

UNIVERSITÉ DE JENDOUBA
FACULTÉ DES SCIENCES JURIDIQUES, ÉCONOMIQUES ET DE GESTION
DÉPARTEMENT INFORMATIQUE

Les Bases de Données Relationnelles

Support de cours

Roukaya Ben Jeddou
FSJEG, Université de Jendouba
CREGO, Université de Bourgogne Europe
rokaya.benjeddou@fsjegj.u-jendouba.tn

Table des matières

Liste des figures	iii
Avant propos	v
1 Introduction aux bases de données	1
Introduction	1
1.1 Contexte et motivation	1
1.2 Historique des SGBD	3
1.3 Types de bases de données	4
1.4 Exemples de domaines d'utilisation	5
1.5 Rôle des SGBD et panorama des solutions existantes	6
Conclusion	7
Exercices corrigés	7
2 Modélisation conceptuelle	9
Introduction	9
2.1 Concepts de base	9
2.2 Exemples pratiques	12
2.3 Règles de conception et bonnes pratiques	13
Conclusion	13
Exercice 1 : Gestion d'une compagnie touristique	13
Exercice 2 : Agence immobilière	16
3 Modélisation relationnelle	21
Introduction	21
3.1 Schéma relationnel	21
3.2 Production du schéma relationnel	22
3.3 Synthèse des règles de traduction	27
3.4 Avantages du modèle relationnel	29
3.5 Limites du modèle relationnel	30
Conclusion	30

Exercices corrigés	31
4 Bases de données relationnelles	35
Introduction	35
4.1 Les concepts fondamentaux	35
4.2 Les contraintes d'intégrité	37
4.3 Les dépendances fonctionnelles	38
4.4 La normalisation	39
Conclusion	40
Exercices corrigés	41
5 Algèbre relationnelle	49
Introduction	49
5.1 Opérations fondamentales	49
5.2 Opérations de jointure	51
Conclusion	52
Exercices corrigés	53
6 Introduction au Langage SQL	63
Introduction	63
6.1 Commandes DDL : CREATE, ALTER, DROP	64
6.2 Commandes DML : INSERT, UPDATE, DELETE, SELECT	65
6.3 Clauses avancées : WHERE, GROUP BY, HAVING, ORDER BY	67
6.4 Contraintes d'intégrité, vues et index	68
6.5 Requêtes imbriquées et jointures	70
6.6 Fonctions d'agrégation	70
6.7 Transactions et gestion de la concurrence	71
Conclusion	72
Exercices corrigés	72
7 Études de cas et exercices SQL	83
7.1 Mini-projet 1 : Gestion d'école	83
7.2 Mini-projet 2 : Gestion d'une bibliothèque	85
7.3 Mini-projet 3 : E-commerce	87
7.4 Travaux pratiques (TP)	89
Conclusion	89
Conclusion générale	91
Bibliographie	93

Liste des figures

1.1	Schéma illustrant BD-SGBD	2
2.1	Schéma illustrant: Entité-Association	11
2.2	Schéma illustrant: Les cardinalités	12
2.3	Dictionnaire de données	17
2.4	Schéma Entité-Association	18
2.5	Schéma Entité-Association	20
3.1	Type entité → Table	23
3.2	Association un à plusieurs → Clé étrangère	24
3.3	Clé étrangère multi-composant	24
3.4	Migration de la clé étrangère	25
3.5	Association un à un par clé étrangère	25
3.6	Association plusieurs à plusieurs→ création d'une relation	26
3.7	Modélisation d'une association cyclique ou réflexive un à plusieurs	26
3.8	Modélisation d'une association cyclique ou réflexive plusieurs à plusieurs	27
3.9	Entité→ Table; Migration clé 1 à N	27
3.10	Migration clé 1 à 1; N à N; Multi-composant	28
3.11	Attributs; Mixte, Rôles	28
3.12	Exemple	29

Avant propos

La maîtrise des bases de données est devenue incontournable dans de nombreux domaines, que ce soit pour la gestion d'entreprises, d'institutions éducatives, d'applications web ou de systèmes d'information complexes. Ce cours a pour objectif de fournir aux étudiants et aux lecteurs une approche progressive et pratique de la conception, de la modélisation et de l'exploitation des bases de données relationnelles.

Organisé en sept chapitres, cet ouvrage commence par une introduction aux concepts fondamentaux des bases de données, en passant par l'historique des systèmes de gestion de bases de données (SGBD), leurs types, et les domaines d'application. Nous avons ensuite abordé la modélisation conceptuelle et relationnelle, accompagnée d'exemples pratiques et d'exercices corrigés pour renforcer la compréhension des lecteurs.

Les chapitres consacrés aux bases de données relationnelles, à l'algèbre relationnelle et au langage SQL permettent de consolider les connaissances théoriques par l'application concrète des commandes DDL, DML et des requêtes avancées. Enfin, le dernier chapitre propose des études de cas, exercices pratiques et mini-projets qui permettent de passer progressivement de la théorie à la pratique.

Nous espérons que ce livre saura accompagner le lecteur dans sa découverte et sa maîtrise des bases de données relationnelles, et qu'il servira de tremplin vers des applications et projets concrets dans le domaine de l'informatique.

Chapitre 1

Introduction aux bases de données

Introduction

Les bases de données sont au cœur de presque toutes les applications modernes, qu'il s'agisse de banques, d'e-commerce, de réseaux sociaux ou de systèmes industriels. Une base de données permet de stocker, organiser et manipuler les données de manière efficace, sécurisée et cohérente. Dans ce chapitre, nous allons présenter le contexte historique des systèmes de gestion de bases de données (SGBD), les types de bases de données, des cas d'usage concrets et un tableau comparatif des SGBD les plus utilisés.

1.1 Contexte et motivation

Avec l'évolution de l'informatique et la croissance rapide des données, les organisations (banques, hôpitaux, administrations, e-commerce) doivent gérer efficacement l'information. Les fichiers traditionnels provoquaient des redondances, des incohérences et une difficulté d'accès aux données. Pour répondre à ces défis, les bases de données (BD) et les systèmes de gestion de bases de données (SGBD) ont été introduits.

Définition

Une **base de données** est un ensemble structuré de données enregistrées sur des supports accessibles par l'ordinateur, représentant des informations du monde réel et pouvant être interrogées et mises à jour par plusieurs utilisateurs.

Définition

Un **Système de Gestion de Base de Données (SGBD)** est un logiciel permettant de créer, manipuler, interroger et sécuriser une base de données.

Un SGBD est un ensemble de logiciels chargés d'assurer le maintien de la cohérence des données entre elles, le contrôle d'intégrité des données accédées, les autorisations d'accès aux données et les opérations classiques sur les données (consultation, insertion, modification, suppression).

Actuellement, la plupart des SGBD fonctionnent selon un mode client/serveur. Le serveur reçoit des requêtes de plusieurs clients et ceci de manière concurrente. Le serveur analyse la requête, la traite et retourne le résultat au client (Figure 1.1).

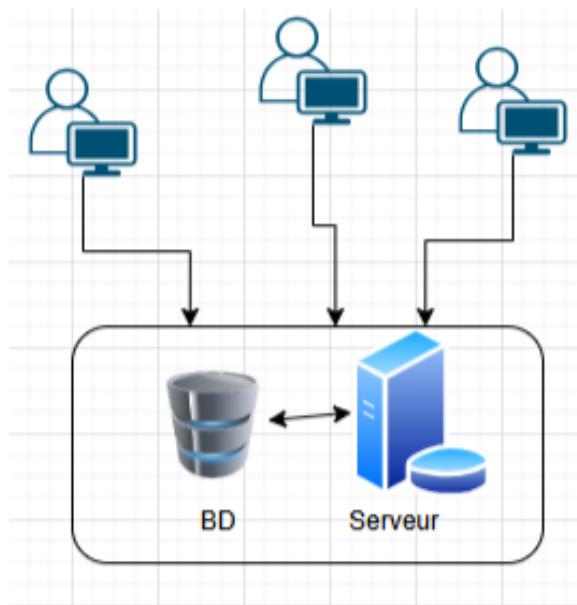


Figure 1.1: Schéma illustrant BD-SGBD

Motivation

Les organisations modernes manipulent des volumes de données de plus en plus importants. L'utilisation d'un SGBD s'impose pour répondre à plusieurs besoins essentiels:

- **Centralisation des données** : éviter la duplication, réduire les incohérences et assurer une source unique de vérité.
- **Accès multi-utilisateurs** : permettre à plusieurs utilisateurs et applications de consulter et modifier les données simultanément, tout en garantissant la cohérence.
- **Sécurité et confidentialité** : contrôle d'accès, authentification et autorisations pour protéger les informations sensibles (banques, santé, administration...).
- **Intégrité des données** : maintien de la fiabilité grâce aux contraintes (clés primaires, référentielles, règles métier).
- **Performance et optimisation des requêtes** : extraction rapide d'informations pour répondre aux besoins métier.
- **Aide à la décision** : analyses, rapports et tableaux de bord facilitant la gouvernance basée sur les données.

Les systèmes d'information modernes reposent ainsi sur des SGBD fiables afin de réduire les coûts opérationnels, améliorer la qualité des données et soutenir la transformation digitale.

1.2 Historique des SGBD

L'évolution des systèmes de gestion de bases de données peut être résumée ainsi:

- **Années 1960** : les bases hiérarchiques et réseaux (IMS, CODASYL). Les données étaient organisées en arbres ou graphes et les applications accédaient directement aux structures.

- **Années 1970** : apparition des bases relationnelles grâce aux travaux d'E. F. Codd (1970). Les données sont représentées sous forme de tables et les relations entre données sont gérées via des clés primaires et étrangères.
- **Années 1980-1990** : généralisation des SGBD relationnels commerciaux (Oracle, IBM DB2, Microsoft SQL Server).
- **Années 1990** : apparition des bases de données orientées objet (OODB). Ces SGBD combinent les concepts des bases relationnelles avec la programmation orientée objet, permettant de stocker directement des objets complexes (images, vidéos, documents, structures hiérarchiques) sans avoir besoin de les transformer en tables.
- **Années 2000** : émergence du **Big Data** et des bases NoSQL pour répondre aux besoins de volume, vitesse et variété des données (MongoDB, Cassandra, Redis).
- **Aujourd’hui** : bases orientées graphes pour les relations complexes (Neo4j, OrientDB), et bases cloud pour la scalabilité et la disponibilité (Amazon RDS, Google Cloud SQL). solutions hybrides, cloud databases, *distributed systems*.

1.3 Types de bases de données

Les principaux types de bases de données incluent :

- **Bases relationnelles (RDBMS)** : données stockées dans des tables avec des relations définies. Exemples : PostgreSQL, MySQL, Oracle, SQL Server.
- **Bases NoSQL** : conçues pour gérer de grandes quantités de données non structurées. Types : document (MongoDB), clé-valeur (Redis), colonnes (Cassandra), graphe (Neo4j).
- **Bases orientées graphe** : adaptées aux relations complexes et aux réseaux sociaux. Exemple : Neo4j, ArangoDB.

- **Bases temps réel et bases cloud :** permettent l'accès rapide aux données massives, souvent distribuées sur plusieurs serveurs. Exemple : Amazon Aurora, Google Big-Query.

1.4 Exemples de domaines d'utilisation

Banque

- Gestion des comptes clients, transactions et prêts.
- Sécurité et intégrité critique : transactions financières avec rollback en cas d'erreur.
- Analyse des risques grâce à des requêtes complexes et des rapports automatisés.

E-commerce

- Gestion des clients, produits, commandes et stocks.
- Requêtes fréquentes pour l'historique des commandes, suivi des livraisons et reporting.
- Big Data et analyse prédictive pour recommander des produits aux clients.

Autres domaines

- Santé : dossiers médicaux électroniques, suivi des patients et statistiques épidémiologiques.
- Transport : gestion des vols, trains, réservations et itinéraires.
- Réseaux sociaux : gestion des relations, publications, likes, commentaires et graphes d'amis.

Remarque: Le modèle relationnel reste le plus utilisé, notamment pour les systèmes transactionnels.

1.5 Rôle des SGBD et panorama des solutions existantes

Les SGBD constituent l'un des piliers technologiques des systèmes d'information modernes. Ils sont conçus pour garantir une gestion efficace, sécurisée et cohérente des données dans divers domaines applicatifs tels que la banque, l'e-commerce, les systèmes hospitaliers ou l'éducation.

Avantages des SGBD

L'adoption d'un SGBD présente de nombreux bénéfices pour les organisations :

- **Réduction de la redondance** et prévention des incohérences.
- **Sécurité et confidentialité** grâce au contrôle d'accès.
- **Accès multi-utilisateurs** avec gestion concurrente fiable.
- **Indépendance des données**, séparation entre logique et stockage.
- **Performances** et optimisation des requêtes.
- **Support des transactions** avec les propriétés ACID.

Ces capacités permettent de répondre aux exigences industrielles en matière de performance, de fiabilité et de continuité de service.

Comparatif de SGBD représentatifs

Le tableau (Table 1.1) présente un aperçu des SGBD largement utilisés dans le monde professionnel.

Cette diversité de solutions permet aux organisations de choisir la technologie la plus adaptée à leurs besoins.

SGBD	Modèle	Licence	Cas d'usage
MySQL	Relationnel	Open Source	Web, ERP
PostgreSQL	Relationnel	Open Source	Scientifique, SIG, SQL avancé
Oracle DB	Relationnel	Commercial	Grandes entreprises
MongoDB	Document (NoSQL)	Open Source	Big Data, applications flexibles
Neo4j	Graphe	Commercial/Open	Réseaux sociaux, recommandations

Table 1.1: Exemples de SGBD populaires et leurs caractéristiques

Avantages majeurs des SGBD

- Réduction de la redondance
- Intégrité et cohérence des données
- Sécurité et contrôle d'accès
- Partage multi-utilisateurs
- Performances et indépendance des données

Conclusion

Ce chapitre introduit les concepts fondamentaux des BD et SGBD, leur évolution historique, les différents modèles existants ainsi que leurs domaines d'application. Les bases de données constituent un pilier incontournable de la transformation numérique.

Exercices corrigés

1. Expliquer la différence entre base de données et SGBD avec un exemple concret.
2. Citer trois avantages des bases de données par rapport aux fichiers traditionnels.
3. Classer les SGBD suivants dans le bon modèle : MySQL, MongoDB, Neo4j, Cassandra.
4. Donner un exemple de cas d'utilisation idéal pour une base NoSQL.

Correction

1. Différence entre base de données et SGBD: Une *base de données* (BD) est un ensemble structuré de données persistantes. Un *Système de Gestion de Base de Données* (SGBD) est le logiciel permettant de créer, manipuler, interroger et sécuriser cette base.

Exemple: La banque "ABC" dispose d'une base de données "ComptesClients". Le logiciel MySQL Server est le SGBD qui gère cette base, exécute les requêtes, garantit l'intégrité et permet les sauvegardes.

2. Trois avantages des BD par rapport aux fichiers traditionnels:

- Accès multi-utilisateur simultané avec gestion de la concurrence.
- Maintien de l'intégrité des données grâce aux contraintes et normalisation.
- Traitement structuré et optimisé des requêtes (SQL, indexation).

3. Classement des SGBD selon leur modèle:

- MySQL → Relationnel
- MongoDB → NoSQL (Document)
- Neo4j → NoSQL (Graphe)
- Cassandra → NoSQL (Colonne/Famille de colonnes)

4. Exemple d'utilisation d'une base NoSQL: Pour une application de réseau social traitant des millions de messages et publications :

- Les données sont semi-structurées et évoluent rapidement → MongoDB ou Cassandra.
- Scalabilité horizontale pour supporter un grand nombre d'utilisateurs simultanés.
- Flexibilité pour gérer les changements fréquents de schéma.

Chapitre 2

Modélisation conceptuelle

Introduction

Dans le chapitre précédent, nous avons présenté les concepts fondamentaux des bases de données et des systèmes de gestion de bases de données. Avant de créer physiquement une base de données dans un SGBD relationnel ou NoSQL, il est indispensable de comprendre et représenter clairement les données et leurs relations. C'est là qu'intervient la *modélisation conceptuelle*, qui permet de transformer les besoins des utilisateurs en un modèle structuré et compréhensible, indépendant du logiciel de base de données choisi.

Le modèle *Entité-Association* est un outil puissant pour représenter de manière graphique les entités, leurs attributs, leurs relations et les contraintes associées. Dans ce chapitre, nous allons explorer ce modèle et apprendre à créer des diagrammes ER afin de les traduire ensuite en *schéma relationnel*.

2.1 Concepts de base

Le modèle Entité-Association (EA en français) ou ER en anglais (Entity Relationship) est un formalisme retenu par l'ISO pour décrire l'aspect conceptuel des données à l'aide d'entités et d'associations.

Entité

Une **entité** représente un objet du monde réel ou abstrait que l'on souhaite modéliser dans la base de données. Exemples : *Client, Produit, Cours, Étudiant.*

Définition

Une **Entité** est un élément du monde réel dont on souhaite conserver des informations dans la base de données.

Par exemple:

- L'étudiant nommé TOTO est une **instance** ou **occurrence** de l'entité ETUDIANT
- l'employé nommé DODO est une instance ou occurrence de l'entité EMPLOYE

Propriété

Une propriété décrit des données élémentaires relatives à une entité. Par exemple, un numéro de carte d'étudiant, le matricule d'un employé, une date de début d'un projet ect. Nous nous considérons que les propriétés qui intéressent un contexte particulier. Les propriétés d'une entité sont également appelées des **attributs** ou des **caractéristiques** de cette entités.

Définition

Une **Propriété** décrit un attribut ou une caractéristique d'une entité.

