CHAPTER 1

1.1 Introduction et Motivation

1.1.1 Omniprésence des bases de données

- Les données sont **omniprésentes**.
  - Dans notre monde moderne, les données sont devenues un élément fondamental, imprégnant presque tous les aspects de notre vie quotidienne et professionnelle. De la simple interaction avec nos smartphones à la gestion complexe des entreprises mondiales, les données sont au cœur de tout. Cette omniprésence est due à la numérisation croissante des informations et à la capacité accrue de collecter, stocker et traiter ces données.
- Exemples d'omniprésence des données:
  - **Banque - Finance - Assurance**: *Banking system*. Les systèmes bancaires modernes reposent entièrement sur des bases de données pour gérer les comptes des clients, les transactions financières, les prêts, les assurances et bien d'autres opérations. Ces bases de données doivent être extrêmement fiables, sécurisées et performantes pour assurer le bon fonctionnement du secteur financier.
  - **Système d'Information Entreprise**: *Information System* (gestion du personnel, des clients, des stocks d'une entreprise). Toute entreprise, quelle que soit sa taille, utilise un système d'information pour gérer ses opérations internes et ses interactions externes. Ces systèmes d'information utilisent des bases de données pour stocker et organiser des informations cruciales telles que les données des employés, les informations sur les clients, la gestion des stocks, les ventes, le marketing, etc. L'efficacité de ces systèmes est directement liée à la performance des bases de données sous-jacentes.
  - **Gestion de réservations**: *Airline reservation system* (Transports, Hotels, Spectacles). Les systèmes de réservation, largement utilisés dans les secteurs du transport aérien, de l'hôtellerie et du spectacle, dépendent fortement des bases de données. Ils permettent de gérer en temps réel les disponibilités, les réservations, les tarifs et les informations relatives aux clients. Ces systèmes doivent supporter un grand nombre de requêtes simultanées et garantir la cohérence des données.
  - **Commerce électronique**: *e-commerce* (Culture, Agro-alimentaire, Bricolage). Les plateformes de commerce électronique, qu'il s'agisse de vente de produits culturels, agro-alimentaires ou de bricolage, utilisent des bases de données pour gérer les catalogues de produits, les commandes des clients, les paiements, les informations d'expédition, et les recommandations personnalisées. La capacité à gérer de grandes quantités de données de produits et de clients est essentielle pour le succès du commerce électronique.

1.1.2 Motivation: Le Déluge de Données

- **Big Data**, **Data Deluge**, **Data Scientist**. L'expression "Big Data" ou "déluge de données" décrit l'explosion du volume, de la variété et de la vitesse des données générées aujourd'hui. Cette quantité massive de données a conduit à l'émergence du métier de "Data Scientist", un expert capable d'extraire de la valeur et des connaissances de ces données en utilisant des techniques d'analyse avancées et des outils spécifiques.

- L'humanité produit **2,5 quintillons** (10^{30} bytes/j). Chaque jour, l'humanité génère une quantité phénoménale de données, estimée à 2,5 quintillions d'octets. Pour mettre cela en perspective, un quintillion est un milliard de milliards de milliards. Ce chiffre illustre l'ampleur du défi posé par le déluge de données.

- **Data Never Sleeps 1.0 VS. 10.0**. Les infographies "Data Never Sleeps" illustrent de manière frappante la quantité de données générées chaque minute sur Internet. Les versions successives, de 1.0 à 10.0 et au-delà, montrent une augmentation exponentielle des activités en ligne, des messages envoyés, des recherches effectuées, des achats en ligne, et bien d'autres interactions numériques, soulignant encore une fois la croissance massive du volume de données.

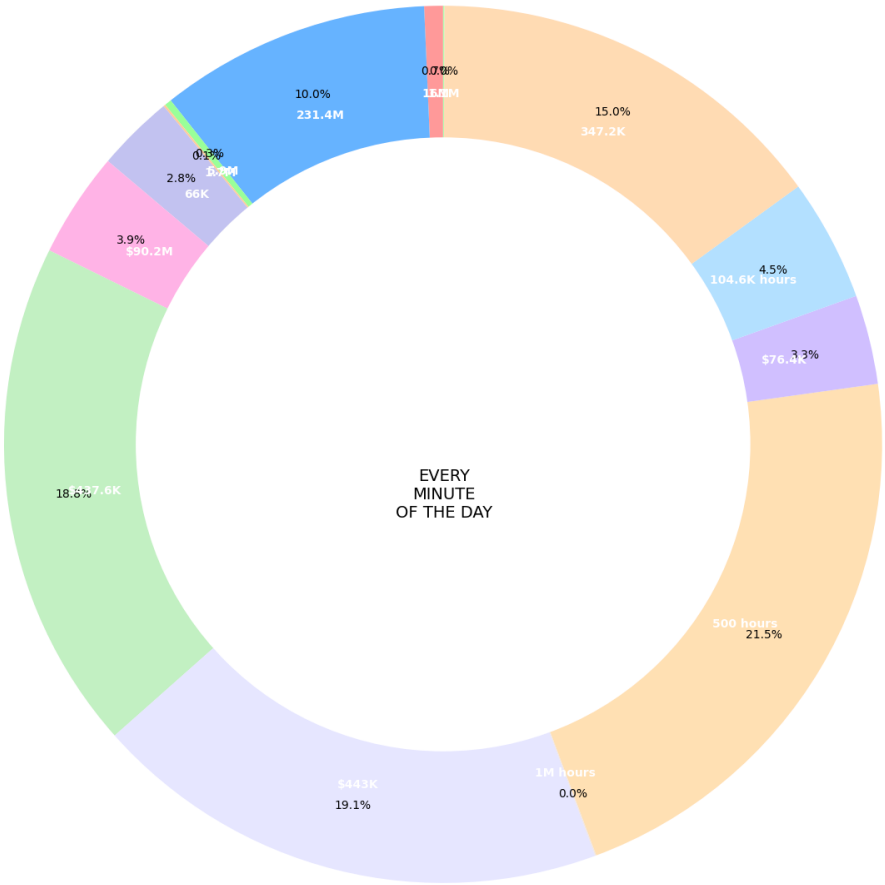


Figure 1.1: Data Never Sleeps

1.1.3 Rôle central des SGBD

- Les **Systèmes de Gestion de Bases de Données (SGBD)** sont au coeur de cette révolution. Face à ce déluge de données, les Systèmes de Gestion de Bases de Données (SGBD) jouent un rôle central. Ils sont les outils fondamentaux qui permettent d'organiser, de stocker, de gérer et d'exploiter efficacement ces masses d'informations. Sans les SGBD, il serait pratiquement impossible de tirer profit du potentiel immense des données.

- Cette révolution a commencé dans les années **1960** ! L'histoire des bases de données remonte aux années 1960, une époque où les ordinateurs commençaient à être utilisés pour gérer des volumes de données de plus en plus importants. C'est durant cette décennie que les premiers concepts et systèmes de gestion de bases de données ont émergé, posant les fondations de ce qui allait devenir un domaine crucial de l'informatique.

- **IL Y A plus de 50 ans !** Il est important de souligner que les bases de données ne sont pas une technologie récente. Plus de 50 ans d'évolution et d'innovation ont permis de développer des systèmes extrêmement sophistiqués et performants, capables de répondre aux exigences les plus pointues du monde moderne en matière de gestion de données.

- **Des données, des données et des données !** La devise pourrait être "des données, des données et encore des données!". L'ère numérique est caractérisée par une croissance exponentielle des données. La capacité à maîtriser et à exploiter ces données est devenue un enjeu majeur pour les organisations de tous types. Les SGBD sont les outils qui permettent de transformer ce flux continu de données en informations précieuses et exploitables.

1.1.4 Que devez-vous connaître après ce cours ?

- Qu'est-ce qu'une **base de données** ? *Database*. Après ce cours, vous devriez être en mesure de définir clairement ce qu'est une base de données. Il ne s'agit pas simplement d'un ensemble de fichiers, mais d'une collection organisée de données, structurée de manière à faciliter son accès, sa gestion et sa mise à jour. Vous comprendrez les concepts fondamentaux qui sous-tendent l'organisation des données dans une base de données.

