie de types d'entités, où certaines entités sont des spécialisations (ou sous-types) d'entités plus générales (ou sur-types). L'héritage est utile pour modéliser des situations où des entités partagent des caractéristiques communes mais ont également des propriétés spécifiques.

6.2 Sur-Entité et Sous-Entité

- **Sur-entité (Superclasse)** : L'entité générale, englobant les caractéristiques communes à toutes les entités de la hiérarchie.
- **Sous-entité (Sous-classe)** : Une spécialisation de la sur-entité, héritant de ses caractéristiques et ajoutant des propriétés spécifiques.

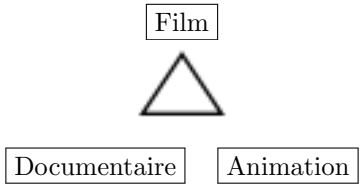
Une instance d'une sous-entité est toujours également une instance de sa sur-entité. Par exemple, si "Film d'animation" est une sous-entité de "Film", alors tout film d'animation est aussi un film.

Example 6.1. Dans notre domaine cinématographique, nous pourrions avoir une hiérarchie d'entités :

- **Film** (sur-entité) : Représente tous les types de films (attributs communs : Titre, Année).
- **Documentaire** (sous-entité de Film) : Spécialisation de Film, avec des attributs spécifiques aux documentaires (ex: Sujet documentaire).
- **Animation** (sous-entité de Film) : Spécialisation de Film, avec des attributs spécifiques aux films d'animation (ex: Techniques d'animation).

6.3 Représentation Graphique

La relation d'héritage (is-a) est généralement représentée par un triangle pointant vers la sur-entité, reliant la sur-entité aux sous-entités. La cardinalité de la relation is-a est généralement (1,1) indiquant qu'une instance de sous-entité correspond à exactement une instance de la sur-entité.



7 Principes de Conception d'un Modèle E/A

La conception d'un bon modèle E/A requiert de suivre certains principes clés pour garantir sa clarté, sa pertinence et son efficacité.

7.1 Se Focaliser sur les Besoins de l'Application et des Utilisateurs

Le modèle E/A doit avant tout répondre aux besoins de l'application et des utilisateurs finaux. Il est essentiel de :

- **Identifier clairement les objectifs de l'application** : Quelles sont les fonctionnalités principales ? Quelles données doivent être gérées ?
- **Comprendre les besoins des utilisateurs** : Quelles informations recherchent-ils ? Comment vont-ils interagir avec la base de données ?
- **S'assurer que les entités et attributs reflètent la réalité métier** : Les éléments modélisés doivent avoir un sens et être pertinents dans le contexte de l'application. Un attribut "GENRE" pourrait ne pas être utile dans une application mini-ciné, mais pertinent dans une application de recommandation de films. De même, stocker l'adresse de chaque acteur peut être inutile pour une application comme Amazon, mais essentiel pour un studio de cinéma.

7.2 Faire Simple

La simplicité est un principe fondamental. Un modèle E/A doit être aussi simple que possible tout en restant suffisamment expressif pour modéliser les données nécessaires. Il faut éviter de complexifier inutilement le modèle en ajoutant des entités, des attributs ou des associations superflues.

7.3 Eviter les Redondances

La redondance, c'est-à-dire la présence de la même information sous différentes formes, est à proscrire. Elle peut engendrer des problèmes d'efficacité (espace de stockage gaspillé, temps de traitement accru) et de qualité des données (données inconsistantes si les copies ne sont pas mises à jour simultanément). Il faut donc :

- **Identifier et éliminer les informations redondantes** : S'assurer que chaque information n'est stockée qu'une seule fois.
- **Normaliser le modèle** : Appliquer les principes de normalisation pour décomposer les entités et relations de manière à minimiser la redondance et les anomalies de mise à jour.

7.4 Choisir Judicieusement Entités, Associations et Attributs

Le choix entre représenter une information comme un attribut, une entité ou une association est crucial. Il faut se poser les bonnes questions :

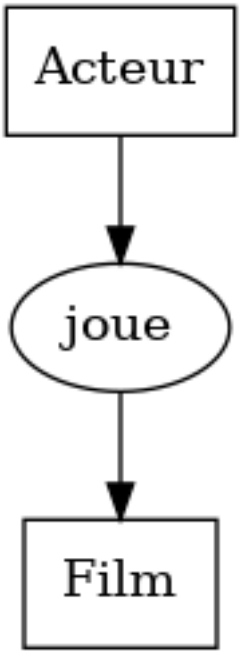
- **Attribut ou Entité ?** : Si une information descriptive est simple et atomique, elle peut être un attribut. Si elle a elle-même des propriétés ou des relations avec d'autres éléments, elle doit être une entité. Par exemple, "Metteur en scène" peut être un attribut de "Film" si l'on ne souhaite stocker que son nom. Mais si l'on veut stocker des informations détaillées sur les metteurs en scène (biographie, filmographie, etc.), "MetteurEnScène" doit devenir une entité à part entière.
- **Association ou Attribut ?** : Parfois, une relation peut être modélisée soit comme une association, soit comme un attribut. Le choix dépend de la complexité de la relation et des informations que l'on souhaite stocker. Si la relation a des attributs propres, il est préférable de la modéliser comme une association. Par exemple, la relation "joue" entre "Acteur" et "Film" pourrait avoir un attribut "Rôle".

