Chapitre 1
Introduction aux bases de données

1.1 Historique

Les Systèmes de Gestion de Bases de Données (SGBD) ont vu le jour dans les années 60 pour gérer d'importants volumes de données de gestion. Il s'agissait de systèmes propriétaires (appartenant à une marque d'ordinateur, par exemple IBM), sur de grands systèmes (main frame) conçus selon un schéma d'organisation "hiérarchique" ou "réseau".

Le modèle hiérarchique
Ce type de modèle permet de représenter des classes ou ensembles d'objets et des relations de type « père-fils » entre ces classes. L'ensemble de ces classes constitue une arborescence. Les SGBDs supportant ce type de structure gèrent les liens entre un « père » et ses « fils ». Ils offrent des primitives pour manipuler et « naviguer » dans de telles structures.

Le modèle réseau
Le modèle réseau permet également la représentation de classes d'objets et de liens de type « père-fils » entre ces classes. Comme son nom l'indique, et à la différence du modèle hiérarchique, il autorise une classe « fille » à avoir plusieurs classes « mères ». Les langages de ces SGBD sont aussi de type navigationnel.

Le modèle relationnel de Codd.
En 1970, Codd, chercheur chez IBM, proposa le **modèle relationnel**. Ce modèle conceptuel constitue un progrès important car il repose sur une représentation unifiée de l'information sous forme de tables. Il dispose d'un fondement mathématique solide avec l'algèbre relationnelle. Il permet une plus grande indépendance entre les applications, les données et le support physique. Il propose une démarche cohérente et unifiée pour la description et pour l'interrogation.

Les SGBD de nos jours
De nombreux SGBD sont aujourd'hui disponibles sur micro ordinateurs. La plupart sont dotés de capacités relationnelles, bien que l'ancêtre des SGBD sur micro, Dbase ne soit qu'un gestionnaire de fichiers structurés avec un langage de programmation. On peut citer FoxPro (clone de DBase), Access (Microsoft) et Paradox (Borland). Sur stations de travail et mini ordinateurs sous Unix, trois ou quatre SGBD relationnels dominent : Oracle, Ingres, Informix et Sybase. DB2 (IBM) est un SGBD relationnel sur main frame…

1.2 Base de données et Système de Gestion de Base de Données

Bases de données

Définition : Une base de données est un ensemble structuré de données enregistrées avec le minimum de redondance pour satisfaire simultanément plusieurs utilisateurs de façon sélective en un temps opportun.

L'approche base de données est due à une triple évolution :

- évolution des entreprises (volumes importants de données, centralisées ou réparties, qui doivent être accessibles en temps utile,...)
- évolution du matériel (accroissement des performances, intégration des composants, diminution des coûts,...)
- évolution des logiciels (les systèmes d'exploitation, les architectures client serveur et les réseaux).

Exemple : Base de données commerciale

Les données sont relatives aux clients, aux produits, aux commandes, aux lignes de commandes... Les requêtes sont très variées :
- "liste des produits qui ont été commandés par un client déterminé" ;
- "quel est le client de la commande numéro X ?" ;
- "quelle est la date de la dernière commande du client s'appelant Y ?".

Systèmes de Gestion de Base de Données

Définition : Un SGBD (ou *DBMS : Data Base Management System*) peut être défini comme un ensemble de logiciels permettant de stocker et d'interroger un ensemble de fichiers interdépendants. Il peut aussi être défini comme un outil permettant de modéliser et de gérer les données d'une entreprise.

Ainsi, un SGBD permet à un utilisateur de communiquer avec une base de données pour :
- décrire et organiser les données sur les mémoires secondaires (disques),
- rechercher, sélectionner et modifier les données.

Un SGBD offre la possibilité à l'utilisateur de manipuler les représentations abstraites des données, indépendamment de leur organisation et de leur implantation sur les supports physiques (mémoires).

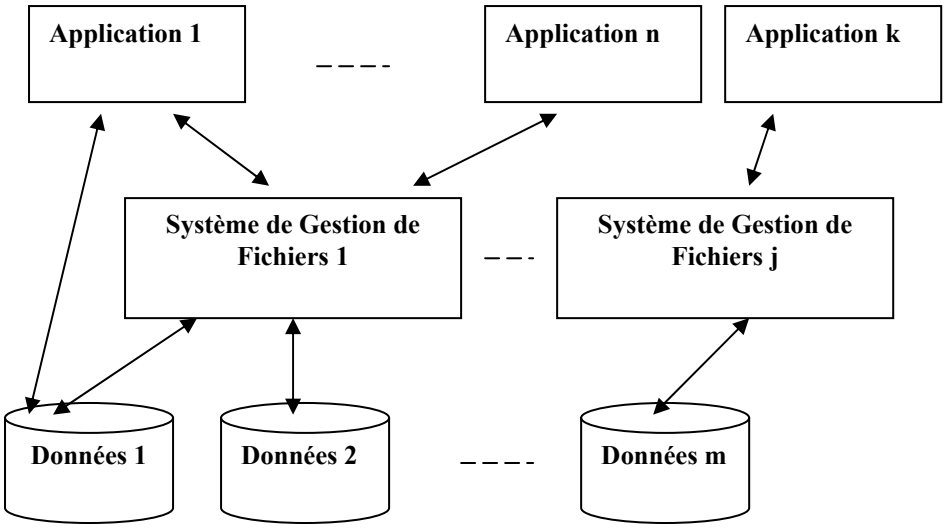
Un SGBD assure
- la **description** des données,
- leur **recherche** et **mise à jour**,
- la **sûreté** : sauvegarde et restauration des données ; limiter les erreurs de saisie et de manipulation,
- la **sécurité** : vérifier les droits d'accès des utilisateurs ; limiter les accès non autorisés,
- l'**intégrité** : définir des règles qui maintiennent l'intégrité de la base de données (contraintes d'intégrité),
- la **concurrence** d'accès : détecter et traiter les cas où il y a conflit d'accès entre plusieurs utilisateurs et les traiter correctement.

1.3. Avantages de l'approche base de données

Un système organisé autour d'une base de données est **centré sur les données**, contrairement aux systèmes de gestion de fichiers (SGF) basés sur les **traitements** (par exemple : traitement de la paye, facturation, gestion des stocks, etc.).

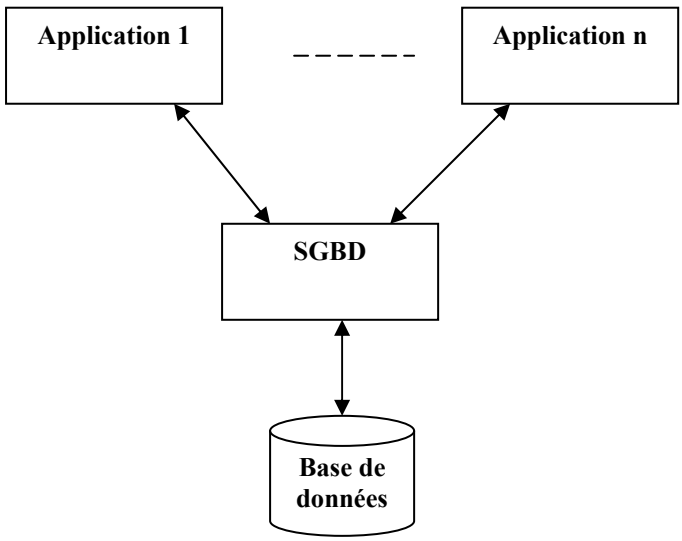
1.3.1. Comparaison SGBD - SGF

Dans l'**approche gestion de fichiers,** les fichiers sont définis pour un ou plusieurs programmes de traitement. Les données d'un fichier sont directement associées à un programme par une description contenue dans le programme de traitement lui-même. Il n'existe aucune indépendance entre le programme et les données. Toute modification de la structure des données nécessite la réécriture du programme.



Gestion des données avec les SGF

Dans l'**approche base de données**, la partie de structuration et de description des données est unifiée et séparée des programmes d'application. Bien sûr la gestion de ces données (stockage, modification, recherche) qui est étroitement dépendante de leur structuration, est fournie par le système de gestion des données, les applications ne communiquant avec les données qu'au travers de l'interface de gestion. D'où l'indépendance entre les données et les applications, qui peuvent être modifiées indépendamment. Le programmeur des applications (l'utilisateur) n'a pas à connaître l'organisation physique des données.



Gestion des données avec les SGBDs

1.3.2. Organisation des données

Dans les deux approches, des fichiers de données sont constitués, permettant le stockage, l'organisation et l'accès aux différents enregistrements.
Ces fichiers peuvent être accédés selon différentes méthodes :

- Méthodes d'accès séquentielles : consistant à lire successivement tous les enregistrements d'un fichier, depuis le premier jusqu'à l'enregistrement désiré.
- Méthodes d'accès sélectives : permettant de lire/écrire tout article au moyen de quelques accès disques, y compris pour de très gros fichiers.
- Méthodes d'accès par hachage : basées sur l'utilisation d'une fonction de calcul qui, appliquée à la clé, détermine l'adresse relative d'une zone appelée paquet dans laquelle est placé l'article.
- Méthodes d'accès indexées : consistant à associer à la clé d'un enregistrement son adresse relative dans le fichier à l'aide d'une « table des matières » appelée **index** du fichier. Ainsi, à partir de la clé de l'enregistrement, un accès rapide est possible par recherche de l'adresse relative dans la table des matières, puis par un accès en relatif à l'enregistrement dans le fichier.

Définition : un index est une table (ou plusieurs tables) permettant d'associer à une clé d'enregistrement l'adresse relative de cet enregistrement.

1.3.3. Apports des SGBD

Les avantages des systèmes de bases de données par rapport aux systèmes traditionnels (méthodes papier ou approche de gestion de fichiers) sont :

- Compacité : plus besoin de fichiers volumineux
- Rapidité : le système est capable de retrouver et de mettre à jour les données beaucoup plus rapidement que de manière manuelle ou avec un système de gestion de fichiers
- Moins de corvées : les travaux ennuyeux de maintenance manuelle des fichiers sont éliminés
- Exactitude : des informations précises et réactualisées sont disponibles à tout moment.

De plus, le contrôle centralisé des données dans une base de données a pour résultat certains autres avantages tels que :

- Les données peuvent être partagées : partager ne signifie pas seulement que les opérations existantes peuvent partager les données de la base de données, mais également que de nouvelles opérations peuvent être développées pour manipuler ces mêmes données. En d'autres termes, il est possible de satisfaire les besoins des données des nouvelles applications sans avoir à ajouter de nouvelles données dans la base de données.
- La redondance peut être réduite : dans les systèmes autres que les bases de données, chaque application possède ses propres fichiers, ce qui peut conduire à une très grande redondance des données.
- L'incohérence peut être évitée : c'est une conséquence du point précédent.
- Les transactions peuvent être gérées : une transaction est une unité logique de travail qui réalise, en général, plusieurs opérations sur la base de données.
- L'intégrité peut être assurée : le problème de l'intégrité consiste à assurer l'exactitude des données de la base de données.

- La sécurité peut être appliquée : c'est le fait de définir des contraintes ou des règles de sécurité qui sont contrôlées chaque fois que l'on essaye d'accéder à des données sensibles.
- Les normes peuvent être appliquées : une représentation normalisée des données est particulièrement souhaitable, car elle permet l'interchangeabilité des données ou la migration des données entre les systèmes.

Exercices : Questions à choix multiples

Q1 : Historique
Les Systèmes de Gestion de Bases de Données (SGBD) ont vu le jour
a) Dans les années 60.
b) Dans les années 70.
c) Dans les années 80.

Q2 : Modèle hiérarchique
Dans le modèle hiérarchique :
a) Des relations de type « père-fils » sont représentées entre classes d'objets.
b) L'ensemble des classes d'objets constitue une arborescence.
c) Une classe « fille » peut avoir plusieurs classes « mères ».

Q3 : Modèle réseau
Dans le modèle réseau :
a) Des liens de type « père-fils » sont représentés entre classes d'objets.
b) L'ensemble des classes d'objets constitue une arborescence.
c) Une classe « fille » peut avoir plusieurs classes « mères ».

Q4 : Modèle relationnel
Le modèle relationnel
a) repose sur une représentation unifiée de l'information sous forme de tables.
b) dispose d'un fondement mathématique solide avec l'algèbre relationnelle.
c) permet une plus grande indépendance entre les applications, les données et le support physique.
d) Représente l'ensemble des classes d'objets sous forme d'arborescence.

Q5 : Comparaison SGF et SGBD
Dans l'approche gestion de fichiers
a) Les fichiers sont définis pour un ou plusieurs programmes de traitement.
b) Les données d'un fichier sont directement associées à un programme par une description contenue dans le programme de traitement lui-même.
c) Il n'existe aucune indépendance entre le programme et les données.
d) Toute modification de la structure des données nécessite la réécriture du programme.
e) la partie de structuration et de description des données est unifiée et séparée des programmes d'application.
f) Il y a indépendance entre données et applications.

Q6 : Comparaison SGF et SGBD
Dans l'approche base de données
a) Les fichiers sont définis pour un ou plusieurs programmes de traitement.
b) Toute modification de la structure des données nécessite la réécriture du programme.
c) la partie de structuration et de description des données est unifiée et séparée des programmes d'application.
d) La gestion des données (stockage, modification, recherche) est fournie par le Système de gestion des données.
e) Il y a indépendance entre données et applications.
f) Le programmeur des applications (l'utilisateur) n'a pas à connaître l'organisation physique des données.

Q7 : BD et SGBD
Une base de données est :

a) Un ensemble de logiciels systèmes permettant de stocker et d'interroger un ensemble de fichiers interdépendants.
b) Un ensemble structuré de données enregistrées avec le minimum de redondance pour satisfaire simultanément plusieurs utilisateurs de façon sélective en un temps opportun.

Q8 : Fonctions d'un SGBD
Parmi les fonctions assurées par un SGBD :

a) la description des données,
b) leur recherche et mise à jour,
c) la sûreté : vérifier les droits d'accès des utilisateurs
d) la sécurité : sauvegarde et restauration des données ; limiter les erreurs de saisie et de manipulation
e) la concurrence d'accès : détecter et traiter les cas où il y a conflit d'accès entre plusieurs utilisateurs et les traiter correctement.

Q9 : Apports des SGBD
Parmi les avantages des systèmes de bases de données par rapport aux systèmes traditionnels (méthodes papier ou approche de gestion de fichiers) :

a) Compacité
b) Rapidité
c) Exactitude
d) Partage des données
e) Redondance des données
f) Intégrité des données
g) Sécurité des données

Q10 : Index
Un index est :

a) Une table permettant d'associer, à une clé d'enregistrement, l'adresse relative de cet enregistrement.
b) Une clé d'enregistrement permettant de retrouver l'adresse relative de cet enregistrement.
c) Une clé permettant de retrouver un enregistrement.

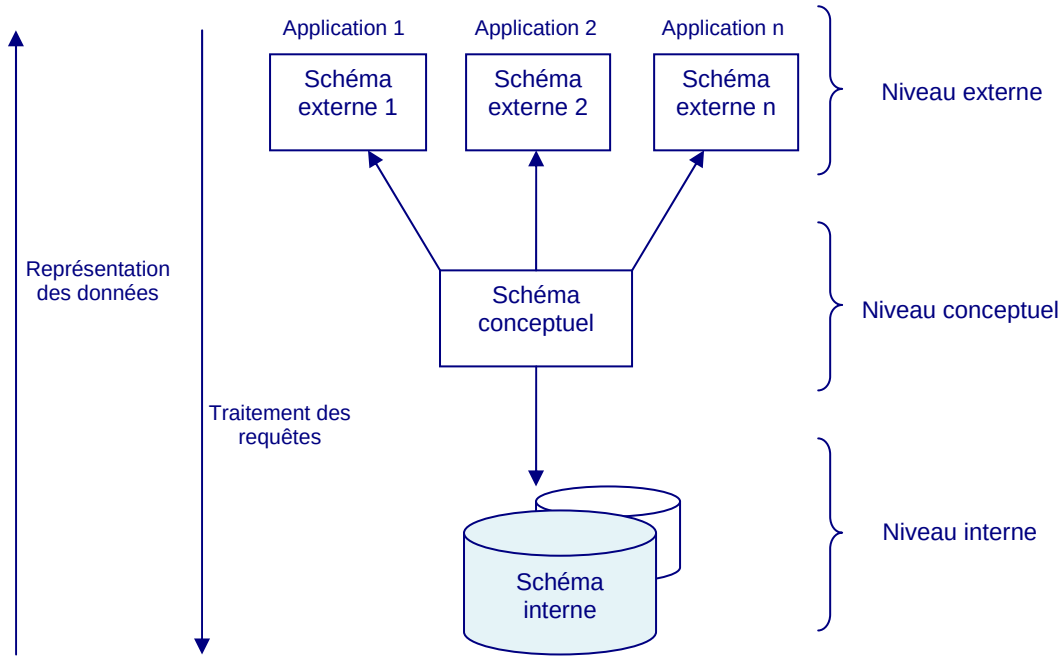
Chapitre 2
Architecture des SGBD

L'architecture ANSI/SPARC, dite aussi architecture de référence, se compose de trois niveaux appelés, respectivement, niveau ***interne***, niveau ***conceptuel*** et niveau ***externe***.

- Le niveau interne (ou niveau physique) est le niveau relatif à la mémoire physique. Il s'agit du niveau où les données sont réellement enregistrées.
- Le niveau externe (ou niveau logique utilisateur) est le niveau relatif aux utilisateurs. Il s'agit du niveau où les utilisateurs voient les données.
- Le niveau conceptuel (ou niveau logique communautaire ou logique) est le niveau intermédiaire entre les deux autres.

Par ailleurs, parmi les objectifs premiers d'un SGBD figurent l'indépendance physique et l'indépendance logique des programmes aux données.

2.1. Les trois niveaux



Les trois niveaux de l'architecture ANSI/SPARC

2.1.1. Le niveau interne

Le niveau interne est une représentation de bas niveau de l'ensemble de la base de données. Elle est constituée de plusieurs occurrences des divers types d'enregistrements internes.
La vue interne est décrite par un **schéma interne**. Ce schéma ne définit pas seulement les divers types d'enregistrements mémoire mais spécifie également les index existants, comment les champs mémoires sont représentés, le séquencement physique des enregistrements, etc.

2.1.2. Le niveau externe

Le niveau externe est le niveau de l'utilisateur. Un utilisateur peut aussi bien être un programmeur d'application que tout autre utilisateur final ayant n'importe quel niveau de compétence. Ce niveau est représenté par plusieurs **schémas externes** appelés aussi **vues**.

2.1.3. Le niveau conceptuel

La vue conceptuelle donne la représentation de l'ensemble des informations contenues dans la base de données. Comme la vue externe, cette représentation est abstraite comparée à la représentation physique des données. En général, elle est également très différente de la vue qu'un utilisateur particulier a de la base de données.
La vue conceptuelle correspond aux diverses occurrences des types d'enregistrements conceptuels. Elle est définie par un **schéma conceptuel**.

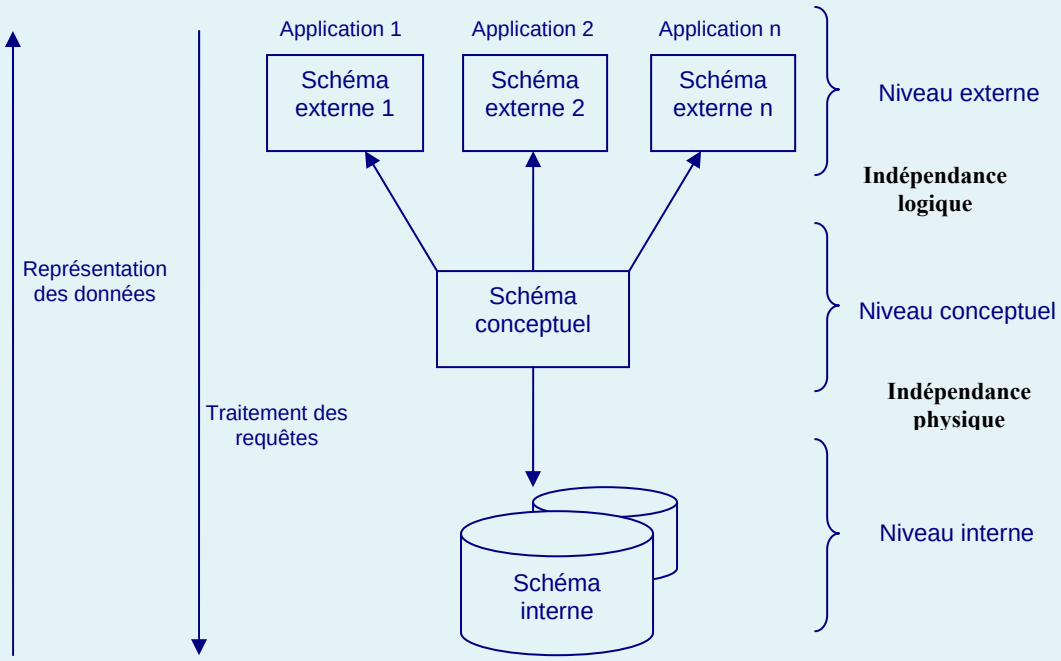
2.2. Indépendance entre les niveaux

2.2.1. Indépendance physique

Un des objectifs essentiels des SGBD est de permettre de réaliser l'indépendance des structures de stockage aux structures de données du monde réel, c'est-à-dire entre le schéma interne et le schéma conceptuel.
Le schéma interne et le schéma conceptuel décrivent les données, mais à des niveaux différents. L'indépendance physique permet donc de modifier le schéma interne sans avoir à modifier le schéma conceptuel, en tenant compte seulement des critères de performance et de flexibilité d'accès.
On pourra par exemple ajouter un index, regrouper deux fichiers en un seul, changer l'ordre ou le codage des données dans un enregistrement, sans mettre en cause les entités et associations définies au niveau conceptuel.