- **Notions élémentaires & vocabulaire**. Vous acquerrez le vocabulaire essentiel pour parler de bases de données. Vous maîtriserez les termes clés tels que "table", "enregistrement", "champ", "clé primaire", "clé étrangère", "requête", "transaction", etc. Ce vocabulaire vous permettra de communiquer efficacement avec d'autres professionnels du domaine et de comprendre la documentation technique.

- **Fonctionnalités de base d'un SGBD**. Vous découvrirez les fonctionnalités essentielles offertes par un Système de Gestion de Base de Données (SGBD). Vous comprendrez comment un SGBD permet de stocker les données de manière persistante, d'effectuer des requêtes pour extraire des informations spécifiques, de mettre à jour les données et de garantir l'intégrité et la sécurité des informations.

- Un peu d'**histoire**. Nous explorerons brièvement l'histoire des bases de données, en retraçant les étapes clés de leur évolution, depuis les premiers systèmes hiérarchiques et réseaux jusqu'aux SGBD relationnels et aux solutions NoSQL plus récentes. Cette perspective historique vous aidera à comprendre les fondements et les motivations qui ont conduit aux technologies actuelles.

- Quelques **systèmes**. Nous passerons en revue quelques exemples de systèmes de gestion de bases de données populaires, qu'il s'agisse de systèmes commerciaux (comme Oracle, SQL Server) ou de systèmes open source (comme MySQL, PostgreSQL). Vous aurez une idée de la diversité des SGBD disponibles et de leurs principales caractéristiques.

- Quelques **métiers**. Enfin, nous aborderons les différents métiers liés aux bases de données, tels que concepteur de bases de données, administrateur de bases de données, développeur d'applications bases de données, et data scientist. Vous prendrez conscience des opportunités professionnelles offertes par ce domaine en constante expansion.

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.* La fonction primordiale d'un SGBD est de fournir un mécanisme fiable et efficace pour stocker les données de manière persistante. Cela signifie que les données sont conservées même après l'arrêt du système ou en cas de panne. Le SGBD gère l'organisation physique des données sur les supports de stockage (disques durs, SSD, etc.) et assure leur disponibilité.
- **Répondre à des questions = requêtes**: *Query answering.* Un SGBD permet d'interroger les données stockées. Les utilisateurs peuvent formuler des questions, appelées "requêtes", pour extraire des informations spécifiques de la base de données. Le SGBD traite ces requêtes et renvoie les résultats pertinents. Le langage de requête standard pour les SGBD relationnels est SQL (Structured Query Language).
- **Mise à jour les données**: *Update.* Les données stockées dans une base de données ne sont pas statiques, elles évoluent constamment. Un SGBD doit permettre de modifier les données existantes, d'ajouter de nouvelles données et de supprimer des données obsolètes ou incorrectes. Ces opérations de mise à jour doivent être réalisées de manière contrôlée et sécurisée pour garantir la cohérence de la base de données.

1.2.2 Fonctionnalités Clés d'un SGBD

- Que fait un SGBD (pour les applications) ? Pour les applications qui utilisent une base de données, le SGBD agit comme une interface centrale pour la gestion des données. Il simplifie considérablement le travail des développeurs en leur fournissant un ensemble de services essentiels.
- Il assure le **stockage**, la **mise à jour** et l'**accès** (requêtes) à de **grandes masses** de données *massive data*. Un SGBD est conçu pour gérer efficacement de très grands volumes de données, allant de gigaoctets à pétaoctets, voire plus. Il assure le stockage, la mise à jour et l'accès aux données, même lorsque celles-ci atteignent des tailles considérables.
- En assurant :
  - **Efficacité**: *performance.* Un SGBD doit être performant, c'est-à-dire capable de traiter rapidement les requêtes et les mises à jour, même lorsque la base de données est volumineuse et que le nombre d'utilisateurs est élevé. L'efficacité est cruciale pour garantir une bonne expérience utilisateur et pour répondre aux exigences des applications temps réel.
  - **Persistance**: *persistency.* La persistance des données est une fonctionnalité fondamentale. Elle garantit que les données stockées dans la base de données sont conservées de manière durable, même en cas d'arrêt du système, de panne matérielle ou logicielle. Les données ne sont pas volatiles et ne disparaissent pas après la fin d'un programme.
  - **Fiabilité**: *reliability.* La fiabilité d'un SGBD est essentielle pour assurer l'intégrité et la disponibilité des données. Un SGBD fiable doit être capable de résister aux pannes (matérielles, logicielles, réseau...) et de se remettre en état de fonctionnement après une panne, sans perte de données ou corruption.
  - **Simplicité d'utilisation**: *convenience.* Un SGBD doit être simple à utiliser pour les développeurs d'applications et les utilisateurs finaux. Il doit offrir des interfaces conviviales et des outils facilitant la création, la gestion et l'interrogation de la base de données. La simplicité d'utilisation réduit le temps de développement et facilite l'adoption du SGBD.

- **Sécurité**: *safety*. La sécurité des données est une préoccupation majeure. Un SGBD doit mettre en œuvre des mécanismes de sécurité pour protéger les données contre les accès non autorisés, les modifications malveillantes et les pertes. Ces mécanismes comprennent le contrôle d'accès, le chiffrement des données, l'audit des opérations, etc.
- **Multi-utilisateur**: *multi-user*. Dans de nombreux contextes, une base de données est partagée par plusieurs utilisateurs et applications simultanément. Un SGBD multi-utilisateur doit gérer les accès concurrents aux données de manière à garantir la cohérence des données et à éviter les conflits. Il utilise des mécanismes de contrôle de concurrence pour gérer les transactions concurrentes.

1.2.3 Gestion de grandes masses de données

- **grandes masses** de données *massive data*. Comme mentionné précédemment, les SGBD sont conçus pour gérer de très grands volumes de données. Cette capacité à gérer des "masses de données" est l'une de leurs caractéristiques distinctives par rapport à d'autres systèmes de stockage de données.
- Gigabytes (10^9 octets) c'est rien ! Pour les SGBD modernes, gérer des gigaoctets de données est devenu une tâche relativement simple. Les systèmes actuels sont capables de traiter des bases de données de plusieurs gigaoctets sans difficulté majeure.
- Terabytes (10^{12} octets) / jour, Petabytes (10^{15} octets). Les SGBD sont désormais confrontés à la gestion de volumes de données bien plus importants, atteignant les téraoctets (milliards de gigaoctets) et même les pétaoctets (millions de gigaoctets). Certaines organisations génèrent des téraoctets de nouvelles données chaque jour, ce qui pose des défis considérables en termes de stockage et de traitement.
- $\rightsquigarrow$ données stockées en mémoire secondaire (disques). En raison de ces volumes massifs, la grande majorité des données d'une base de données est stockée en mémoire secondaire, principalement sur des disques durs ou des SSD. La mémoire vive (RAM) est généralement utilisée pour stocker les données les plus fréquemment utilisées ou les données en cours de traitement.
- $\rightsquigarrow$ gestion des disques et buffer. Un SGBD doit gérer efficacement l'accès aux données stockées sur disque. Il utilise des techniques de gestion de buffer (mémoire tampon) pour minimiser les accès disque, qui sont beaucoup plus lents que les accès mémoire. Le buffer cache les données fréquemment utilisées en mémoire vive, ce qui accélère considérablement les opérations de lecture et d'écriture.
- Quintillions (10^{30} octets) *very massive data*. À l'avenir, les SGBD devront peut-être faire face à des volumes de données encore plus gigantesques, atteignant les quintillions d'octets. Ces "très grandes masses de données" nécessiteront des approches de stockage et de traitement encore plus innovantes.
- $\rightsquigarrow$ besoin de nouveaux supports de stockage. Pour gérer les volumes de données futurs, il est probable que de nouveaux supports de stockage émergeront, offrant des capacités encore plus importantes et des performances améliorées. La recherche sur les technologies de stockage continue d'évoluer pour répondre à ces besoins croissants.
- $\rightsquigarrow$ de nouveaux mécanismes de stockage et d'accès aux données. Au-delà des supports de stockage, de nouveaux mécanismes de stockage et d'accès aux données seront nécessaires. Cela pourrait impliquer des architectures de bases de données distribuées, des techniques de compression et de déduplication de données plus sophistiquées, et des algorithmes d'indexation et de recherche encore plus performants.