- **Cardinalité des associations** : Choisir les bonnes cardinalités est essentiel pour refléter fidèlement les règles métier. Se demander si un acteur peut jouer plusieurs rôles dans un même film, ou si un film peut être dirigé par plusieurs metteurs en scène, permet de déterminer les cardinalités appropriées.

8 Illustrations et Exemples

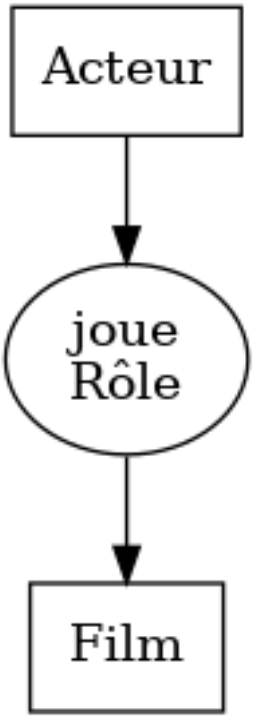
Tout au long de ce chapitre, nous avons utilisé l'exemple d'une application de gestion de films et d'acteurs pour illustrer les concepts du modèle E/A. Les diagrammes suivants récapitulent certains des modèles E/A que nous avons évoqués :

- **Modèle simple : Acteur et Film**



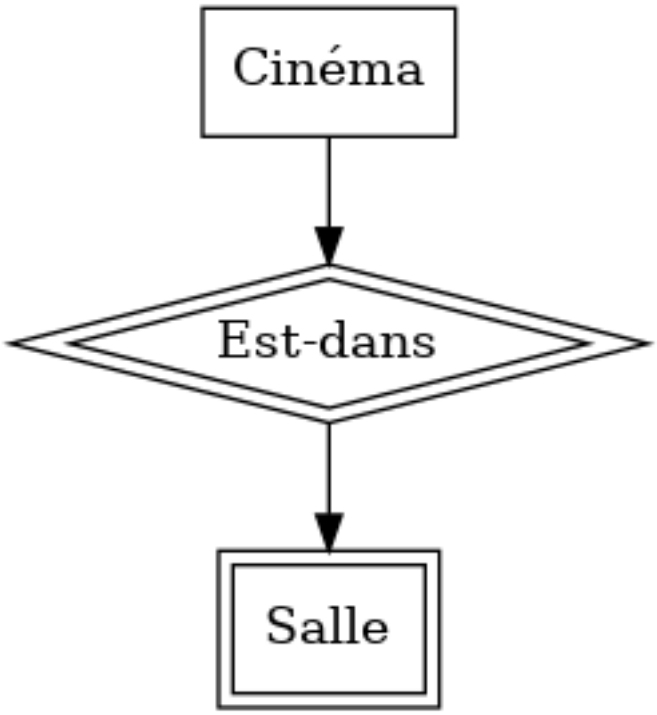
figureModèle E/A simple représentant les entités Acteur et Film avec une association.

- **Modèle avec association et attribut d'association**



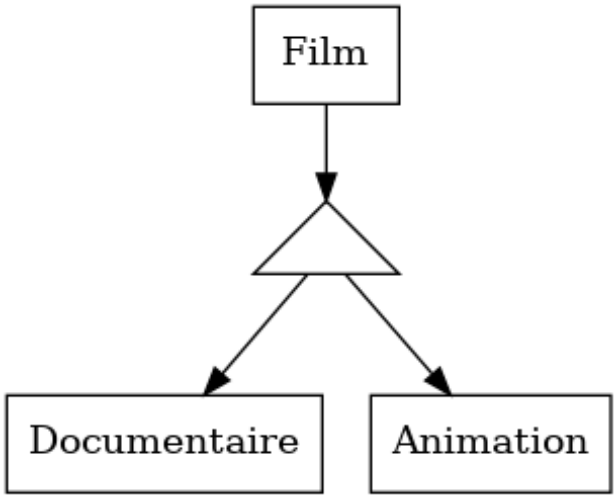
figureModèle E/A illustrant l'association "joue" entre Acteur et Film, avec l'attribut "Rôle" sur l'association.

- **Modèle avec entité faible**



figureModèle E/A montrant une entité faible "Salle" dépendante de l'entité forte "Cinéma".

- **Modèle avec héritage**



figureModèle E/A illustrant l'héritage, avec "Documentaire" et "Animation" comme spécialisations de "Film".

9 Conclusion

La modélisation conceptuelle avec le modèle Entité-Association est une étape fondamentale dans le cycle de vie d'une base de données. Elle permet de définir clairement la structure des données, les relations entre elles et les contraintes à respecter. Un modèle E/A bien conçu facilite la communication entre les différentes parties prenantes du projet, sert de base à la conception logique et physique de la base de données, et contribue à la qualité et à la performance de l'application finale. En suivant les principes de conception présentés et en utilisant les concepts clés du modèle E/A (entités, attributs, associations, cardinalités, clés, entités faibles, héritage), il est possible de créer des modèles de données robustes, clairs et adaptés aux besoins spécifiques de chaque application.