2.2.2. Indépendance logique

Le schéma conceptuel résulte d'une synthèse des vues particulières de chaque groupe de travail utilisant la base de données, c'est-à-dire d'une intégration de schémas externes. Ainsi, chacun doit pouvoir se concentrer sur les éléments constituant son centre d'intérêt, c'est-à-dire qu'un utilisateur doit pouvoir ne connaître qu'une partie des données de la base au travers de son schéma externe, encore appelé **vue**.
L'indépendance logique est donc la possibilité de modifier un schéma externe (une vue) sans modifier le schéma conceptuel. Elle assure aussi l'indépendance entre les différents utilisateurs, chacun percevant une partie de la base via son schéma externe, selon une structuration voire un modèle particulier.
Il doit par exemple être possible d'ajouter des attributs, d'en supprimer d'autres, d'ajouter et de supprimer des associations, d'ajouter ou de supprimer des entités dans des schémas externes sans modifier la plus grande partie des applications.



Types d'indépendances et modèles d'une base de données

2.3. Importance de la conception d'une base de données

Pour pouvoir mettre en œuvre les différents besoins des utilisateurs, l'équipe de conception doit planifier soigneusement les étapes de la conception. Diverses méthodes de conception (Merise, UML …) existent définissant un ensemble d'étapes à suivre et de formalismes à adopter.

2.3.1. Etapes de conception

La conception d'une base de données suit généralement les trois étapes suivantes :

a) **l'analyse des besoins** consiste à identifier les besoins de l'entreprise et à les convertir en besoins d'un système. C'est une phase au cours de laquelle on rassemble des informations qui serviront à la modélisation des données.

b) **la modélisation des données** comportant :

1. L'expression de la sémantique de la représentation du monde réel (niveau d'abstraction conduisant à la définition du schéma conceptuel) par des données et les liens entre ces données
2. La traduction du schéma conceptuel dans un modèle de données de type hiérarchique, en réseau, relationnel …
3. La normalisation consistant à représenter les objets du modèle de données sous forme de tables normalisées afin de réduire la quantité de données redondantes présentes dans la base de données.
4. L'expression de l'utilisation (définition des vues, des accès logiques …)

c) **Expression de l'implantation en machine** (choix des méthodes d'accès, des chemins d'accès …) permettant d'assurer de bonnes performances d'exploitation.

2.3.2. Equipe de conception et distribution des tâches

On met en place une équipe de conception constituée d'une ou de plusieurs personnes. L'équipe de conception typique d'un projet relativement important comporte principalement :

- un chef de projet responsable de la gestion globale du projet
- des concepteurs responsables des tâches de

    - définition des besoins opérationnels, des données et des fonctions
    - analyse/modélisation des données
    - modélisation des processus/fonctions
- des développeurs
    - responsables du codage
    - conduisant les sessions de tests
- un administrateur de la base de données qui
    - administre la base de données
    - détermine l'emplacement physique de la base de données
- des utilisateurs finaux qui
    - interagissent avec le système
    - effectuent des tâches quotidiennes.

2.3.3. Outils de conception automatisée

Les outils de conception automatisée, encore appelés CASE (Computer Aided Systems Engineering) ou AGL (Atelier de Génie Logiciel) sont des applications à interface graphique (GUI : Graphic User Interface) qui facilitent la conception et la création d'une base de données.
Oracle Designer est, par exemple, l'un des AGL les plus puissants du marché.
Parmi les fonctionnalités de tels outils :

- l'identification des besoins de l'activité et de l'utilisateur
- la modélisation des entités et de leurs relations
- la génération d'un langage destiné à créer les objets de la base de données
- la modélisation du flux de données d'un organisme
- la modélisation des processus opérationnels
- le contrôle des versions de la base de données et de l'application
- la génération de la documentation et des comptes rendus des utilisateurs.

Exercices : Questions à choix multiples

Q1 : Les trois niveaux
L'architecture ANSI/SPARC se compose des niveaux suivants :

a) Niveau interne, niveau externe et niveau conceptuel.
b) Niveau physique, niveau logique et niveau logique utilisateur.
c) Niveau conceptuel et niveau externe.

Q2 : Les indépendances entre niveaux
Parmi les objectifs premiers d'un SGBD figurent

a) L'indépendance conceptuelle des programmes aux données.
b) L'indépendance logique conceptuelle.
c) L'indépendance physique des programmes aux données.
d) L'indépendance logique des programmes aux données.

Q3 : Niveau externe
Le niveau externe est :

a) Le niveau de l'utilisateur.
b) Une représentation par plusieurs schémas externes appelés aussi vues.
c) Une représentation de bas niveau de l'ensemble de la base de données.

Q4 : Niveau interne
Le niveau interne est :

a) Une représentation de bas niveau de l'ensemble de la base de données.
b) constitué de plusieurs occurrences des divers types d'enregistrements internes.
c) décrit par un schéma interne.
d) le séquencement physique des enregistrements, comment les champs mémoires sont représentés, les index existants…

Q5 : Niveau conceptuel
La vue conceptuelle

a) Donne la représentation de l'ensemble des informations contenues dans la base de données.
b) est abstraite comparée à la représentation physique des données.
c) est très différente de la vue qu'un utilisateur particulier a de la base de données.
d) correspond aux diverses occurrences des types d'enregistrements conceptuels.
e) est définie par un schéma conceptuel.

Q6 : Indépendance physique
L'indépendance physique permet

a) De modifier le schéma interne sans avoir à modifier le schéma conceptuel.
b) De modifier le schéma conceptuel sans avoir à modifier le schéma interne.
c) De modifier le schéma conceptuel sans avoir à modifier les schémas externes.

Q7 : Indépendance logique
L'indépendance logique permet

a) De modifier le schéma interne sans avoir à modifier le schéma conceptuel.
b) De modifier le schéma conceptuel sans avoir à modifier le schéma interne.
c) De modifier le schéma conceptuel sans avoir à modifier les schémas externes.

d) De modifier un schéma externe sans avoir à modifier le schéma conceptuel.

Q8 : Etapes de conception
Parmi les étapes suivies lors de la conception d'une base de données :

a) Perception du monde réel et capture des besoins.
b) Elaboration du schéma conceptuel.
c) Conception et affinement du schéma logique.
d) Elaboration du schéma physique.

Q9 : Affinement du schéma logique
L'affinement du schéma logique consiste à :

a) S'assurer que le schéma logique obtenu est un « bon » schéma.
b) S'assurer que le schéma logique obtenu est un schéma comportant toutes les redondances d'informations nécessaires.
c) Utiliser la théorie de la normalisation pour obtenir de « bons » schémas relationnels.
d) Regrouper ou décomposer les tables de manière à représenter fidèlement le monde réel modélisé.

Q10 : Equipe de conception
L'équipe de conception typique d'un projet relativement important comporte principalement :

a) des chargés de cas responsables des processus d'activités
b) un chef de projet responsable de la gestion globale du projet
c) des concepteurs et des développeurs
d) un administrateur de la base de données
e) des utilisateurs finaux

Chapitre 3
Le modèle Entités/Associations

Les aspects importants de la réalité à représenter, ou domaines d'application, doivent être décrits d'une manière abstraite, indépendante de toute technologie. Le modèle Entité-association permet de décrire un domaine d'application sous la forme d'ensembles d'entités, dotées de propriétés et en association les unes avec les autres, et ce, sans référence aux notions techniques de tables, colonnes ou index.

3.1. Concepts de base

3.1.1. Entité

Une entité est un objet pourvu d'une existence propre et conforme aux choix de gestion de l'entreprise.
Le domaine d'application est perçu comme étant constitué d'entités concrètes ou abstraites. Ainsi, dans le contexte du commerce, on peut cerner un domaine d'application dans lequel on repère des clients, des commandes et des produits. On considère que chacun d'eux est une entité du domaine et que chaque entité appartient à une classe ou type d'entités. On définit naturellement quatre types d'entités qu'on nommera CLIENT, COMMANDE et PRODUIT. On représentera ces entités d'une manière graphique, comme indiqué sur la figure 1.

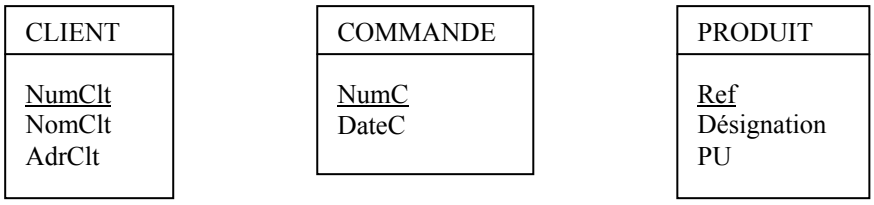


Figure 1 : Formalisme des entités

Ces quelques exemples montrent qu'une entité peut correspondre à des objets concrets inanimés (produits), des objets concrets animés (clients) ou des événements (commandes).
Une occurrence d'une entité est un élément individualisé appartenant à cette entité.

3.1.2. Association

Une association entre entités est une association perçue dans le réel entre deux ou plusieurs entités. Une association est dépourvue d'existence propre.
Une commande est liée au client qui l'a passée ; il existe donc une association entre cette commande et ce client. On dira que toutes les associations de cette nature appartiennent au type d'association *Passer* entre les deux entités CLIENT et COMMANDE.
Lorsqu'une entité intervient dans un type d'association, on dit qu'elle joue un rôle. On utilisera d'ailleurs ce terme pour désigner une des extrémités d'une association. Une même entité peut jouer deux rôles dans une même association.
On représentera une association d'une manière graphique, comme indiqué sur la figure 2.

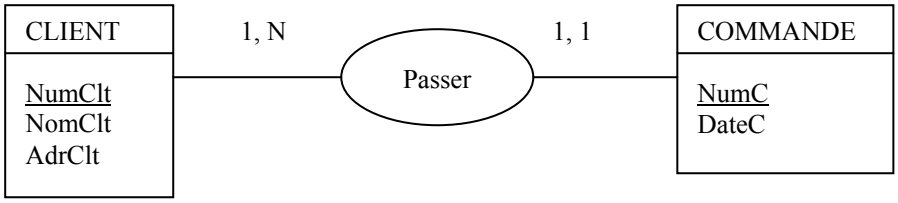


Figure 2 : Formalisme d'une association

Propriétés d'une association

a) Classe fonctionnelle d'une association

Soient deux entités E et F et une association A entre E et F.

Cette propriété décrit le nombre maximum d'occurrences de l'entité F pour chaque occurrence de l'entité E et inversement. On est ainsi amené à définir trois classes fonctionnelles d'associations : un à plusieurs, un à un et plusieurs à plusieurs.

- association de **type 1:1** (ou un-à-un) si à une occurrence de l'entité E peut correspondre par l'association A au plus une occurrence de l'entité F et que, réciproquement à une occurrence de l'entité F ne peut correspondre au plus qu'une occurrence de l'entité E.
- association de **type 1:n** (ou un-à-plusieurs) : si à une occurrence de l'entité E peut correspondre par l'association A plusieurs occurrences de l'entité F mais à une occurrence de l'entité au plus une occurrence de l'entité E.
- association de **type n:n** (ou plusieurs-à-plusieurs) : si à une occurrence de l'entité E peuvent correspondre plusieurs occurrences de l'entité F et réciproquement.

b) Type d'associations obligatoire ou facultatif

On peut imposer qu'une association soit obligatoire pour une entité qui y participe.

c) Cardinalités d'une association

Chaque entité participant à une association y est caractérisée par un couple de valeurs min-max appelé cardinalités.

d) Dimension d'une association

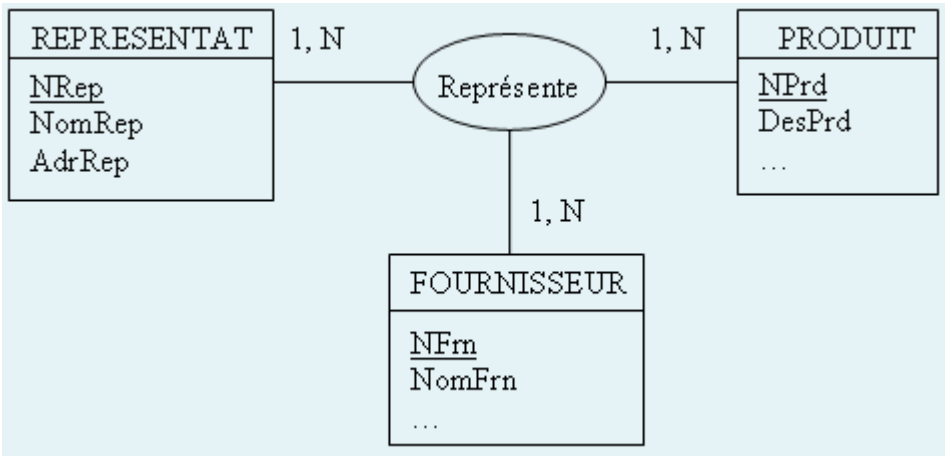
C'est le nombre d'entités participant à l'association. Une association entre deux entités est appelée association binaire. Une association entre trois entités est appelée association ternaire. Une association entre n entités est appelée association n-aire.

Exemple 1 : entre les entités ETUDIANT et ENSEIGNEMENT on peut considérer l'association binaire INSCRIT pour traduire le fait qu'un étudiant est inscrit à un enseignement.

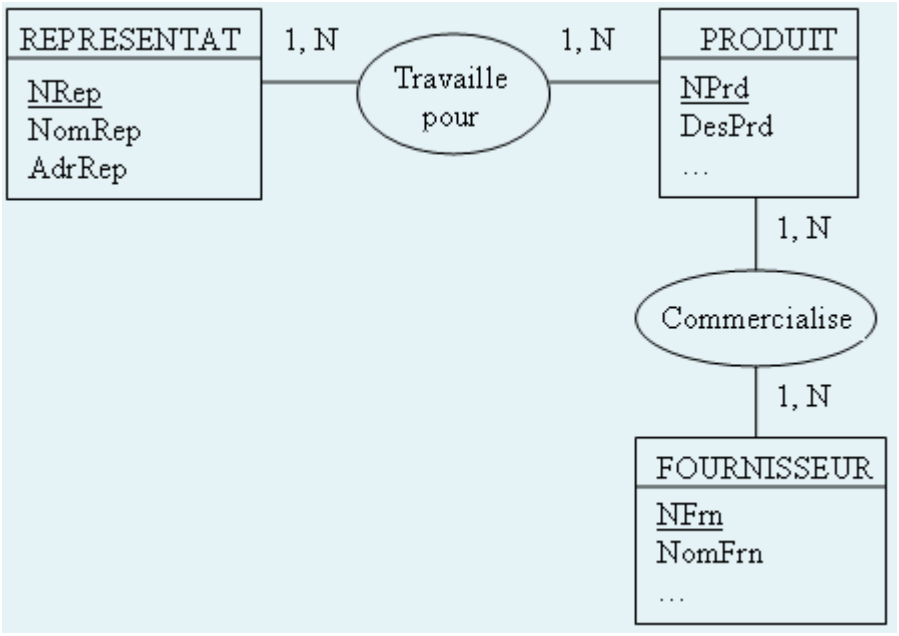
INSCRIT

ETUDIANT _____ENSEIGNEMENT

Exemple 2:



L'association ternaire « Représente » ne peut être décomposée en deux associations binaires que si l'on sait que tout représentant travaillant pour un fournisseur représente **tous** les produits de ce fournisseur. La décomposition sera la suivante :



e) Association réflexive

C'est une association d'une entité sur elle-même. En effet, il est parfaitement possible d'établir une association entre une entité et elle-même, définissant par là une association cyclique.

Exemple: pour traduire le fait que Irène Curie est la fille de Marie Curie on pourra utiliser une association A-POUR-MERE entre les deux entités représentant ces personnes.

A-POUR-MERE

PERSONNE_____ PERSONNE

3.1.3 Attributs

Un attribut ou une propriété est une donnée élémentaire que l'on perçoit sur une entité ou sur une association entre objets.
Chaque client est caractérisé par un numéro, un nom et une adresse. On modélisera ces faits en dotant l'entité CLIENT des attributs NumClient, Nom et Adresse.
On spécifiera le type de chaque attribut : numérique, caractère, date, etc. ainsi que sa longueur.
Un attribut d'une association est une propriété qui dépend de toutes les entités intervenant dans l'association.

Exemple: L'association INSCRIT définie entre les entités ETUDIANT et MODULE a pour attribut l'année de première inscription de l'étudiant à l'enseignement. Cette année d'inscription est attribut de l'association et non de l'une des entités, car il faut connaître l'étudiant et le module pour pouvoir la déterminer.

3.1.4. Les identifiants

En général, une entité est dotée d'un attribut qui identifie les entités de ce type. L'entité CLIENT, par exemple, possède un attribut NumClient tel qu'à tout instant les occurrences de l'entité CLIENT ont des valeurs de NumClient distinctes. En d'autres termes, étant donné une valeur quelconque de NumClient, on a la garantie qu'il n'y aura, à aucun moment, pas plus d'une occurrence de l'entité CLIENT possédant cette valeur. On dira que NumClient est un identifiant de CLIENT. En outre, il peut arriver qu'une entité possède plus d'un identifiant. Dans ce cas, l'un d'eux peut être déclaré primaire tandis que tous les autres sont secondaires.

Il se peut que la valeur d'un attribut ne soit pas connue au moment où les informations sur l'entité sont enregistrées. Si on admet que cet attribut puisse ne pas avoir de valeur pour certaines occurrences de l'entité, on le déclarera facultatif. Sinon cet attribut est obligatoire.

L'identifiant d'une entité est un attribut particulier de l'entité tel qu'à chaque valeur de la propriété corresponde une et une seule occurrence de l'entité.
L'identifiant d'une association est l'identifiant obtenu par concaténation des identifiants des entités participant à la relation.
Dans le diagramme E/A, les clés sont soulignées.

3.1.5. Les Cardinalités

La cardinalité d'une entité par rapport à une association s'exprime par deux nombres appelés cardinalité minimale et cardinalité maximale.

La cardinalité minimale (égale à 0 ou 1) est le nombre de fois minimum qu'une occurrence d'une entité participe aux occurrences de l'association.
Si la cardinalité minimale est égale à 0, c'est qu'il existe parmi toutes les occurrences de l'entité au moins une occurrence ne participant pas aux occurrences de l'association.
Si la cardinalité minimale est égale à 1, ceci correspond au fait que chaque occurrence de l'entité participe toujours à une occurrence de l'association.

La cardinalité maximale indique le nombre de fois maximum qu'une occurrence de l'entité participe aux occurrences de la relation.

Remarques: Le minimum m peut valoir 0, 1 ou un entier strictement plus grand que 1. Le maximum M peut valoir 1 ou une valeur n>1, n n'étant souvent pas précisé de manière numérique, faute de connaissance suffisante.

Exemple:
L'association Appartient entre PROPRIETAIRE et VEHICULE a pour cardinalités (0,n) du côté de l'entité VEHICULE et (1,n) du côté de l'entité PROPRIETAIRE car certains véhicules sont abandonnés (0,n) mais il faut posséder au moins un véhicule pour être propriétaire (1,n).

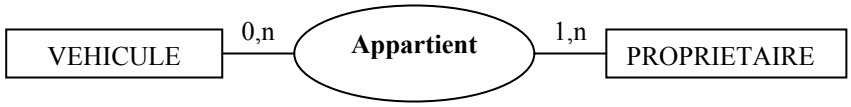


Figure 3 : Exemple d'association de type n : n

Contrainte d'identité fonctionnelle (CIF): Quand on détermine, entre une association et une entité, *une cardinalité présentant les valeurs 0,1 ou 1,1*, l'association est particulière. On l'appellera alors *Contrainte d'identité fonctionnelle* (CIF) : Cette association particulière n'est en général pas nommée. Elle indique que l'une des entités est totalement déterminée par la connaissance de l'autre ; par exemple si on connaît une commande bien précise, on connaît un client bien précis...