1.2.4 Efficacité

- **efficacité** *performance*. L'efficacité, ou la performance, est un critère essentiel pour un SGBD. Un SGBD performant doit être capable de traiter les requêtes et les mises à jour rapidement, en minimisant les temps de réponse et en optimisant l'utilisation des ressources système.

- Le système doit être capable de gérer plusieurs milliers de requêtes par seconde ! ces requêtes pouvant être complexes. Dans les applications modernes, les SGBD doivent souvent supporter un très grand nombre de requêtes simultanées, parfois plusieurs milliers par seconde. De plus, ces requêtes peuvent être complexes, impliquant des jointures entre plusieurs tables, des calculs complexes, et des analyses de données sophistiquées.

- $\rightsquigarrow$ optimisation - optimisation - optimisation. Pour atteindre de telles performances, l'optimisation est un aspect crucial dans la conception et la mise en œuvre d'un SGBD. L'optimisation intervient à différents niveaux : optimisation des requêtes (choix du plan d'exécution le plus efficace), optimisation du stockage (organisation des données sur disque), optimisation de l'accès aux données (utilisation de buffers et d'index), etc. L'optimisation est un processus continu et itératif.

1.2.5 Persistance

- **persistance** *persistency.* La persistance, comme nous l'avons vu, est la propriété qui garantit que les données stockées dans la base de données sont conservées de manière durable.

- Consultation des palmarès à chaque demande: la réponse peut changer ... en fonction des mises à jour ! Prenons l'exemple d'une base de données contenant les palmarès de compétitions sportives. Lorsque l'on consulte le palmarès, on s'attend à obtenir les résultats les plus récents. Si des mises à jour sont effectuées après une nouvelle compétition, la réponse à la même requête devra refléter ces mises à jour. La persistance garantit que les modifications apportées aux données sont prises en compte lors des consultations ultérieures.

- $\rightsquigarrow$ Toute modification des données faite par un programme est enregistrée sur disque et persiste au delà de la seule exécution du programme. Lorsque un programme effectue une opération de mise à jour (insertion, modification, suppression) sur une base de données, cette modification est enregistrée de manière persistante sur le disque. Même si le programme se termine ou si le système est redémarré, les modifications sont conservées et seront visibles lors des prochaines consultations de la base de données.

1.2.6 Fiabilité

- **fiabilité** (pannes) *reliability.* La fiabilité, dans le contexte des SGBD, concerne la capacité du système à fonctionner correctement et à garantir l'intégrité des données même en cas de pannes ou d'incidents.

- Résistance aux pannes: pannes matérielles, logicielles, électriques, ..., incendie, ... Les SGBD doivent être conçus pour résister à différents types de pannes, qu'il s'agisse de pannes matérielles (disque dur, mémoire, processeur, alimentation électrique), de pannes logicielles (bugs, erreurs système), de problèmes électriques, ou même d'événements plus graves comme un incendie.

- $\rightsquigarrow$ reprise sur panne: *failure recovery.* En cas de panne, un SGBD fiable doit être capable de se redémarrer et de reprendre son activité de manière cohérente. Le processus de "reprise sur panne" (failure recovery) permet de restaurer la base de données dans un état consistant et de minimiser la perte de données.

- $\rightsquigarrow$ journal: *logs, write ahead, checkpoints.* Pour assurer la reprise sur panne, les SGBD utilisent un "journal" (log) des transactions. Le principe du "write-ahead logging" consiste à enregistrer les modifications dans le journal avant de les appliquer réellement à la base de données. Des "points de contrôle" (checkpoints) sont régulièrement effectués pour synchroniser le journal et la base de données et accélérer la reprise en cas de panne.

1.2.7 Sécurité

- **sécurité** *safety.* La sécurité est un aspect crucial des SGBD. Elle vise à protéger les données stockées dans la base de données contre les accès non autorisés, les modifications malveillantes et les divulgations.

- Les données de la base peuvent être critiques pour l'entreprise, pour les clients, pour les états. Les données stockées dans une base de données peuvent être de nature très sensible et critique pour une organisation, ses clients ou même des États. Il peut s'agir d'informations financières, de données personnelles, de secrets commerciaux, d'informations stratégiques, etc. La protection de ces données est essentielle.
- $\rightsquigarrow$ protéger les données d'accès malveillants. Un SGBD doit mettre en place des mécanismes pour empêcher les accès non autorisés aux données. Cela implique de contrôler qui peut accéder à quelles données et quelles opérations ils sont autorisés à effectuer (lecture, écriture, suppression, etc.).
- $\rightsquigarrow$ contrôle d'accès, vues: *access grant*. Les SGBD utilisent des mécanismes de "contrôle d'accès" pour gérer les autorisations des utilisateurs. On peut définir des rôles et des privilèges pour chaque utilisateur ou groupe d'utilisateurs. Les "vues" sont des tables virtuelles qui peuvent être utilisées pour restreindre l'accès à certaines données ou pour simplifier la présentation des données aux utilisateurs.
- La protection de la vie privée (RGPD) est un autre sujet *privacy*. En plus de la sécurité contre les accès malveillants, la protection de la vie privée est devenue une préoccupation majeure, en particulier avec des réglementations comme le RGPD (Règlement Général sur la Protection des Données) en Europe. Les SGBD doivent intégrer des fonctionnalités pour aider à garantir la conformité avec ces réglementations, notamment en matière de gestion du consentement, de droit à l'oubli, de minimisation des données, etc.

1.2.8 Multi-utilisateur

- **multi-utilisateur** *multiuser*. La capacité à gérer des accès multi-utilisateurs est une caractéristique essentielle des SGBD. Dans de nombreux environnements, plusieurs utilisateurs et applications doivent accéder à la base de données simultanément.
- Accès à la base de données partagé par de nombreux utilisateurs et de nombreux programmes. Dans un système d'information d'entreprise, par exemple, de nombreux employés peuvent accéder à la base de données en même temps, depuis leurs postes de travail ou via des applications web. Des programmes automatiques peuvent également interagir avec la base de données en arrière-plan.
- Nécessité de contrôler l'accès (écriture/lecture) pour assurer la cohérence des données et des traitements. Lorsque plusieurs utilisateurs accèdent et modifient les données simultanément, il est crucial de contrôler ces accès pour éviter les conflits et garantir la cohérence des données. Si deux utilisateurs modifient le même enregistrement en même temps sans contrôle, cela peut conduire à des données incohérentes ou à des pertes de modifications.
- $\rightsquigarrow$ contrôle de la concurrence: *concurrency control*. Les SGBD utilisent des mécanismes de "contrôle de concurrence" pour gérer les transactions concurrentes. Ces mécanismes permettent de sérialiser les transactions ou d'utiliser des techniques de verrouillage pour garantir que les transactions concurrentes n'interfèrent pas les unes avec les autres et que la base de données reste dans un état cohérent.

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

- Exécution 1: Voici une première séquence d'opérations pour le retrait de M. Dupont :
  1. Lire_bd Cpte_Dupont $\rightarrow$. Lecture du solde du compte Dupont (Cpte_Dupont) et stockage dans une variable locale (Am). Supposons que Am = 100.
  2. := - 50. Débit de 50 euros du solde local (Am). Am devient 50.
  3. Ecrire_bd AM $\rightarrow$ Cpte_Dupont. Écriture du nouveau solde (Am = 50) dans la base de données pour le compte Dupont (Cpte_Dupont).
- Exécution 2: Voici une deuxième séquence d'opérations, exécutée en même temps que la première,

pour le retrait de Mme Dupont:

i Lire_bd Cpte_Dupont → AMme. Lecture du solde du compte Dupont (Cpte_Dupont) et stockage dans une variable locale (AMme). Supposons que AMme = 100 (lu avant la mise à jour de M. Dupont dans l'exemple d'exécution concurrente).

ii AMme := AMme - 20. Débit de 20 euros du solde local (AMme). AMme devient 80.

iii Ecrire_bd AMme → Cpte_Dupont. Écriture du nouveau solde (AMme = 80) dans la base de données pour le compte Dupont (Cpte_Dupont).