Exemples : pour l'entité *Client* → *Nom, Prénom, Adresse, Téléphone.*

Identifiant

Propriété ou groupe de propriétés qui sert à identifier une entité. L'identifiant d'une entité est choisi par l'analyste de façon à ce que deux occurrences de cette entité ne puissent pas avoir le même identifiant.

Définition

Un **Identifiant** est un attribut (ou un ensemble d'attributs) permettant d'identifier de manière unique une entité.

Par exemple:

- le numéro d'employé sera l'identifiant de l'entité EMPLOYE
- ID_Client pour identifier chaque client de manière unique.

Association

C'est une représentation d'un lien entre deux entités ou plus (figure 2.1). Une association peut avoir des propriétés particulières Par exemple, la date d'emprunt d'un livre.

Définition

Une **Association** représente un lien entre deux ou plusieurs entités.

Exemple : un client peut passer plusieurs commandes → relation *Passe* entre *Client* et *Commande*.

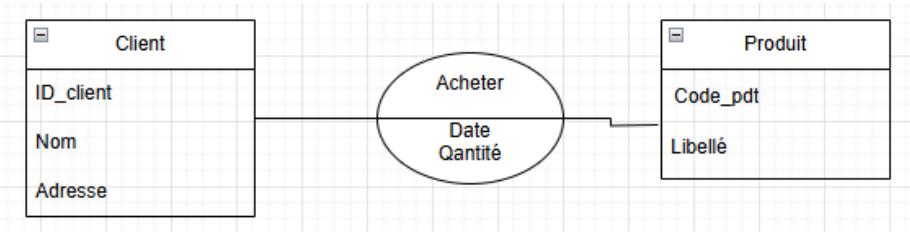


Figure 2.1: Schéma illustrant: Entité-Association

Cardinalité

La cardinalité d'une association pour une entité constituante est constituée d'une borne minimale et d'une borne maximale:

- **Minimale** : nombre minimum de fois qu'une occurrence de l'entité participe aux occurrences de l'association, généralement **0** ou **1**.

- **Maximale** : nombre maximum de fois qu'une occurrence de l'entité participe aux occurrences de l'association, généralement **1** ou **N**.

Définition

La **cardinalité** indique le nombre d'occurrences possibles dans une association pour chaque entité.

Exemples:

- 1:1 → un étudiant a un seul ID de carte d'étudiant.
- 1:N → un professeur enseigne plusieurs cours.
- N:M → un étudiant peut suivre plusieurs cours et un cours peut avoir plusieurs étudiants.

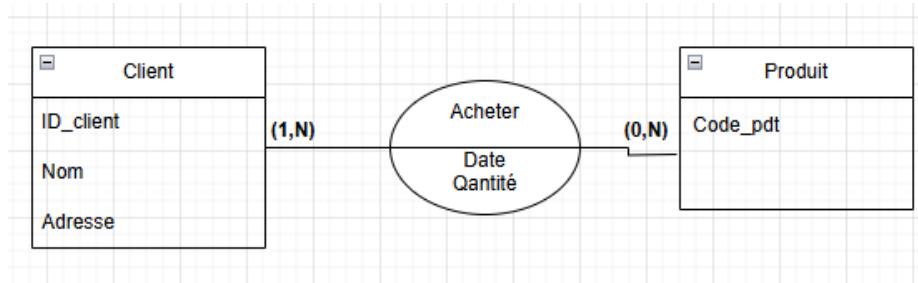


Figure 2.2: Schéma illustrant: Les cardinalités

2.2 Exemples pratiques

Bibliothèque

Entités : Livre, Auteur, Emprunteur, Prêt Relations :

- Livre – Auteur : N:M
- Livre – Emprunteur : N:M via Prêt
- Prêt : inclut date de début et date de retour

École

Entités : Étudiant, Cours, Professeur, Département Relations :

- Étudiant – Cours : N:M
- Professeur – Cours : 1:N
- Étudiant – Département : N:1

2.3 Règles de conception et bonnes pratiques

- Identifier toutes les entités pertinentes avant de créer les associations.
- Définir un identifiant (ou plusieurs) pour chaque entité afin d'assurer l'unicité.
- Préciser les cardinalités pour chaque association.
- Éviter la redondance et respecter la *normalisation*.
- Documenter les hypothèses et règles métier pour faciliter la maintenance.

Conclusion

La modélisation conceptuelle est essentielle pour : - Comprendre les besoins métiers avant de créer des bases de données, - Identifier les entités, attributs et associations, - Prévenir les erreurs de conception et faciliter la transition vers la modélisation relationnelle. Par conséquent, la maîtrise du modèle Entité Association est une étape incontournable avant de passer à SQL et à la création de tables relationnelles.

Exercice 1 : Gestion d'une compagnie touristique

Une agence de voyage organise des circuits touristiques dans divers pays. Les règles de gestion suivantes ont été collectées auprès du responsable :

- RG1 : On garde trace de tous les clients connus même s'ils n'ont pas participé à des circuits touristiques depuis longtemps.
- RG2 : On répertorie un seul hôtel par ville.
- RG3 : Toutes les villes sont désignées par des noms distincts.
- RG4 : Il y a un seul accompagnateur par voyage.
- RG5 : Toute nuit pendant le circuit est passée dans un hôtel.
- RG6 : Tout circuit concerne au moins deux villes.
- RG7 : Toutes les villes répertoriées ne sont pas obligatoirement utilisées dans un circuit à chaque période.
- RG8 : À une même date, aucun circuit ne part plus d'une fois d'une même ville ni n'arrive plus d'une fois dans une même ville.
- RG9 : Les demandes de réservations donnent lieu à des réponses positives dans la mesure des places disponibles.
- RG10 : Un client ne peut obtenir une réservation qu'après une réponse positive et le versement d'un acompte.
- RG11 : Une réservation ne sera définitive qu'après le règlement du solde dû par un deuxième versement.
- RG12 : Après une date limite D1, les réservations non confirmées sont annulées.
- RG13 : Après une date limite D2 :
 - S'il n'y a pas assez de réservations définitives, le circuit est annulé et les clients ayant réservé définitivement sont remboursés.
 - S'il n'y a aucune réservation définitive, le circuit est annulé.
 - S'il y a assez de réservations, le circuit est maintenu.

Travail demandé :

1. Établir une version simplifiée du dictionnaire de données : entités, clés, propriétés, signification (présenter sous forme de tableau).
2. Lister les associations reliant les entités.
3. Proposer le schéma Entité-Association (E-A) correspondant et décrire les associations et leurs cardinalités.

Correction de l'Exercice 1

1. Figure 2.3
2.
 - Type association : Accompagne Relie les circuits et les accompagnateurs. Un circuit a un et un seul accompagnateur ; un accompagnateur peut accompagner au cours d'une saison plusieurs circuits.
 - Type association : Arrivée Relie les déplacements et les villes. Un déplacement a toujours une et une seule ville d'arrivée ; une ville est au moins ville d'arrivée d'un déplacement d'un circuit et peut être ville d'arrivée de plusieurs déplacements de circuits différents.
 - Type association : Concerne Relie les paiements et les réservations. Un paiement concerne toujours une et une seule réservation ; une réservation peut avoir fait l'objet au minimum de zéro paiement et au maximum de deux paiements.
 - Type association : Départ Relie les déplacements et les villes. Un déplacement a toujours une et une seule ville de départ ; une ville est au moins ville de départ d'un déplacement d'un circuit et peut être ville de départ de plusieurs déplacements de circuits différents.
 - Type association : Origine Relie les circuits et les villes. Elle désigne pour un circuit sa ville de départ et sa ville de retour (qui est la même puisqu'il s'agit d'un circuit). Un circuit est donc toujours lié à une et une seule ville

par cette association, tandis qu'une ville ne peut être origine d'aucun circuit au minimum, mais peut être, au maximum, départ de plusieurs circuits.

- Type association : Pour Relie les réservations et les circuits. Une réservation est effectuée pour un et un seul circuit ; pour un circuit, il peut y avoir plusieurs réservations.
- Type association: Programme Relie les circuits et les déplacements. Un déplacement concerne un et un seul circuit ; un circuit comporte au moins un déplacement, en général, il en comporte plusieurs.
- Type association: Réserve Relie les réservations et les clients. Une réservation est effectuée par un et un seul client. Un client présent dans la base peut n'avoir aucune réservation en cours, comme il peut en avoir plusieurs.
- Type association : Situation Relie les villes et les pays. Une ville est située dans un pays et un pays présent dans la base comporte au moins une ville de la base et peut-être plusieurs.

3. Figure 2.4

Exercice 2 :Gestion d'une agence immobilière

Une agence immobilière souhaite créer une base de données afin de gérer les biens immobiliers mis à sa disposition et d'effectuer différentes analyses statistiques ou fiscales.

Pour chaque logement, on enregistre : l'adresse, le nom du propriétaire, le type (maison / appartement), le nombre de pièces, la surface habitable, l'état (neuf, bon état, très bon état, à rénover), l'objectif de gestion (vente ou location), le prix de mise en vente ou le prix de location mensuelle, la date de disponibilité, la ville, etc.

Chaque propriété peut comporter un ou plusieurs garages. Ceux-ci sont caractérisés par leur type (box, emplacement numéroté, etc.) et peuvent, dans certains cas, avoir une adresse différente de celle du logement.

Entité	Clé	Propriétés
Accompagnateur	nom-accomp	nom-accomp : Nom de l'accompagnateur
Circuit	nocircuit	nocircuit : Numéro du circuit noplaces : Nombre de places prévues pour le circuit prix : Prix du circuit date-dep : Date de départ du circuit dateret : Date de retour du circuit etatcircuit : Etat du circuit
Client	noclient	noclient : Numéro du client nom-client : Nom du client
Déplacement	num-déplacement	num-déplacement : numéro du déplacement date : Date du déplacement heure-A : Heure d'arrivée dans la ville but du déplacement heure-D : Heure de départ de la ville origine du déplacement
Paiement	nopaiement	nopaiement : Numéro du paiement typevers : Type de versement montant : Montant du versement
Pays	nom-pays	nom-pays : Nom du pays
Réservation	nores	nores : Numéro de la réservation etates : Etat de la réservation datereserv : Date de la réservation montantot : Montant total déjà versé pour cette réservation
Ville	nom-ville	nom-ville : Nom d'une ville hôtel : Nom de l'hôtel d'hébergement dans la ville

Figure 2.3: Dictionnaire de données

Une personne, identifiée par son nom et son adresse, peut mettre en vente ou en location un de ses logements auprès de l'agence.

Un logement à vendre (resp. à louer) peut être acheté (resp. loué) par une personne. Pour chaque transaction de vente, l'agence touche une commission : une partie fixe + un pourcentage variable (entre 3% et 5%) selon le montant de la transaction et les négociations.

Une fois vendu ou loué, un logement devient indisponible pour d'autres clients. Un locataire peut donner son préavis ; dans ce cas, le logement redevient disponible après un délai de trois mois.

L'agence organise et suit les visites effectuées par des clients (acheteurs ou locataires

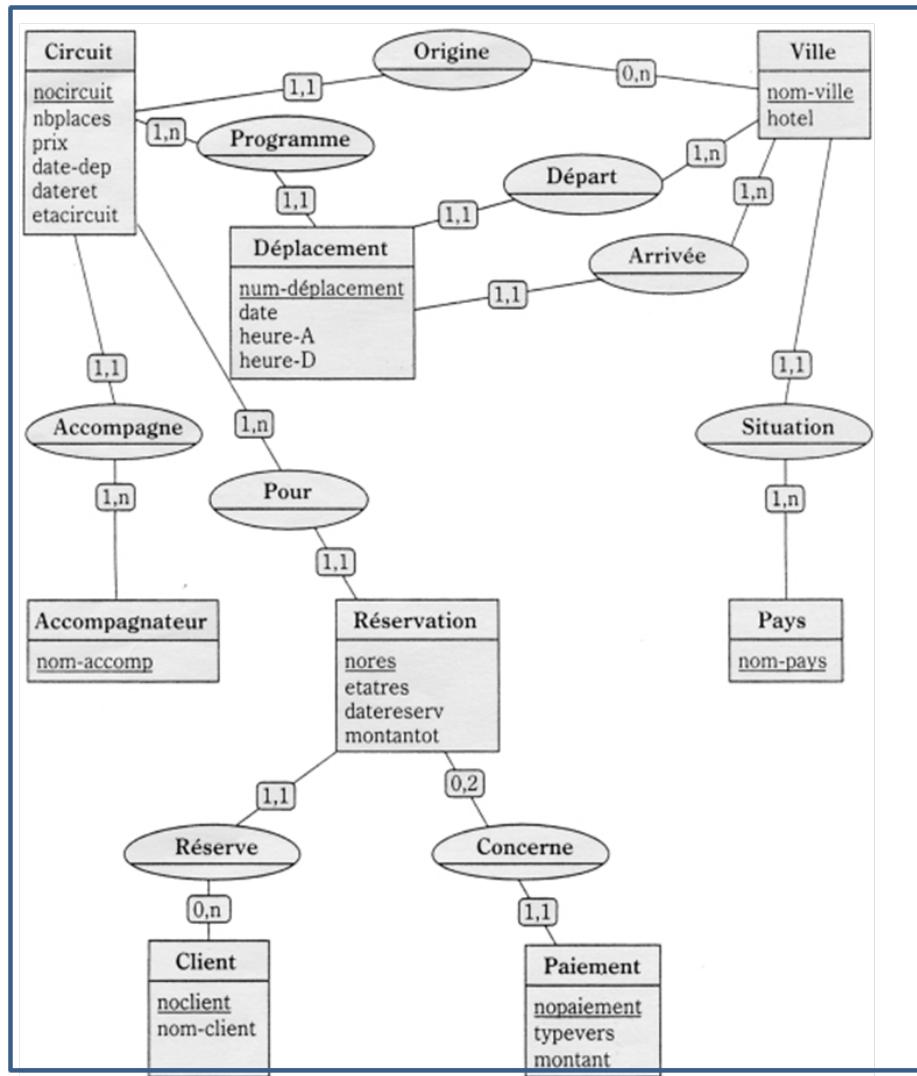


Figure 2.4: Schéma Entité-Association

potentiels).

Travail demandé :

1. Reformuler l'énoncé du problème sous la forme de règles claires. Formuler des hypothèses, si nécessaire, pour compléter les informations manquantes.
2. Réaliser le modèle E-A du problème.

Correction de l'Exercice 2

1. • Il y a des logements qui ont un type, un nombre de pièces, une surface, un état, un objectif de gestion, un prix, une disponibilité, une date de disponibilité.

Les logements sont proposés à la location ou à la vente par des propriétaires, possèdent des garages et sont situés à des adresses. On fait l'hypothèse supplémentaire que les logements sont codifiés, ce qui permettra de les identifier facilement.

- Il y a des garages, qui ont un type. Les garages appartiennent aux logements et sont situés à une adresse. On fait l'hypothèse que les garages ont un numéro, ce qui est indispensable pour identifier plusieurs garages d'un même logement (qui peuvent être situés à une même adresse).
- Il y a des adresses, qui ont un numéro de rue, une rue et une ville. Les adresses servent à situer un logement, un garage ou une personne. On fait l'hypothèse que l'agence n'est pas internationale.
- Il y a des personnes, qui ont un nom et un prénom. Les personnes vivent à une adresse.
- Il y a des propriétaires, qui sont des personnes. Les propriétaires louent ou vendent des logements.
- Il y a des clients, qui sont des personnes. Les clients font des visites et louent ou achètent des propriétés.
- Il y a des visites, qui ont une date. Les visites sont faites par des clients dans des logements.
- Il y a des transactions de location ou de vente, qui ont une date, une commission et un montant (le montant n'étant pas forcément égal au prix de mise en vente).
- Une transaction concerne un logement, un client et un propriétaire.

2. Figure 2.5

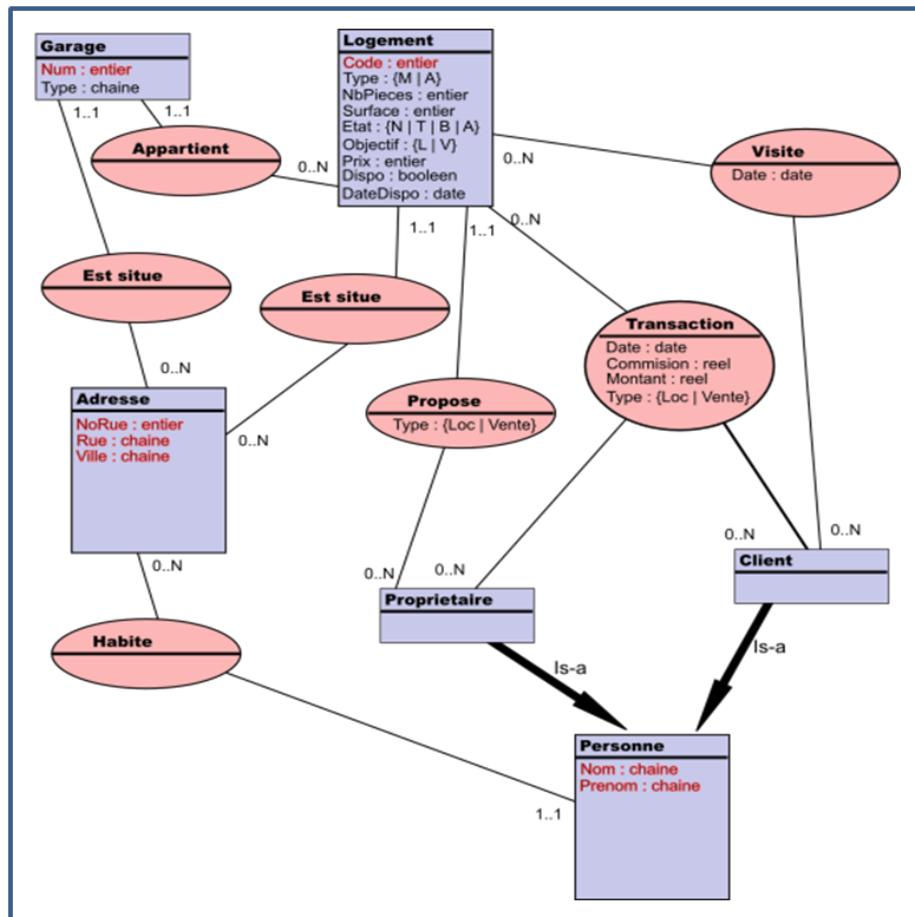


Figure 2.5: Schéma Entité-Association

Chapitre 3

Modélisation relationnelle

Introduction

Introduit en 1970 par Edgar F. Codd, le modèle relationnel a révolutionné la gestion des données en proposant une représentation logique reposant sur des relations (tables), simples à manipuler et indépendantes des considérations physiques. C'est aujourd'hui le modèle le plus utilisé dans les Systèmes de Gestion de Bases de Données (SGBD).