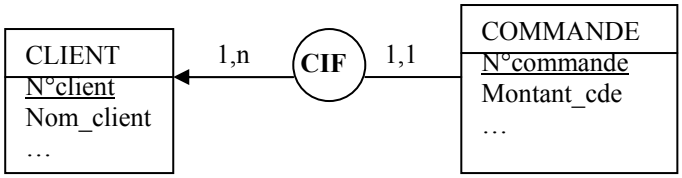


Figure 4 : Exemple de relation de type CIF

3.2. Diagramme Entité-Association

Un diagramme Entité-Association (E/A) décrit la structure d'ensemble d'une base de données en combinant les objets graphiques suivants :

- des **rectangles** qui représentent des ensembles d'objets, c'est-à-dire des **entités** concrètes ou abstraites (par exemple : lecteur, ouvrage, compte bancaire, client...)
- des **attributs** relatifs aux entités (le nom, l'adresse, le titre, la cote, numéro,...)
- des **ellipses**, qui représentent des **associations** ("a emprunté", "possède le compte", "suit le cours de",...)
- des **rôles** qui relient les entités aux associations annotés par les cardinalités.

Dans ce diagramme les identifiants sont soulignés.

Exemple 1 : Base de données commerciale

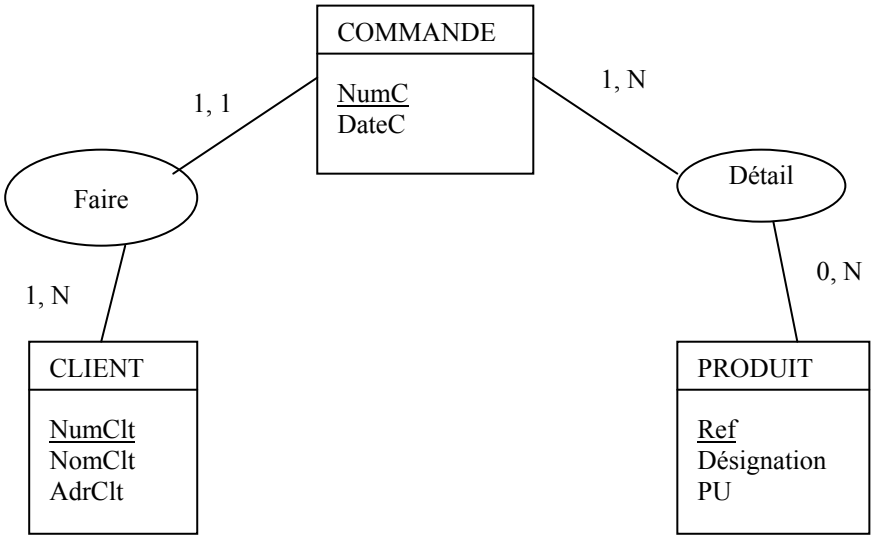


Figure 5 : Modèle E/A d'une base de données commerciale

Exemple 2 : Base de données matrimoniale

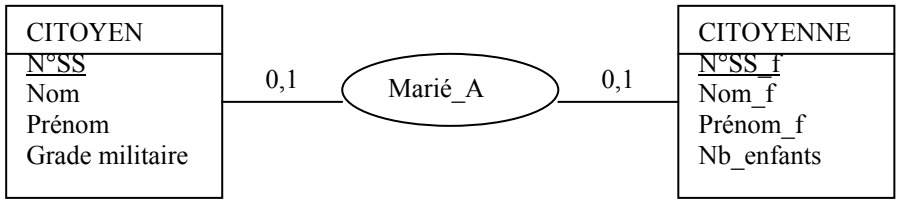


Figure 6 : Modèle E/A d'une base de données matrimoniale

Exemple 3 : Base de données d'enseignement

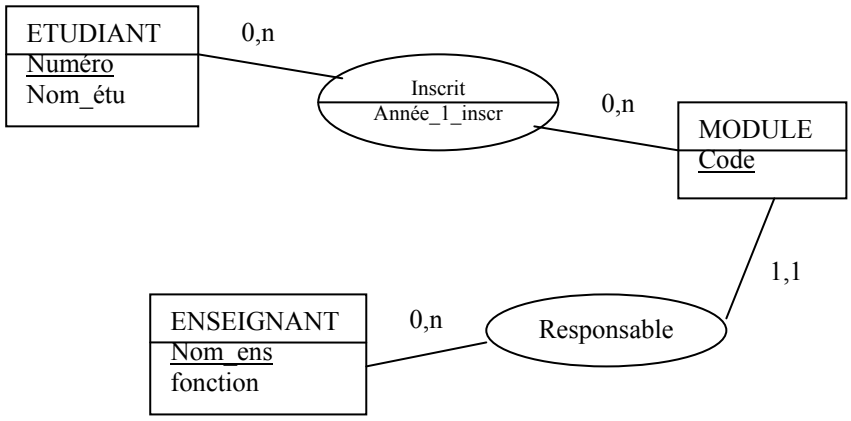


Figure 7 : Le modèle E/A d'une base de données d'enseignement

Le diagramme ENTITE-ASSOCIATION ci-dessus modélise entre deux entités ETUDIANT et MODULE, une association INSCRIT qui traduit qu'un étudiant est inscrit à un module.
Pour une entité ENSEIGNANT, il exprime qu'un enseignant est RESPONSABLE d'un module.

3.3. Dictionnaire des données

Le **dictionnaire des données** liste les entités et leurs attributs, en spécifiant le domaine de chacun ainsi que leur catégorie :

- données élémentaires (information stockée)
- données d'information déduite ou calculée d'utilisation fréquente (ce qui évite de refaire le calcul plusieurs fois) ainsi que les règles de calcul
- données calculées de type situation ou historique (total HT des commandes par mois...)
- paramètres utilisés dans des cas particuliers (TVA) ...

Il se présente sous forme d'une grille d'analyse :

Nom de donnée	Format	Type					Règle de calcul	Contrainte d'intégrité	Document
		Elémentaire	Calculée	Paramètre	Signalétique	Situation			
Nom client	Alpha	X			X				Facture
Code postal	Num	X			X				Facture
Ville client	Alpha	X			X				Facture
…									…
Total HT	Num		X			X	Somme		Facture
Taux TVA	Num			X				18,60%	Facture

3.4. Règles de validation

Ces règles doivent être respectées pour la cohérence du modèle Entité-Association. Il s'agit de règles de vérification et de normalisation d'un modèle E/A

Règle 1 : Existence d'un identifiant pour chaque entité.

Règle 2 : Toutes les propriétés d'une entité, autres que l'identifiant, doivent être en dépendance fonctionnelle complète et directe de l'identifiant.

Règle 3 : Toutes les propriétés d'une association doivent dépendre complètement de l'identifiant de l'association ; chaque attribut doit dépendre de tout l'identifiant et non d'une partie de cet identifiant.

Règle 4 : Un attribut ne peut apparaître qu'une seule fois dans un même modèle E/A, c'est ainsi qu'il ne peut qualifier qu'une seule entité ou une association.

Règle 5 : Les attributs qui sont le résultat d'un calcul ne doivent pas, en principe, figurer dans un modèle E/A sauf s'ils sont indispensables à la compréhension de celui-ci.

3.5. Généralisation et hiérarchie

Un ensemble d'entités E_1 est un sous-ensemble de E_2 si toute occurrence de E_1 est aussi une occurrence de E_2. L'ensemble d'entités E_1 hérite des attributs de E_2.

Un ensemble d'entités E est une généralisation de E_1, E_2, E_n si chaque occurrence de E est aussi une occurrence d'une et une seule entité E_1, E_2, ..., E_n. Les ensembles E_1, E_2, ..., E_n sont des spécialisations de l'ensemble d'entités E. Les ensembles d'entité E_1, E_2, E_n héritent des *attributs de E* et possèdent en outre des attributs spécifiques qui expriment leur **spécialisation**.

Notation "EST-UN" (*IS A)* : B "EST-UN" A si l'ensemble A est une extension de B ou B un cas particulier de A.

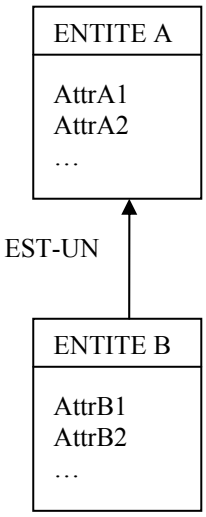


Figure 8 : La relation « EST-UN »

Exemple 1 : L'ensemble des VEHICULES est une généralisation de l'ensemble des AUTOMOBILES et des CYCLES.

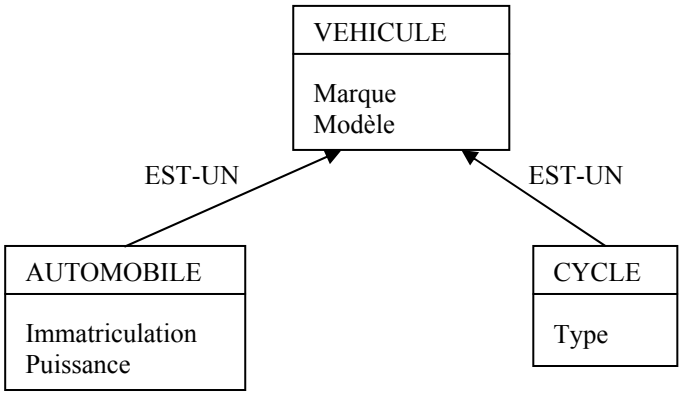


Figure 9

Exemple 2 : L'ensemble des PILOTES est un sous-ensemble de l'ensemble des EMPLOYES d'une compagnie aérienne.

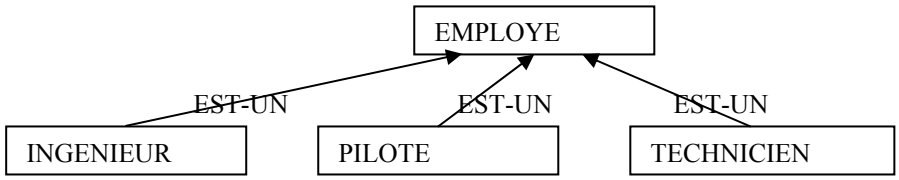


Figure 10

Exercices

Partie 1 : Questions à choix multiples

Q1 : Entité
Une entité est :

a) Un objet concret pourvu d'une existence propre et conforme aux choix de gestion de l'entreprise.
b) Un objet abstrait pourvu d'une existence propre et conforme aux choix de gestion de l'entreprise.
c) Un objet concret ou abstrait pourvu d'une existence propre et conforme aux choix de gestion de l'entreprise.

Q2 : Association
Une association entre entités est :

a) Une relation perçue dans le monde réel entre deux ou plusieurs entités.
b) Un objet concret pourvu d'une existence propre.

Q3 : Cardinalités
La cardinalité d'une entité par rapport à une association s'exprime par :

a) La cardinalité moyenne indiquant le nombre moyen d'occurrences qu'une occurrence de l'entité participe aux occurrences de l'association.
b) La cardinalité minimale et la cardinalité maximale qui sont respectivement le nombre de fois minimum et le nombre de fois maximum qu'une occurrence d'une entité participe aux occurrences de l'association.

Q4 : Identifiants
L'identifiant d'une entité est :

a) Un attribut particulier de l'entité tel qu'à chaque occurrence de l'entité corresponde une et une seule valeur de l'attribut.
b) Un attribut particulier de l'entité tel qu'à chaque valeur de l'entité corresponde une et une seule valeur des attributs.
c) Un attribut particulier de l'entité tel qu'à chaque valeur de l'attribut corresponde une et une seule occurrence de l'entité.

Q5 : Schéma E/A
Parmi les objets graphiques décrivant la structure d'ensemble d'une base de données dans un diagramme Entité-Association (E/A) :

a) des **rectangles** qui représentent des ensembles d'objets, c'est-à-dire des **entités** concrètes ou abstraites (par exemple : lecteur, ouvrage, compte bancaire, client...)
b) des **tables** qui représentent ces objets selon le modèle relationnel
c) des **ellipses**, qui représentent des **associations** ("a emprunté", "possède le compte", "suit le cours de",...)
d) des **rôles** qui relient les entités aux associations annotés par les cardinalités.

Partie 2 :

Exercice 2.1 (Centre médical)

On vous donne un schéma E/A représentant des visites dans un centre médical. Répondez aux questions suivantes **en fonction des caractéristiques de ce schéma** (i.e.: indiquez si la situation décrite est conforme avec ce schéma, indépendamment de sa vraisemblance).

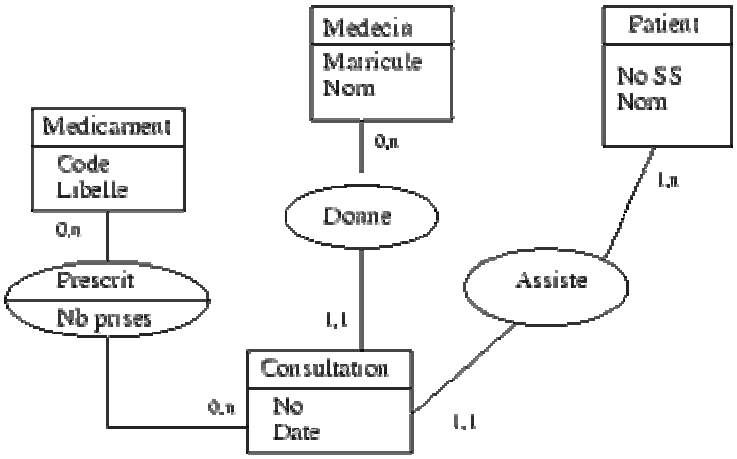


Figure 1.1 : Centre médical

A) Un patient peut-il effectuer plusieurs visites ?

B) Un médecin peut-il recevoir plusieurs patients dans la même consultation ?

C) Peut-on prescrire plusieurs médicaments dans une même consultation ?

D) Deux médecins différents peuvent-ils prescrire le même médicament ?

Exercice 2.2 (Tournoi de tennis)

Le second schéma représente des rencontres dans un tournoi de tennis.

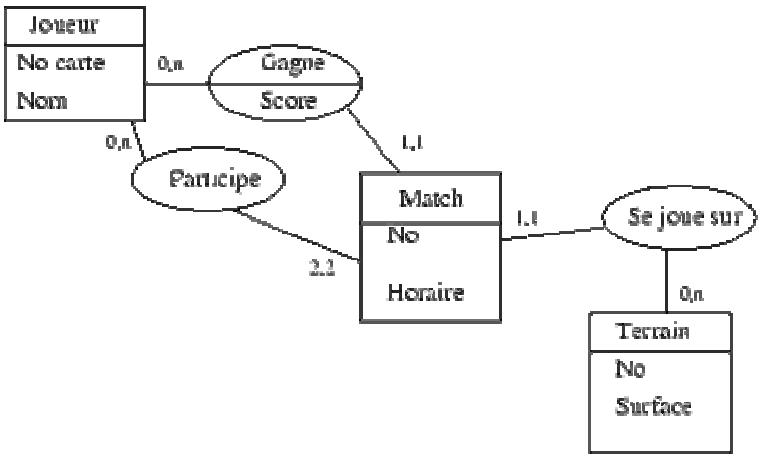


Figure 1.2 : Tournoi de tennis

A) Peut-on jouer des matchs de double ?
B) Un joueur peut-il gagner un match sans y avoir participé ?
C) Peut-il y avoir deux matchs sur le même terrain à la même heure ?

Exercice 2.3 (Un journal)

Voici le schéma E/A du système d'information (très simplifié) d'un quotidien.

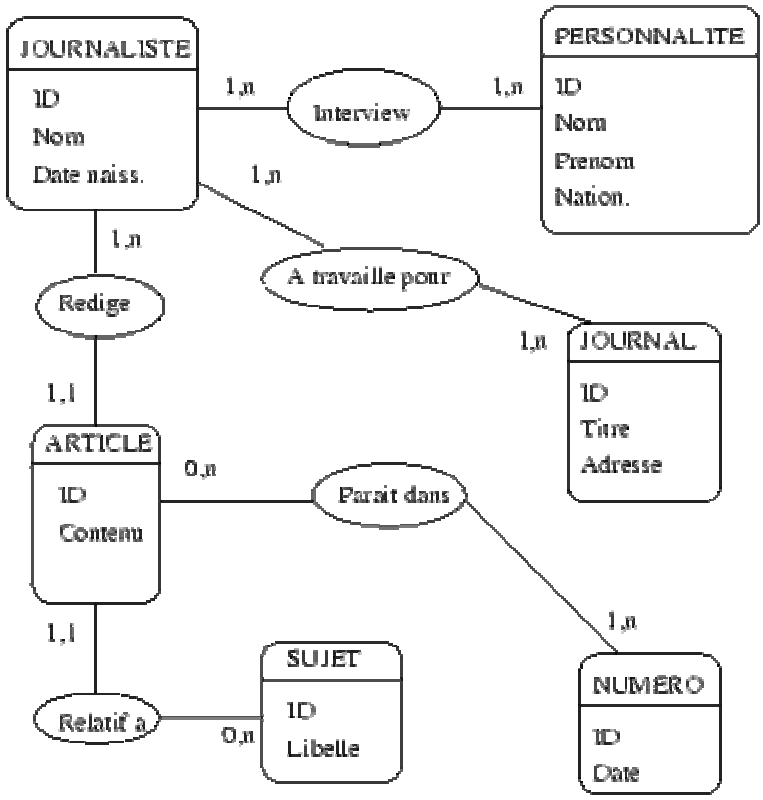


Figure 1.3 : Journal

A) Un article peut-il être rédigé par plusieurs journalistes ?
B) Un article peut-il être publié plusieurs fois dans le même numéro ?
C) Peut-il y avoir plusieurs articles sur le même sujet dans le même numéro ?

Partie 3 :
Construire le modèle E/A relatif aux bases de données suivantes en donnant les différentes entités, associations et attributs.

Exercice 3.1

Le propriétaire d'un garage de voitures souhaite utiliser une base de données pour traiter les informations concernant les clients, leurs voitures et les réparations effectuées sur ces voitures. On connaît :
- des voitures : le n° d'immatriculation, la marque, le type, l'année.
- des clients : le nom, le prénom, le n° de téléphone.
- des réparations : le n° de réparation, la date, le montant total.

Exercice 3.2

Une base de données doit être conçue pour étudier l'utilisation des ressources informatiques d'une entreprise. L'entreprise est présente sur plusieurs sites géographiques.
A partir de certains sites, les utilisateurs (décrits par un numéro, un profil, un nom et un prénom) accèdent à des applications informatiques (décrites par un numéro, un nom et un domaine). Il est possible qu'un même utilisateur accède à une application à partir de sites différents. Chaque site a ses propres applications et il est décrit par son numéro et son nom.
Chaque application demande un ensemble de logiciels (décrits par le nom et le producteur) mis à la disposition de tous les sites et matériels (décrits par un numéro, un type, une désignation et un constructeur) propres à chaque site.

Exercice 3.3

Une fédération sportive désire informatiser l'organisation de ses tournois.
Les clubs de la fédération sont dotés d'un numéro et d'un nom. Chaque club attribue à ses équipes un numéro unique au sein du club. Chaque joueur d'un club appartient à une seule équipe. Un joueur est décrit par un numéro matricule attribué par la fédération, son nom, son prénom, son adresse, son "numéro de maillot" et sa "place" sur le terrain.
Un tournoi est décrit par un numéro et sa date. Chaque tournoi est organisé par un club de la fédération. Au cours d'un tournoi, les différentes équipes qui y participent s'affrontent dans des matchs. Un match est décrit par un numéro au sein du tournoi et son résultat.

Chapitre 4
Le modèle relationnel

Le modèle relationnel a été proposé par Codd au début des années 70, avec comme objectif essentiel l'accroissement de l'indépendance des programmes vis-à-vis de la représentation des données.

I. Concepts de base

I.1. Domaine

Un domaine est un ensemble de valeurs.
Exemple:
Domaines D1 = {entiers}.
D2 = {chaînes de caractères}.

I.2 Attribut

Un attribut est une variable prenant ses valeurs dans un domaine.
Exemple :

attribut A1=NCl à valeurs dans D1,
attribut A2=NomCl à valeurs dans D2,
attribut A3=AdrCl à valeurs dans D2.