- Exécution concurrente = 1, i, 2, ii, 3, iii. Dans une exécution concurrente, les opérations des deux retraits peuvent s'entrelacer. Par exemple, l'ordre d'exécution pourrait être : opération 1 (lecture de M. Dupont), opération i (lecture de Mme Dupont), opération 2 (débit de M. Dupont), opération ii (débit de Mme Dupont), opération 3 (écriture de M. Dupont), opération iii (écriture de Mme Dupont). Dans ce scénario, le solde final du compte Dupont serait de 80 euros, alors qu'il aurait dû être de 100 - 50 - 20 = 30 euros. Ce problème est dû à l'absence de contrôle de concurrence, où les opérations de retrait interfèrent entre elles. Un SGBD avec contrôle de concurrence approprié éviterait cette situation et garantirait un solde final correct de 30 euros.

1.2.9 Fiabilité - qualité

- **fiabilité - qualité** *data quality.* La fiabilité ne se limite pas à la résistance aux pannes. Elle englobe également la "qualité des données", c'est-à-dire l'exactitude, la validité, la cohérence et la complétude des données stockées dans la base de données.

- données ≠ informations. Il est important de distinguer les "données" des "informations". Les données brutes ne sont que des faits ou des chiffres isolés. Les informations sont des données interprétées et contextualisées, qui prennent du sens pour les utilisateurs. Un SGBD doit garantir la qualité des données pour que celles-ci puissent être transformées en informations fiables.

- données + propriétés = représentation d'informations. Pour que des données brutes deviennent des informations utiles, il faut leur ajouter des "propriétés", telles que la structure, le format, les relations entre les données, les contraintes d'intégrité, etc. Un SGBD permet de définir et de gérer ces propriétés pour garantir la qualité des données.

- $\rightsquigarrow$ contraintes d'intégrité. Les "contraintes d'intégrité" sont des règles qui définissent les propriétés que doivent respecter les données stockées dans la base de données. Elles permettent de garantir la validité et la cohérence des données. Par exemple, une contrainte d'intégrité peut imposer qu'un champ "âge" soit toujours un nombre positif, ou qu'une clé primaire soit unique.

- $\rightsquigarrow$ vérification automatique de l'intégrité. Un SGBD doit être capable de vérifier automatiquement le respect des contraintes d'intégrité lors des opérations de mise à jour. Si une opération de mise à jour viole une contrainte d'intégrité, le SGBD doit rejeter l'opération et empêcher la corruption des données. Cette vérification automatique est essentielle pour maintenir la qualité des données dans le temps.

Example 1.3. • Le César de la meilleure actrice est attribué à une femme. Ceci est un exemple de contrainte d'intégrité. Dans une base de données gérant les récompenses cinématographiques, on peut définir une contrainte qui stipule que le lauréat du "César de la meilleure actrice" doit être de sexe féminin.

- Le César de la meilleure actrice ne peut pas être attribué à deux actrices la même année. Ceci est un autre exemple de contrainte d'intégrité. On peut définir une contrainte qui assure que, pour chaque année, il n'y a qu'une seule actrice qui reçoit le César de la meilleure actrice. Si l'on

> essayait d'insérer un deuxième César de la meilleure actrice pour la même année, la contrainte serait violée, et l'opération serait rejetée par le SGBD.

1.2.10 Simplicité d'utilisation

- **simplicité d'utilisation** *convenience*. Malgré leur complexité interne, les SGBD doivent être simples à utiliser pour les développeurs d'applications, les administrateurs et les utilisateurs finaux. La simplicité d'utilisation est essentielle pour faciliter le développement d'applications, l'écriture de requêtes, la maintenance et l'optimisation de la base de données.
- Développement d'applications / Ecriture de requêtes. Les SGBD fournissent des outils et des langages de programmation qui simplifient le développement d'applications qui accèdent à la base de données. Le langage SQL, en particulier, est un langage puissant et relativement simple pour formuler des requêtes et manipuler les données.
- Maintenance et optimisation. Les SGBD offrent des fonctionnalités pour faciliter la maintenance de la base de données, telles que la sauvegarde et la restauration, la gestion des utilisateurs et des droits d'accès, la surveillance des performances, etc. Des outils d'optimisation aident à améliorer les performances du SGBD et des requêtes.
- $\rightsquigarrow$ Langage de haut niveau & déclaratif: **SQL**. SQL (Structured Query Language) est le langage standard pour interagir avec les SGBD relationnels. C'est un langage de "haut niveau", car il permet d'exprimer des requêtes de manière abstraite, sans se soucier des détails de l'implémentation. C'est également un langage "déclaratif", car on spécifie ce que l'on veut obtenir (les résultats de la requête), et non pas comment l'obtenir (l'algorithme à suivre). Le SGBD se charge d'optimiser l'exécution de la requête.

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

- $\rightsquigarrow$ Schéma & Instance. La notion de "schéma" et d'"instance" est fondamentale dans les SGBD. Le schéma décrit la structure de la base de données (tables, colonnes, types de données, relations, etc.), tandis que l'instance représente les données elles-mêmes à un moment donné, conformes au schéma. La séparation schéma/instance permet de modifier les données sans changer la structure de la base de données, et vice-versa.
- $\rightsquigarrow$ 3 niveaux d'abstraction. Les SGBD reposent sur une architecture à trois niveaux d'abstraction : le niveau externe (vues), le niveau logique et le niveau physique. Cette architecture permet d'assurer l'"indépendance des données", c'est-à-dire la possibilité de modifier le schéma physique ou logique sans impacter les applications qui utilisent la base de données (dans une certaine mesure).
- $\rightsquigarrow$ Principe d'indépendance physique. L'"indépendance physique" est le principe qui permet de modifier l'organisation physique des données (par exemple, le format de stockage sur disque, l'utilisation d'index) sans avoir à modifier les applications qui utilisent la base de données. Cela offre une grande flexibilité et permet d'optimiser les performances du SGBD sans perturber les applications existantes.

1.3 Principes des SGBD

1.3.1 Schéma versus instance

- **schéma** versus **instance**. Comme mentionné précédemment, les concepts de schéma et d'instance sont centraux dans les bases de données. Ils permettent de distinguer la structure des données de leur contenu effectif.

- **schéma** = description de la structuration des données. Le schéma d'une base de données est une description formelle de la façon dont les données sont organisées. Il définit les types de données que la base contiendra, les relations entre ces données, et les contraintes qui doivent être respectées. On peut le voir comme le plan, ou la structure, de la base de données.
- le schéma est une donnée. Il est important de noter que le schéma lui-même est stocké et géré comme des données au sein du SGBD. On peut interroger le schéma, le modifier, le sauvegarder, tout comme on le ferait avec les données de la base elle-même. Cette "métadonnée" (donnée sur les données) est essentielle pour le fonctionnement du SGBD.
- $\rightsquigarrow$ stocké + modifiable + interrogeable. Le fait que le schéma soit une donnée implique qu'il peut être stocké de manière persistante, qu'il peut être modifié au cours du temps (évolution du schéma), et qu'il peut être interrogé pour connaître la structure de la base de données.
- **instance** (du schéma) = données organisées en se conformant au schéma. L'instance d'une base de données, à un instant donné, est l'ensemble des données effectives stockées dans la base. Ces données sont organisées conformément au schéma défini. On peut voir l'instance comme le "remplissage" du schéma avec des données concrètes.