3.1 Schéma relationnel

Le **schéma relationnel** est l'ensemble des relations (tables) permettant de modéliser un domaine réel. Chaque relation représente soit :

- une **entité** du monde réel (ex : Client, Produit),
- une **association** entre entités (ex : Vente).

Exemple de transformation

- CLIENT(IdCli, Nom, Ville)
- PRODUIT(IdPro, Nom, Prix, Qstock)
- VENTE(IdCli, IdPro, Date, Qte)

Modèle E-A	Modèle Relationnel
Entité	Relation (table)
Occurrence d'une entité	Nuplet (ligne)
Attribut simple	Attribut atomique
Attribut composite	Ensemble d'attributs
Attribut multivalué	Relation séparée
Entité faible	Relation avec clé étrangère
Association 1:N	Ajout de clé étrangère
Association N:N	Nouvelle relation
Association >3 entités	Nouvelle relation
Occurrence d'association	Nuplet
Héritage	Vue ou table(s) spécialisée(s)

Table 3.1: Passage du modèle E-A au modèle relationnel

Représentation tabulaire

Table 3.2: Table CLIENT

IdCli	Nom	Ville
X	Mohamed	Tunis
Y	Karim	Tunis
Z	Yacine	Béja
A	Amine	Paris

Table 3.3: Table PRODUIT

IdPro	Nom	Prix	Qst
P	Auto	100	10
Q	Moto	100	10
R	Velo	100	10
S	Pedalo	100	10

Table 3.4: Table VENTE

IdCli	IdPro	Date	Qte
X	P	-	1
Y	P	-	4
Y	Q	-	5
Z	Q	-	6

3.2 Production du schéma relationnel

La production du schéma relationnel à partir d'un modèle conceptuel de données s'appuie sur un ensemble de règles de transformation. Ces règles permettent de convertir les entités, les associations et les cardinalités afin d'obtenir un modèle logique de données cohérent et exploitable au niveau d'un SGBD.

Les Figures 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7 et 3.8 illustrent progressivement ces principales règles de dérivation :

- la transformation d'un type entité en une table relationnelle (Figure 3.1);
- le traitement des associations de type « un à plusieurs » par la migration de clé étrangère (Figure 3.2);

- la gestion des clés étrangères multi-composants (Figure 3.3);
- la migration de la clé étrangère selon la cardinalité dominante (Figure 3.4);
- la conversion des associations « un à un » (Figure 3.5);
- la création d'une table associative dans le cas d'une relation « plusieurs à plusieurs » (Figure 3.6);
- la modélisation d'associations réflexives ou cycliques « un à plusieurs » (Figure 3.7);
- la modélisation d'associations réflexives « plusieurs à plusieurs » (Figure 3.8).

Ces transformations garantissent une structure logique conforme aux contraintes du MCD tout en assurant l'intégrité des données dans la base relationnelle.

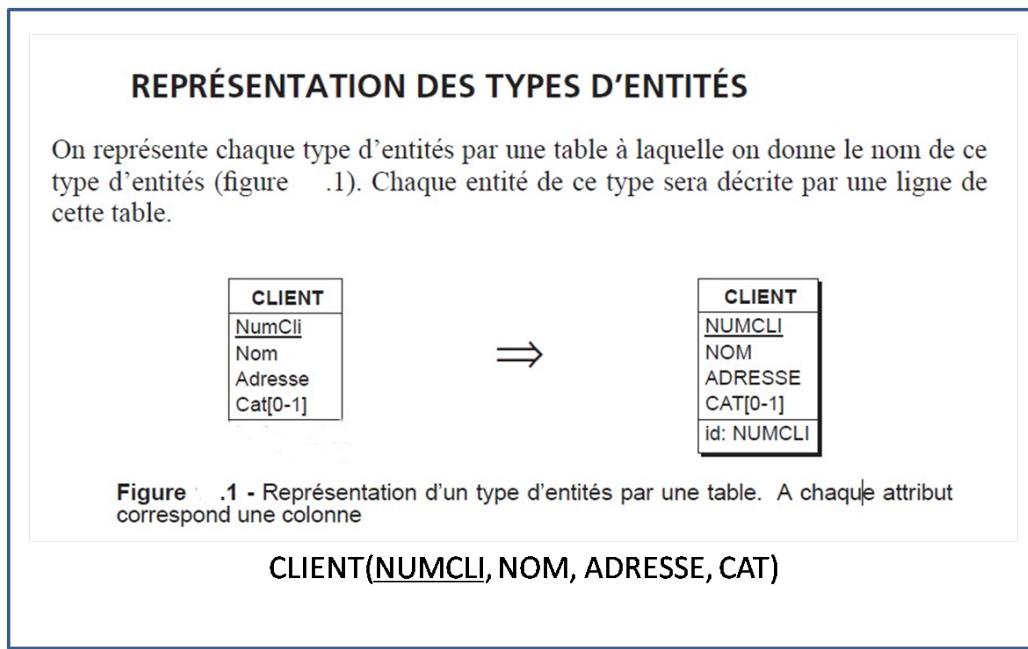


Figure 3.1: Type entité → Table

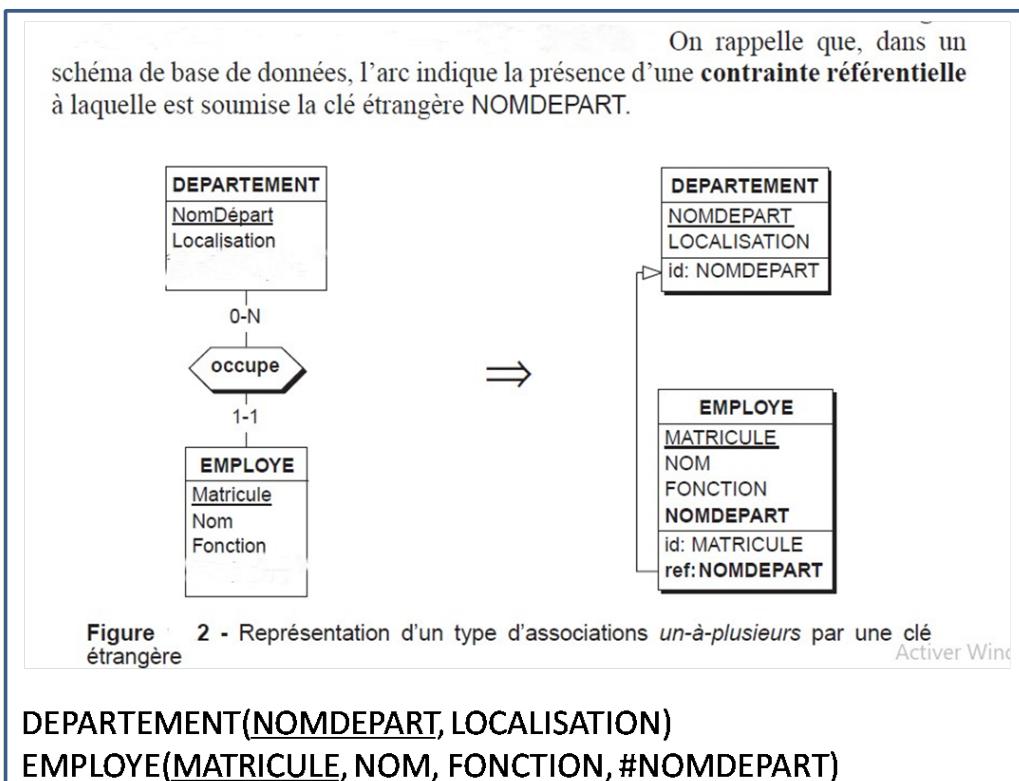


Figure 3.2: Association un à plusieurs → Clé étrangère

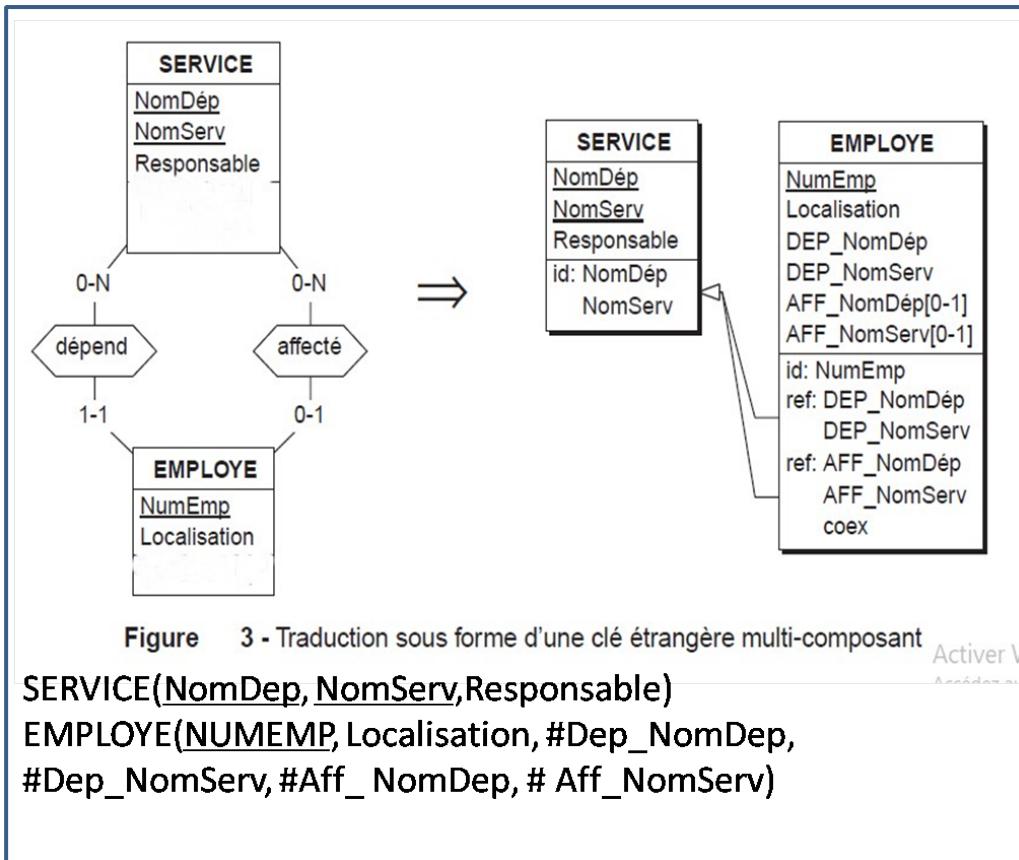


Figure 3.3: Clé étrangère multi-composant

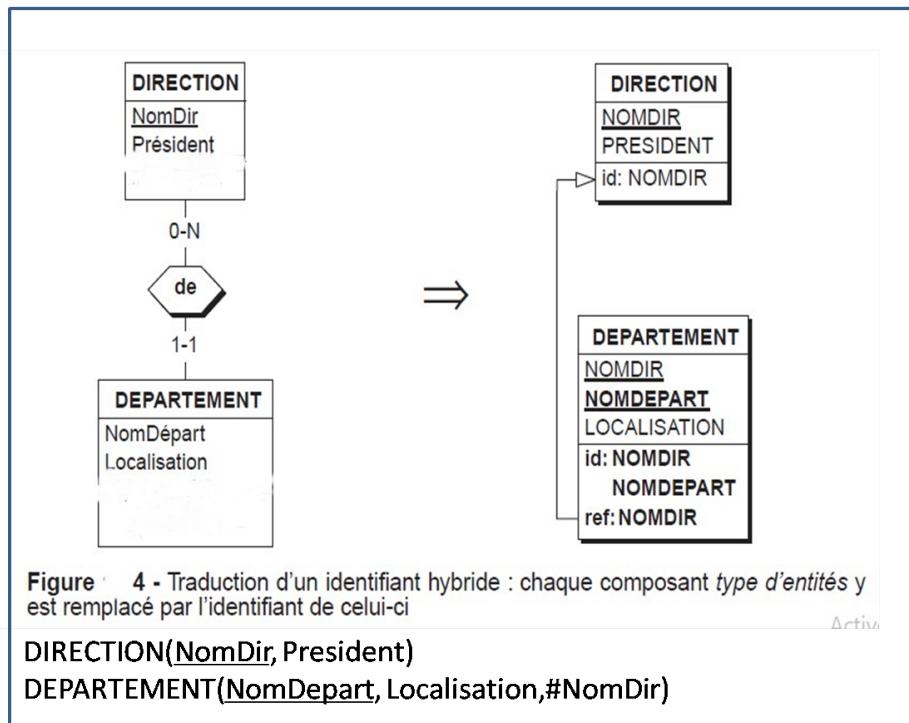


Figure 3.4: Migration de la clé étrangère

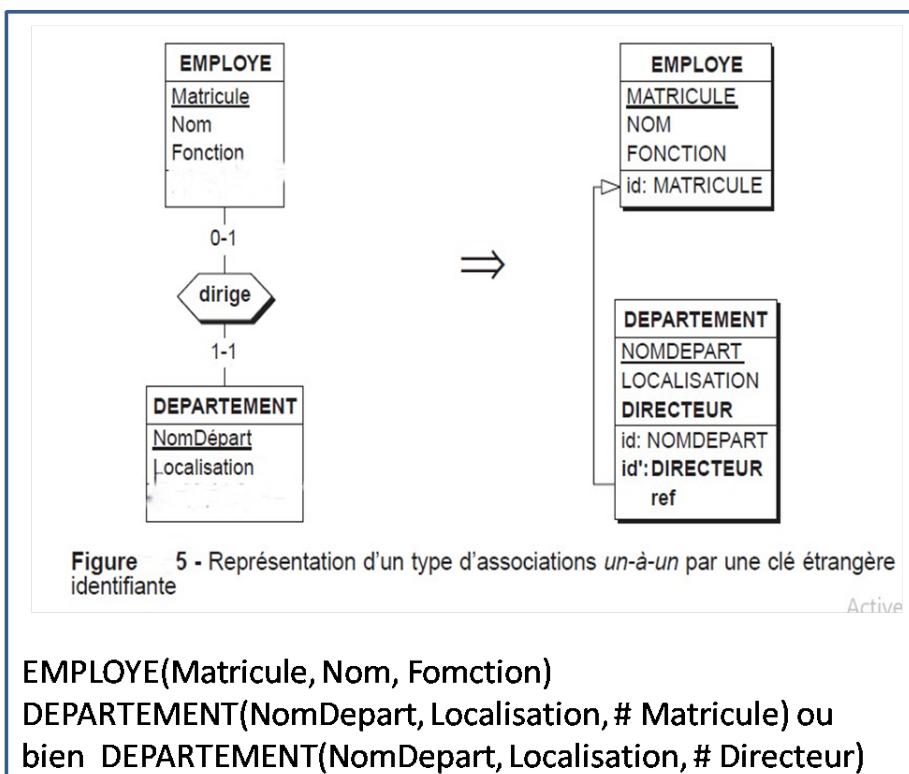


Figure 3.5: Association un à un par clé étrangère

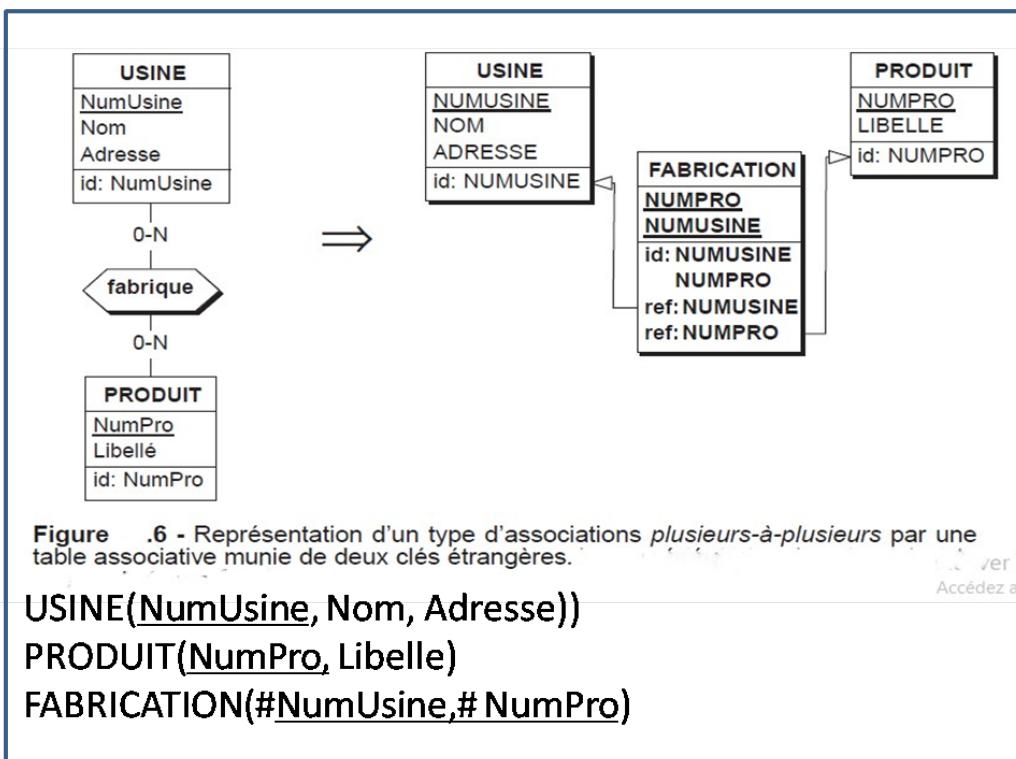


Figure 3.6 - Représentation d'un type d'associations plusieurs-à-plusieurs par une table associative munie de deux clés étrangères.