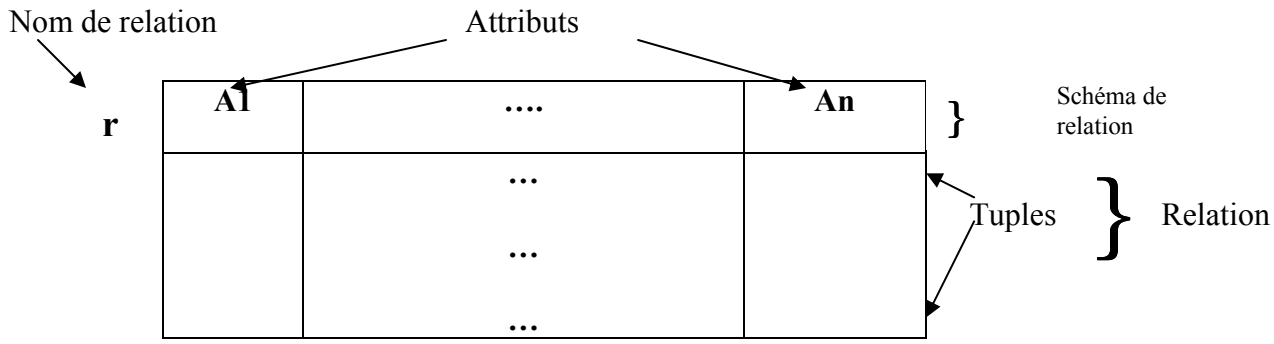
I.3 Relation

Une relation n-aire sur les attributs A1, A2,..., An, de domaines respectifs D1, D2,..., Dn, est un sous-ensemble du produit cartésien des domaines D1, D2, ..., Dn.
Un élément appartenant à une telle relation sera appelé n-uplet ou Tuple. Il sera noté (d1,d2,….dn) où di $\in$ Di, $\forall$ $1 \leq i \leq n$.
L'ensemble des n-uplets d'une relation sera appelé extension de la relation.
Exemple:
Soit r une extension constituée de trois tuples :

r = {(2140, ALI, TUNIS), (1123, SALAH, SFAX), (3425, MHAMED, GABES)}.

I.4 Représentation d'une relation

Chaque tuple (n-uplet) de la relation (appelée aussi Table) est écrit dans une ligne d'un tableau, dont les noms de colonnes sont les attributs de la relation.
Chaque tuple est unique. Les duplications ne sont pas autorisées. L'ordre des tuples est indifférent.



Exemple:
Table Client :

NCl	NomCl	AdrCl
2140	ALI	TUNIS
1123	SALAH	SFAX
3425	MHAMED	GABES

I.5 Schéma de relation /Contraintes d'intégrité

Le schéma **R** d´une relation **r** est la liste des attributs de **r** avec, pour chacun, son domaine, parfois sous-entendu.
Exemple:
Le schéma de r (voir exemple paragraphe précédent) est R = (NCl : D1, NomCl : D2, AdrCl: D2), écrit en abrégé R = (NCl, NomCl, AdrCl).

On dit que r est une relation de schéma R (attention : *ne pas confondre r et R*).
Chaque relation est vue comme un tableau dont les colonnes désignent les attributs et les lignes les n-uplets.
Lorsqu'on repère chaque attribut d'une relation par un nom, l'ordre des colonnes n'est pas important.
On pourra parfois attacher à une relation un ensemble *P* de propriétés que doit vérifier chacun de ses tuples. Ces propriétés sont appelées **Contraintes d'intégrité**. A un schéma de relation sont donc rattachées des contraintes d'intégrité.

Exemples
CLIENT(NCl, NomCl, AdrCl, DateNaissance)
Pour ce schéma de relation, la date de naissance du client doit être inférieur à la date du jour.

COMMANDE (NCmd, DateCmd, DateLivr)
Pour ce schéma de relation, la date de la livraison (DateLivr) doit être supérieure à la date de la commande (DateCmd).

I. 6 Clé d'une relation

Une des contraintes d'intégrité d'un schéma est l'unicité d'identification des n-uplets d'une relation. Cette identification unique est assurée par la notion de **clé** de relation.

Une clé peut être composée d'un seul attribut ou d'une liste d'attributs qui caractérise un tuple (n-uplet) de la relation de manière unique.
Une relation peut avoir plusieurs clés. Une clé comportant un minimum d'attributs sera choisie comme étant ***clé primaire***, les autres clés possibles sont appelées ***clés candidates***. Par convention, la clé primaire d'une relation est soulignée dans un schéma de relation.

Exemple: Client (NCl, NomCl, PrenomCl, AdrCl)

NCl	**NomCl**	**PrenomCl**	**AdrCl**
2140	ALI	Mohamed	TUNIS
1123	SALAH	Ali	SFAX
453	SALAH	Zied	TUNIS
3425	MHAMED	Fatma	GABES

- (NCl), (NomCl, PrenomCl) sont des clés.
- (NCl) est clé primaire.
- (NomCl, PrenomCl) est une clé candidate. Par contre (NomCl) n'est pas une clé à elle seule.

Remarque : si on choisit pour clé (NomCl, AdrCl), la modélisation ne permet pas des homonymes habitant la même ville.

I.7 Clé étrangère

Définition
Une clé étrangère est un ensemble d'une ou de plusieurs colonnes d'une table qui fait référence à une clé primaire d'une autre table. Toutes les valeurs des clés étrangères apparaissent dans une autre relation comme valeurs d'une clé. Par convention, la clé étrangère d'une relation précédée par le symbole # dans un schéma de relation.

Exemple
Soient les schémas de relations suivants

Client(NCl,NomCl,AdrCl) - *Désigne l'ensemble des clients.*
Commande (NCmd, DateCmd, #NCl) - *Désigne l'ensemble des commandes.*

L'attribut NCl dans la table Commande est une clé étrangère. Il prend ses valeurs dans le domaine de valeurs de l'attribut NCl qui se trouve, dans le schéma de relation Client. Une commande est toujours passée par un Client existant dans la base de données.

I.8 Schéma de base de données relationnelle

Une base de données relationnelle est une collection de relations. L'ensemble des schémas des relations de la collection est appelé schéma relationnel de la base. Formellement, un schéma de base de données relationnelle B est un ensemble de schémas de relations R1, R2,..., Rp. Une base de données b de schéma B est un ensemble de relations r1, r2, .., rp de schémas respectifs R1, R2, ..., Rp.

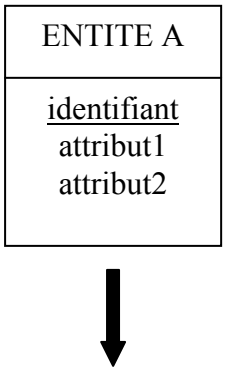
II. Traduction d'un modèle Entité - Association en modèle relationnel

L'absence de SGBD supportant directement le modèle Entité-Association amène à transformer le schéma conceptuel en un schéma conforme au modèle de données du SGBD cible. Dans ce paragraphe, nous présentons les règles qui permettent de passer d'un schéma Entité-Association à un schéma relationnel.

II.1 Traduction des entités

Toute entité est traduite selon les trois règles suivantes :

- L'entité se transforme en une relation.
- L'identifiant de l'entité devient la clé primaire de la relation.
- Les propriétés de l'entité deviennent des attributs de la relation.



ENTITE A (identifiant, attribut1, attribut2)

II.2 Traduction des associations

Nous distinguons deux catégories d'associations : les associations binaires et les associations n-aires. La traduction d'une association s'effectue selon les cardinalités relatives aux entités participant à l'association. Plusieurs cas peuvent se présenter.

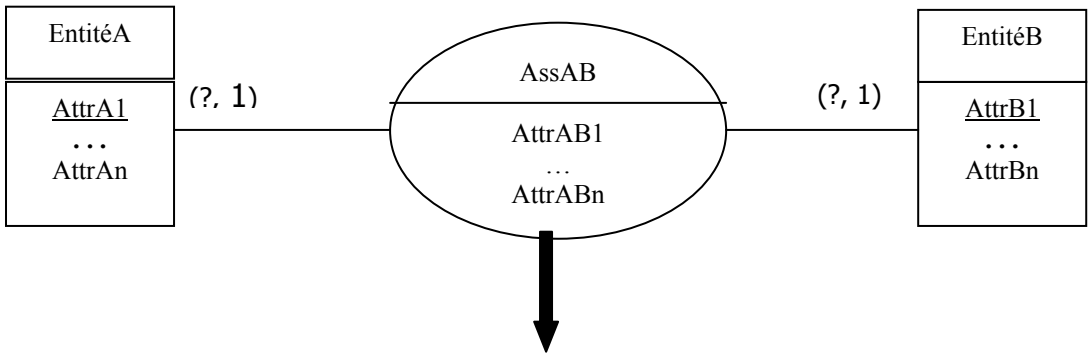
a. Traduction des associations binaires

Soient deux entités A et B reliées par une association AssAB

Cas1 : Association Un-à-Un
Cardinalité entité A 0, 1 ou 1, 1 et Cardinalité entité B 0, 1 ou 1, 1
Pour ce type d'association deux traductions sont possibles :

- ***Solution1 :*** Les deux entités et l'association seront transformées en une seule relation contenant les attributs des deux entités ainsi que les attributs éventuels de l'association, l'identifiant de l'entité A ou de l'entité B sera choisie comme clé primaire de la nouvelle relation. Cette solution est surtout utilisée dans le cas où les deux entités ont des cardinalités 1,1 qui ne sont pas sujettes à des modifications dans le temps (Voir exemple).
- ***Solution2 :*** Les deux entités seront transformées en deux relations. Une de ces deux relatons sera choisie et étendue par la liste des attributs éventuels de

l'association ainsi que de l'identifiant de l'autre entité en tant que clé étrangère. Ce choix se base sur la séquence temporelle de création des entités. L'entité qui sera créée en second lieu aura comme clé étrangère l'identifiant de l'entité créée en premier lieu. Cette solution est la plus adaptée dans le cas où une ou les deux cardinalités minimales sont nulles.



Solution1 : R(AttrA1,…,AtrrAn,AtrrB1,…,AtrrBn, A***ttrAB1,…,AttrABn***)

Attributs de l'Entité A — Attributs de l'Entité B — Attributs de l'association

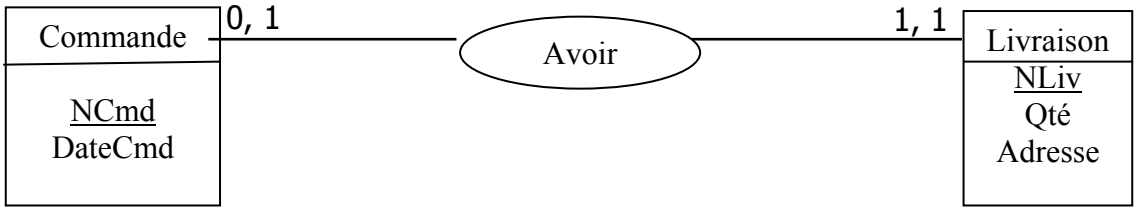
Solution2 :

EntitéA(AttrA1,…,AtrrAn)
EntitéB(AtrrB1,…,AtrrBn, ***#AttrA1***, ***AttrAB1,…,AttrABn***)

Clé étrangère — Attributs de l'association

Exemples :

Exemple1



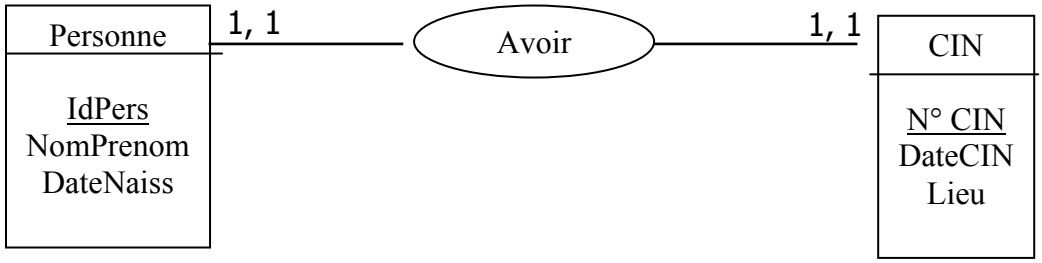
Le modèle relationnel correspondant est le suivant :

Commande (NCmd, DateCmd)
Livraison (NLiv, Qté, Adresse, # NCmd)

Clé étrangère

La relation 'Livraison' a comme clé étrangère l'identifiant de 'Commande' car la création d'une livraison survient après la création d'une commande.

Exemple2



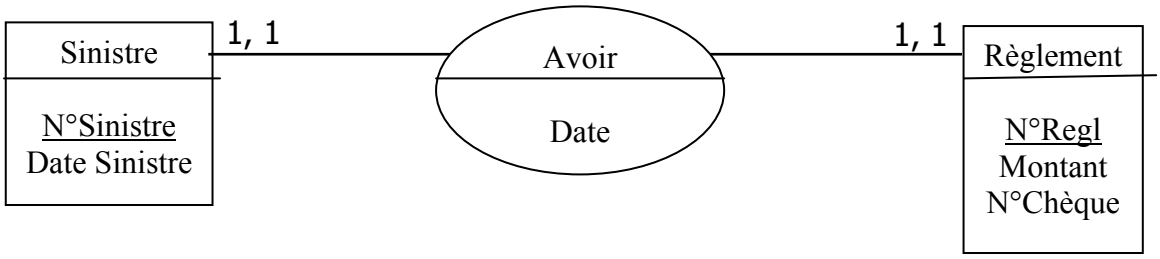
Le modèle relationnel correspondant est le suivant :

Personne (IdPers, NomPrenom, DateNaiss)
CIN (N° CIN, DateCIN, Lieu, # IdPers)

Clé étrangère

La relation 'CIN' a comme clé étrangère l'identifiant de 'Personne' en supposant que la création d'une CIN survient après la création d'une personne. Il est possible également d'utiliser la deuxième solution et de fusionner les deux tables 'Personne' et 'CIN' car les cardinalités 1,1 de chaque côté ne risquent pas de changer dans le temps. En effet, une personne a une et une seule CIN et une CIN correspond à une et une seule personne; et cette règle ne risque pas de changer dans l'avenir.

Exemple3



Le modèle relationnel correspondant est le suivant :

Sinistre (N°Sinistre, Date Sinistre)
Règlement (N°Regl, Montant, N° Chèque, Date, # N°Sinistre)

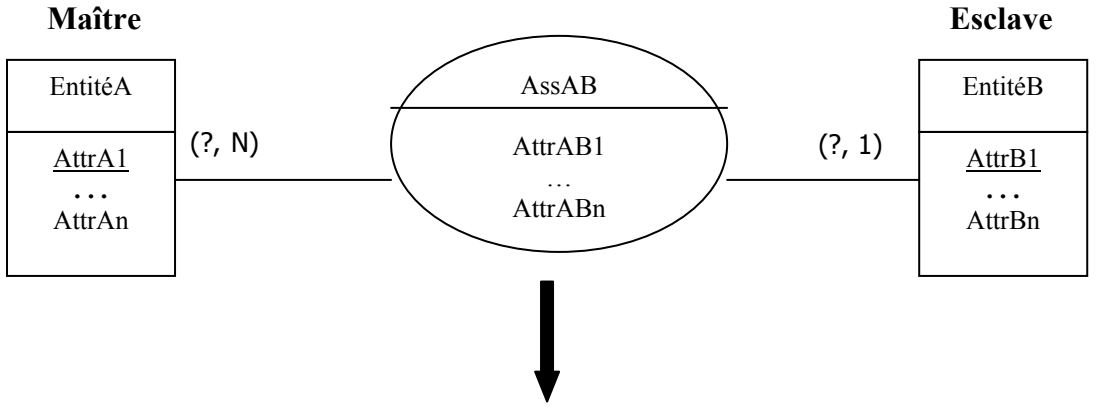
Clé étrangère

La relation 'Règlement' a comme clé étrangère l'identifiant de 'Sinistre' car un règlement fait obligatoirement référence au sinistre qui lui a donné naissance.

Cas2 : Association Un-à-plusieurs (Maître-Esclave) :
Cardinalité entité A (Maître) 0, N ou 1, N et Cardinalité entité B (Esclave) 0, 1 ou 1, 1

Les règles de traduction de ce type d'association sont les suivantes :

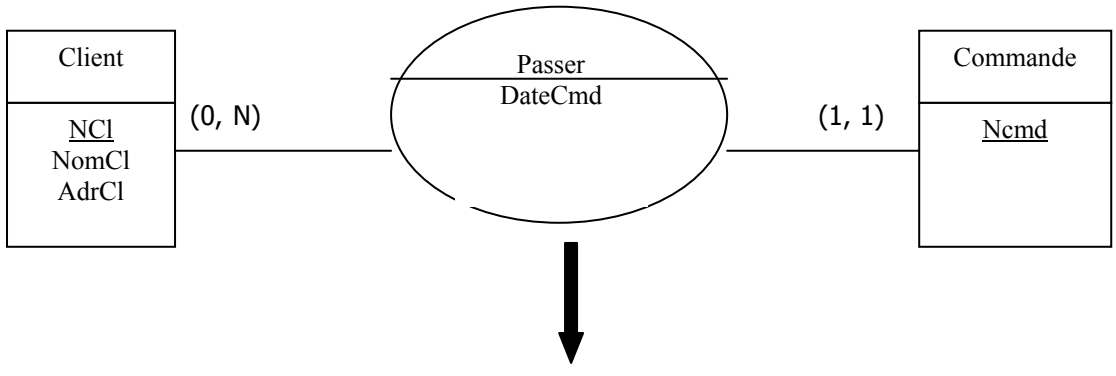
- L'entité Maître (Entité A) devient la relation Maître.
- L'entité Esclave (Entité B) devient la relation Esclave.
- L'identifiant de l'entité Maître devient attribut de la relation Esclave. Cet attribut est désigné comme clé étrangère.
- Les attributs éventuels de l'association (AssAB) migrent vers la relation esclave et deviennent ses attributs.



EntitéA(AttrA1,…,AtrrAn)
EntitéB(AtrrB1,…,AtrrBn, ***#AttrA1***, ***AttrAB1,…,AttrABn***)

Clé étrangère Attributs de l'association

Exemple

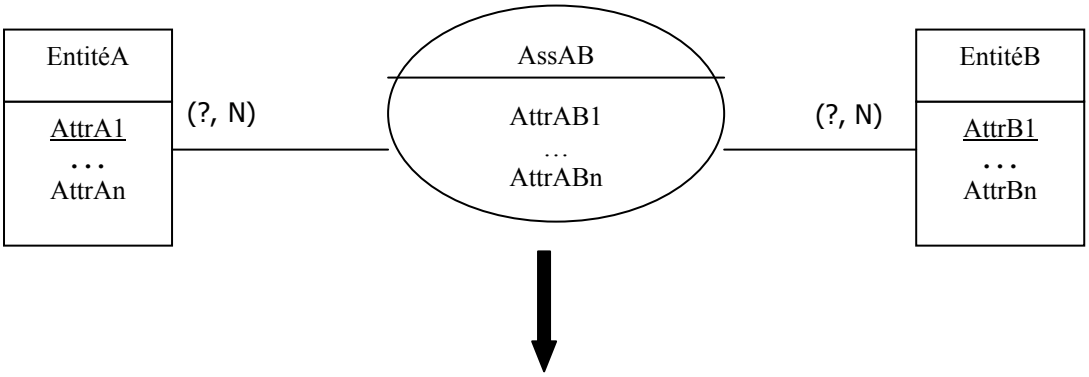


Client (NCl, NomCl, AdrCl)
Commande (NCmd, ***#NCl***, **DateCmd**)

Cas 3 : Association Plusieurs-à-Plusieurs:
Cardinalité entité A 0, N ou 1, N et Cardinalité entité B 0, N ou 1, N

Les règles de traduction de ce type d'association sont les suivantes :

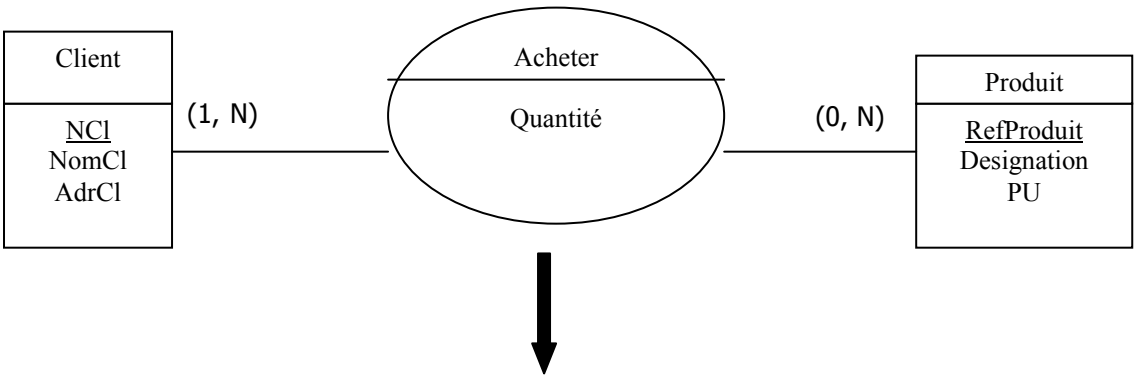
- Chaque entité (Entité A et Entité B) devient une relation.
- L'association sera transformée aussi en une relation ayant comme clé la concaténation des deux clés issues des entités A et B. Les attributs éventuels de l'association seront stockés dans cette relation en tant qu'attributs.