schéma = récipient instance = contenu

1.3.2 Les Trois Niveaux d'Abstraction

- **Les Trois Niveaux d'Abstraction**. L'architecture des SGBD est souvent décrite en termes de trois niveaux d'abstraction, permettant de gérer la complexité et d'assurer l'indépendance des données.
  - **Niveau externe (vues)**: description et manipulation des données dédiées à un groupe d'utilisateurs et applications. Le niveau externe, ou niveau des vues, est le niveau le plus proche des utilisateurs et des applications. Il définit un ensemble de "vues" qui sont des présentations personnalisées de la base de données pour différents groupes d'utilisateurs ou applications. Chaque vue peut masquer certaines données ou présenter les données d'une manière spécifique. Cela permet de simplifier l'accès aux données pour chaque type d'utilisateur et d'assurer une certaine sécurité en limitant l'accès à certaines informations.
  - **Niveau logique**: description et manipulation des données "haut niveau". Le niveau logique, ou niveau conceptuel, décrit la structure globale de la base de données, sans se soucier des détails de l'implémentation physique. Il définit les entités, les attributs, les relations, et les contraintes d'intégrité. Le schéma logique est généralement exprimé à l'aide d'un modèle de données (par exemple, le modèle relationnel). C'est le niveau de description utilisé par les concepteurs de bases de données et les développeurs d'applications.
  - **Niveau physique (interne)**: organisation et stockage des données en mémoire secondaire. Le niveau physique, ou niveau interne, décrit comment les données sont réellement stockées sur les supports de stockage (disques durs, SSD...). Il s'agit des détails de l'organisation des fichiers, des index, des structures de données utilisées pour optimiser l'accès aux données, du format de stockage, etc. Ce niveau est principalement concerné par l'efficacité et les performances du SGBD.
- **Utilisateurs versus Niveaux**. Différents types d'utilisateurs interagissent avec les SGBD à différents niveaux d'abstraction.
  - Utilisateurs λ, Développeurs d'Applications $\Longrightarrow$ Vues, Niveau Logique. Les utilisateurs finaux ("utilisateurs lambda") et les développeurs d'applications interagissent principalement avec le niveau externe (vues) et le niveau logique. Ils utilisent des vues pour accéder aux données de manière simplifiée et adaptée à leurs besoins. Ils manipulent les données en se basant sur le schéma logique, sans se préoccuper des détails de stockage physique.
  - Développeurs BD, Administrateur BD $\Longrightarrow$ Niveaux Logique & Physique. Les développeurs de bases de données et les administrateurs de bases de données (DBA) sont plus concernés par les niveaux logique et physique. Les développeurs de BD conçoivent le schéma logique et les

contraintes d'intégrité. Les DBA sont responsables de l'implémentation physique de la base de données, de l'optimisation des performances, de la sécurité, de la sauvegarde et de la restauration, etc. Ils doivent maîtriser les aspects logiques et physiques du SGBD.

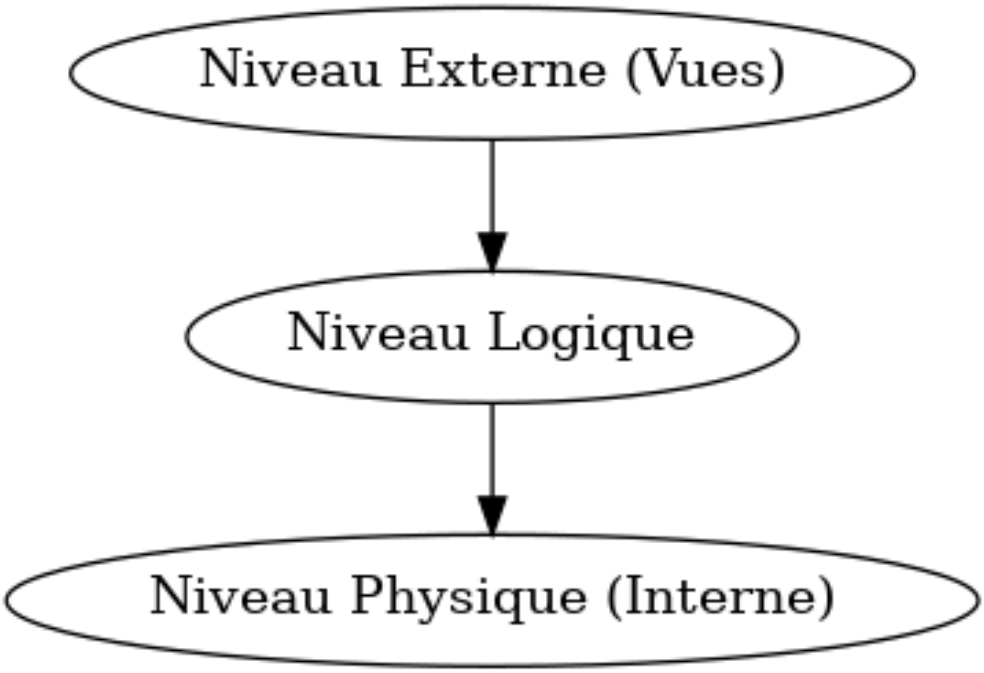


Figure 1.2: Architecture 3 niveaux

1.3.3 Indépendance physique

- **Indépendance physique**. L'indépendance physique des données est un principe fondamental de l'architecture à trois niveaux. Elle permet de modifier le niveau physique sans affecter le niveau logique et externe.
- $\begin{array}{c|c} \textbf{Programmes d'application} & \textbf{Invariants} \\ \hline \textbf{Schéma Physique} & \text{Modification} \end{array}$ L'indépendance physique signifie que les programmes d'application et le schéma logique restent "invariants" (inchangés) même si l'on modifie le schéma physique. On peut changer la manière dont les données sont stockées physiquement sans avoir à réécrire les applications ou à modifier le schéma logique.

Example 1.4. Exemple: Considérons l'exemple d'une base de données contenant les palmarès de festivals de cinéma.

- Initialement = BD Palmarès festivals français. Au départ, la base de données ne contient que les palmarès des festivals français.
- $\rightsquigarrow$ schéma physique = fichier non trié, pas d'index *heapfile*. Initialement, le schéma physique peut être très simple : les données sont stockées dans un "fichier tas" (heapfile), c'est-à-dire un fichier non trié, sans index. Dans ce cas, pour répondre à une requête, le SGBD doit parcourir séquentiellement tout le fichier.

  Combien d'Oscars pour Adjani =

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

Par exemple, la requête SQL ci-dessus, qui cherche à compter le nombre de prix (Palme d'Or - Palm.MA) remportés par Isabelle Adjani (Palm.Act='Adjani'), sera exécutée en parcourant tout le fichier de palmarès.

- Qcq années + tard = BD Palmarès festivals internationaux. Quelques années plus tard, on décide d'étendre la base de données pour inclure les palmarès des festivals internationaux. Le volume de données augmente considérablement.
- $\rightsquigarrow$ schéma physique = fichier indexé (Arbre B+) *index file*. Pour améliorer les performances des requêtes, on décide de modifier le schéma physique. On crée un "fichier indexé" (index file), en utilisant par exemple une structure d'index de type Arbre B+. Un index permet d'accéder plus rapidement aux données en fonction de certaines clés de recherche.

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

La même requête SQL, exécutée après la modification du schéma physique et la création d'un index sur le champ "Acteur", sera maintenant beaucoup plus rapide. Le SGBD pourra utiliser l'index pour localiser rapidement les palmarès concernant Isabelle Adjani, sans avoir à parcourir tout le fichier. Cependant, la requête SQL elle-même n'a pas changé, et les applications qui utilisent cette requête continuent de fonctionner 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**. La première génération de SGBD, dans les années 1960 et 1970, était basée sur le "modèle réseau". Ce modèle permettait de représenter des relations complexes entre les données, mais il était assez complexe à mettre en œuvre et manquait d'indépendance des données.
  - IDS (General Electric), APL, DMS 1100, ..., ADABAS (Software AG). Parmi les SGBD réseau les plus connus, on peut citer IDS (Integrated Data Store) développé par General Electric, APL, DMS 1100, et ADABAS.
- **60-70 SGBD Hiérarchique**. En parallèle des SGBD réseau, les "SGBD hiérarchiques" ont également émergé dans les années 1960 et 1970. Ce modèle organisait les données sous forme d'arbres, avec une structure hiérarchique rigide. Il était plus simple que le modèle réseau, mais moins flexible pour représenter des relations complexes.
  - ISM, IBM (www.software.ibm.com). Le SGBD hiérarchique le plus emblématique est IMS (Information Management System) d'IBM, toujours utilisé aujourd'hui dans certains systèmes legacy.
  - gestion de pointeurs entre enregistrements. Dans les SGBD hiérarchiques et réseaux, les relations entre les enregistrements étaient généralement implémentées à l'aide de "pointeurs". Les pointeurs étaient des adresses mémoire qui reliaient les enregistrements entre eux. Cette approche était efficace en termes de performance, mais rendait le schéma physique très dépendant du schéma logique, limitant l'indépendance des données.
  - problème : pas d'**indépendance physique**. L'un des principaux problèmes des SGBD hiérarchiques et réseaux était le manque d'indépendance physique. Toute modification de l'organisation physique des données pouvait avoir un impact important sur les applications et nécessiter des modifications du schéma logique et des programmes.