USINE(NumUsine, Nom, Adresse))
PRODUIT(NumPro, Libelle)
FABRICATION(#NumUsine,# NumPro)

Figure 3.6: Association plusieurs à plusieurs→ création d'une relation

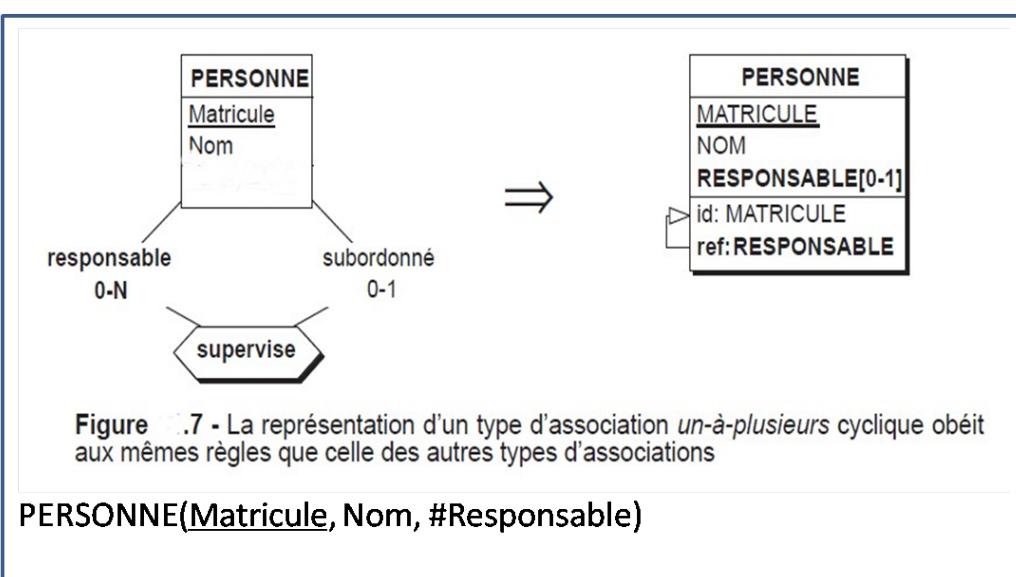


Figure 3.7 - La représentation d'un type d'association un-à-plusieurs cyclique obéit aux mêmes règles que celle des autres types d'associations

PERSONNE(Matricule, Nom, #Responsable)

Figure 3.7: Modélisation d'une association cyclique ou réflexive un à plusieurs

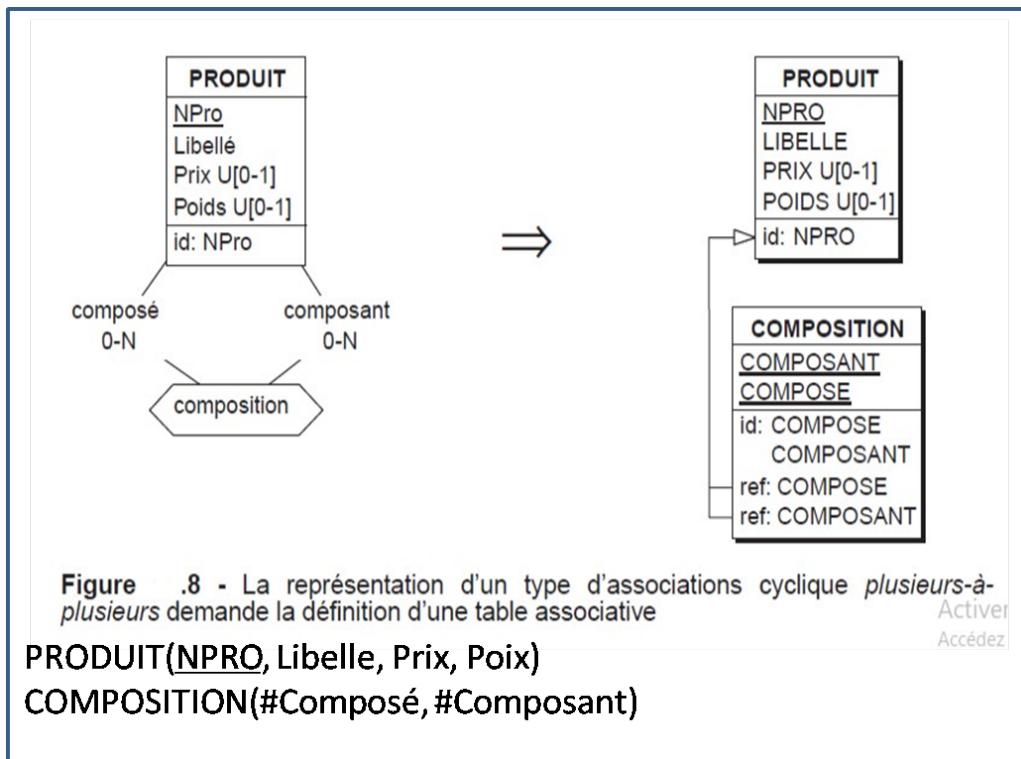


Figure 3.8: Modélisation d'une association cyclique ou réflexive plusieurs à plusieurs

3.3 Synthèse des règles de traduction

Les figures 3.9, 3.10, 3.11 et 3.12 constituent un récapitulatif des principales règles de traduction d'un schéma conceptuel en structures de tables.

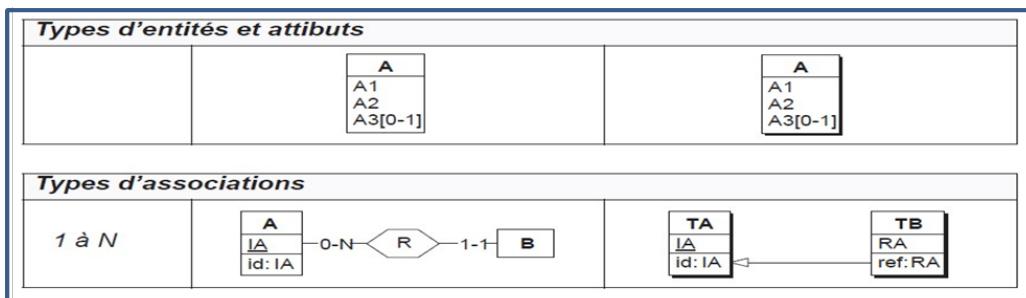


Figure 3.9: Entité→ Table; Migration clé 1à N

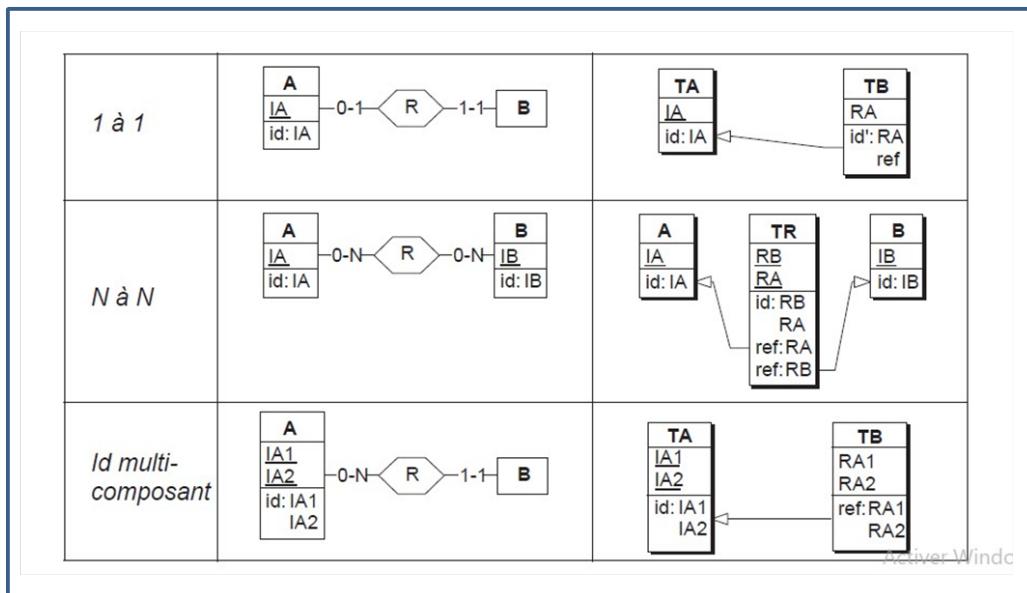


Figure 3.10: Migration clé 1 à 1; N à N; Multi-composant

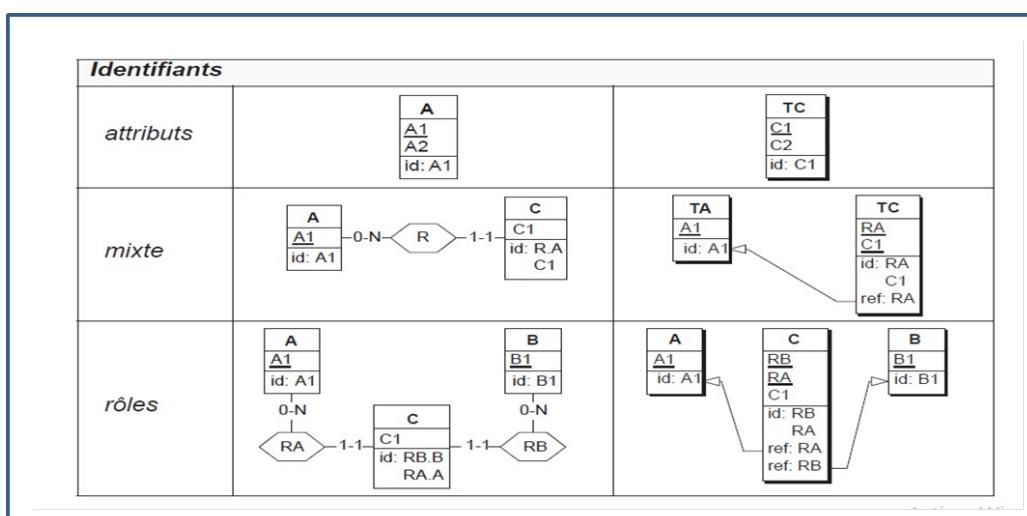


Figure 3.11: Attributs; Mixte, Rôles

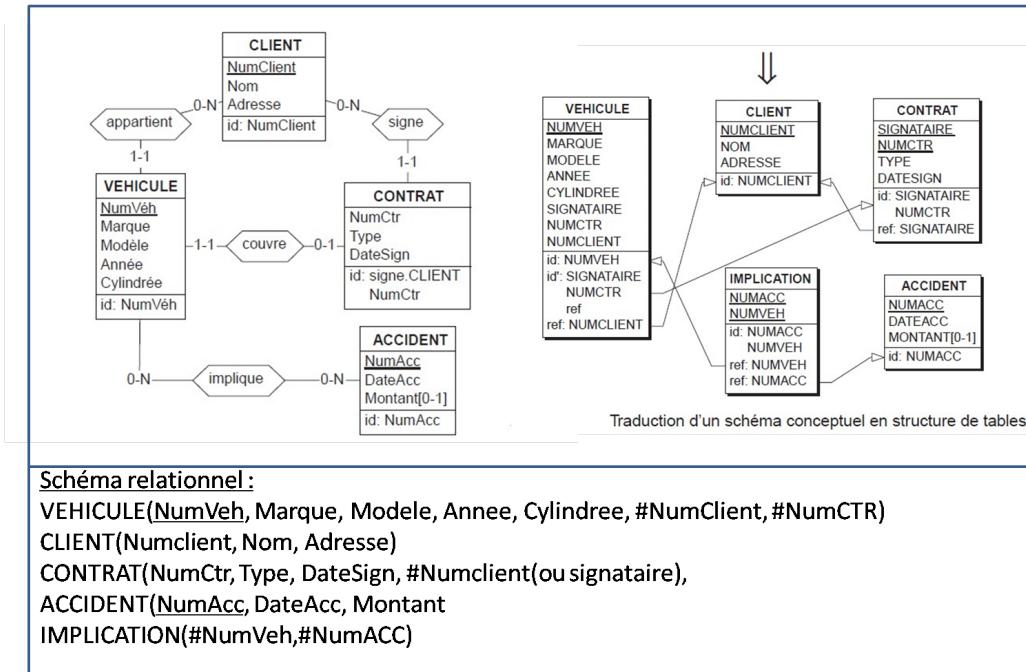


Figure 3.12: Exemple

3.4 Avantages du modèle relationnel

Le modèle relationnel s'est imposé comme standard dans le monde des SGBD pour plusieurs raisons:

Avantages majeurs

- **Simplicité** : représentation intuitive sous forme de tables.
- **Puissance des opérations** : algèbre relationnelle et SQL.
- **Indépendance physique** : optimisation gérée par le SGBD.
- **Indépendance logique** : utilisation de vues.
- **Maintien de l'intégrité** : contraintes au niveau du schéma.
- **Langage universel** : SQL, normalisé par l'ISO/IEC 9075, assure la portabilité entre SGBD.

3.5 Limites du modèle relationnel

Malgré ses atouts, le modèle relationnel présente certaines limites :

Limites

- Difficulté à gérer des données **semi-structurées ou non structurées** (ex. JSON, multimédia).
- Problèmes de **scalabilité horizontale** pour les très grands volumes de données.
- Complexité accrue dans les systèmes hautement distribués.

Ces limites ont conduit à l'émergence des systèmes **NoSQL**, tout en conservant le modèle relationnel comme base des systèmes transactionnels (OLTP).

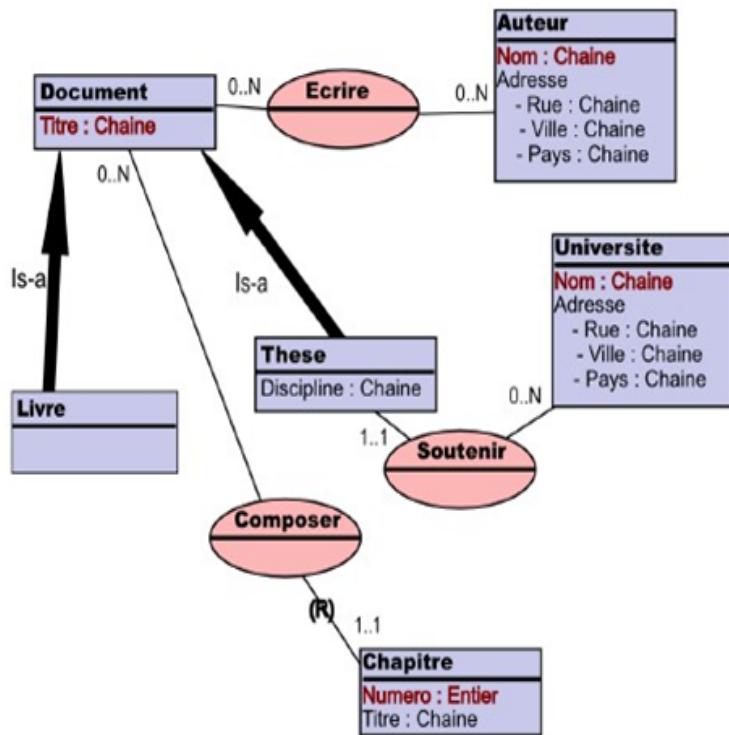
Conclusion

Le modèle relationnel permet une représentation logique claire et normalisée des données. Il est indépendant de la manière dont les données sont physiquement stockées et constitue la base des systèmes transactionnels modernes.

Exercices corrigés

Exercice 1

Soit le schéma E-A suivant :



Travail demandé: Quel est le modèle relationnel qui correspondent à ce modèle E-A ?

Correction

Listing 3.1: Modèle relationnel des documents

```

1 -- Relations
2 Auteur(#Nom:Chaine , AdRue:Chaine , AdVille:Chaine , AdPays:
      Chaine)
3 Universite(#Nom:Chaine , AdRue:Chaine , AdVille:Chaine , AdPays
           :Chaine)
4
  
```

```

5      Document(#Titre:Chaine, Discipline:Chaine, Univ=>Universite,
6          Type:{These|Livre})
7
8      Chapitre(#Numero:Entier, #TitreDoc=>Document, TitreChap:
9          Chaine)
10
11     AuteursDocs(#Titre=>Document, #Nom=>Auteur)

```

Exercice 2: La gestion d'une compagnie touristique

Travail demandé: Transformer le modèle E-A du chapitre précédent en modèle relationnel.