EntitéA(AttrA1,…,AtrrAn)
EntitéB(AtrrB1,…,AtrrBn
AssAB(***#AttrA1***, ***#AttrB1***, ***AttrAB1,…,AttrABn***)

Clé primaire et clés étrangères Attributs de l'association

Exemple :

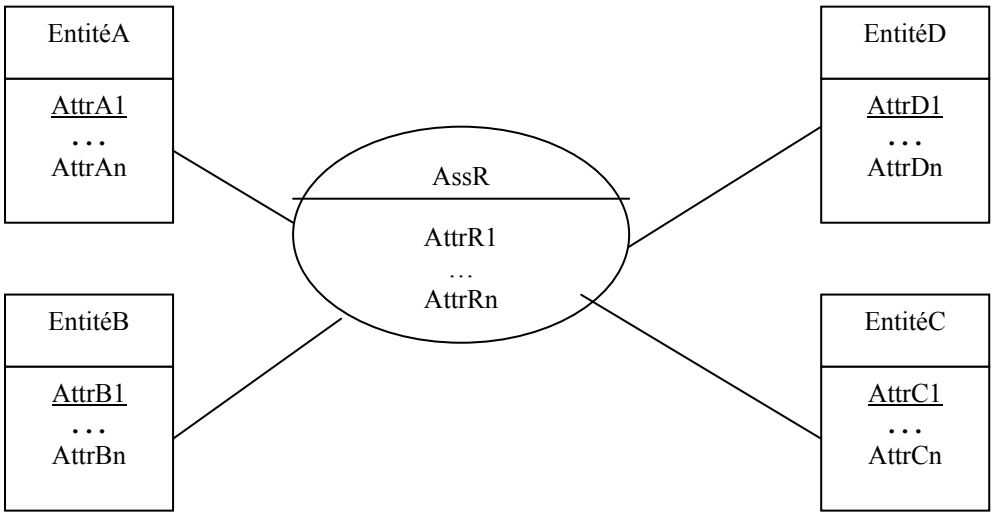


Le modèle relationnel correspondant est le suivant :

Client (NCl, NomCl, AdrCl)
Produit (RefProduit, Designation, PU)
Acheter (#NCl, #RefProduit, Quantite)

b. Traduction des associations n-aires

Ce type d'association sera transformé en une relation ayant comme liste d'attributs la liste des clés des relations correspondantes aux entités qui participent à cette association en plus de ses attributs éventuels. Une clé minimale sera choisie parmi la liste des attributs ainsi constituée.



EntitéA(AttrA1,…,AttrAn)
EntitéB(AtrrB1,…,AttrBn)
EntitéC(AttrC1,…,AttrCn)
EntitéD(AtrrD1,…,AttrDn)
AssR(***#AttrA1***, ***#AttrB1***, ***#AttrC1***, ***#AttrD1***, ***AttrR1,…,AttrRn***)

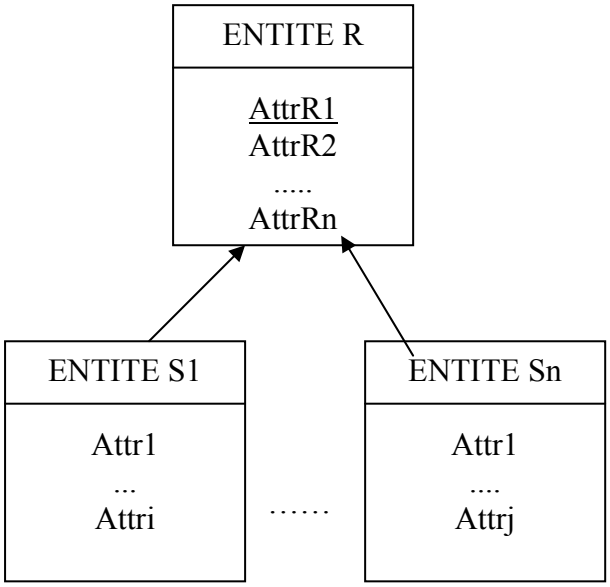
Clé primaire et clés étrangères — Attributs de l'association

II.3 Traduction du lien is-a

La traduction du lien is-a peut se faire selon plusieurs règles. Dans ce qui suit, nous considérerons une entité mère R avec n entités filles S1, S2, ….Sn.
La traduction d'un lien is-a se fait selon l'une des trois règles suivantes :

R1 : Représentation de l'entité mère et de ses entités filles

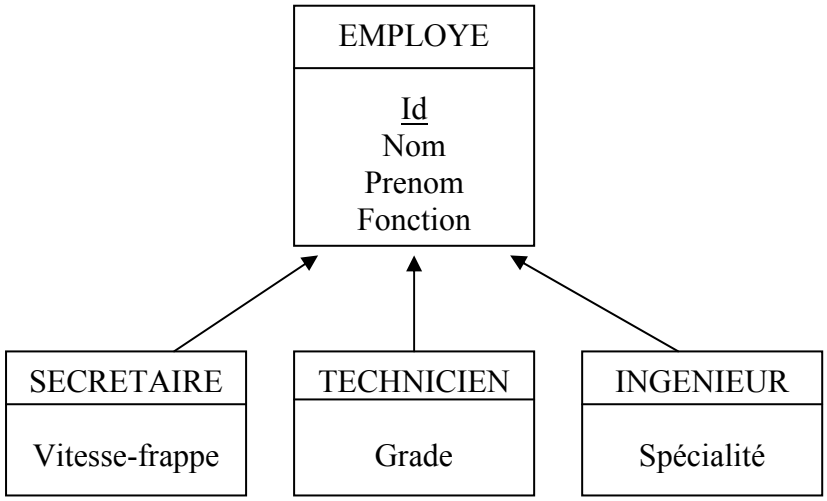
- L'entité mère sera transformée en une nouvelle relation avec ses attributs.
- Chaque entité fille Si sera transformée en une relation comportant comme clé primaire l'identifiant de l'entité mère et comme attributs les attributs de Si.



EntitéR(AttrR1,…,AtrrRn)
EntitéS1(#AtrrR1,Attr1,…,Attri)
…
EntitéSn(#AtrrR1,Attr1,…,Attrj)

Cette règle est adaptée pour tout type de spécialisation ce qui permettra de représenter l'entité mère et les entités filles explicitement.

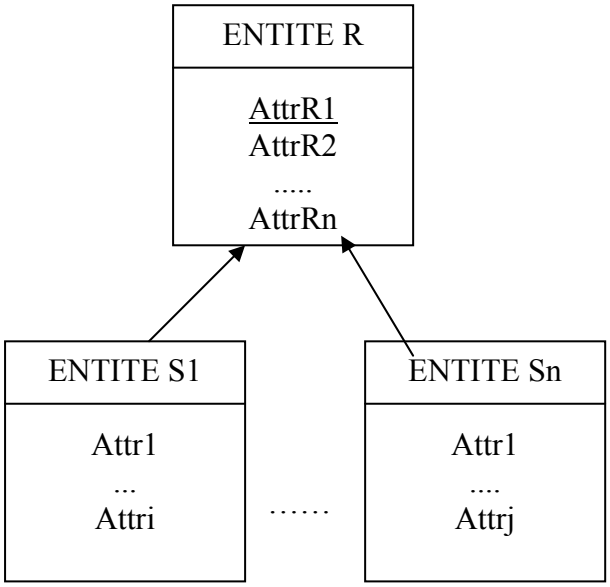
Exemple



EMPLOYE(Id, Nom, Prénom, Fonction)
SECRETAIRE(#Id, Vitesse-frappe)
TECHNICIEN(#Id, Grade)
INGENIEUR(#Id, Spécialité)

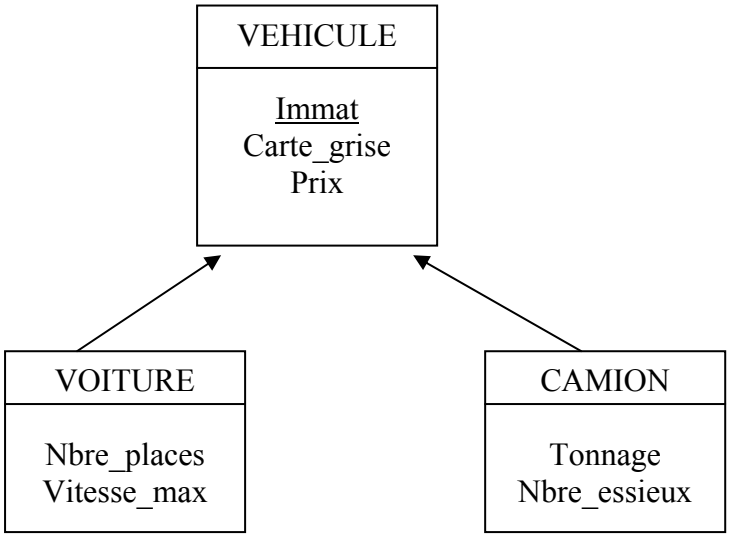
R2 : Pas de représentation de l'entité mère

Chaque entité fille Si sera transformée en une relation comportant comme clé primaire l'identifiant de l'entité mère et comme attributs les attributs de Si en plus des attributs de l'entité mère.



EntitéS1(AtrrR1,Attr1,…,Attri, AttrR2,….,AttRn)
…
EntitéSn(AtrrR1,Attr1,…,Attrj, AttrR2,….,AttRn)

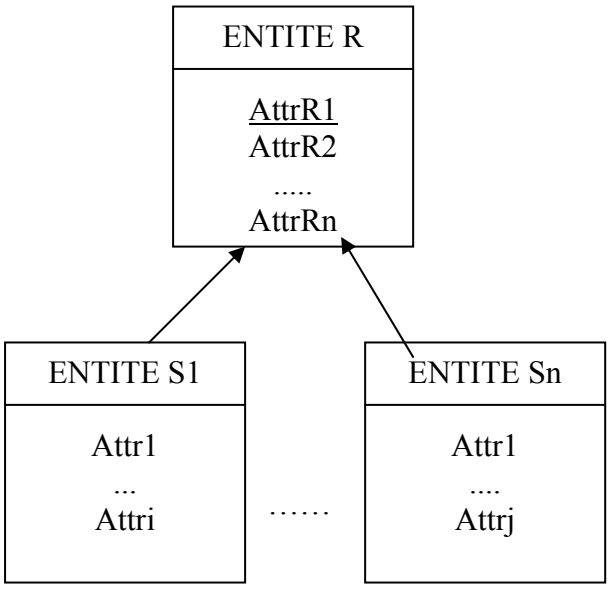
Exemple



VOITURE(Immat, Carte_grise, Prix, Nbre_place, Vitesse_max)
CAMION(Immat, Carte_grise, Prix, Tonnage, Nbre_essieux)

Cette règle pose un problème lorsque les sous-entités ne sont pas disjointes. Dans ce cas, il peut y avoir duplication de certaines données. Certains problèmes d'incohérence peuvent alors avoir lieu.
Cette règle est applicable donc, dans le cas de sous-entités sont totalement disjointes, tels que Homme, Femme ➔ Personne
ou aussi, Alimentaire, Habillement, Electroménager ➔ Article.
Pour le cas, Etudiant, Employé ➔ Personne cette règle conduirait à dupliquer les données héritées pour des employés étudiants.

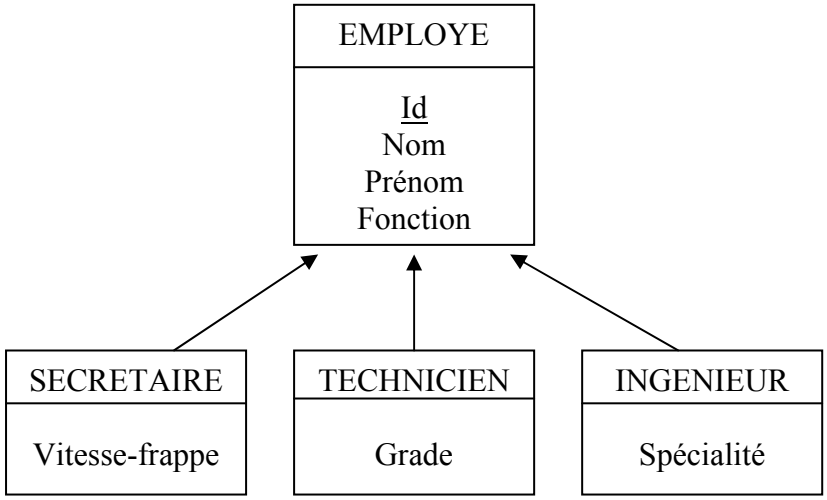
R3 : Fusion des entités filles et de l'entité mère
L'entité mère et ses entités filles seront transformées toutes en une seule relation ayant comme clé primaire l'identifiant de l'entité mère et comme attributs les attributs de toutes les entités (mère et filles).



R (AtrrR1, AttrR2,….,AttRn, Attr1,…,Attri, Attr1,…,Attrj)

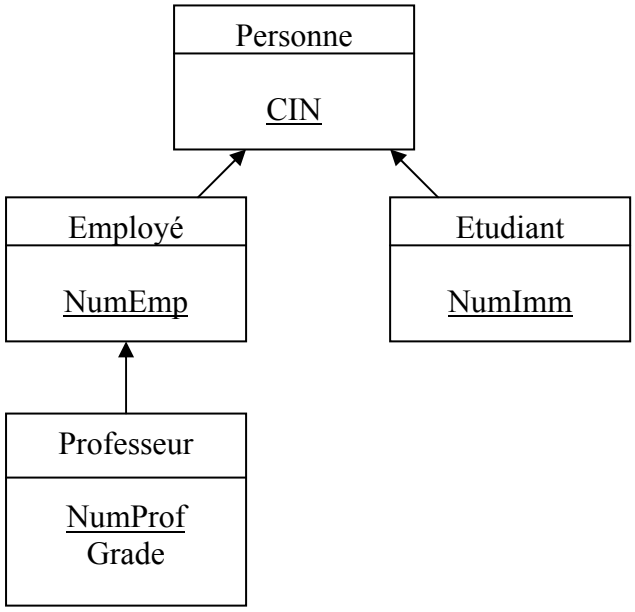
Le problème posé par cette règle est que certains attributs risquent d'avoir une valeur nulle.
Par exemple, pour la hiérarchie Homme, Femme ➔ Personne, suite à l'utilisation de cette règle les attributs spécifiques aux hommes seront nuls pour les femmes et vice versa.
En utilisant cette règle par exemple pour la hiérarchie Etudiant, Employé ➔ Personne, tout étudiant non employé aura les attributs spécifiques aux étudiants nuls, et tout employé non étudiant aura les attributs d'étudiants nuls.

Exemple 1



EMPLOYE(Id, Nom, Prénom, Fonction, Vitesse-frappe, Grade, Spécialité)

Exemple 2
Soit l'exemple suivant :



Pour traduire cette hiérarchie nous utilisons deux règles :
Pour le deuxième niveau de la hiérarchie Professeur Employé nous pouvons utiliser la troisième règle et nous obtiendrons la relation suivante :
Employé (NumEmp, NumProf,Grade)
Pour le premier niveau de la hiérarchie nous utilisons la première règle, nous obtiendrons alors comme modèle relationnel final :
Personne (CIN)
Employé (#CIN, NumEmp, NumProf, Grade)
Etudiant (#CIN, NumImm)

III. Conclusion

Ce chapitre a été consacré à la présentation des principaux concepts du modèle relationnel et aux règles de passage du modèle entité association au modèle relationnel. Le chapitre suivant sera dédié à l'algèbre relationnelle et ses langages prédicatifs.

Exercices

Exercice 1 :

Au niveau d'un faculté, on dispose d'un réseau Intranet comportant un ensemble d'informations utiles aux différents utilisateurs. Afin d'assister les étudiants et les aider à comprendre leurs cours, chaque enseignant met à leur disposition les cours qu'il enseigne ainsi qu'un ensemble d'exercices qui ont été proposés dans des examens antérieurs (précédents) relativement à ces cours.
Un cours est relatif à une matière identifiée par un code et décrite par une désignation. Une matière peut avoir plusieurs cours. Ce dernier est identifié par un code et décrit par un titre et une adresse sur Intranet.
Un cours peut être élaboré par plusieurs enseignants. Un enseignant est identifié par son numéro de carte d'identité et il est décrit par son nom, son grade et sa spécialité.
A un cours, sont associé s plusieurs examens dont chacun est identifié par un numéro, un type et une date de déroulement. Chaque examen comprend plusieurs exercices. Chacun est identifié par un numéro et possède un barème relativement à un examen. Il est à noter qu'un exercice peut être repris dans plusieurs examens.

❶ Construire le modèle E/A relatif à cette base en donnant les différentes entités, associations et attributs.

❷ Traduire ce modèle selon les règles du modèle relationnel.

Exercice 2

Soit la base de données suivante :
Bus (Codb, Marque, Modèle, Nmat, Nbpass, Nbpdeb)
Service (Cods, Desgs)
Employé (Codemp, Nom, Adr, Fct, Datrec)
Affectation_bus (#Codb, #Cods, DataffB)
Affectation_employé (#Codemp, #Cods, DataffE)
Ligne (Codl, Desgl, #Cods, Nbkm)

Description de la base de données

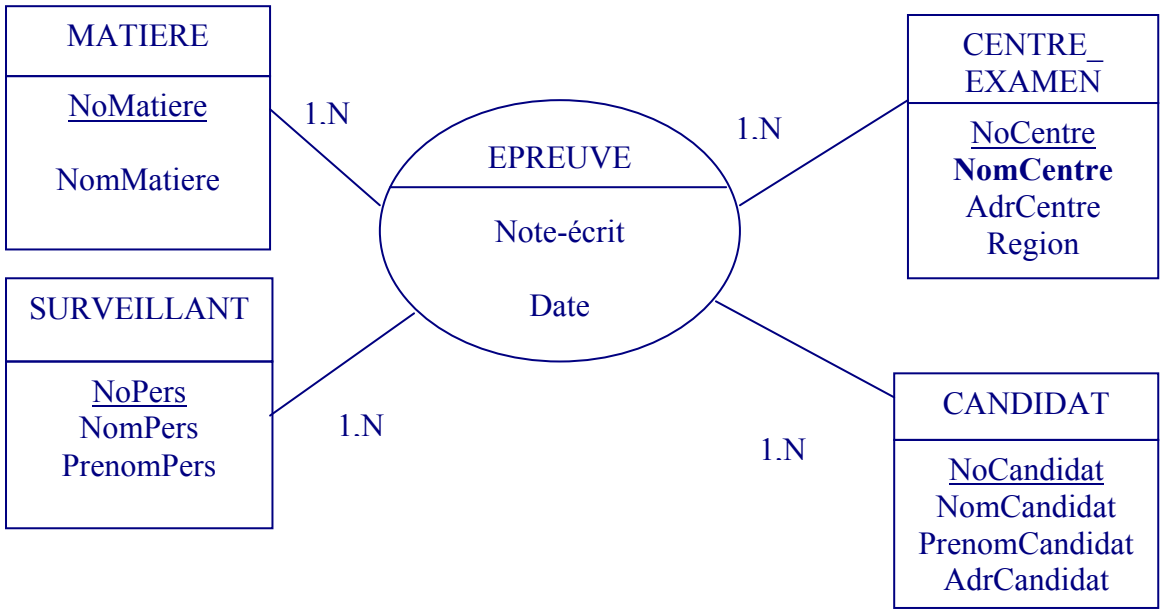
Code	Description
CODB	Code du bus
Marque	Marque du bus
Modèle	Modèle du bus
Nmat	Numéro d'immatriculation du bus
Nbpass	Nombre de places assises d'un bus
Nbpdeb	Nombre de places debout d'un bus
Cods	Code du service
Desgs	Désignation du service

Codemp	Code de l'employé
Nom	Nom de l'employé
Adr	Adresse de l'employé
Fct	Fonction de l'employé (Chauffeur ou contrôleur)
Datrec	Date de recrutement de l'employé
DataffB	Date d'affectation du bus à un service
DataffE	Date d'affectation de l'employé à un service
Codl	Code de la ligne
Desgl	Désignation de la ligne
Nbkm	Nombre de kilomètres de parcours d'une ligne

Déduire à partir du modèle relationnel ci-dessus, le modèle E/R correspondant.

Exercice 3

Transformer ce modèle entité/Association en un modèle relationnel.



Exercice 4

On souhaite informatiser une partie de la gestion d'une école.
Le personnel de cette école est composé de professeurs et de secrétaires. Chaque membre du personnel est identifié par un numéro matricule, par son nom , son prénom et son adresse.
L'école est composée de locaux (identifiés par un numéro) qui sont soit des bureaux (dans ce cas ils sont pourvus d'un unique téléphone), soit des salles de cours (qui comprennent un certain nombre de places). Un bureau est occupé par un professeur et/ou plusieurs secrétaires.
Un étudiant est doté d'un numéro matricule étudiant. On souhaite également disposer dans la base de données, des nom, prénom et adresse des étudiants inscrits dans l'école.
Un étudiant s'inscrit dans une unique année d'étude, identifiée par un code et un nom.