- **70- SGBD RELATIONNEL** *Relational DBMS*. Dans les années 1970, Edgar F. Codd a introduit le "modèle relationnel", qui a révolutionné le domaine des bases de données. Le modèle relationnel proposait une approche plus simple, plus formelle et plus flexible pour la gestion des données.
  - modèle logique simple : données = table. Dans le modèle relationnel, les données sont organisées sous forme de "tables" (ou relations). Une table est un tableau à deux dimensions, composé de lignes (enregistrements) et de colonnes (attributs). Ce modèle logique simple est facile à comprendre et à utiliser.
  - formalisation mathématique : théorie des ensembles ou logique. Le modèle relationnel repose sur des fondements mathématiques solides, issus de la théorie des ensembles et de la logique. Cette formalisation a permis de développer des langages de requête puissants et d'optimiser l'exécution des requêtes de manière rigoureuse.
  - un langage normalisé = **SQL** (Structured Query Language). Le langage SQL (Structured Query Language) est devenu le langage de requête standard pour les SGBD relationnels. SQL est un langage déclaratif et de haut niveau, qui permet d'interroger et de manipuler les données de manière simple et efficace. La normalisation de SQL a favorisé l'interopérabilité entre différents SGBD relationnels.
- **SGBD relationnel non normalisé**. Au fil du temps, des extensions au modèle relationnel ont été proposées pour gérer des données plus complexes, comme les données imbriquées ou les données semi-structurées. Les "SGBD relationnels non normalisés" ont introduit des fonctionnalités permettant de stocker des tables dans des colonnes de tables, par exemple.
  - une extension du modèle relationnel : des tables dans des tables ! Cette approche permet de représenter des structures de données plus riches que les simples tables plates du modèle relationnel de base. Elle a été utilisée dans certains SGBD pour gérer des documents complexes ou des données XML.
- **SGBD orienté objet (OO)**. Dans les années 1980 et 1990, avec l'essor de la programmation orientée objet, les "SGBD orientés objet" (SGBD OO) ont émergé. Ces systèmes cherchaient à intégrer les concepts de l'orienté objet (objets, classes, héritage, encapsulation, etc.) dans les bases de données.
  - modèle inspiré par les concepts des langages objets. Le modèle de données des SGBD OO était basé sur les objets, les classes, l'héritage, le polymorphisme, etc. L'objectif était de réduire l' "impédance" entre les langages de programmation orientés objet et les bases de données, et de faciliter le développement d'applications OO manipulant des données persistantes.
  - identité d'objet, héritage, méthodes ... Les SGBD OO offraient des fonctionnalités telles que l'identité d'objet (chaque objet a un identifiant unique et persistant), l'héritage (possibilité de définir des classes filles héritant des propriétés des classes mères), les méthodes (opérations spécifiques associées aux objets), etc.
- **Modèle semi-structuré et XML**. Avec l'arrivée du Web et l'échange de données sur Internet, le "modèle semi-structuré" a gagné en importance. Ce modèle permet de gérer des données dont la structure n'est pas rigide ou complètement définie à l'avance, comme les documents XML.
  - information incomplète, schéma souple. Dans les données semi-structurées, certaines informations peuvent être manquantes, la structure peut varier d'un document à l'autre, ou le schéma peut être implicite ou évoluer fréquemment. Le modèle semi-structuré offre plus de flexibilité que le modèle relationnel pour gérer ce type de données.
  - motivé par l'échange de données par les services web. Le modèle semi-structuré et le format XML sont devenus très populaires pour l'échange de données entre applications et services web, car ils permettent de transmettre des données structurées tout en conservant une certaine flexibilité.
- **Modèle de graphes et RDF**. Plus récemment, le "modèle de graphes" a connu un essor important, notamment avec le développement du Web sémantique et des réseaux sociaux. Ce modèle représente les données sous forme de graphes, composés de nœuds (entités) et d'arêtes (relations).

  - information incomplète, possibilité de raisonnement, schéma souple et interopérables. Le modèle de graphes est bien adapté pour représenter des données complexes, interconnectées et potentiellement incomplètes. Il permet de faire du "raisonnement" sur les données (inférence de nouvelles relations), et il est naturellement "interopérable" grâce à des standards comme RDF (Resource Description Framework).
  - motivé par la publication et le partage de données sur le web. Le modèle de graphes et RDF sont au cœur du Web sémantique, qui vise à publier et à partager des données structurées sur le Web de manière interopérable et exploitable par les machines. Les graphes de connaissances (knowledge graphs) comme ceux utilisés par Google ou Facebook sont des exemples d'applications du modèle de graphes à grande échelle.

- **NoSQL**. Le terme "NoSQL" (Not Only SQL) est apparu pour désigner une catégorie de SGBD qui s'éloignent du modèle relationnel traditionnel et du langage SQL. Les SGBD NoSQL visent à répondre aux besoins spécifiques des applications web modernes, en termes de scalabilité, de performance, de flexibilité et de gestion de grands volumes de données non structurées ou semi-structurées.
  - Adéquation aux besoins actuels. Les SGBD NoSQL sont souvent mieux adaptés que les SGBD relationnels pour certaines applications web qui nécessitent une grande scalabilité horizontale (capacité à distribuer la base de données sur plusieurs serveurs), de hautes performances en lecture/écriture, et la gestion de données non structurées (documents, graphes, etc.).
  - "SQL" = SGBDs relationnels classiques. Dans le contexte de NoSQL, "SQL" fait référence aux SGBD relationnels classiques, comme Oracle, MySQL, PostgreSQL, etc.
  - NoSQL= Ne pas utiliser les SGBDs relationnels classiques. L'expression "NoSQL" suggère l'idée de "ne pas utiliser seulement les SGBD relationnels classiques". Il ne s'agit pas de rejeter complètement les SGBD relationnels, mais de reconnaître qu'il existe d'autres types de SGBD qui peuvent être plus pertinents pour certains cas d'usage.
  - NoSQL $\neq$ Ne pas utiliser le langage SQL. Contrairement à ce que le nom pourrait laisser penser, "NoSQL" ne signifie pas "ne pas utiliser SQL". Certains SGBD NoSQL proposent des langages de requête qui s'inspirent de SQL ou qui permettent d'utiliser des dialectes de SQL pour interroger les données.
  - NoSQL = Ne pas utiliser **seulement** les SGBDs relationnels. La signification précise de "NoSQL" est donc "Not Only SQL" : ne pas utiliser seulement les SGBD relationnels, mais explorer et utiliser d'autres types de SGBD (documentaires, clé-valeur, colonnaires, graphes, etc.) en fonction des besoins de l'application.

1.4.2 Quelques systèmes relationnels

- **Systèmes historiques**. Parmi les premiers SGBD relationnels, on peut citer :

- System R, IBM-San José- www.mcjones.org/System$_R$.$SystemR, développéparIBMdanslesannées1970, estconsidéréco$
  $Ingres)www.ingres.com/products/ingres - database.php.Ingres, développéàl'universitédeBerkeleyàlamêmeépoque,$

- **Systèmes propriétaires**. De nombreux SGBD relationnels commerciaux ont été développés, parmi lesquels :
  - Oracle Database www.oracle.com. Oracle Database est l'un des SGBD relationnels les plus utilisés au monde, notamment dans les grandes entreprises. Il offre de nombreuses fonctionnalités avancées et une grande scalabilité.
  - Microsoft SQL Server, Microsoft www.microsoft.com. Microsoft SQL Server est un autre SGBD relationnel majeur, particulièrement populaire dans l'environnement Windows. Il est étroitement intégré aux technologies Microsoft.
  - DB2, MaxDB, 4D, dBase, Informix, Sybase ... D'autres SGBD relationnels propriétaires importants incluent DB2 (IBM), MaxDB (SAP), 4D, dBase, Informix (IBM), Sybase (SAP), etc. Chacun de ces systèmes a ses propres caractéristiques et points forts.