Correction

Listing 3.2: Modèle relationnel du système de gestion de circuits touristiques

```

1      -- Relations
2
3      Pays(#nom-pays)
4
5      Ville(#nom-ville, hotel, nom-pays=>Pays(nom-pays))
6      WITH nom-pays NOT NULL
7
8      Accompagnateur(#nom-accomp)
9
10
11     Circuit(#nocircuit, nbplaces, prix,
12             nom-ville=>Ville(nom-ville),
13             date-dep, daterep, etatcircuit,
14             nom-accomp=>Accompagnateur(nom-accomp))
15     WITH nom-accomp, nom-ville NOT NULL
16
17     Deplacement(nocircuit=>Circuit(nocircuit), #num-deplacement,
18                 date, heure-A, heure-D,

```

```

18    villedepart=>Ville(nom-ville),
19    villearrivee=>Ville(nom-ville))
20    WITH nocircuit, villedepart, villearrivee NOT NULL
21
22    Client(#noclient, nomclient)
23
24    Reservation(#nores, etatres, datereserv, montantot,
25    nocircuit=>Circuit(nocircuit),
26    noclient=>Client(noclient))
27    WITH noclient, nocircuit NOT NULL
28
29    Paiement(#nopaiement, typevers, montant,
30    nores=>Reservation(nores))
31    WITH nores NOT NULL
32
33    -- Contraintes d intégrité référentielles
34    PROJ(Pays, nom-pays) IN PROJ(Ville, nom-pays)
35    AND PROJ(Accompagnateur, nom-accomp) IN PROJ(Circuit, nom-
36        accomp)
37    AND PROJ(Circuit, nocircuit) IN PROJ(Deplacement, nocircuit)
38    AND PROJ(Ville, nom-ville) IN PROJ(Deplacement, villedepart)
39    AND PROJ(Ville, nom-ville) IN PROJ(Deplacement, villearrivee
        )
      AND PROJ(Circuit, nocircuit) IN PROJ(Reservation, nocircuit)

```

Exercice2 : Agence immobilière

Travail demandé: Transformer le modèle E-A du chapitre précédent en modèle relationnel.

Correction

Listing 3.3: Modèle relationnel du système de gestion immobilière

```
1      -- Relations  
2  
3      LOGEMENT(CODE , TYPE , NBPIECES , SURFACE , ETAT , OBJECTIF , PRIX  
4          , DISPO , DATEDISPO ,  
5          #NORUE=>ADRESSE , #RUE=>ADRESSE , #VILLE=>ADRESSE ,  
6          #NOM=>PROPRIETAIRE , #PRENOM=>PROPRIETAIRE)  
7  
8      GARAGE(NUM , #CODE=>LOGEMENT , TYPE ,  
9          #NORUE=>ADRESSE , #RUE=>ADRESSE , #VILLE=>ADRESSE)  
10  
11  
12      ADRESSE(NORUE , RUE , VILLE)  
13  
14  
15      PROPRIETAIRE(NOM , PRENOM ,  
16          #NORUE=>ADRESSE , #RUE=>ADRESSE , #VILLE=>ADRESSE)  
17  
18  
19      CLIENT(NOM , PRENOM ,  
20          #NORUE=>ADRESSE , #RUE=>ADRESSE , #VILLE=>ADRESSE)  
21  
22  
23      VISITE(#CODE=>LOGEMENT ,  
24          #NOM=>CLIENT , #PRENOM=>CLIENT ,  
25          #NORUE=>CLIENT , #RUE=>CLIENT , #VILLE=>CLIENT ,  
26          DATE)  
27  
28  
29      TRANSACTION(#CODE=>LOGEMENT ,  
30          #NOMP=>PROPRIETAIRE , #PRENOMP=>PROPRIETAIRE ,  
31          #NORUEP=>PROPRIETAIRE , #RUEP=>PROPRIETAIRE , #VILLEP=>  
32          PROPRIETAIRE ,  
33          #NOMC=>CLIENT , #PRENOMC=>CLIENT ,  
34          #NORUEC=>CLIENT , #RUEC=>CLIENT , #VILLEC=>CLIENT ,  
35          DATE , COMMISSION , MONTANT , TYPE)
```

Chapitre 4

Bases de données relationnelles

Introduction

Dans ce chapitre, nous allons introduire les principes fondamentaux des bases de données relationnelles, proposé par E. F. Codd en 1970, sur lequel reposent les SGBD modernes comme PostgreSQL, MySQL ou Oracle. Nous allons présenté les notions fondamentales du modèle relationnel, ainsi que les dépendances fonctionnelles, les règles d'intégrité et les formes normales.

4.1 Les concepts fondamentaux

Le modèle relationnel repose sur quelques concepts fondamentaux:

Domaine

Un domaine est un ensemble de valeurs atomiques d'un type sémantique donné.

- NOM_VILLE = {Tunis, Paris, Rome}
- NUM_ELV = {1,2,...,2000}
- NUM_ANNEE = {1,2,...,2000}

Relation

Une relation est un sous-ensemble du produit cartésien :

$$R \subseteq D_1 \times D_2 \times \cdots \times D_n$$

- ELEVE \subseteq NOM_ELV \times PREN_ELV \times DATE_NAISS
- INSCRIPT \subseteq NOM_ELV \times NOM_SPORT
- TRAJET \subseteq NOM_VILLE \times NOM_VILLE

n-uplets (tuples)

Un n -uplet est un élément d'une relation \rightarrow représente un fait.

Attributs

Chaque composante d'une relation est un attribut.

Clé primaire

identifiant unique d'un tuple dans une relation.

Clé étrangère

attribut qui établit un lien logique entre deux relations.

Exemple :

ID_Client	Nom	Ville
C001	Dupont	Paris
C002	Martin	Lyon

Ici, la table CLIENT est une relation. L'attribut ID_Client constitue la clé primaire.

Schéma relationnel

Le **schéma relationnel** est la description formelle de la structure de la base. Il définit les tables, leurs attributs, les clés et les relations entre elles.

- 1 relation → 1 table
- 1 attribut → 1 colonne
- 1 n-uplet → 1 ligne

Exemple :

CLIENT(*ID_Client*,*Nom*,*Ville*)

COMMANDE(*ID_Commande*,*DateCommande*,*ID_Client*)

où la clé primaire est soulignée et la clé étrangère (*ID_Client*) relie la table COMMANDE à CLIENT.

4.2 Les contraintes d'intégrité

Les contraintes d'intégrité assurent la cohérence logique des données dans une base relationnelle:

- **Intégrité de clé :** chaque table doit avoir une clé primaire unique et non nulle.
- **Intégrité référentielle :** une clé étrangère doit toujours référencer une clé primaire existante.
- **Intégrité de domaine :** les valeurs d'un attribut doivent appartenir à un domaine valide.

Exemple SQL :

```

1      CREATE TABLE Client (
2          ID_Client CHAR(4) PRIMARY KEY ,
3          Nom VARCHAR(50) ,
4          Ville VARCHAR(30)
5      );
6
7      CREATE TABLE Commande (
8          ID_Commande CHAR(5) PRIMARY KEY ,
9          DateCommande DATE ,
10         ID_Client CHAR(4) ,
11         FOREIGN KEY (ID_Client) REFERENCES Client(ID_Client)
12     );

```

4.3 Les dépendances fonctionnelles

Soit $R(A_1, \dots, A_n)$ et $X, Y \subseteq \{A_1, \dots, A_n\}$:

$X \rightarrow Y \Rightarrow$ à chaque valeur de X correspond une valeur unique de Y

Définition

Une **dépendance fonctionnelle** $A \rightarrow B$ signifie que la valeur de l'attribut A détermine **uniquement** la valeur de l'attribut B .

Clé

Un attribut ou ensemble minimal d'attributs déterminant tous les autres.

- Plusieurs clés possibles \rightarrow clés candidates
- Une seule clé choisie \rightarrow clé primaire

Clé étrangère

Référence à une clé primaire d'une autre relation.

- PRODUIT(#no_cat) réfère à CATEG(no_cat)

4.4 La normalisation

L'objectif de la normalisation est de réduire la redondance et d'éviter les anomalies de mise à jour d'insertion ou de suppression.

La normalisation permet:

- Éliminer les données répétitives
- Créer plusieurs tables reliées par des FK

Première forme normale (1NF)

- Chaque attribut contient des valeurs atomiques.
- Les colonnes répétitives ou multi-valeurs sont séparées.

Exemple avant 1NF :

```
1 Commande(ID_Commande, ID_Client, Produits)
2   1, 101, [Laptop, Souris]
3   2, 102, [Clavier]
```

Après 1NF :

```
1 Commande(ID_Commande, ID_Client, Produit)
2   1, 101, Laptop
3   1, 101, Souris
4   2, 102, Clavier
```

Deuxième forme normale (2NF)

- Être en 1NF.
- Tous les attributs non-clés doivent dépendre de la clé primaire entière.

Exemple :

1 CommandeProduit (ID_Commande , ID_Produit , NomProduit , Prix)

Ici, NomProduit et Prix dépendent uniquement de ID_Produit, pas de la combinaison (ID_Commande, ID_Produit). **Solution :** créer une table Produit séparée et ne garder que ID_Produit dans CommandeProduit.

Troisième forme normale (3NF)

- Être en 2NF.
- Aucun attribut non-clé ne dépend d'un autre attribut non-clé (pas de dépendance transitive).

Exemple :

1 Commande (ID_Commande , ID_Client , Ville)

Ville dépend de ID_Client et non de ID_Commande → violation 3NF. **Solution :** supprimer Ville de Commande et conserver cette information dans la table Client.

Conclusion

La modélisation relationnelle et la normalisation permettent:

- D'assurer la cohérence et l'intégrité des données,
- D'éviter les anomalies et redondances,
- De préparer un schéma relationnel prêt à être implémenté dans un SGBD relationnel.

La maîtrise de la normalisation est essentielle pour concevoir des bases de données robustes et efficaces.

Exercices corrigés

Exercice 1

Soit la relation $R(film, metteur-en-scene, acteur, cinema, salle, seance)$.

1. Interprétations en français des DF données :

- $Cinema, salle, seance \rightarrow film$.

Dans un même cinéma, pour une salle donnée et une séance donnée, le film projeté est déterminé (il n'y a qu'un film par combinaison cinéma+salle+séance).

- $film \rightarrow metteur-en-scene$.

Un film a un metteur en scène unique (le film détermine son metteur en scène).

- $film, cinema \rightarrow salle$.

Dans un cinéma donné, un film est projeté dans une salle déterminée (la paire film+cinéma détermine la salle).

2. DF correspondant aux interprétations demandées :

- «Il ne passe pas 2 films différents projetés durant la même séance d'un même cinéma.»

DF : $cinema, seance \rightarrow film$.

- «Un acteur ne peut jouer dans 2 films différents projetés durant la même séance d'un même cinéma.»

DF : $cinema, seance, acteur \rightarrow film$.

Exercice 2

Relations :

Bureau(NumBureau, NumTelephone, Taille)

Occupant(NumBureau, PersonneID)

Materiel(NumBureau, NumPC)

DF fournies :

$$\text{DF}_{Bureau} = \{ \text{NumBureau} \rightarrow \text{NumTelephone}, \text{Taille}; \quad \text{NumTelephone} \rightarrow \text{NumBureau} \}$$

$$\text{DF}_{Occupant} = \{ \text{NumBureau} \rightarrow \text{PersonneID} \}$$

$$\text{DF}_{Materiel} = \{ \text{NumPC} \rightarrow \text{NumBureau} \}$$

1. «Un bureau peut contenir plusieurs postes téléphoniques..»

Vérification : Non, la DF $\text{NumBureau} \rightarrow \text{NumTelephone}$ impose qu'à chaque bureau correspond un seul numéro de téléphone. De plus $\text{NumTelephone} \rightarrow \text{NumBureau}$ et la précédente impliquent une correspondance bijective (1–1).

Correction pour que la contrainte soit vérifiée : Il faut **supprimer** la DF $\text{NumBureau} \rightarrow \text{NumTelephone}$. Conserver $\text{NumTelephone} \rightarrow \text{NumBureau}$ (chaque téléphone appartient à un bureau) permet à un même bureau d'avoir plusieurs téléphones (plusieurs NumTelephone associés au même NumBureau).

2. «Il y a une et une seule personne par bureau.»

Vérification : Oui. $\text{NumBureau} \rightarrow \text{PersonneID}$ dans $\text{DF}_{Occupant}$ garantit qu'à un NumBureau correspond exactement une PersonneID (une personne par bureau).

3. «Un bureau contient un seul ordinateur.»

Vérification : Non. La DF donnée est $\text{NumPC} \rightarrow \text{NumBureau}$ (chaque PC est dans un bureau), cela n'empêche pas qu'un bureau contienne plusieurs PC.

Correction pour que la contrainte soit vérifiée : Ajouter $\text{NumBureau} \rightarrow \text{NumPC}$ (ou remplacer la DF existante par une bijection $\text{NumBureau} \leftrightarrow \text{NumPC}$) afin d'imposer qu'un bureau ait un et un seul PC.

2. Clés minimales possibles :

- **Bureau** : d'après les DF, $\text{NumBureau} \rightarrow \text{NumTelephone}$, Taille et $\text{NumTelephone} \rightarrow \text{NumBureau}$. Ainsi NumBureau est une clé ; NumTelephone est aussi une clé (les deux sont candidats et minimaux).
- **Occupant** : $\text{NumBureau} \rightarrow \text{PersonneID}$ donc NumBureau est clé minimale de Oc-

cupant.

- **Materiel** : NumPC → NumBureau donc NumPC est clé minimale de Materiel.

Exercice 3

Relation $R(\text{UtilisateurID}, \text{Nom}, \text{Prénom}, \text{AdresseEmail}, \text{Login}, \text{Passwd}, \text{ServeurMail})$.

1. DF exprimant les contraintes :

- (a) «On peut déduire le nom et le prénom d'un utilisateur à partir de son identificateur.»

$$\text{UtilisateurID} \rightarrow \text{Nom}, \text{Prénom}.$$

- (b) «Un utilisateur (identifié par son identificateur) possède un seul login et un seul password par serveur de mails.»

$$(\text{UtilisateurID}, \text{ServeurMail}) \rightarrow \text{Login}, \text{Passwd}.$$

- (c) «Une adresse email est associée à un et un seul identificateur d'utilisateur. (Un utilisateur peut avoir plusieurs adresses.)»

$$\text{AdresseEmail} \rightarrow \text{UtilisateurID}.$$

- (d) «Une adresse email est associée à un et un seul serveur de mails.»

$$\text{AdresseEmail} \rightarrow \text{ServeurMail}.$$

2. Clés minimales de R :

Calcul de fermeture : $\text{AdresseEmail}^+ = \{\text{AdresseEmail} \Rightarrow \text{UtilisateurID}, \text{ServeurMail}\};$

par $\text{UtilisateurID} \rightarrow \text{Nom}, \text{Prénom}$ on obtient $\text{Nom}, \text{Prénom}$;

par $(\text{UtilisateurID}, \text{ServeurMail}) \rightarrow \text{Login}, \text{Passwd}$ on obtient $\text{Login}, \text{Passwd}$. Donc AdresseEmail^+

contient tous les attributs : AdresseEmail est une clé minimale.

D'autres combinaisons minimales n'apparaissent pas (par ex. UtilisateurID seul ne permet pas d'obtenir AdresseEmail ni Login/Passwd), donc la clé minimale est :

{ AdresseEmail }.

3. Forme normale de R :

La clé est AdresseEmail. On a AdresseEmail → UtilisateurID et UtilisateurID → Nom,Prénom. Ainsi il existe une dépendance transitive :

AdresseEmail → UtilisateurID → Nom,Prénom,

donc AdresseEmail → Nom,Prénom indirectement. Cela constitue une *dépendance transitive* d'un attribut clé vers des attributs non-primes via un attribut non-prime (UtilisateurID), donc la relation **n'est pas en 3NF**. Elle est en revanche en 2NF (clé simple) mais **pas en 3NF**.

Remarque de normalisation (décomposition recommandée) :

Pour obtenir la 3NF/BCNF, on peut décomposer en :

Utilisateur(UtilisateurID, Nom, Prénom)

Email(AdresseEmail, UtilisateurID, ServeurMail, Login, Passwd)

Cette décomposition élimine la transitive et est sans perte d'information (décomposition en 3NF/BCNF selon les FDs souhaitées).

Exercice 4

Relation Commande(numCommande, numClient, nomClient, date, numProduit, nomProduit)
avec DF :

DF1 : numCommande → numClient, date, numProduit

DF2 : numClient → nomClient

DF3 : numProduit → nomProduit

1. Redondances et anomalies :

- *Redondances* : nomClient est répété pour chaque tuple ayant le même numClient ; nomProduit est répété pour chaque tuple ayant le même numProduit.
- *Anomalies d'insertion* : pour insérer un client (numClient, nomClient) sans commande il faut une table client séparée, sinon on ne peut pas ajouter le client indépendant d'une commande.
- *Anomalies de mise à jour* : si le nom d'un client change, il faut mettre à jour toutes les lignes où apparaît ce numClient (risque d'incohérence).
- *Anomalies de suppression* : la suppression de la dernière commande d'un client supprimerait l'information sur le client si elle n'existe que dans cette relation.

2. Clé(s) :

D'après DF1, numCommande détermine numClient, date, numProduit. Ainsi, avec les DF fournies, numCommande détermine tous les autres attributs et est donc **clé candidate minimale** de la relation. (Remarque : DF1 implique qu'une commande contient exactement un produit ; si en pratique une commande peut contenir plusieurs produits, DF1 est alors incorrecte et la clé serait (numCommande, numProduit).)

3. Forme normale :

- 1NF : respectée (attributs atomiques).
- 2NF : triviale car la clé est simple (numCommande), donc 2NF est respectée.

- 3NF : **non respectée** : on a des dépendances non-triviales $\text{numClient} \rightarrow \text{nomClient}$ et $\text{numProduit} \rightarrow \text{nomProduit}$ où numClient et numProduit sont *non-clés* déterminés par la clé numCommande ; ces DF introduisent des dépendances transitives clef \rightarrow attribut non-prime \rightarrow autre attribut non-prime, donc violation de 3NF (et donc pas BCNF).