Le programme d'une année d'étude consiste en un ensemble de cours (décrits par un code , un intitulé, et un nombre d'heures). Un cours peut regrouper plusieurs années d'étude.
On souhaite voir figurer dans la base de données le titulaire du cours.

❶ Construire le modèle E/A relatif à cette base en donnant les différentes entités, associations et attributs.

❷ Traduire ce modèle selon les règles du modèle relationnel.

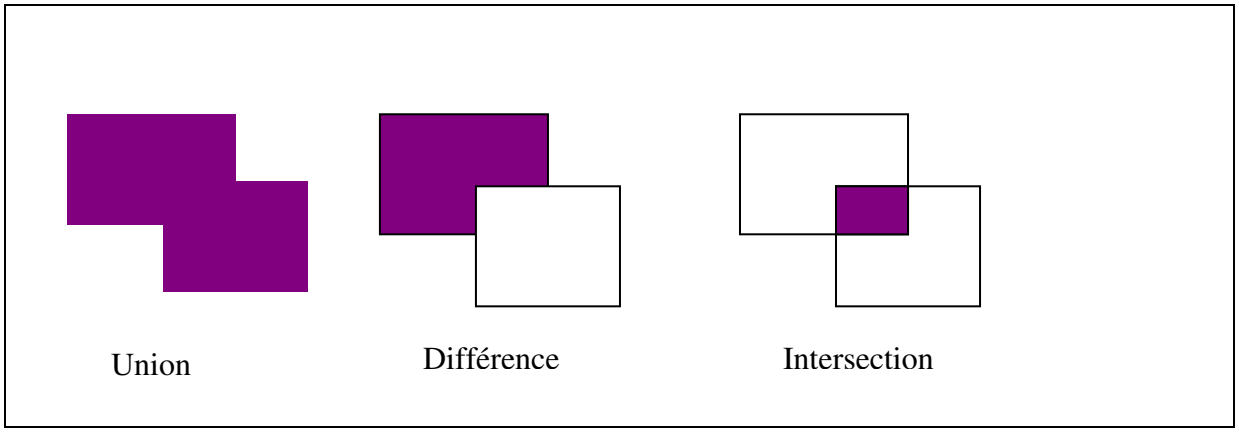
Chapitre 5
Algèbre relationnelle

I- Introduction à l'Algèbre relationnelle

L'algèbre relationnelle est une collection d'opérateurs permettant de réaliser des opérations sur des relations. Elle permet par exemple de sélectionner certains enregistrements d'une relation satisfaisant une condition ou encore de regrouper des enregistrements de relations différentes.
Le résultat de l'application d'un opérateur sur une ou deux relations est une nouvelle relation. Cette propriété est appelée **fermeture**. Elle implique notamment qu'il n'y a pas de doublons dans le résultat et permet l'écriture d'expressions de calcul.
Toute manipulation pouvant être souhaitée par les utilisateurs devrait pouvoir être exprimée par une expression algébrique. Cette propriété est appelée **complétude.**
Etant donnée, que le modèle relationnel est basé sur la théorie des ensembles, l'algèbre relationnelle utilise les opérateurs classiques de manipulation des ensembles (union, intersection, différence et produit cartésien) et introduit des opérateurs propres aux bases de données (renommage, sélection, projection, jointure et division).
Ces opérateurs sont soit unaires soit binaires. Les opérateurs unaires impliquent une seule opérande: sélection (noté σ), projection (π), renommage (α)
Les opérateurs binaires impliquent deux opérandes: produit cartésien ($\times$), jointures ($\bowtie$), union ($\cup$), intersection ($\cap$), différence ($-$), division ($/$).

II- Opérateurs ensemblistes

Les opérateurs ensemblistes correspondent aux opérateurs habituels de la théorie des ensembles, définis sur des tables de même schéma (union-compatibles).
Ces opérateurs sont l'union, l'intersection et la différence. Le produit cartésien porte sur des relations qui ne sont pas de même schémas. Le schéma suivant illustre l'effet des trois premiers opérateurs sur des tables de mêmes schémas. Le résultat étant la partie colorée.



II-1 Union-compatibilité

Deux relations sont dites union-compatibles si elles ont le même schéma de relation, c'est à dire qu'elles ont le même nombre d'attributs et que ceux-ci ont le même domaine.

II-2 Opérateur union (∪)

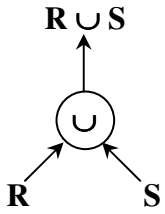
Définition :

Soient r et s, deux relations de schémas respectifs R et S. Les schémas R et S doivent être union-compatibles. L'union des deux relations R ∪ S produit une nouvelle relation de schéma identique à R et à S possédant les enregistrements appartenant à R ou à S ou aux deux relations.

Description :

- Type opération: binaire
- Syntaxe : R ∪ S
- Notation fonctionnelle : Union (R,S)
- Sémantique : réunit dans une même relation les tuples de R et ceux de S (sans doublons)
- Schéma : schéma(R ∪ S) = schéma(R) = schéma(S)
- Pré-condition : schéma(R) = schéma(S)

Représentation graphique



Exemple : Soit la base de données relationnelle (BD commerciale) suivante :

Produit (NP, LibP, Coul, Poids, PU, Qtes)	*Désigne l'ensemble des produits.*
Client (NCl, NomCl, AdrCl)	*Désigne l'ensemble des clients.*
Commande (NCmd, DateCmd, #NCl)	*Désigne l'ensemble des commandes.*
Ligne_Cmd (#NCmd, #NP, Qte)	*Désigne l'ensemble des lignes commandes.*

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

Exemple : Soit la BD commerciale.

Supposons maintenant que nous disposons de 2 tables produit produit1 et produit2 exprimant le fait que les produits sont stockés dans deux dépôts différents.
Question : Lister tous les produits.
Réponse : Réaliser l'union des deux tables de produit.

Produit1

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900

Produit2

NP	LibP	Coul	Poids	PU	Qtes
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

Produit1 ∪ Produit2

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

II-3 Opérateur intersection (∩)

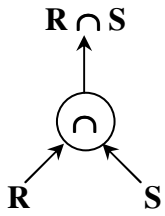
Définition :

Soient r et s, deux relations de schémas respectifs R et S. Les schémas R et S doivent être union-compatibles. L'intersection des deux relations R ∩ S produit une nouvelle relation de schéma identique à R et à S possédant les enregistrements appartenant conjointement à R et à S.

Description :

- Type opération: binaire
- Syntaxe : R ∩ S
- Notation fonctionnelle : Inter (R,S)
- Sémantique : sélectionne les tuples qui sont à la fois dans R et S
- Schéma : schéma (R ∩ S) = schéma (R) = schéma (S)
- Pré-condition : schéma(R) = schéma(S)

Représentation graphique



Exemple : Soit la BD commerciale.

Supposons maintenant que nous disposons de 2 tables produit produit1 et produit2 donnant respectivement les produits achetés par le client1 et le client2
Question : Lister tous les produits identiques achetés par les 2 clients.
Réponse : Réaliser l'intersection des deux tables produit1 et produit2.

Produit1

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900

Produit2

NP	LibP	Coul	Poids	PU	Qtes
P002	Prise	Blanc	1.2	1.500	1000
P004	Peinture	Blanc	25	33.000	900
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

Produit1 ∩ Produit2

NP	LibP	Coul	Poids	PU	Qtes
P002	Prise	Blanc	1.2	1.500	1000
P004	Peinture	Blanc	25	33.000	900

II-4 Opérateur différence (-)

Définition :

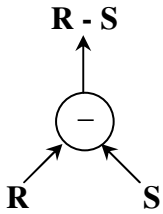
Soient r et s, deux relations de schémas respectifs R et S. Les schémas R et S doivent être union-compatibles. La différence des deux relations R - S produit une nouvelle relation de schéma identique à R ou à S possédant les enregistrements présents dans R mais pas dans S.

Description :

- Type opération: binaire
- Syntaxe : R – S
- Notation fonctionnelle : Diff (R,S)

- Sémantique : sélectionne les tuples de R qui ne sont pas dans S
- Schéma : schéma (R – S) = schéma (R) = schéma (S)
- Pré-condition : schéma (R) = schéma (S)

Représentation graphique



Exemple : Soit la BD commerciale.

Supposons maintenant que nous disposons de 2 tables produit produit1 et produit2 donnant respectivement les produits achetés par le client1 et le client2
Question : Lister tous les produits achetés par le client1 et que le client2 n'a pas acheté.
Réponse : Réaliser la différence entre les deux tables de produit.

Produit1

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250

Produit2

NP	LibP	Coul	Poids	PU	Qtes
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

Produit1 - Produit2

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900

II-5 Opérateur produit cartésien (x)

Le produit cartésien est un opérateur issu de la théorie des ensembles défini comme suit : si A et B sont deux ensembles, leur produit cartésien A x B contient toutes les paires (a, b) avec a ∈ A et b ∈ B. Ceci signifie que le produit cartésien permet d'obtenir toutes les combinaisons possibles entre les éléments de deux ensembles. Dans le cadre de l'algèbre relationnelle nous définirons donc le produit cartésien comme suit :

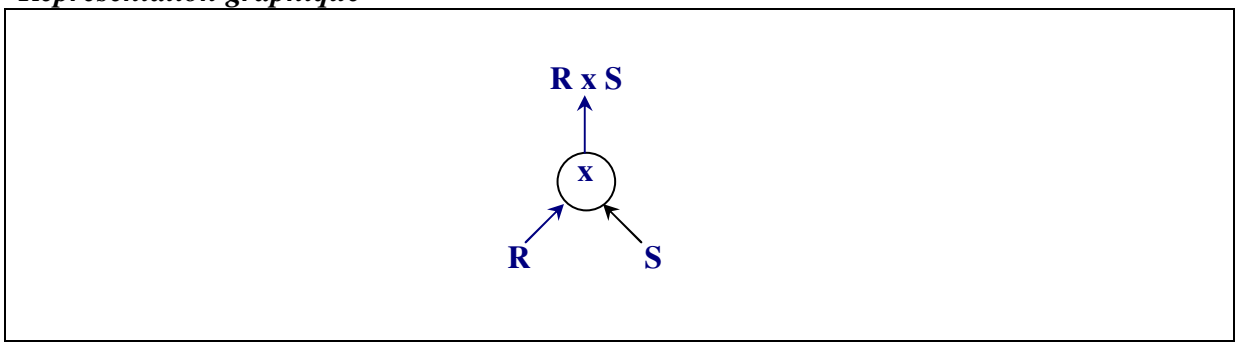
Définition :

Soient r et s, deux relations de schémas respectifs R et S. Les schémas R et S doivent être disjoints c'est à dire ne pas avoir d'attributs communs. Le produit cartésien des deux relations R x S produit une nouvelle relation de schéma Z égal à l'union des schémas R et S et possédant comme enregistrements, la concaténation des enregistrements de R avec ceux de S.

Description :

- Type opération: binaire
- Syntaxe : R x S
- Sémantique : chaque tuple de R est combiné avec chaque tuple de S
- Schéma : schéma (R × S) = schéma(R) ∪ schéma(S)
- Pré-condition: R et S n'ont pas d'attributs de même nom
  (sinon, renommage des attributs avant de faire le produit).

Représentation graphique



Exemple : Soit la BD commerciale.

Supposons maintenant que nous disposons de 2 tables produit et client.
Question : Lister tous les achats possibles des clients (produits pouvant être achetés par tous les clients).
Réponse : Réaliser le produit cartésien entre les deux tables produit et client.
Pour simplifier, nous avons réduit le nombre de tuples.

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000

Client X Produit

NCl	NomCl	AdrCl	NP	LibP	Coul	Poids	PU	Qtes
CL01	BATAM	Tunis	P001	Robinet	Gris	5	18.000	1200
CL01	BATAM	Tunis	P002	Prise	Blanc	1.2	1.500	1000
CL02	BATIMENT	Tunis	P001	Robinet	Gris	5	18.000	1200
CL02	BATIMENT	Tunis	P002	Prise	Blanc	1.2	1.500	1000
CL03	AMS	Sousse	P001	Robinet	Gris	5	18.000	1200
CL03	AMS	Sousse	P002	Prise	Blanc	1.2	1.500	1000

III Opérateurs propres aux Bases de Données

III-1 Renommage (α)

Définition

Le renommage ou l'affectation permet de renommer les attributs d'une relation pour résoudre des problèmes de compatibilité entre noms d'attributs de deux relations opérandes d'une opération binaire.

Description :

- Type opération: unaire
- Syntaxe : α [ancien_nom : nouveau_nom] R
- Sémantique : les tuples de R avec un nouveau nom de l'attribut
- Schéma : schéma (α [n, m] R) le même schéma que R avec n renommé en m
- Pré-condition : le nouveau nom n'existe pas déjà dans R

Exemple : Soit la BD commerciale.

Supposons maintenant que nous disposons de la table produit.
Question : Renommer l'attribut LibP par l'attribut DésigP.
Réponse : Réaliser le renommage de l'attribut LibP par l'attribut DésigP.

Produit2 = α [LibP: DésigP] produit1

Produit1

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250

Produit2

NP	DésigP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250

III-2 Sélection (σ)

Définition :

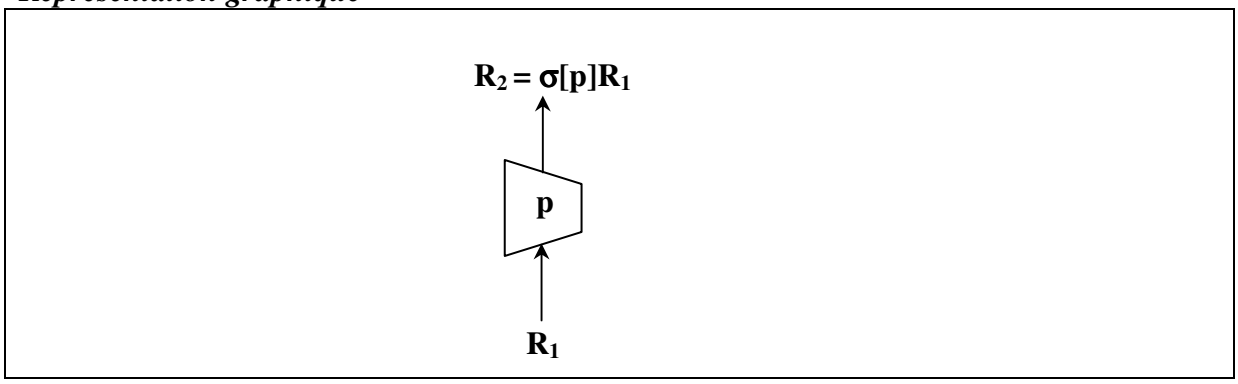
La sélection appelée encore restriction est un opérateur unaire qui prend en entrée une relation r de schéma R et produit en sortie une nouvelle relation de même schéma R ayant comme enregistrements ceux de r satisfaisant la condition de sélection. Le but étant de sélectionner un ensemble de tuples d'une relation, en fonction d'un critère de sélection (prédicat ou expression logique de prédicats).

La condition de sélection utilise les opérateurs de comparaison (=, <, <=, >, >=, !=), les connecteurs logiques (et, ou, non) et les parenthèses.

Description :

- Type opération: unaire
- Syntaxe: **σ [p] R**
  p: prédicat de sélection (condition de sélection)
- Notation fonctionnelle : R[Prédicat]
- Sémantique : crée une nouvelle relation de population l'ensembles des tuples de R qui satisfont le prédicat p
- Schéma: Schéma (résultat) = Schéma (opérande)
- Population: population (résultat) ⊆ population (opérande)

Représentation graphique



Exemple 1: Lister tous les ouvrages dont le genre est BD et dont l'éditeur est Eyrolles

LIVRE

CodeOuv	Titre	Genre	Editeur	Collection
255	Bases de données relationnelles	BD	Eyrolles	Info
786	Réseaux Téléinformatiques	Réseaux	Hermes	Telec
355	Bases de données Orientés Objet	BD	Dunod	Info
800	Bases de données réparties	BD	Eyrolles	Info
241	Programmation C++	Programmation	Dunod	Info

Livre_BD = σ [Genre = 'BD' et Editeur='Eyrolles'] (LIVRE)

CodeOuv	Titre	Genre	Editeur	Collection
255	Bases de données relationnelles	BD	Eyrolles	Info
355	Bases de données réparties	BD	Eyrolles	Info

Exemple 2 : Soit la BD commerciale.

Supposons maintenant que nous disposons de la table produit.
Question : Lister tous les produits dont le prix unitaire est < 33.000.
Réponse : Il faut réaliser une sélection sur les tuples dont le prix unitaire est < 33.000 :
σ [p] Produit avec p=PU < ***33.000.***

Produit2 = σ [p] Produit avec p=PU < ***33.000***

Produit1

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250

Produit2

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P005	Poignée	Gris	3	12.000	1300

✓ Donner les clients de la ville de Sousse.

σ [Ville = 'Sousse'] Client ➡

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

➡

NCl	NomCl	AdrCl
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse

✓ Donner la liste des commandes dont la date est supérieure à '01/01/2004'.

σ[DateCmd>'01/01/2004'] Commande

Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

NCmd	DateCmd	NCl
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C005	11/03/2004	CL03

✓ Donner la liste des produits dont le prix est compris entre 20 et 50.

σ[(PU >= 20) **AND** (PU <= 50)] Produit;

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

NP	LibP	Coul	Poids	PU	Qtes
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P006	Serrure	Jaune	2	47.000	1250
P008	Fer	Noir	50	90.000	800

III-3 Projection (π)

Définition

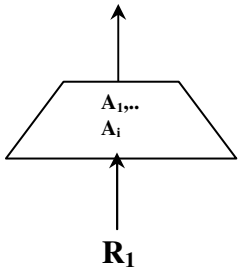
La projection est un opérateur unaire qui prend en entrée une relation r de schéma R (A_1; A_2; ...; A_n) et produit en sortie une nouvelle relation de schéma (A_1; A_2; .. ;A_i; A_j) inclus dans R ayant comme enregistrements ceux de r restreints à ce sous-schéma (A_1; A_2; .. ;A_i; A_j). Le but de la projection étant de ne retenir que certains attributs dans une relation (c'est-à-dire un ensemble de colonnes). A l'issue d'une projection, la relation résultante peut contenir des doublons.

Description :

- Type opération: unaire
- Syntaxe: **π [attributs] R**
  attributs: liste l'ensemble des attributs de R à conserver dans le résultat.
- Notation fonctionnelle : R{liste d'attributs}
- Sémantique : crée une nouvelle relation de population l'ensemble des tuples de R réduits aux seuls attributs de la liste spécifiée
- Schéma: Schéma (résultat) ⊆ schéma (opérande)
- Résultat : nombre tuples (résultat) = nombre tuples (opérande) (en comptant les doublons)

Représentation graphique

$R_2 = \pi [A_1, A_2, \ldots A_i] R_1$



Exemple : Soit la BD commerciale.

Supposons maintenant que nous disposons de la table produit.
Question : Lister toutes les désignations de produit.
Réponse : Il faut réaliser une projection sur la table produit pour ne garder que l'attribut LibP : **π [LibP] (Produit1)**

Produit2 = π [LibP] (Produit1)

Produit1

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250

Produit2

LibP
Robinet
Prise
Câble
Peinture
Poignée
Serrure

✓ Donner la liste des clients.

π [*] Client

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

✓ Donner l'ensemble des numéros des produits qui ont été commandés (NP seulement).

π [NP] Ligne_Cmd

NP
P001
P004
P006
P002
P007
P001
P002
P004
P005
P008
P001
P002

Exemple2 : Lister tous les titres de livres

On ne veut que l'attribut Titre de la table livre

π [Titre] (LIVRE)

Titre
Bases de données relationnelles
Réseaux Téléinformatiques
Bases de données Orientés Objet
Programmation C++

III-4 Division (/)

Définition

Le résultat de la division d'une relation R(X,Y) par une relation S (Y) est une relation Q(X) définie par :

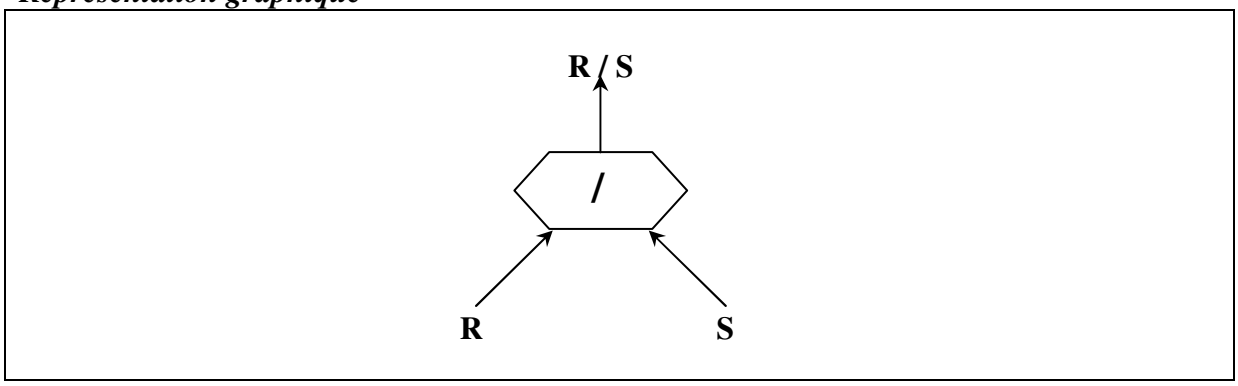
1) le schéma de Q est constitué de tous les attributs de R n'appartenant pas à S.
2) les tuples q_j de Q tels que, quels que soit les tuples s_i de S, le tuple (q_j,s_i) est un tuple de R (c'est-à-dire $QXS \subseteq R$).