- **Systèmes libres**. De nombreux SGBD relationnels open source sont également disponibles :
  - PostgreSQL www.enterprisedb.com. PostgreSQL est un SGBD relationnel open source très puissant et riche en fonctionnalités. Il est réputé pour sa conformité aux standards, sa robustesse et ses performances.
  - MariaDB (MySQL) www.mariadb.org. MariaDB est un fork open source de MySQL, un autre SGBD relationnel open source très populaire, notamment pour les applications web. MariaDB vise à rester compatible avec MySQL tout en offrant des améliorations et des fonctionnalités supplémentaires.
  - Firebird, Ingres, HSQLDB, Derby. D'autres SGBD relationnels open source incluent Firebird, Ingres (qui est devenu open source), HSQLDB (base de données relationnelle en Java), Derby (base de données relationnelle en Java, projet Apache), etc. Ces systèmes offrent des alternatives open source aux SGBD commerciaux.

1.5 Les différents acteurs

1.5.1 Acteurs autour des SGBD

- **Développeurs de SGBD**. Le domaine des bases de données implique différents types d'acteurs, chacun ayant un rôle spécifique. Les "développeurs de SGBD" sont ceux qui conçoivent et implémentent les systèmes de gestion de bases de données eux-mêmes.
  - Concevoir et développer les systèmes de gestion de bases de données (du futur !). Les développeurs de SGBD sont des experts en informatique, en algorithmique, en systèmes d'exploitation, etc. Ils travaillent sur les fonctionnalités internes des SGBD, l'optimisation des performances, la gestion de la concurrence, la sécurité, etc. Ils sont également impliqués dans la recherche et le développement de nouvelles technologies de bases de données pour répondre aux défis futurs.
- **Concepteur de bases de données**. Les "concepteurs de bases de données" (ou architectes de données) sont responsables de la conception du schéma logique des bases de données, c'est-à-dire de la structure des données, des relations, des contraintes d'intégrité, etc.
  - Concevoir le schéma logique de la base de données. Les concepteurs de bases de données travaillent en étroite collaboration avec les utilisateurs et les experts métier pour comprendre les besoins en données de l'organisation. Ils utilisent des méthodes de modélisation des données (comme le modèle Entité-Association ou le modèle UML) pour concevoir un schéma logique efficace et adapté aux besoins.
- **Développeur d'applications base de données**. Les "développeurs d'applications bases de données" sont ceux qui créent les applications qui utilisent les bases de données pour stocker et manipuler les données.
  - Développer les programmes (requêtes, mises à jour et autres) qui opèrent sur la base de données et répondent aux besoins des applications. Les développeurs d'applications bases de données utilisent des langages de programmation (comme Java, Python, C, etc.) et des interfaces d'accès aux bases de données (comme JDBC, ODBC, etc.) pour écrire des programmes qui interrogent, mettent à jour et traitent les données stockées dans les SGBD. Ils doivent maîtriser le langage SQL et les principes de la programmation d'accès aux données.
- **Administrateur de la base de données**. Les "administrateurs de bases de données" (DBA) sont responsables de l'exploitation, de la maintenance et de la gestion des SGBD en production.
  - Paramétrer le système, maintenance (tunning) des applications. Les DBA installent et configurent les SGBD, gèrent les utilisateurs et les droits d'accès, assurent la sécurité des données, effectuent les sauvegardes et les restaurations, surveillent les performances, optimisent les requêtes (tunning), gèrent l'espace disque, etc. Ils sont les garants du bon fonctionnement et de la disponibilité des bases de données.

1.6 Objectifs du Cours

1.6.1 Objectif principal

- Un SGBD relationnel est un **système complexe** ! Un Système de Gestion de Base de Données relationnel (SGBDR) est un logiciel extrêmement complexe, intégrant de nombreuses fonctionnalités et mécanismes sophistiqués pour garantir l'efficacité, la fiabilité, la sécurité et la cohérence des données. La complexité interne d'un SGBDR est considérable, même si l'interface utilisateur et les outils de développement cherchent à simplifier son utilisation.
- **facile à utiliser - difficile à bien utiliser !** Paradoxalement, les SGBDR sont conçus pour être "faciles à utiliser" dans le sens où ils offrent des interfaces conviviales, des langages de requête simples (comme SQL), et des outils qui facilitent le développement d'applications. Cependant, il est "difficile de bien utiliser" un SGBDR si l'on veut tirer pleinement parti de ses capacités, optimiser les performances, garantir la sécurité et la fiabilité, et concevoir des bases de données robustes et évolutives. La maîtrise approfondie des SGBDR demande des connaissances techniques pointues et de l'expérience.

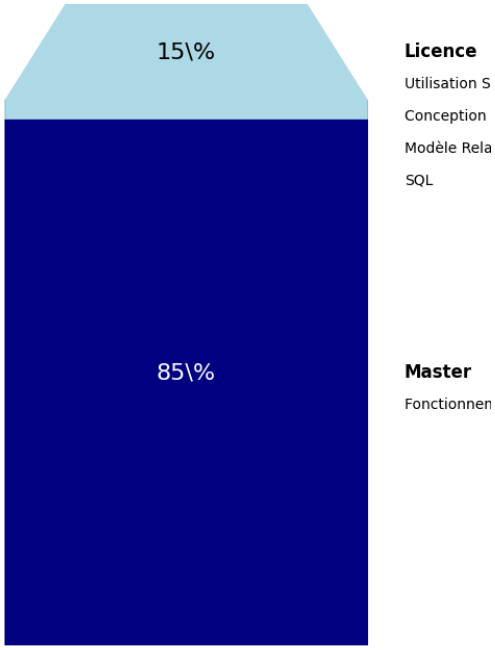


Figure 1.3: Iceberg des SGBD

1.7 Plan du cours

1.7.1 Bases de Données - Plan du cours

- **Conception d'une base de données**. La première partie du cours sera consacrée à la conception des bases de données, c'est-à-dire à la définition du schéma logique.

  - Processus de conception. Nous étudierons les étapes du processus de conception d'une base de données, depuis la collecte des besoins jusqu'à la validation du schéma. Nous verrons les différentes phases de la conception conceptuelle, logique et physique.
  - Le modèle Entité-Association. Nous apprendrons à utiliser le modèle Entité-Association (E-A) pour la modélisation conceptuelle des données. Le modèle E-A est un outil graphique puissant pour représenter les entités, les attributs et les relations dans un domaine d'application.

- **Modèle de données**. La deuxième partie du cours portera sur le modèle de données relationnel, qui est le modèle dominant pour les SGBD.
  - Le Modèle Relationnel. Nous étudierons en détail les concepts du modèle relationnel : tables, colonnes, clés primaires, clés étrangères, relations, etc. Nous verrons comment organiser les données sous forme de tables et comment définir les relations entre les tables.
  - Du Modèle E-A au modèle relationnel. Nous apprendrons à traduire un schéma conceptuel exprimé en modèle E-A en un schéma logique relationnel. Nous verrons les règles de conversion et les bonnes pratiques pour passer du modèle conceptuel au modèle logique.
  - SQL : Langage de définition (LDD). Nous introduirons le langage SQL, en nous concentrant sur la partie LDD (Langage de Définition de Données). Le LDD de SQL permet de créer, de modifier et de supprimer les objets de la base de données (tables, vues, index, etc.) et de définir le schéma logique.

- **Interrogation d'une base de données**. La troisième partie du cours sera consacrée à l'interrogation des bases de données, c'est-à-dire à l'extraction d'informations à partir des données stockées.
  - Algèbre relationnelle. Nous étudierons l'algèbre relationnelle, qui est un langage formel de manipulation des relations (tables). L'algèbre relationnelle est à la base du langage SQL et permet de comprendre les opérations fondamentales de requête.
  - SQL : Langage de manipulation (LMD). Nous approfondirons le langage SQL, en nous concentrant sur la partie LMD (Langage de Manipulation de Données). Le LMD de SQL permet d'interroger les données (requêtes SELECT), d'insérer, de modifier et de supprimer des données. Nous verrons les différentes clauses de SQL (SELECT, FROM, WHERE, GROUP BY, ORDER BY, JOIN, etc.) et les techniques de requête avancées.