4. Proposition de décomposition en BCNF (sans perte d'information):

Deux cas selon l'interprétation de DF1 :

Cas A — DF1 considérée exacte (une commande comporte exactement un produit). On conserve la sémantique donnée par les DF. Décomposer en:

Commande($\text{numCommande}, \text{numClient}, \text{date}, \text{numProduit}$)
 Client($\text{numClient}, \text{nomClient}$)
 Produit($\text{numProduit}, \text{nomProduit}$)

Vérification :

- $\text{numClient} \rightarrow \text{nomClient}$ est préservée dans Client.
- $\text{numProduit} \rightarrow \text{nomProduit}$ est préservée dans Produit.
- Commande a pour clé numCommande ; aucune dépendance non triviale violent BCNF n'y reste (si on a seulement $\text{numCommande} \rightarrow$ autres attributs).
- Cette décomposition est *sans perte d'information* (joindre Commande avec Client et Produit restitue la relation initiale) et conserve les dépendances fonctionnelles données.

Cas B — Si en réalité une commande peut contenir plusieurs produits (DF1 est incorrecte): Il faut alors supposer que chaque ligne de commande est identifiée

par (numCommande, numProduit). Décomposition recommandée :

Commande(numCommande, numClient, date)
LigneCommande(numCommande, numProduit)
Client(numClient, nomClient)
Produit(numProduit, nomProduit)

- Cette décomposition est en BCNF (chaque relation a ses DF triviales ou les déterminants sont clés).
- Elle est sans perte d'information (jointure naturelle reconstruira les lignes).
- *Remarque de dépendances* : dans ce schéma on ne préserve plus la DF erronée $\text{numCommande} \rightarrow \text{numProduit}$ (qui ne devrait pas exister si des commandes ont plusieurs produits) — mais on préserve DF2 et DF3.

Si on respecte strictement les DF fournies (DF1 affirme qu'une commande a un seul produit), la décomposition du **Cas A** est correcte et conduit à la BCNF en conservant les dépendances. Si la réalité autorise plusieurs produits par commande, il faut utiliser le **Cas B**, qui est la décomposition normale en BCNF pour un modèle commande/lignes de commande.

Chapitre 5

Algèbre relationnelle

Introduction

L'algèbre relationnelle est un langage théorique permettant de décrire les opérations de manipulation des données dans une base de données relationnelle. Elle constitue un cadre formel pour définir et transformer les requêtes, et représente ainsi la base conceptuelle des langages déclaratifs comme SQL.

L'objectif de ce chapitre est de comprendre les opérations fondamentales sur les relations, de formaliser les requêtes avant leur implémentation en SQL, et de fournir les outils nécessaires pour raisonner sur les requêtes et optimiser leur exécution.

5.1 Opérations fondamentales

Sélection (σ)

Filtrer les lignes d'une relation selon une condition.

$$\sigma_{condition}(Relation)$$

Exemple : Tous les étudiants inscrits au cours 101

$$\sigma_{ID_Cours=101}(Inscription)$$

Projection (π)

Sélectionner certaines colonnes (attributs) d'une relation.

$$\pi_{attributs}(Relation)$$

Exemple : Nom et prénom des étudiants

$$\pi_{Nom, Prenom}(Etudiant)$$

Union (\cup)

Combiner deux relations compatibles (mêmes attributs).

$$Relation_1 \cup Relation_2$$

Exemple : Clients de deux boutiques différentes

Différence (-)

Différence entre deux relations compatibles.

$$Relation_1 - Relation_2$$

Exemple : Clients qui n'ont pas passé de commande

Produit cartésien (\times)

Toutes les combinaisons possibles de deux relations.

$$Relation_1 \times Relation_2$$

Exemple : Coupler chaque étudiant avec chaque cours

5.2 Opérations de jointure

Jointure naturelle (\bowtie)

Combine deux relations en utilisant les attributs communs.

$$Etudiant \bowtie Inscription$$

Exemple : Nom des étudiants inscrits au cours 101

$$\pi_{Nom, Prenom}(\sigma_{ID_Cours=101}(Etudiant \bowtie Inscription))$$

Jointure theta (\bowtie_θ)

Jointure avec une condition spécifique.

$$Relation_1 \bowtie_\theta Relation_2$$

Exemple : Tous les produits dont le prix est supérieur à 50 et vendus dans une commande

Division (\div)

Trouver les entités d'une relation qui satisfont toutes les conditions d'une autre relation.

Exemple : Étudiants inscrits à tous les cours proposés

Exemples:

Soient les relations:

$$Etudiant(ID_Etudiant, Nom, Prenom)$$

$$Inscription(ID_Etudiant, ID_Cours)$$

$\text{Cours}(ID_{\text{Cours}}, Nom_{\text{Cours}})$

1- Sélection: Donner tous les étudiants inscrits au cours 101

$\sigma_{ID_{\text{Cours}}=101}(\text{Inscription})$

2- Projection: Donner les noms des étudiants

$\pi_{Nom, Prenom}(\text{Etudiant})$

3- Jointure: Noms des étudiants inscrits au cours 101

$\pi_{Nom, Prenom}(\text{Etudiant} \bowtie \sigma_{ID_{\text{Cours}}=101}(\text{Inscription}))$

4- Division: Trouver les étudiants inscrits à tous les cours.

$\pi_{ID_{\text{Etudiant}}}(\text{Inscription}) \div \pi_{ID_{\text{Cours}}}(\text{Cours})$

Conclusion

L'algèbre relationnelle représente le fondement théorique des systèmes de gestion de bases de données relationnelles. Elle repose sur un ensemble d'opérateurs permettant de manipuler les relations de manière formelle. Grâce à la sélection, la projection, la jointure, l'union, la différence ou la division, il devient possible d'exprimer des requêtes complexes de façon structurée et logique. La compréhension de ces notions est essentielle pour appréhender le fonctionnement des langages de requête tels que SQL et concevoir des requêtes performantes. Par conséquent, l'algèbre relationnelle constitue une étape clé dans la modélisation et la mise en œuvre efficace des bases de données.

Exercices corrigés

Exercice 1

Soit le schéma de BD suivant:

Client(**ID_Client**, *Nom*, *Prenom*)

Commande(**ID_Commande**, #*ID_Client*, *Date_Commande*)

Produit(**ID_Produit**, *Nom_Produit*, *Prix*)

Ligne_Commande(**ID_Commande**, **ID_Produit**, *Quantite*)

1. Sélectionner tous les clients ayant passé au moins une commande.

$\pi_{ID_Client, Nom, Prenom}(Client) \bowtie \pi_{ID_Client}(Commande)$

ou de manière équivalente :

$\pi_{Nom, Prenom}(Client \bowtie Commande)$

2. Lister les produits commandés par un client donné (exemple : **ID_Client** = 5).

$\pi_{Nom_Produit}((\sigma_{ID_Client=5}(Commande) \bowtie Ligne_Commande) \bowtie Produit)$

3. Trouver les étudiants inscrits à tous les cours.

$\pi_{ID_Etudiant}(Inscription) \div \pi_{ID_Cours}(Cours)$

4. Combiner les clients de deux boutiques différentes (union).

$Client_Boutique1 \cup Client_Boutique2$

5. Identifier les clients qui n'ont jamais passé de commande.

$$\pi_{ID_Client, Nom, Prenom}(Client) - \pi_{ID_Client, Nom, Prenom}(Client \bowtie Commande)$$

Exercice 2

Soit le schéma relationnel suivant:

$$\begin{aligned} CJH(\mathbf{IdCours}, Jour, Heure) \\ CS(\mathbf{IdCours}, \#IdSalle) \\ ENA(\mathbf{IdEtudiant}, Nom, Adresse) \\ CEN(\#IdCours, \#IdEtudiant, Note) \end{aligned}$$

Description des relations :

- **CJH** : le cours dont l'identifiant apparaît dans la première composante d'un n-uplet a lieu le jour spécifié dans la deuxième composante et à l'heure indiquée dans la troisième.
- **CS** : le cours de la première composante a lieu dans la salle indiquée dans la seconde composante.
- **ENA** : les étudiants dont l'identifiant apparaît comme première composante ont un nom et une adresse dans les deux autres.
- **CEN** : l'étudiant (deuxième composante) a obtenu la note (troisième composante) au cours spécifié (première composante).

1. Les opérateurs algébriques

1.1. Projections :

$$R1 = \pi_{IdCours}(CJH) \quad R2 = \pi_{IdEtudiant}(ENA)$$

1.2. Restriction :

$$R3 = \sigma_{IdCours='Algo'}(CEN)$$

1.3. Jointure :

$$R4 = CJH \bowtie_{CJH.IdCours=CS.IdCours} CS$$

1.4. Division :

$$R5 = \pi_{IdEtudiant, IdCours}(CEN)$$

$$R6 = R5 \div R1$$

$$R6 = \{ x \in \pi_{R5}(IdEtudiant) \text{ tel que } (x, u) \in R5 \text{ pour tout } u \in R1 \}$$

1.5. Enchaînement d'opérations :

$R7 = R2 \times R1$: ensemble de toutes les inscriptions possibles

$R8 = R7 - R5$: inscriptions manquantes

$R9 = \pi_{IdEtudiant}(R5)$: étudiants inscrits à certains cours

$R10 = \pi_{IdEtudiant}(R8)$: étudiants non inscrits à certains cours

$R11 = R9 - R10$: étudiants inscrits à tous les cours

1.6. Comparaison : Les résultats $R6$ et $R11$ sont équivalents : ils représentent tous deux l'ensemble des étudiants inscrits à **tous les cours**.

2. Le langage algébrique

2.1. Donner les noms des étudiants qui suivent le cours ‘Algo’.

$$\pi_{Nom}(ENA \bowtie_{ENA.IdEtudiant=CEN.IdEtudiant} \sigma_{IdCours='Algo'}(CEN))$$

2.2. Donner les notes en ‘Archi’ des étudiants dont le nom est ‘Titi’.

$$\pi_{Note}(\sigma_{Nom='Titi'}(ENA) \bowtie_{ENA.IdEtudiant=CEN.IdEtudiant} \sigma_{IdCours='Archi'}(CEN))$$

2.3. Donner les couples (jour, heure) pour lesquels la salle ‘S1’ est occupée par un cours.

$$\pi_{Jour, Heure}(\sigma_{IdSalle='S1'}(CS) \bowtie CJH)$$

2.4. Donner les identifiants des étudiants qui n’ont que des notes ‘A’.

$$\pi_{IdEtudiant}(CEN) - \pi_{IdEtudiant}(\sigma_{Note \neq 'A'}(CEN))$$

2.5. Donner la salle où se trouve ‘Toto’ le lundi à 9h.

$$\pi_{IdSalle}(\sigma_{Jour='Lundi' \wedge Heure=9}(CJH) \bowtie CS \bowtie (\sigma_{Nom='Toto'}(ENA) \bowtie CEN))$$

Exercice 3

Soit le schéma relationnel suivant:

$$\begin{aligned} & CJH(\mathbf{IdCours}, Jour, Heure) \\ & CS(\mathbf{IdCours}, \#IdSalle) \\ & ENA(\mathbf{IdEtudiant}, Nom, Adresse) \\ & CEN(\#IdCours, \#IdEtudiant, Note) \end{aligned}$$

Description des relations :

- **CJH** : le cours dont l’identifiant apparaît dans la première composante d’un n-uplet a lieu le jour indiqué et à l’heure correspondante.
- **CS** : le cours de la première composante a lieu dans la salle indiquée dans la seconde composante.
- **ENA** : chaque étudiant est identifié par un identifiant, un nom et une adresse.
- **CEN** : chaque ligne indique la note obtenue par un étudiant à un cours.

Exemple de contenu des tables

	IdCours	Jour	Heure
CJH	Algo	Lundi	9
	Archi	Mardi	10
	BD	Mercredi	8

	IdCours	IdSalle
CS	Algo	S1
	Archi	S2
	BD	S1

	IdEtudiant	Nom	Adresse
ENA	E1	Toto	Paris
	E2	Titi	Lyon
	E3	Tata	Nice

	IdCours	IdEtudiant	Note
CEN	Algo	E1	A
	Archi	E1	B
	BD	E1	A
	Algo	E2	A
	Archi	E2	A
	Algo	E3	B

1. Les opérateurs algébriques

1.1. Projections :

$$R1 = \pi_{IdCours}(CJH) \Rightarrow \{Algo, Archi, BD\}$$

$$R2 = \pi_{IdEtudiant}(ENA) \Rightarrow \{E1, E2, E3\}$$

1.2. Restriction :

$$R3 = \sigma_{IdCours='Algo'}(CEN) \Rightarrow$$

<i>IdCours</i>	<i>IdEtudiant</i>	<i>Note</i>
Algo	E1	A
Algo	E2	A
Algo	E3	B

1.3. Jointure :

$$R4 = CJH \bowtie_{CJH.IdCours=CS.IdCours} CS$$

$$\Rightarrow$$

<i>IdCours</i>	<i>Jour</i>	<i>Heure</i>	<i>IdSalle</i>
Algo	Lundi	9	S1
Archi	Mardi	10	S2
BD	Mercredi	8	S1

1.4. Division :

$$R5 = \pi_{IdEtudiant, IdCours}(CEN)$$

$$R6 = R5 \div R1$$

$$\Rightarrow R6 = \{E1\}$$

(car seul E1 est inscrit à tous les cours : Algo, Archi, BD)

1.5. Suite d'opérations équivalente :

$$R7 = R2 \times R1$$

$$R8 = R7 - R5$$

$$R9 = \pi_{IdEtudiant}(R5) = \{E1, E2, E3\}$$

$$R10 = \pi_{IdEtudiant}(R8) = \{E2, E3\}$$

$$R11 = R9 - R10 = \{E1\}$$

$$\Rightarrow R11 = R6 = \text{étudiants inscrits à tous les cours}$$

1.6. Comparaison :

Les résultats $R6$ et $R11$ sont identiques. Ils représentent les étudiants

qui suivent **tous les cours** proposés.

2. Le langage algébrique

2.1. Donner les noms des étudiants qui suivent le cours ‘Algo’.

$$\pi_{Nom}(\sigma_{ENA \bowtie_{ENA.IdEtudiant=CEN.IdEtudiant} \sigma_{IdCours='Algo'}}(CEN)) \Rightarrow \{Toto, Titi, Tata\}$$

2.2. Donner les notes en ‘Archi’ des étudiants dont le nom est ‘Titi’.

$$\pi_{Note}(\sigma_{Nom='Titi'}(ENA) \bowtie_{ENA.IdEtudiant=CEN.IdEtudiant} \sigma_{IdCours='Archi'}(CEN)) \Rightarrow \{A\}$$

2.3. Donner les couples (jour, heure) pour lesquels la salle ‘S1’ est occupée par un cours.

$$\pi_{Jour, Heure}(\sigma_{IdSalle='S1'}(CS) \bowtie CJH) \Rightarrow \{(Lundi, 9), (Mercredi, 8)\}$$

2.4. Donner les identifiants des étudiants qui n’ont que des notes ‘A’.

$$\pi_{IdEtudiant}(CEN) - \pi_{IdEtudiant}(\sigma_{Note \neq 'A'}(CEN)) \Rightarrow \{E2\}$$

2.5. Donner la salle où se trouve ‘Toto’ le lundi à 9h.

$$\pi_{IdSalle}(\sigma_{Jour='Lundi' \wedge Heure=9}(CJH) \bowtie CS \bowtie (\sigma_{Nom='Toto'}(ENA) \bowtie CEN)) \Rightarrow \{S1\}$$

Exercice 4

Soit la relation suivante :

	Numero	Nom	Prenom
Personne (Numero, Nom, Prenom)	1	Musk	Elon
	2	Jolie	Angelina
	3	Beckham	David
	4	Obama	Barack

1. $\sigma_{\text{Numero} < 3 \wedge \text{Nom} \neq \text{'Musk'}}(\text{Personne})$

Résultat :

Numero	Nom	Prenom
2	Jolie	Angelina

2. $\pi_{\text{Nom}, \text{Prenom}}(\sigma_{\text{Numero} > 1}(\text{Personne}))$

Résultat :

Nom	Prenom
Jolie	Angelina
Beckham	David
Obama	Barack

3. $\pi_{\text{Prenom}}(\sigma_{\text{Prenom} \neq \text{'Angelina'}}(\pi_{\text{Nom}, \text{Prenom}}(\text{Personne})))$

Résultat :

Prenom
Elon
David
Barack

Exercice 5

On considère les relations suivantes:

PERSONNE(CIN, NOM, Prenom, Adresse)

Voiture(NCarteGrise, #CIN, Modele)

Moto(NCarteGrise, #CIN, Modele)

1. Afficher les personnes qui possèdent une voiture mais pas de moto :

$$\pi_{CIN,NOM,Prenom,Adresse}((\text{PERSONNE} \bowtie \text{Voiture})) - \pi_{CIN,NOM,Prenom,Adresse}((\text{PERSONNE} \bowtie \text{Moto}))$$

2. Afficher les personnes qui possèdent une voiture et une moto :

$$\pi_{CIN,NOM,Prenom,Adresse}((\text{PERSONNE} \bowtie \text{Voiture}) \bowtie \text{Moto})$$

3. Afficher les personnes qui ne possèdent ni voiture ni moto :

$$\text{PERSONNE} - \pi_{CIN,NOM,Prenom,Adresse}(\text{PERSONNE} \bowtie (\text{Voiture} \cup \text{Moto}))$$

Exercice 6

Soit le schéma de base de données de la bibliothèque suivant :

- **Etudiant(NumEtd, NomEtd, PrenomEtd, AdresseEtd)**
- **Livre(NumLivre, TitreLivre, #NumAuteur, #NumEditeur, #NumTheme, AnneeEdition)**
- **Auteur(NumAuteur, NomAuteur, AdresseAuteur)**
- **Editeur(NumEditeur, NomEditeur, AdresseEditeur)**
- **Theme(NumTheme, IntituléTheme)**
- **Prêt(#NumEtd, #NumLivre, DatePret, DateRetour)**

1. Le nom, le prénom et l'adresse de l'étudiant nommé ‘Einstein’ :

$$\pi_{NomEtd,PrenomEtd,AdresseEtd}(\sigma_{NomEtd='Einstein'}(\text{Etudiant}))$$

2. Le numéro de l'auteur ‘Shakespeare’ :

$$\pi_{NumAuteur}(\sigma_{NomAuteur='Shakespeare'}(Auteur))$$

3. La liste des livres de l'auteur numéro 121 :

$$\pi_{TitreLivre}(\sigma_{NumAuteur=121}(Livre))$$

4. Les livres de l'auteur nommé ‘Shakespeare’ :

$$\pi_{TitreLivre}(Livre \bowtie \sigma_{NomAuteur='Shakespeare'}(Auteur))$$

5. Le numéro de l'auteur du livre ‘The Art of Data’ :

$$\pi_{NumAuteur}(\sigma_{TitreLivre='The Art of Data'}(Livre))$$

6. Le nom et l'adresse de l'auteur du livre ‘The Art of Data’ :

$$\pi_{NomAuteur, AdresseAuteur}(Auteur \bowtie \sigma_{TitreLivre='The Art of Data'}(Livre))$$

7. Les livres de l'auteur ‘Shakespeare’ édités chez ‘HarperCollins’ :

$$\pi_{TitreLivre}(\sigma_{NomAuteur='Shakespeare'}(Auteur) \bowtie Livre \bowtie \sigma_{NomEditeur='HarperCollins'}(Editeur))$$

8. Les livres des auteurs ‘Shakespeare’ ou ‘Tolkien’ :

$$\pi_{TitreLivre}(Livre \bowtie \sigma_{NomAuteur='Shakespeare' \vee NomAuteur='Tolkien'}(Auteur))$$

9. Les livres qui n'ont jamais été empruntés :

$$Livre - \pi_{NumLivre}(Prêt)$$

Chapitre 6

Introduction au Langage SQL

Introduction

Le Structured Query Language (SQL) est le langage standard pour interagir avec les bases de données relationnelles. Il permet de **créer**, **manipuler** et **interroger** les données de manière déclarative, c'est-à-dire en indiquant le résultat souhaité sans décrire le processus exact d'exécution.