La division traite les requêtes de style «les … tels que TOUS les …»

Description :

- Type opération: binaire
- Syntaxe: **R / S**
  soient R(A1, …, An) et S(A1, …, Am) avec n>m et A1, …, Am des attributs de même nom dans R et S
  $R / S = \{ <a_{m+1}, a_{m+2}, \ldots, a_n> / \forall <a1, a2, \ldots, am> \in S, \exists <a1, a2, \ldots, am, am+1, am+2, \ldots, an> \in R\}$
- Notation fonctionnelle : Div(R,S)
- Sémantique : crée une nouvelle relation de population des tuples dont la concaténation avec tous les n-uplets de S appartiennent à R.
- Schéma: schéma (résultat) ⊆ schéma (opérande)
- Résultat : nombre tuples (résultat) **<=** nombre tuples (opérande)

Représentation graphique



Exemple 1 : Soit la BD commerciale
Question : Quels sont les commandes qui portent sur tous les produits
Réponse : Diviser la relation Ligne_Cmd par la relation produit (ne contenant que NP).

Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P002	300
C001	P003	100
C001	P004	200
C001	P005	550
C001	P006	50
C001	P007	100
C001	P008	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

Produit

NP
P001
P002
P003
P004
P005
P006
P007
P008

Ligne_Cmd / Produit

NCmd
C001

Exemple 2 :

Quels sont les étudiants qui ont réussi tous les cours ?

R

Etudiant	Cours	Réussi
François	BDR	Oui
Jacques	BDR	Oui
Pierre	BDR	Non
François	Prog	Oui
Pierre	Prog	Oui
Jacques	Math	Oui
François	Math	Oui

V

Cours	Réussi
BDR	Oui
Prog	Oui
Math	Oui

R / V
François

III-5 Jointure (⋈)

Définition

Soient r et s deux relations de schémas respectifs R et S. La jointure de R et S, selon une condition que doivent vérifier les valeurs des tuples, est l'ensemble des tuples du produit cartésien R x S satisfaisant cette condition. Donc on peut la considérer comme un produit cartésien suivi d'une sélection.

La relation résultant de la jointure possède comme schéma l'union des deux schémas R et S et comme enregistrements la concaténation des enregistrements de R avec ceux de S qui répondent à la condition de sélection.

La condition de sélection utilise les opérateurs de comparaison (=;<; <=; >; >=; !=), les connecteurs logiques (et, ou, non) et les parenthèses.

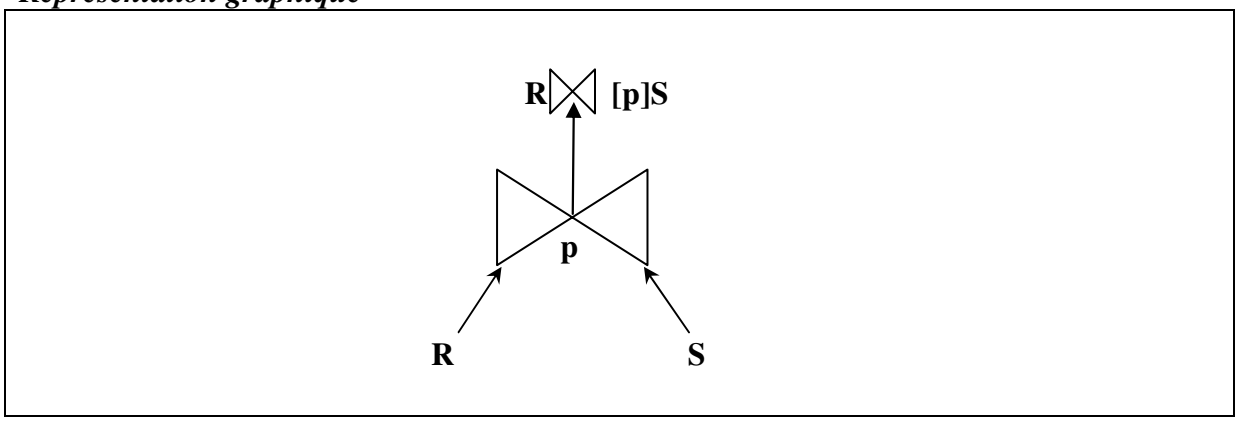
Lorsque le critère de sélection est l'égalité, on parle d'équi-jointure sinon on parle de Théta-jointure.

Description :

- Type opération: binaire
- But: créer toutes les combinaisons significatives entre tuples de deux relations (le critère de combinaison est explicitement défini en paramètre de l'opération)
- Syntaxe : **R ⋈ [p]S**
  p: prédicat de sélection (condition de jointure)

- Notation fonctionnelle : Join(R,S/Prédicat)
- Sémantique : combine certains tuples qui répondent à une condition
- Schéma : schéma (R ⋈ [p] S) = schéma (R) ∪ schéma (S)

Représentation graphique



III-5.1 Equi-Jointure

L'Equi-jointure de r et s de schéma R et S sur les attributs Ai et Bj est une jointure selon la condition valeur Ai = valeur Bj. Les attributs Ai et Bj, appelés colonnes de jointure doivent avoir des domaines compatibles. S'ils ont des noms identiques on ajoute le nom de la relation (Exp: R.Ai, NomTable.attribut)
La relation résultant de l'équi-jointure a comme tuples la concaténation des tuples de R et de S s'ils ont la même valeur pour les attributs communs. Le but de l'Equi-jointure est de créer toutes les combinaisons significatives (portant la même valeur pour les attributs de même nom) entre tuples de deux relations.

Exemple : Soit la BD commerciale.
Nous disposons des deux relations Client et commande.
Question : Lister les clients qui ont passé des commandes.
Réponse : Equi-jointure avec comme critère : Client.NumCli=Commande.NumCli

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

Commande ⋈[Commande.NCl=Client.NCl]Client

NCmd	DateCmd	NCl	NCl	NomCl	AdrCl
C001	10/12/2003	CL02	CL02	BATIMENT	Tunis
C002	13/02/2004	CL05	CL05	PRODELEC	Tunis
C003	15/01/2004	CL03	CL03	AMS	Sousse
C004	03/09/2003	CL10	CL10	MELEC	Monastir
C005	11/03/2004	CL03	CL03	AMS	Sousse

III-5.2 Théta-Jointure

La Théta-jointure de r et s de schéma R et S sur les attributs Ai et Bj est une jointure selon la condition Ai *opérateur* Bj, avec *opérateur* … {<; <=; >; >=; != }
La relation résultant de la Theta-jointure (θ-jointure) a comme tuples la concaténation des tuples de R et de S dont les valeurs vérifient la condition.

Exemple : Soit la BD commerciale.
Nous disposons des deux relations Client et commande.
Question : Quels sont les clients qui n'ont pas passé de commandes le « 10/12/2003 »
Réponse : Théta-jointure avec comme critère : Client.NCl =Commande.NCl et Commande.DateCmd ! = '10/12/2003'.

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse

Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL03
C003	15/01/2004	CL03

Commande ⋈[Commande.NCl =Client.NCl et Commande.DateCmd!='10/12/20003']Client

NCmd	DateCmd	NCl	NCl	NomCl	AdrCl
C002	13/02/2004	CL03	CL03	PRODELEC	Tunis
C003	15/01/2004	CL03	CL03	AMS	Sousse

III-5.2 Jointure naturelle

La jointure naturelle de r et s de schéma R et S sur les attributs Ai et Bj est une Equi-jointure de r et de s sur tous les attributs de même nom dans R et S (Ai = Bj) suivie de la projection qui élimine les doublures des attributs (les attributs de même nom n'apparaissent qu'une seule fois dans la relation résultante).
Le but de la jointure naturelle est de créer toutes les combinaisons entre les tuples de deux relations qui ont au moins un attribut de même nom Si de plus R et S n'ont pas d'autres attributs en commun, on peut omettre le paramètre de la jointure, et écrire simplement
R⋈ S.

Exemple : Soit la BD commerciale.
Nous disposons des deux relations Client et commande.
Question : Quels sont les clients qui ont passé des commandes.
Réponse : Jointure naturelle.

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

Commande ⋈ Client

NCmd	DateCmd	NCl	NCl	NomCl	AdrCl
C001	10/12/2003	CL02	CL02	BATIMENT	Tunis
C002	13/02/2004	CL05	CL05	PRODELEC	Tunis
C003	15/01/2004	CL03	CL03	AMS	Sousse
C004	03/09/2003	CL10	CL10	MELEC	Monastir
C005	11/03/2004	CL03	CL03	AMS	Sousse

IV Combinaisons et équivalences

IV-1 Combinaison des opérateurs

Les opérateurs produisent en résultat des nouvelles relations, il sera alors possible d'impliquer ces relations dérivées dans d'autres opérations pour obtenir de nouveaux résultats.
L'imbrication des opérateurs algébriques permettra de composer la plupart des requêtes qui manipulent les bases de données relationnelles. Ces requêtes forment les langages d'interrogation qui sont non procéduraux, c'est à dire qu'une interrogation se contente de spécifier les données requises en résultat, alors que le SGBD prend le soin de rechercher ces données.

IV-2 Propriétés des opérateurs relationnels

Propriété 1 : Associativité

- $R1 \cup (R2 \cup R3) = (R1 \cup R2) \cup R3$
- $R1 \ |X| \ (R2 \ |X| \ R3) = (R1 \ |X| \ R2) \ |X| \ R3$

Propriété 2 : Commutativité

- $R1 \cup R2 = R2 \cup R1$
- $R1 \ |X| \ R2 = R2 \ |X| \ R1$
- $R1 \cap R2 = R2 \cap R1$
- $R[Ai \ \theta \text{ valeur}] \{Y\} = R \{Y\}[Ai \ \theta \text{ valeur}]$ si $Ai \in Y$
- $R1 [Ai \ \theta \text{ valeur}] \ |X| \ R2 = (R1 \ |X| \ R2) [Ai \ \theta \text{ valeur}]$
- $R1 \{Y\} \ |X| \ R2 = (R1 \ |X| \ R2) \{Y\}$ si attribut de jointure $\in Y$
- $R1 [Ai \ \theta \text{ valeur}] \cup R2 = (R1 \cup R2) [Ai \ \theta \text{ valeur}]$
- $R1 \{Y\} \cup R2 \{Y\} = (R1 \cup R2) \{Y\}$

Propriété 3 : Distributivité

- $(R1 \cup R2) \ |X| \ R3 = (R1 \ |X| \ R3) \cup (R2 \ |X| \ R3)$ si R1, R2, R3 de même schéma

- R1 ∪ (R2) |X| R3)=(R1 ∪ R3) |X| (R2 ∪ R3) si R1, R2, R3 de même schéma

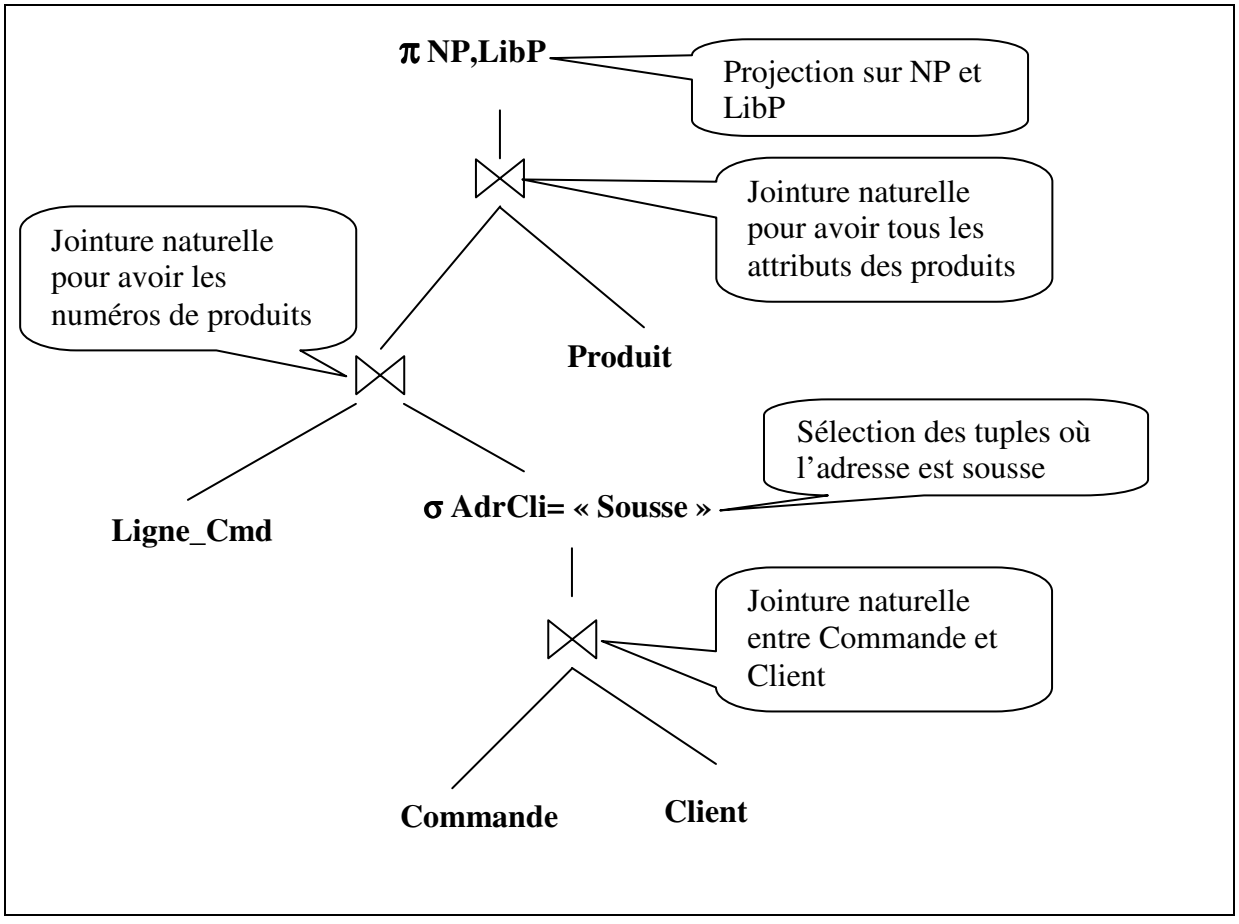
IV-3 Equivalence des opérateurs

- σ [p1] (σ [p2] R) = σ [p2] (σ [p1] R) = σ [p2 et p1] R
- σ [p] (π [a] R) = π [a] (σ [p] R) si attributs(p) ⊆ a

- … etc.

V- Arbres d'expression pour l'algèbre relationnelle

Il est possible de représenter les expressions de l'algèbre relationnelle sous forme d'arbre. Les feuilles sont étiquetées par les opérandes c'est des relations spécifiques ou des variables représentant des relations. Chaque nœud intérieur est étiqueté par un opérateur, accompagné par son paramètre.
Exemple :
Question : Lister le numéro, le libellé des produits commandés par des clients de sousse.
La réponse est indiquée par l'arbre suivant :



Conclusion

Le langage d'interrogation le plus courant, supportant l'algèbre relationnelle est le SQL (Structured Query Langage) qui est devenu une norme de l'ANSI (American National Standards Institute) et qui est adopté par la majorité des SGBD relationnels.

Exercices

Exercice 1
Soit la relation suivante :
PERSONNE

Nom	Age	Ville
Ali	29	Tunis
Amira	32	Sousse
Olfa	54	Tunis
Salah	13	Sfax
Samia	40	Sousse

1) Donner les résultats des requêtes suivantes :

- Requête 1 : s [Age > 30] PERSONNE
- Requête 2 : p [Age] PERSONNE
- Requête 3 : p [Nom] (s [ville = Tunis]PERSONNE)

2) Exprimer les requêtes suivantes en algèbre relationnelle :

- Requête 4 : les personnes (nom, âge, ville) qui habitent Tunis
- Requête 5 : les personnes (nom, âge) qui ont moins de 30 ans
- Requête 6 : les villes dans la relation PERSONNE
- Requête 7 : les noms des personnes habitant Sousse et âgées de plus de 35 ans

Exercice 2
La base de données d'un festival cinématographique est constituée des relations suivantes :
SALLE (Nom_salle, Adresse_salle)
FILM (Titre_film, Réalisateur_film, Producteur_film)
ACTEUR (Nom_Acteur, #Titre_film)
ABONNE (Num_Ab, Nom_Ab, Prenom_Ab, Categorie_Ab)
VU (#Nom_salle, Horaire_projection, #Titre_film,# Num_Ab)
PREFERER (#Num_Ab, #Titre_film)

Exprimer les requêtes suivantes en utilisant les opérateurs de l'algèbre relationnelle
:

- Requête 1 : Dans quelle salle et à quelle heure peut-on voir le film ''Mad city'' ?
- Requête 2 : Quels sont les films réalisés par ''Youssef Chahine ?
- Requête 3 : Quels sont les acteurs du film ''Papillon'' ?
- Requête 4 : Quels sont les films vus par tous les abonnés ?
- Requête 5 : Quels sont les films préférés de la catégorie d'abonnés annuels ?
- Requête 6 : Quels sont les abonnés qui ont vu tous les films?

Exercice 3
Nous considérons toujours la base de données exemple . Répondre aux questions suivantes en utilisant les opérateurs algébriques.

1- Insérer les tuples suivants dans la relation client

NCl	NomCl	AdrCl
CL15	Bonprix	Tunis
CL18	SMT	Tunis
CL19	ATB	Monastir

2- Supprimer les tuples suivants de la relation produit

NCl	NomCl	AdrCl
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis

3- Quels sont les produits qui n'ont jamais été commandés
4- Quels sont les produits qui n'ont jamais été commandés par le client « BATAM »
5- Quels sont les numéros des clients qui ont acheté les mêmes produits que le client « AMS »

Exercice 4
Soit le schéma suivant de base de données :
JOUEUR (Nom , Prénom, Age, Nationalité)
RENCONTRE (#Nomgagnant, #Nomperdant,Tournoi, Date, Score)
GAIN (#NomJoueur, Tournoi, Date, Prime, #NomSponsor)
SPONSOR (Nom, Adresse, ChiffreAffaires)
Questions :
Exprimer sur cette base de données les requêtes suivantes à l'aide de l'algèbre relationnelle. Donner les expressions fonctionnelles correspondantes :

a- Noms et primes des joueurs sponsorisés par Peugeot entre 1985 et 1990.
b- Noms et âges des joueurs ayant participés au tournoi de Roland Garros de 1989.
c- Noms et nationalités des joueurs sponsorisés par Peugeot et ayant gagné à Roland Garros.

Chapitre 6
Normalisation d'une base de données relationnelle

I. Introduction

Un schéma de relation est décrit par la liste de ses attributs et de leurs contraintes d'intégrité éventuelles. La constitution de la liste d'attributs du schéma ne peut pas se faire n'importe comment pour ne pas occasionner de redondance, avec toutes ses implications (perte de place, risques d'incohérence et de perte d'informations). Les formes normales des relations et les mécanismes pour les construire permettent d'obtenir des relations non redondantes. Ces mécanismes sont fondés sur les notions de clés de relations et de dépendances entre données. Ces dernières fournissent les conditions d'application d'un processus dit de **normalisation** qui mène à des formes « correctes » de relations qui sont les formes normales.

Considérons la relation suivante

PRODUIT(Refproduit, LibelleProduit, PU, Quantité, NumService, Adresse, Capacité) qui est visiblement redondante.