- **Dépendances & Conception de schéma**. La dernière partie du cours abordera les notions de dépendances fonctionnelles et la conception de schémas relationnels de qualité.
  - Dépendances fonctionnelles. Nous étudierons les dépendances fonctionnelles, qui sont des contraintes sémantiques sur les données. Les dépendances fonctionnelles permettent de formaliser les relations entre les attributs et de détecter les redondances et les anomalies potentielles dans un schéma relationnel.
  - Schéma sous forme normale. Nous verrons les concepts de formes normales (1NF, 2NF, 3NF, BCNF, etc.) et les techniques de normalisation des schémas relationnels. La normalisation vise à éliminer les redondances et à améliorer la qualité et la flexibilité des schémas. Nous apprendrons à concevoir des schémas relationnels bien normalisés et adaptés aux besoins des applications.

1.8 Bibliographie

1.8.1 Une Petite Bibliographie

- **Bases de données**, G. Gardarin, Editions Eyrolles. Un ouvrage de référence en français sur les bases de données, couvrant les aspects conceptuels, techniques et pratiques.

- **Système de gestion des bases de données**, H. Korth et A. Silberschatz, McGraw-Hill. Un manuel classique et très complet sur les SGBD, souvent utilisé dans les cursus universitaires.

- **A first course in Database System**, J. Ullman et J. Widom. Un autre manuel de référence, plus orienté vers l'apprentissage des concepts fondamentaux des bases de données.
- **Fondements des bases de données**, Serge Abiteboul. Un livre de référence en français, écrit par un expert reconnu du domaine, abordant les fondements théoriques et les aspects avancés des bases de données.
- **Database System Concepts**, Korth, Silberschatz. La version originale en anglais du manuel "Système de gestion des bases de données" mentionné précédemment.
- **Database Systems - Concepts, Languages and Architectures**, Atzeni *et al.* http://dbbook.dia.uniroma3.it/dbbook.pdf. Un manuel de bases de données disponible gratuitement en ligne, couvrant un large éventail de sujets, des concepts de base aux architectures avancées.

CHAPTER 2

2.1 Introduction au Modèle Entité-Association

2.1.1 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.

2.1.2 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.

2.1.3 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.

2.2 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.

2.2.1 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

2.2.2 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.

2.2.3 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.

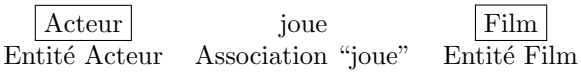
Les associations permettent de modéliser les interactions entre les entités du monde réel. Chaque association est spécifiée par :

- **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.

2.3 Contraintes de Cardinalité

2.3.1 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.

2.3.2 Types de Cardinalités

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

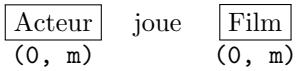
- **Un-à-un (1:1)** : Une instance de A est liée à au plus une instance de B, et réciproquement.
- **Un-à-plusieurs (1:m)** : Une instance de A est liée à zéro, une ou plusieurs instances de B, mais une instance de B est liée à au plus une instance de A.
- **Plusieurs-à-plusieurs (m:m)** : Une instance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

2.3.3 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 2.7. 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.

2.4 Clés (Identifiants) d'une Entité

2.4.1 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.

2.4.2 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.

2.4.3 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 2.8. 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**.

2.5 Entités Faibles

2.5.1 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.

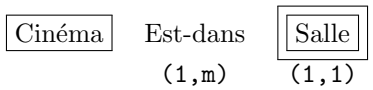
2.5.2 Identification Relative

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 2.9. 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**.

2.5.3 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.



2.6 Héritage (is-a)

2.6.1 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.

2.6.2 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.

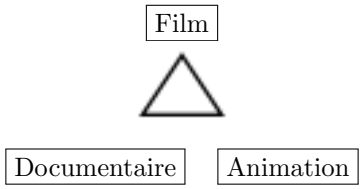
Une instance d'une sous-entité est toujours également une instance de sa sur-entité. Par exemple, si "Film d'animation" est une sous-entité de "Film", alors tout film d'animation est aussi un film.

Example 2.10. 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).

2.6.3 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.



2.7 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.

2.7.1 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.

2.7.2 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.

2.7.3 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.

2.7.4 Choisir Judicieusement Entités, Associations et Attributs

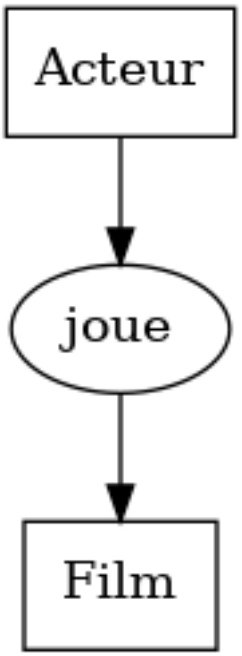
Le choix entre représenter une information comme un attribut, une entité ou une association est crucial. Il faut se poser les bonnes questions :

- **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.

2.8 Illustrations et Exemples

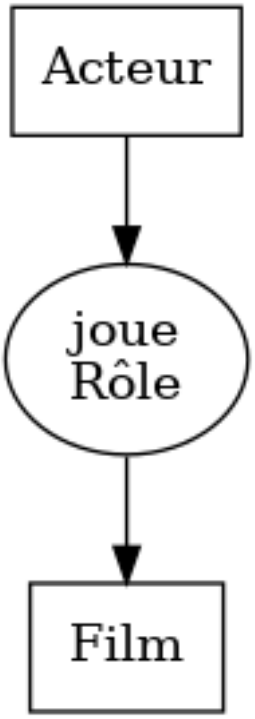
Tout au long de ce chapitre, nous avons utilisé l'exemple d'une application de gestion de films et d'acteurs pour illustrer les concepts du modèle E/A. Les diagrammes suivants récapitulent certains des modèles E/A que nous avons évoqués :

- **Modèle simple : Acteur et Film**



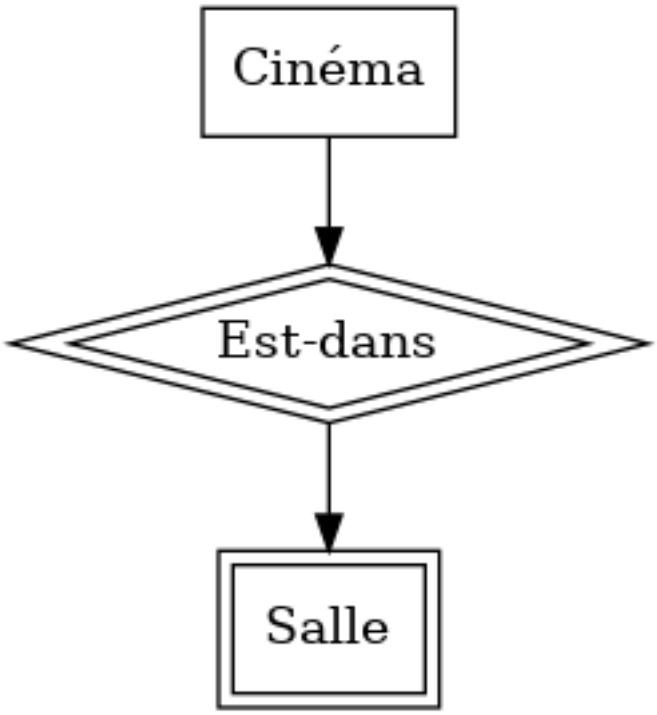
figureModèle E/A simple représentant les entités Acteur et Film avec une association.

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



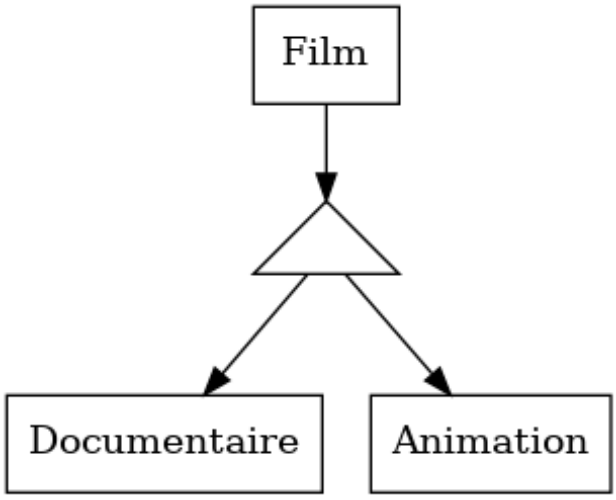
figureModèle E/A illustrant l'association "joue" entre Acteur et Film, avec l'attribut "Rôle" sur l'association.

- **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".

2.9 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.

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.