SQL se compose de plusieurs sous-langages :

- **DDL (Data Definition Language)** : pour définir la structure des tables, contraintes et vues ;
- **DML (Data Manipulation Language)** : pour insérer, mettre à jour, supprimer et sélectionner des données ;
- **DCL (Data Control Language)** : pour gérer les droits d'accès ;
- **TCL (Transaction Control Language)** : pour gérer les transactions et la concurrence.

Par conséquent, la maîtrise de SQL est indispensable pour tout utilisateur, concepteur ou administrateur de bases de données.

6.1 Commandes DDL : CREATE, ALTER, DROP

CREATE TABLE

Cette instruction SQL crée la table Client avec les colonnes suivantes :

- ID_Client : identifiant unique du client (**clé primaire**), de type CHAR(4).
- Nom : nom du client, de type VARCHAR(50).
- Ville : ville de résidence du client, de type VARCHAR(30).

La contrainte PRIMARY KEY sur ID_Client garantit l'unicité de chaque client dans la table.

Énoncé : Crée une nouvelle table et définit ses colonnes et contraintes.

```

1      -- Creation de la table Client
2
3      CREATE TABLE Client (
4          ID_Client CHAR(4) PRIMARY KEY,    -- identifiant unique
5          Nom VARCHAR(50),                -- nom du client
6          Ville VARCHAR(30)             -- ville de residence
7      );

```

Remarque : le type de données doit être choisi en fonction des valeurs attendues (VARCHAR pour texte, CHAR pour code fixe, DATE pour dates).

ALTER TABLE

Ces instructions SQL illustrent comment modifier la structure de la table Client :

Énoncé: Ajouter une nouvelle colonne Telephone de type VARCHAR(15) pour stocker le numéro de téléphone des clients.

```

1      -- Ajouter une colonne Telephone
2
3      ALTER TABLE Client
4          ADD COLUMN Telephone VARCHAR(15);

```

Permet de modifier la structure d'une table existante.

Énoncé: Changer le type de la colonne Ville de VARCHAR(30) à VARCHAR(50), permettant ainsi de stocker des noms de villes plus longs.

```

1      -- Modifier un _type de colonne
2      ALTER TABLE Client
3      ALTER COLUMN Ville TYPE VARCHAR(50);
```

DROP TABLE

Supprime complètement une table et toutes ses données.

Énoncé: Supprimer complètement la table Client de la base de données, ainsi que toutes les données qu'elle contient. La commande DROP TABLE est définitive et ne peut pas être annulée, il convient donc de l'utiliser avec précaution.

```
1      DROP TABLE Client;
```

6.2 Commandes DML : INSERT, UPDATE, DELETE, SELECT

INSERT INTO

Insère de nouvelles lignes dans une table.

Énoncé: Insérer un client avec l'ID_Client 'C001', le nom 'Dupont' et la ville 'Paris'.

Puis, insérer simultanément plusieurs clients.

```

1      INSERT INTO Client (ID_Client, Nom, Ville)
2          VALUES ('C001', 'Dupont', 'Paris');
3
4      -- Inserer plusieurs lignes en une seule commande
5      INSERT INTO Client (ID_Client, Nom, Ville)
6          VALUES
7              ('C002', 'Martin', 'Lyon'),
8              ('C003', 'Bernard', 'Marseille');
```

UPDATE

Met à jour des données existantes.

Énoncé: mettre à jour la ville du client dont l'ID_Client est 'C001', en la remplaçant par 'Lille'. Modifier toutes les lignes où la ville est actuellement 'Marseille', en la remplaçant par 'Nice'.

```

1      -- Modifier la ville d un client
2      UPDATE Client
3          SET Ville = 'Lille'
4          WHERE ID_Client = 'C001';
5
6      -- Mise a jour multiple
7      UPDATE Client
8          SET Ville = 'Nice'
9          WHERE Ville = 'Marseille';

```

DELETE

Supprime des enregistrements.

Énoncé: Supprimer le client dont l'ID_Client est 'C003'. Supprimer tous les clients dont la ville est 'Nice'.

```

1      -- Supprimer un client specifique
2      DELETE FROM Client
3          WHERE ID_Client = 'C003';
4
5      -- Supprimer tous les clients d une ville
6      DELETE FROM Client
7          WHERE Ville = 'Nice';

```

SELECT

Extrait des données selon certains critères.

Énoncé: Récupérer toutes les lignes de la table Client, en affichant uniquement les colonnes Nom et Ville. Récupérer uniquement les clients dont la ville est 'Paris', en affichant les colonnes Nom et Ville.

```

1      -- Selection simple
2      SELECT Nom, Ville
3          FROM Client;
4
5      -- Selection avec condition
6      SELECT Nom, Ville
7          FROM Client
8      WHERE Ville = 'Paris';

```

Énoncé : Afficher les clients dont le nom commence par 'M'.

```

1      SELECT *
2          FROM Client
3      WHERE Nom LIKE 'M%';

```

6.3 Clauses avancées : WHERE, GROUP BY, HAVING, ORDER BY

Énoncé : Compter le nombre de clients dans chaque ville et ne conserver que celles où il y a plus d'un client.

```

1
2      SELECT Ville, COUNT(*) AS NbClients
3          FROM Client
4      GROUP BY Ville
5      HAVING COUNT(*) > 1
6      ORDER BY NbClients DESC;

```

Énoncé de la requête SQL avec agrégation et filtre

Cette requête permet d'obtenir des statistiques sur les clients regroupés par ville :

- **Objectif :** Compter le nombre de clients dans chaque ville et ne conserver que celles où il y a plus d'un client.
- **Fonctionnement :**
 - GROUP BY Ville regroupe les clients par ville.
 - COUNT(*) AS NbClients calcule le nombre de clients dans chaque groupe.
 - HAVING COUNT(*) > 1 filtre les groupes pour ne conserver que ceux comptant plus d'un client.
 - ORDER BY NbClients DESC trie les résultats par nombre de clients décroissant.
- **Résultat:** La liste des villes ayant plus d'un client, avec le nombre de clients correspondant, triée du plus grand au plus petit.

6.4 Contraintes d'intégrité, vues et index

Contraintes

- PRIMARY KEY : identifie de manière unique chaque ligne.
- FOREIGN KEY : garantit la cohérence entre tables.
- UNIQUE, NOT NULL, CHECK : contrôlent les valeurs acceptables.

Vues (VIEW)

Cette instruction SQL crée une **vue** appelée ClientsParis. Elle permet d'obtenir une représentation virtuelle de la table Client ne contenant que les clients résidant à Paris.

```

1      -- Vue des clients a Paris
2      CREATE VIEW ClientsParis AS
3          SELECT * FROM Client
4              WHERE Ville = 'Paris';

```

Cette instruction SQL crée une **vue** appelée ClientsParis :

- **Objectif :** Obtenir une représentation virtuelle de la table Client ne contenant que les clients résidant à Paris.
- **Fonctionnement :**
 - CREATE VIEW ClientsParis AS crée une vue nommée ClientsParis.
 - La sous-requête SELECT * FROM Client WHERE Ville = 'Paris' définit les données que la vue va présenter.
- **Remarque :** Une vue ne stocke pas les données de façon physique, mais sert de table virtuelle pour simplifier l'accès aux informations fréquemment utilisées.

Index

```
1    -- Index sur la colonne Ville  
2    CREATE INDEX idx_ville ON Client(Ville);
```

Cette instruction SQL crée un **index** sur la colonne Ville de la table Client :

- **Objectif :** Accélérer les requêtes qui filtrent ou trient les clients par ville.
- **Fonctionnement :**
 - CREATE INDEX idx_ville ON Client(Ville) crée un index nommé idx_ville sur la colonne Ville.
 - L'index permet au système de gestion de bases de données de rechercher plus rapidement les lignes correspondant à une ville donnée.
- **Remarque :** Un index améliore les performances des requêtes de lecture mais peut légèrement ralentir les opérations d'insertion, mise à jour et suppression.

6.5 Requêtes imbriquées et jointures

Énoncé : Les noms des clients ayant passé au moins une commande. Elle utilise une sous-requête pour identifier les clients présents dans la table Commande.

```

1      -- Sous-requête
2
3      SELECT Nom
4
5      FROM Client
6
7      WHERE ID_Client IN (
8          SELECT ID_Client FROM Commande
9      );

```

Énoncé : Les noms des clients ainsi que la date de chacune de leurs commandes. Elle utilise une jointure entre les tables Client et Commande sur la clé ID_Client pour associer chaque client à ses commandes.

```

1      -- Jointure interne
2
3      SELECT c.Nom, co.DateCommande
4
5      FROM Client c
6
7      JOIN Commande co ON c.ID_Client = co.ID_Client;

```

6.6 Fonctions d'agrégation

Énoncé : Calcule le salaire moyen des employés pour chaque ville. Cette requête utilise l'opérateur GROUP BY sur la colonne Ville pour regrouper les employés par ville et la fonction d'agrégation AVG(Salaire) pour calculer le salaire moyen par groupe.

```

1      SELECT Ville, AVG(Salaire) AS SalaireMoyen
2
3      FROM Employe
4
5      GROUP BY Ville;

```

Énoncé : Statistiques globales sur les salaires des employés. Cette requête utilise les fonctions d'agrégation COUNT(*) pour compter le nombre total d'employés, MAX(Salaire) pour trouver le salaire le plus élevé et MIN(Salaire) pour trouver le salaire le plus bas.

```

1      SELECT COUNT(*), MAX(Salaire), MIN(Salaire)

```

```
2   FROM Employe;
```

6.7 Transactions et gestion de la concurrence

```
1 BEGIN TRANSACTION;
2
3     UPDATE Compte SET Solde = Solde - 500 WHERE ID = 'A123';
4     UPDATE Compte SET Solde = Solde + 500 WHERE ID = 'B456';
5
6     COMMIT;
7
8     -- En cas d'erreur
9     ROLLBACK;
```

Cette séquence SQL illustre la gestion des transactions pour garantir la cohérence des données :

- **Début de transaction :** BEGIN TRANSACTION; démarre une transaction, permettant de regrouper plusieurs opérations en une seule unité logique.
- **Opérations sur les comptes :**
 - La première commande UPDATE retire 500 unités du solde du compte A123.
 - La deuxième commande UPDATE ajoute 500 unités au solde du compte B456.
- **Validation de la transaction :** COMMIT; confirme toutes les modifications effectuées dans la transaction.
- **Annulation en cas d'erreur :** ROLLBACK; permet d'annuler toutes les opérations de la transaction si une erreur survient, assurant ainsi l'intégrité des données.

Les systèmes de gestion de bases de données (SGBD) utilisent des mécanismes tels que le verrouillage (*locking*) et l'isolation pour éviter les conflits entre transactions simultanées.

Conclusion

Le langage SQL constitue l'outil principal pour manipuler et interroger les bases de données relationnelles. Il permet de créer et de modifier la structure des tables, d'insérer, mettre à jour ou supprimer des données, ainsi que de récupérer des informations à travers des requêtes simples ou complexes. SQL offre également des mécanismes avancés tels que les vues, les index et la gestion des transactions, afin de garantir la performance, la sécurité et l'intégrité des données. La maîtrise de SQL est donc essentielle pour concevoir, administrer et exploiter efficacement les bases de données relationnelles.

Exercices corrigés

Soit la base de données suivante :

- **film**(IDFilm, Titre, MetteurEnScene, Genre, Annee)
- **cinema**(NumCinema, NomCinema, AdresseCinema, CodePostalCinema, VilleCinema, TelephoneCinema)
- **jouedans**(IDFilm, IDActeur)
- **acteur**(IDActeur, NomActeur)
- **affiche**(NumCinema, IDFilm)

Écrire en SQL les requêtes suivantes.

Requêtes simples

1. Nom des cinémas
2. Nom des cinémas sans doublon
3. Nom des cinémas du 5^{ième} arrondissement (CP = 75005)
4. Nom des cinémas situés à Besançon

5. Nom des cinémas situés à BESANCON
6. Information sur les cinémas dont le nom contient 'cin'
7. Nom et ville des cinémas qui n'ont pas de numéro de téléphone
8. Titre des films dont l'année de sortie est supérieure ou égale à 1928
9. Titre des films dont l'année de sortie est comprise entre 1928 et 1980
10. Numéro des cinémas projetant le film N°5
11. Numéro des cinémas projetant le film N°5 ou le film N°4
12. Nom des cinémas qui ont un numéro de téléphone

Requêtes avec jointure

1. Numéro des cinémas projetant le film “Le bal”
2. Nom et adresse des cinémas projetant le film “Le bal”
3. Nom des metteurs en scène des films à l'affiche du cinéma Le Grand Rex
4. Nom des acteurs apparaissant dans la distribution des films à l'affiche dans le 5^{ième} arrondissement (CP = 75005)
5. Titre des films avec Humphrey Bogart à l'affiche à Paris
6. Titre des comédies musicales à l'affiche à Paris et nom du cinéma où ils sont à l'affiche
7. Nom des cinémas ayant le même code postal que le cinéma nommé “Le Champo”

Requêtes de comptage

1. Nombre de cinémas à Paris
2. Pour chaque cinéma, nombre de films à l'affiche

3. Nom des cinémas ainsi que le nombre de films à l'affiche dans ceux-ci
4. Numéro des cinémas ayant au moins deux films à l'affiche
5. Nom des cinémas ayant au moins deux films à l'affiche
6. Numéro des cinémas qui ont autant de films à l'affiche que le cinéma N°1
7. Pour chaque cinéma, nombre de films à l'affiche (0 si aucun film)
8. Pour chaque acteur, nombre de films dans lesquels il joue (0 si aucun film)
9. Pour chaque cinéma, nombre de films à l'affiche sans genre
10. Pour chaque ville, nombre de cinémas dont le numéro de téléphone est défini

Requêtes pour aller plus loin

1. Nom des cinémas de Paris ayant à l'affiche une comédie et une comédie musicale
2. Nom des cinémas n'ayant pas de film de genre comédie musicale à l'affiche

Correction

Soit la base de données suivante :

- **film**(IDFilm, Titre, MetteurEnScene, Genre, Annee)
- **cinema**(NumCinema, NomCinema, AdresseCinema, CodePostalCinema, VilleCinema, TelephoneCinema)
- **jouedans**(IDFilm, IDActeur)
- **acteur**(IDActeur, NomActeur)
- **affiche**(NumCinema, IDFilm)

Écrire en SQL les requêtes suivantes.

Requêtes simples

1. Nom des cinémas

```
1   SELECT NomCinema FROM cinema;
```

2. Nom des cinémas sans doublon

```
1   SELECT DISTINCT NomCinema FROM cinema;
```

3. Nom des cinémas du 5^{ème} arrondissement (CP = 75005)

```
1   SELECT NomCinema FROM cinema WHERE  
      CodePostalCinema = 75005;
```

4. Nom des cinémas situés à Besançon

```
1   SELECT NomCinema FROM cinema WHERE VilleCinema  
      = 'Besancon';
```

5. Nom des cinémas situés à BESANCON (insensible à la casse selon SGBD)

```
1   SELECT NomCinema FROM cinema WHERE UPPER(  
      VilleCinema) = 'BESANCON';
```

6. Information sur les cinémas dont le nom contient 'cin'

```
1   SELECT * FROM cinema WHERE NomCinema LIKE '%cin  
      %';
```

7. Nom et ville des cinémas qui n'ont pas de numéro de téléphone

```
1   SELECT NomCinema, VilleCinema FROM cinema WHERE  
      TelephoneCinema IS NULL;
```

8. Titre des films dont l'année de sortie est supérieure ou égale à 1928

```
1   SELECT Titre FROM film WHERE Annee >= 1928;
```

9. Titre des films dont l'année de sortie est comprise entre 1928 et 1980

```
1      SELECT Titre FROM film WHERE Annee BETWEEN 1928
          AND 1980;
```

10. Numéro des cinémas projetant le film N°5

```
1      SELECT NumCinema FROM affiche WHERE IDFilm = 5;
```

11. Numéro des cinémas projetant le film N°5 ou le film N°4

```
1      SELECT NumCinema FROM affiche WHERE IDFilm IN
          (4,5);
```

12. Nom des cinémas qui ont un numéro de téléphone

```
1      SELECT NomCinema FROM cinema WHERE
          TelephoneCinema IS NOT NULL;
```

Requêtes avec jointure

1. Numéro des cinémas projetant le film "Le bal"

```
1      SELECT a.NumCinema
2          FROM affiche a
3          JOIN film f ON a.IDFilm = f.IDFilm
4          WHERE f.Titre = 'Le bal';
```

2. Nom et adresse des cinémas projetant le film "Le bal"

```
1      SELECT c.NomCinema, c.AdresseCinema
2          FROM cinema c
3          JOIN affiche a ON c.NumCinema = a.NumCinema
4          JOIN film f ON a.IDFilm = f.IDFilm
5          WHERE f.Titre = 'Le bal';
```

3. Nom des metteurs en scène des films à l'affiche du cinéma Le Grand Rex

```

1      SELECT DISTINCT f.MetteurEnScene
2          FROM film f
3              JOIN affiche a ON f.IDFilm = a.IDFilm
4              JOIN cinema c ON a.NumCinema = c.NumCinema
5                  WHERE c.NomCinema = 'Le Grand Rex';

```

4. Nom des acteurs apparaissant dans les films à l'affiche dans le 5^{ème} arrondissement
(CP = 75005)

```

1      SELECT DISTINCT ac.NomActeur
2          FROM acteur ac
3              JOIN playedans j ON ac.IDActeur = j.IDActeur
4              JOIN film f ON j.IDFilm = f.IDFilm
5              JOIN affiche a ON f.IDFilm = a.IDFilm
6              JOIN cinema c ON a.NumCinema = c.NumCinema
7                  WHERE c.CodePostalCinema = 75005;

```

5. Titre des films avec Humphrey Bogart à l'affiche à Paris

```

1      SELECT DISTINCT f.Titre
2          FROM film f
3              JOIN playedans j ON f.IDFilm = j.IDFilm
4              JOIN acteur ac ON j.IDActeur = ac.IDActeur
5              JOIN affiche a ON f.IDFilm = a.IDFilm
6              JOIN cinema c ON a.NumCinema = c.NumCinema
7                  WHERE ac.NomActeur = 'Humphrey Bogart' AND c.
VilleCinema = 'Paris';

```

6. Titre des comédies musicales à l'affiche à Paris, et nom du cinéma

```

1      SELECT f.Titre, c.NomCinema
2          FROM film f
3              JOIN affiche a ON f.IDFilm = a.IDFilm
4              JOIN cinema c ON a.NumCinema = c.NumCinema
5                  WHERE f.Genre = 'Comedie musicale' AND c.
VilleCinema = 'Paris';

```

7. Nom des cinémas ayant le même code postal que le cinéma nommé 'Le Champo'

```

1      SELECT NomCinema
2      FROM cinema
3      WHERE CodePostalCinema = (SELECT
4          CodePostalCinema FROM cinema WHERE NomCinema
5          = 'Le Champo');

```

Requêtes de comptage

1. Nombre de cinémas à Paris

```

1      SELECT COUNT(*) FROM cinema WHERE VilleCinema =
2          'Paris';

```

2. Pour chaque cinéma, nombre de films à l'affiche

```

1      SELECT c.NumCinema, COUNT(a.IDFilm) AS NbFilms
2      FROM cinema c
3      LEFT JOIN affiche a ON c.NumCinema = a.
4          NumCinema
5      GROUP BY c.NumCinema;

```

3. Nom des cinémas et nombre de films à l'affiche

```

1      SELECT c.NomCinema, COUNT(a.IDFilm) AS NbFilms
2      FROM cinema c
3      LEFT JOIN affiche a ON c.NumCinema = a.
4          NumCinema
5      GROUP BY c.NomCinema;

```

4. Numéro des cinémas ayant au moins deux films à l'affiche

```

1      SELECT c.NumCinema
2      FROM cinema c
3      JOIN affiche a ON c.NumCinema = a.NumCinema
4      GROUP BY c.NumCinema
5      HAVING COUNT(a.IDFilm) >= 2;

```

5. Nom des cinémas ayant au moins deux films à l'affiche

```

1      SELECT c.NomCinema
2          FROM cinema c
3              JOIN affiche a ON c.NumCinema = a.NumCinema
4                  GROUP BY c.NomCinema
5                      HAVING COUNT(a.IDFilm) >= 2;

```

6. Numéro des cinémas qui ont autant de films à l'affiche que le cinéma N°1

```

1      SELECT c.NumCinema
2          FROM cinema c
3              JOIN affiche a ON c.NumCinema = a.NumCinema
4                  GROUP BY c.NumCinema
5                      HAVING COUNT(a.IDFilm) = (
6                          SELECT COUNT(*) FROM affiche WHERE NumCinema =
7                                          1
8                      );

```

7. Pour chaque cinéma, nombre de films (0 si aucun film)

```

1      SELECT c.NumCinema, COUNT(a.IDFilm) AS NbFilms
2          FROM cinema c
3              LEFT JOIN affiche a ON c.NumCinema = a.
4                  NumCinema
5                      GROUP BY c.NumCinema;

```

8. Pour chaque acteur, nombre de films (0 si aucun film)

```

1      SELECT ac.NomActeur, COUNT(j.IDFilm) AS NbFilms
2          FROM acteur ac
3              LEFT JOIN playedans j ON ac.IDActeur = j.
4                  IDActeur
5                      GROUP BY ac.NomActeur;

```

9. Pour chaque cinéma, nombre de films à l'affiche sans genre

```

1      SELECT c.NumCinema , COUNT(f.IDFilm) AS
2          NbFilmsSansGenre
3
4      FROM cinema c
5
6      JOIN affiche a ON c.NumCinema = a.NumCinema
7
8      JOIN film f ON a.IDFilm = f.IDFilm
9
10     WHERE f.Genre IS NULL
11
12     GROUP BY c.NumCinema;

```

10. Pour chaque ville, nombre de cinémas avec un numéro de téléphone

```

1      SELECT VilleCinema , COUNT(*) AS NbCinemas
2
3      FROM cinema
4
5      WHERE TelephoneCinema IS NOT NULL
6
7      GROUP BY VilleCinema;

```

Requêtes pour aller plus loin

1. Nom des cinémas de Paris ayant à l'affiche une comédie et une comédie musicale

```

1      SELECT c.NomCinema
2
3      FROM cinema c
4
5      WHERE c.VilleCinema = 'Paris'
6
7      AND EXISTS (
8
9          SELECT 1 FROM affiche a
10
11         JOIN film f ON a.IDFilm = f.IDFilm
12
13         WHERE a.NumCinema = c.NumCinema AND f.Genre = ,
14             Comedie'
15
16     )
17
18     AND EXISTS (
19
20         SELECT 1 FROM affiche a
21
22         JOIN film f ON a.IDFilm = f.IDFilm
23
24         WHERE a.NumCinema = c.NumCinema AND f.Genre = ,
25             Comedie musicale'
26
27     );

```

2. Nom des cinémas n'ayant pas de film de genre comédie musicale à l'affiche

```
1      SELECT c.NomCinema
2      FROM cinema c
3      WHERE NOT EXISTS (
4          SELECT 1 FROM affiche a
5          JOIN film f ON a.IDFilm = f.IDFilm
6          WHERE a.NumCinema = c.NumCinema AND f.Genre = 'Comedie musicale'
7      );
```


Chapitre 7

Études de cas et exercices SQL

Introduction

Cette partie pratique permet de mettre en application tous les concepts SQL et de bases de données relationnelles vus précédemment. Elle contient :

- des mini-projets complets pour appliquer les concepts vus dans les chapitres précédents,
- des exercices corrigés SQL,
- des travaux pratiques sur PostgreSQL ou MySQL.

7.1 Mini-projet 1 : Gestion d'école

Description

Créer une base de données pour gérer :

- Les étudiants et leurs informations personnelles
- Les cours proposés par l'école
- Les inscriptions aux cours
- Les notes et résultats

Création des tables

```

1      CREATE TABLE Etudiant (
2          ID_Etudiant SERIAL PRIMARY KEY,
3          Nom VARCHAR(50),
4          Prenom VARCHAR(50),
5          DateNaissance DATE
6      );
7
8      CREATE TABLE Cours (
9          ID_Cours SERIAL PRIMARY KEY,
10         NomCours VARCHAR(50),
11         Professeur VARCHAR(50)
12     );
13
14     CREATE TABLE Inscription (
15         ID_Inscription SERIAL PRIMARY KEY,
16         ID_Etudiant INT REFERENCES Etudiant(ID_Etudiant),
17         ID_Cours INT REFERENCES Cours(ID_Cours),
18         DateInscription DATE
19     );
20
21     CREATE TABLE Note (
22         ID_Note SERIAL PRIMARY KEY,
23         ID_Inscription INT REFERENCES Inscription(ID_Inscription),
24         Valeur DECIMAL(5,2)
25     );

```

Exemples de requêtes SQL

```

1      -- Lister tous les etudiants inscrits a un cours specifique
2      SELECT e.Nom, e.Prenom, c.NomCours
3      FROM Etudiant e
4      JOIN Inscription i ON e.ID_Etudiant = i.ID_Etudiant
5      JOIN Cours c ON i.ID_Cours = c.ID_Cours

```

```
6 WHERE c.NomCours = 'Mathematiques';  
7  
8     -- Compter le nombre d etudiants par cours  
9     SELECT c.NomCours, COUNT(*) AS NbEtudiants  
10    FROM Cours c  
11    JOIN Incription i ON c.ID_Cours = i.ID_Cours  
12    GROUP BY c.NomCours;
```

Mini-exercices

1. Afficher le nom et prénom des étudiants nés après 2005.
2. Afficher tous les cours avec plus de 5 étudiants inscrits.
3. Créer une vue EtudiantsActifs des étudiants inscrits à plus de 2 cours.

7.2 Mini-projet 2 : Gestion d'une bibliothèque

Description

La base doit gérer :

- Les livres et leurs informations
- Les membres de la bibliothèque
- Les emprunts de livres

Création des tables

```
1 CREATE TABLE Livre (  
2     ID_Livre SERIAL PRIMARY KEY,  
3     Titre VARCHAR(100),  
4     Auteur VARCHAR(50),  
5     AnneePublication INT  
6 );
```

```

7
8     CREATE TABLE Membre (
9         ID_Membre SERIAL PRIMARY KEY ,
10        Nom VARCHAR(50) ,
11        Prenom VARCHAR(50) ,
12        DateInscription DATE
13    );
14
15    CREATE TABLE Emprunt (
16        ID_Empunt SERIAL PRIMARY KEY ,
17        ID_Livre INT REFERENCES Livre(ID_Livre) ,
18        ID_Membre INT REFERENCES Membre(ID_Membre) ,
19        DateEmprunt DATE ,
20        DateRetour DATE
21    );

```

Exemples de requêtes SQL

```

1      -- Lister tous les livres empruntes par un membre
2
3      SELECT m.Nom , m.Prenom , l.Titre
4
5      FROM Membre m
6
7      JOIN Emprunt e ON m.ID_Membre = e.ID_Membre
8
9      JOIN Livre l ON e.ID_Livre = l.ID_Livre
10
11     WHERE m.Nom = 'Martin';
12
13
14      -- Compter le nombre d'emprunts par livre
15
16      SELECT l.Titre , COUNT(*) AS NbEmprunts
17
18      FROM Livre l
19
20      JOIN Emprunt e ON l.ID_Livre = e.ID_Livre
21
22      GROUP BY l.Titre;

```

Mini-exercices

1. Lister les membres ayant emprunté plus de 3 livres.

2. Identifier les livres jamais empruntés.
3. Créer une vue LivresEmpruntes pour les livres actuellement empruntés.

7.3 Mini-projet 3 : E-commerce

Description

Créer une base pour gérer :

- Les clients et leurs informations
- Les produits et leur stock
- Les commandes et détails des commandes

Création des tables

```
1   CREATE TABLE Client (
2       ID_Client SERIAL PRIMARY KEY,
3       Nom VARCHAR(50),
4       Prenom VARCHAR(50),
5       Email VARCHAR(100),
6       Ville VARCHAR(50)
7   );
8
9   CREATE TABLE Produit (
10      ID_Produit SERIAL PRIMARY KEY,
11      NomProduit VARCHAR(100),
12      Prix DECIMAL(10,2),
13      Stock INT
14  );
15
16  CREATE TABLE Commande (
17      ID_Commande SERIAL PRIMARY KEY,
18      ID_Client INT REFERENCES Client(ID_Client),
```

```

19      DateCommande DATE
20  );
21
22  CREATE TABLE DetailsCommande (
23      ID_Detail SERIAL PRIMARY KEY,
24      ID_Commande INT REFERENCES Commande(ID_Commande),
25      ID_Produit INT REFERENCES Produit(ID_Produit),
26      Quantite INT,
27      PrixTotal DECIMAL(10,2)
28  );

```

Exemples de requêtes SQL

```

1      -- Lister toutes les commandes avec informations client et
2          total
3
4      SELECT c.Nom, c.Prenom, com.DateCommande, SUM(d.PrixTotal)
5          AS TotalCommande
6
7      FROM Commande com
8
9      JOIN Client c ON com.ID_Client = c.ID_Client
10
11     JOIN DetailsCommande d ON com.ID_Commande = d.ID_Commande
12
13     GROUP BY c.Nom, c.Prenom, com.DateCommande
14
15     ORDER BY TotalCommande DESC;
16
17
18
19      -- Produits en rupture de stock
20
21      SELECT NomProduit, Stock
22
23      FROM Produit
24
25      WHERE Stock = 0;

```

Mini-exercices

1. Lister tous les clients habitant à Paris ou Lyon.
2. Trouver le produit le plus vendu et le montant total généré.
3. Créer une vue CommandesClients affichant nom, prénom, date commande et total.

4. Supprimer les commandes avant le 1er janvier 2025.
5. Ajouter une contrainte CHECK pour que Quantité soit toujours > 0.

7.4 Travaux pratiques (TP)

1. Installer PostgreSQL ou PhpMyAdmin.
2. Créer la base et les tables des mini-projets.
3. Insérer au moins 10 enregistrements par table.
4. Réaliser des requêtes : SELECT simples et avancées (WHERE, JOIN, GROUP BY, HAVING), INSERT, UPDATE, DELETE.
5. Créer des vues et index pour optimiser les performances.
6. Tester les transactions avec COMMIT et ROLLBACK.
7. Bonus : ajouter des contraintes CHECK pour valider les données.

Conclusion

Les études de cas et exercices permettent :

- D'appliquer les concepts théoriques à des situations concrètes,
- De développer la maîtrise des requêtes SQL, des transactions et de l'administration,
- De préparer les étudiants à gérer des projets complets en bases de données relationnelles.

Conclusion générale

Au terme de ce parcours, le lecteur a acquis une compréhension solide des bases de données relationnelles, depuis la conception jusqu'à l'exploitation et l'optimisation des systèmes.

Le manuscrit a commencé par introduire les concepts fondamentaux des bases de données, le rôle des SGBD, et les différents types de modèles. La modélisation conceptuelle et relationnelle a permis de formaliser les données et leurs relations, avec une attention particulière portée aux règles de conception et à la normalisation pour garantir la cohérence et l'intégrité des données.

L'apprentissage de l'algèbre relationnelle et du langage SQL a constitué une étape essentielle pour manipuler efficacement les données. Les chapitres pratiques ont présenté les commandes DDL et DML, les requêtes avancées avec jointures et agrégations, ainsi que la gestion des transactions, des vues, des index et des contraintes d'intégrité. Ces compétences sont indispensables pour concevoir des bases performantes et faciles à maintenir.

Les études de cas et mini-projets (gestion d'école, bibliothèque, e-commerce) ont permis de mettre en pratique l'ensemble des notions étudiées, en simulant des projets réels. Les exercices corrigés et travaux pratiques ont renforcé la capacité du lecteur à analyser, interroger et administrer des bases de données relationnelles, tout en développant une approche méthodique de résolution de problèmes.

Ce travail constitue un guide progressif pour former des étudiants capables de concevoir, exploiter et optimiser des bases de données relationnelles. Il offre une base pour aborder des sujets plus avancés, tels que les bases de données NoSQL, le Big Data ou l'optimisation complexe des requêtes dans des environnements professionnels.

Bibliographie

- Abiteboul, S., Nguyen, B., and Bras, Y. L. (2020). Introduction aux bases de données relationnelles. Polycopié de cours CPGE Informatique commune. <https://inria.fr>. Consulté en Juillet 2025.
- Date, C. J. (2019). *Database Design and Relational Theory: Normal Forms and All That Jazz*. Apress, 2nd edition. Original work published 2019.
- Gardarin, G. (2003). *Bases de données*. Eyrolles, Paris, 5e tirage edition.
- Pelleau, M. (2023). Bases de données. Support de cours de Master Informatique. <https://univ-cotedazur.fr>. Consulté en Juillet 2025.