RefProduit	LibelleProduit	PU	Quantité	NumService	Adresse	Capacité
P1	CH7	23.510	300	S1	Sousse	9000
P1	CH7	23.510	500	S2	Tunis	6000
P3	VIS12	0.150	900	S4	Sousse	2000

Cette relation présente certaines anomalies :

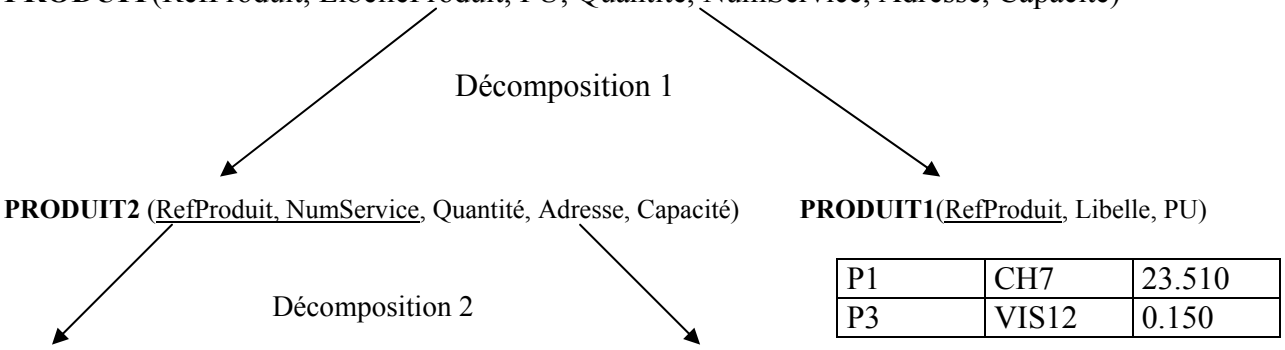
- *Redondance* : un produit apparaît autant de fois qu'il sera livré par un service
- *Mise à jour* : faute de redondance, les mises à jour conduiront à des risques d'incohérence et de non intégrité.
- *Insertion et suppression* : l'insertion la suppression ou le transfert d'attributs pourra faire apparaître des valeurs nulles

On peut dire qu'une Base de Données relationnelle est 'correcte' ou normalisée si :

- chaque relation décrit une information élémentaire avec les seuls attributs qui lui sont directement liés
- il n'y a pas de redondance d'information qui peuvent produire des problèmes de mise à jour

La relation Produit peut être décomposée en trois relations non redondantes :

PRODUIT(RefProduit, LibelleProduit, PU, Quantité, NumService, Adresse, Capacité)



P1	CH7	23.510
P3	VIS12	0.150

P1	S1	300
P1	S2	500
P3	S4	900

S1	Sousse	9000
S2	Tunis	6000
S4	Sousse	2000

Le résultat final de la décomposition est donc les relations suivantes :

PRODUIT1 (RefProduit, LibelleProduit, PU) contient les données relatives aux produits.
PRODUIT21 (RefProduit, NumService, Quantité) contient les données relatives aux produits distribués par des services.
PRODUIT22 (NumService, Adresse, Capacité) contient les données relatives aux services.

II. Dépendance fonctionnelle

II.1 Définition

Un attribut ou une liste d'attributs Y dépend fonctionnellement d'un attribut ou d'une liste d'attributs X dans une relation R, si étant donnée une valeur de X, il ne lui est associé qu'une seule valeur de Y dans tout tuple de R.
On notera une telle dépendance fonctionnelle

X → Y (X détermine Y ou Y dépend fonctionnellement de X).

Exemple
PRODUIT (RefProduit, LibelleProduit, PU, Quantité, NumService, Adresse, Capacité)
Pour cette relation, les dépendances fonctionnelles suivantes sont vérifiées :
RefProduit → LibelleProduit
RefProduit → PU
NumService → Adresse, Capacité
RefProduit, NumService → Quantité

II.2 Propriétés des dépendances fonctionnelles

Des axiomes et des règles d'inférence permettent de découvrir de nouvelles dépendances à partir d'un ensemble initial. Dans ce que suit nous considérons R une relation. Les trois premières propriétés sont connues sous le nom « Axiomes d'Armstrong »

Propriété 1 : Réflexivité
$Y \subseteq X \Rightarrow X \rightarrow Y$
Tout ensemble d'attributs détermine lui-même ou une partie de lui-même.
Propriété 2 : Augmentation
$X \rightarrow Y \Rightarrow X, Z \rightarrow Y, Z$

Si X détermine Y, les deux ensembles d'attributs peuvent être enrichis par un même troisième.

Propriété 3 : Transitivité

$X \rightarrow Y$ et $Y \rightarrow Z \Rightarrow X \rightarrow Z$.

Propriété 4 : Union

$X \rightarrow Y$ et $X \rightarrow Z \Rightarrow X \rightarrow Y, Z$

Propriété 5 : Pseudo-transitivité

$X \rightarrow Y$ et $W,Y \rightarrow Z \Rightarrow W,X \rightarrow Z$

Propriété 6 : Décomposition

$X \rightarrow Y$ et $Z \subseteq Y \Rightarrow X \rightarrow Z$

II.3 Dépendance fonctionnelle élémentaire

Une Dépendance fonctionnelle $X \rightarrow Y$ est élémentaire si pour tout $X' \subset X$ la dépendance fonctionnelle $X' \rightarrow Y$ n'est pas vraie. En d'autres termes, Y ne dépend pas fonctionnellement d'une partie de X (X est la plus petite quantité d'information donnant Y).

Exemple :

RefProduit, LibelleProduit $\rightarrow$ PU n'est pas élémentaire car il suffit d'avoir la référence du produit pour déterminer le prix unitaire.

II.4 Dépendance fonctionnelle canonique

Une Dépendance fonctionnelle $X \rightarrow Y$ est canonique si sa partie droite ne comporte qu'un seul attribut et un ensemble F de dépendances fonctionnelles est canonique si chacune de ses dépendances est canonique.

II.5 Clé d'une relation

Comme défini au chapitre 4, la clé d'une relation est l'ensemble d'attributs dont les valeurs permettent de caractériser les n-uplets de la relation de manière unique.

Formellement :

Un attribut ou une liste d'attributs X est une clé pour la relation R(X,Y,Z) si

- Y et Z dépendent fonctionnellement de X dans R : $X \rightarrow Y, Z$.
- et $X \rightarrow Y, Z$ est élémentaire.

Une relation peut avoir plusieurs clés. Une clé sera choisie et désignée comme **clé primaire**. Les autres seront appelées **clés candidates**.

Un attribut d'une relation R est appelé **attribut clé** s'il appartient au moins à une clé de R.
Un attribut est dit **attribut non clé** s'il n'appartient pas à une clé de R.

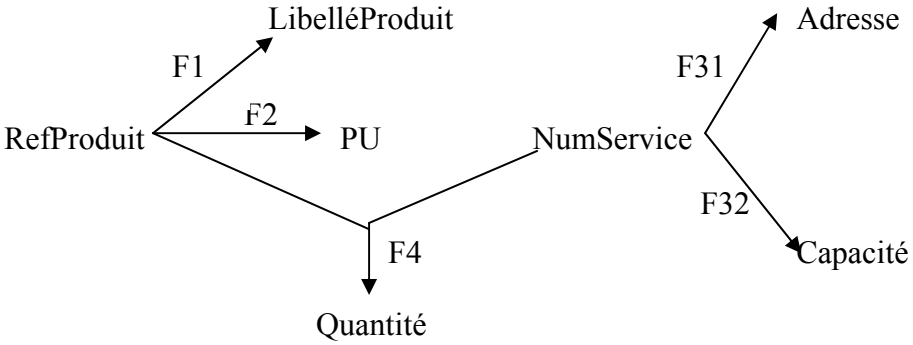
II.6 Graphe des Dépendances Fonctionnelles

Les dépendances fonctionnelles peuvent être représentées à l'aide d'un graphe dont les nœuds sont les attributs impliqués dans les dépendances et les arcs les dépendances elles-mêmes. Les arcs sont orientés de la partie gauche de la dépendance vers sa partie droite.

L'origine d'un arc peut être multiple mais sa cible doit être un nœud unique. De ce fait il est nécessaire d'avoir pour la construction d'un graphe de dépendance fonctionnelle un ensemble canonique de dépendances fonctionnelles.

Exemple :
F1 : RefProduit → LibelleProduit
F2 : RefProduit → PU
F3 : NumService → Adresse, Capacité
F4 : RefProduit, NumService → Quantité
La dépendance fonctionnelle F3 n'est pas canonique, il faut donc la décomposer en deux dépendances fonctionnelles F31 et F32 :
F31 : NumService → Adresse
F32 : NumService → Capacité

Le graphe des dépendances est le suivant :

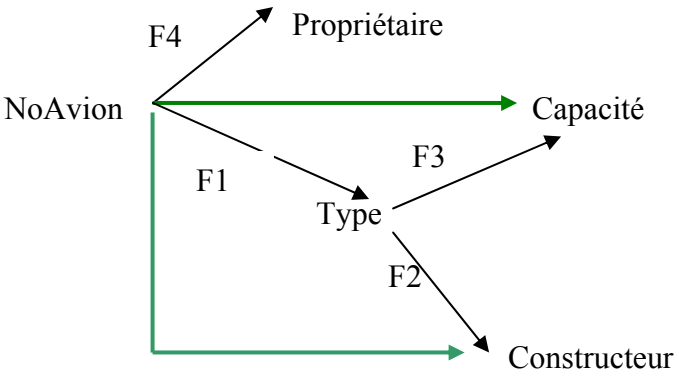


Le graphe des dépendances fonctionnelle d'une relation R permet de trouver les clés de R, qui est l'ensemble (minimal) des nœuds du graphe minimum à partir desquels on peut atteindre tous les autres nœuds (via les dépendances fonctionnelles).

II.7 Fermeture transitive et Couverture Minimale

- **Fermeture Transitive**

La fermeture transitive F+ d'un ensemble F de dépendances fonctionnelles est l'ensemble des dépendances fonctionnelles élémentaires qui peuvent être produites par application des axiomes d'Armstrong sur l'ensemble F.
Exemple :
Avion(NoAvion, Type, Constructeur, Capacité, Propriété)
Avec les dépendances fonctionnelles suivantes :
F1 : NoAvion → Type
F2 : Type→ Constructeur
F3 : Type→ Capacité
F4 : NoAvion → Propriétaire
F= {F1, F2, F3, F4}



Les dépendances fonctionnelles en gras sont déduites par transitivité.
La fermeture transitive de F est
$F^+ = F \cup \{NoAvion \rightarrow Constructeur\} \cup \{NoAvion \rightarrow Capacité\}$

- **Couverture Minimale**

C'est un Ensemble F de dépendances fonctionnelles associé à un ensemble d'attributs vérifiant les propriétés suivantes :

1. Aucune dépendance dans F n'est redondante
2. Toute dépendance fonctionnelle élémentaire des attributs est dans la fermeture transitive

Exemple
La couverture minimale de l'ensemble F de l'exemple précédent est
F^ = {NoAvion → Type ; NoAvion → Propriétaire ;
Type→ Constructeur ; Type→ Capacité}

II-8 Méthode formelle de décomposition

- **Définition**

Une relation R (X,Y ,Z) est décomposable selon (X,Y) et (X,Z) s'il existe deux relations R1 et R2 telle que

- R1 = π [X,Y](R) et R2=π [X,Z](R)
- Et R = R1 ⋈ R2 (R est la jointure naturelle de R1 et R2)

- **Théorème de Heath**

R(X, Y, Z) est décomposable sans perte d'information en R1(X, Y) et R2(X, Z) si X → Y.

Le processus de décomposition est réversible c'est dire il est nécessaire de pouvoir reconstituer la relation initiale par jointure naturelle de celles obtenues par décomposition. Les dépendances fonctionnelles entre les données représentent les contraintes d'intégrité; une décomposition doit le préserver.

Exemple :
Avion(NoAvion, Type, Constructeur, Capacité, Propriétaire)
Avec les dépendances fonctionnelles suivantes :
F1 : NoAvion → Type
F2 : NoAvion → Propriétaire
F3 : Type→ Constructeur
F4 : Type→ Capacité

La décomposition de la relation Avion dans les deux relations suivantes :
Avion (NoAvion, Type, Propriétaire) avec F1 et F2
Modèle (Type, Constructeur, Capacité) avec F3 et F4
préserve les dépendances fonctionnelles et ceci par application du théorème de Heath.

Par contre la décomposition de la relations dans les relations suivantes :
Avion1 (NoAvion, #Type)
Avion2 (#Type, Propriétaire)
Avion3 (Type,Constructeur, Capacité)
Ne préserve pas les dépendances fonctionnelles car F2 sera perdue.

III. Les trois premières Formes Normales et la Forme Normale de Boyce Codd

Les formes normales ont été définies pour permettre la décomposition des relations sans perte d'informations en utilisant la notion de dépendance fonctionnelle. Dans ce cours nous présenterons les trois premières formes normales et celle dite de Boyce Codd.

III-1 Première Forme Normale (1FN)

Une relation R est en première forme normale et notée 1FN si chaque attribut de R a un domaine simple, c'est à dire dont les valeurs sont atomiques et monovaluées.
Cette définition permet d'exclure les relations ayant des attributs dont les valeurs seraient des ensembles ou des listes de valeurs.

Exemple :
LIVRE (No-ISBN, Titre, Auteurs, Editeur)
Cette relation n'est pas en 1FN car l'attribut ''Auteurs'' est multivalué. Un auteur ne peut pas y être traité d'une façon individuelle (exemple: tri des livres par nom d'auteur).
Cette relation peut par exemple être transformée en la nouvelle relation :
LIVRE (No-ISBN, Titre, Auteur1, Auteur2, Auteur 3, Editeur)

ETUDIANT (Nom, Prénom, Adresse(Rue, Ville)) n'est pas en 1FN car l'attribut Adresse n'est pas atomique.
Cette relation peut par exemple être transformée en la nouvelle relation suivante:
ETUDIANT (Nom, Prénom, Rue, Ville)

III-2 Deuxième Forme Normale (2FN)

Une relation R est en deuxième forme normale (2FN) si et seulement si

- elle est en première forme normale,
- et que tout attribut n'appartenant pas à une clé ne dépendra d'aucun sous-ensemble de clé (ne dépend pas d'une partie d'une clé).

Exemple
Soit la relation CLIENT avec ses dépendances fonctionnelles
CLIENT (NomCl, AdrCl, RefProduit, PU)
F1 : NomCl, RefProduit → PU
F2 : NomCl →AdrCl
La clé de la relation est (NomClt, RefProduit)
Suite à F2, une partie de la clé (NomClt) détermine un attribut n'appartenant pas à la clé. Cette relation n'est donc pas en 2FN. Elle pourra être décomposée en :
CLIENT (NomCl, AdrCl)
PRODUIT (#NomCl, RefProduit, PU)

III-3 Troisième Forme Normale (3FN)

Une relation R est en troisième forme normale (3FN) si et seulement si

- elle est en deuxième forme normale,
- et que tout attribut n'appartenant pas à une clé ne dépendra pas d'un attribut non clé.

La troisième forme normale permettra d'éliminer les redondances dues aux dépendances transitives.

La décomposition en 3FN est sans perte d'informations et préserve les DF.

Exemples :
Exemple1 :

CLIENT (NomCl, ChiffreAffaire, Ville, Pays)
Avec les dépendances fonctionnelles suivantes :
F1 : NomCl→ChiffreAffaire
F2 : NomCl→Ville
F3 : Ville→Pays

La relation CLIENT n'est pas en 3FN à cause des dépendances fonctionnelles F2 et F3.
Cette relation doit être décomposée en deux relations :

CLIENT(NomCl, ChiffreAffaire, #Ville)
ADRESSE(Ville, Pays)

Exemple2 :
VOITURE (NumVoiture, Marque, Type, Puissance, Couleur)
F1 : NumVoiture → Marque, Type, Puissance, Couleur
F2 : Type→Marque
n'est pas en 3FN. En effet, l'attribut non clé TYPE détermine MARQUE (F2). Cette relation peut ainsi être décomposée en deux relations :

VOITURE (NumVoiture, #Type, Couleur, Puissance)
MODELE (Type, Marque)

III-4 Forme Normale de Boyce-codd (BCNF)

Une relation R est en BCNF si et seulement si les seules dépendances fonctionnelles élémentaires qu'elle comporte sont celles dans lesquelles une clé détermine un attribut.
En d'autres termes une relation est en BCNF , si elle est en 3FN et qu'aucun attribut membre de la clé ne dépend fonctionnellement d'un attribut non membre de la clé.

Exemple :
ADRESSE (Ville, Rue, CodePostal)
Cette relation présente les DF suivantes :

Ville, Rue → CodePostal
CodePostal → Ville

Elle est en 3FN (car elle est en deuxième forme normale, et tout attribut n'appartenant pas à une clé ne dépendra pas d'un attribut non clé).
Cette relation n'est pas en BCNF car l'attribut ''Ville'' (qui fait partie de la clé) dépend fonctionnellement de CodePostal (qui est un attribut non membre de la clé).

IV. Normalisation par synthèse

IV-1 Introduction

La normalisation est le processus par lequel les données sont organisées dans une base de données. Ceci comprend la création de relations et l'établissement de liens entre ces relations selon certaines règles. Ces règles sont conçues pour protéger les données et rendre l'utilisation de la base de données plus souple en éliminant deux facteurs : la redondance et la dépendance incohérente.

La redondance de données gaspille de l'espace disque et crée des problèmes de maintenance. Si des données existant à plus d'un emplacement doivent être modifiées, il est nécessaire qu'elles le soient de la même manière à chacun de ces emplacements. Il est beaucoup plus facile de prendre en compte la modification d'une adresse client si cette donnée n'est enregistrée que dans la table Clients et à aucun autre emplacement dans la base de données.

Qu'est-ce qu'une " dépendance incohérente" ? Alors qu'il serait naturel pour l'utilisateur de chercher l'adresse d'un client dans la table Clients, il ne penserait pas à y chercher le salaire de l'employé qui rend visite à ce client. Son salaire est lié à (ou dépendant de) l'employé lui-même, et devrait donc être enregistré dans la table Employés. Des dépendances incohérentes peuvent compliquer l'accès aux données ; le chemin de ces données peut être manquant ou interrompu.

Il donc nécessaire de pouvoir d'intégrer dans une base de données uniquement des relation avec le degré de forme normale est le plus élevé possible Il existent deux algorithmes de conception de schémas relationnels en troisième forme normale ou celle de Boyce Codd. Le premier procède par décomposition et le second par synthèse à partir du graphe des dépendances fonctionnelles. Dans le cadre de ce cours nous allons présenter uniquement le second algorithme.

IV-2 Principe

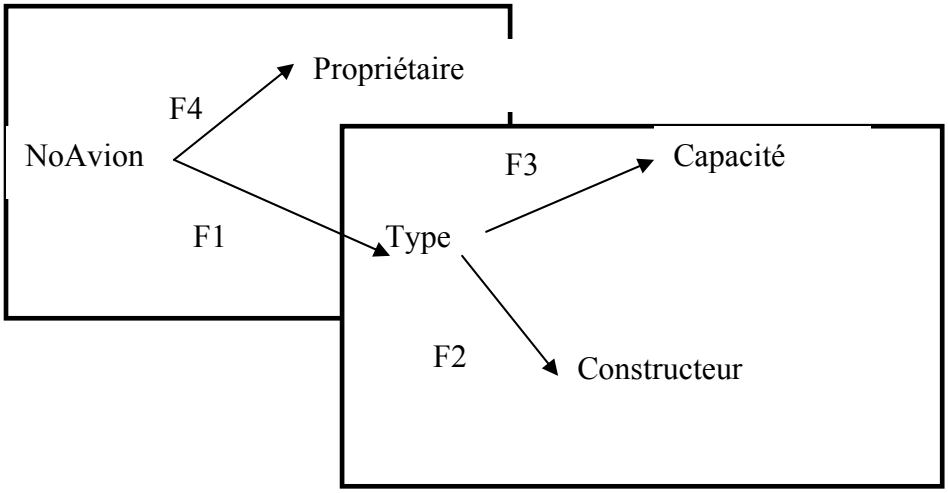
Le processus de la normalisation par la synthèse permet de synthétiser des schémas relationnels à partir des attributs de la relation universelle et de ses dépendances fonctionnelles.
Le principe est le suivant :

1. Trouver une couverture minimale des dépendances fonctionnelles
2. Regrouper les attributs isolés (ceux qui ne sont ni source ni cible d'aucun arc du graphe) au sein d'une relation dont tous les attributs sont clés ;
3. Tant que le graphe n'est pas vide :
   - Rechercher le plus grand ensemble d'attributs X1, …….,Xn dépendant fonctionnellement de X, X étant un attribut ou une liste d'attributs.
   - Constituer une relation d'attributs (X,X1,……,Xn) ; cette relation a X comme clé candidate et elle est forcément en 3FN car X1,……,Xn dépendent fonctionnellement de X et il n'y a pas de dépendances transitives puisque nous sommes partis de la couverture minimale des dépendances.
   - Eliminer les dépendances (X→X1), …,(X→Xn) du graphe
   - Eliminer les attributs isolés dans le graphe
   - Refaire à partir de la troisième étape

IV-3 Exemples

Exemple 1

Soit la relation universelle suivante :
AVION (NoAvion, Propriétaire, Type, Constructeur, Capacité) et sa couverture minimale (voir paragraphe II.4)
Avec les dépendances fonctionnelles suivantes : F= {F1, F2, F3, F4}
F1 : NoAvion → Type F4 : NoAvion → Propriétaire
F2 : Type→ Constructeur F3 : Type→ Capacité



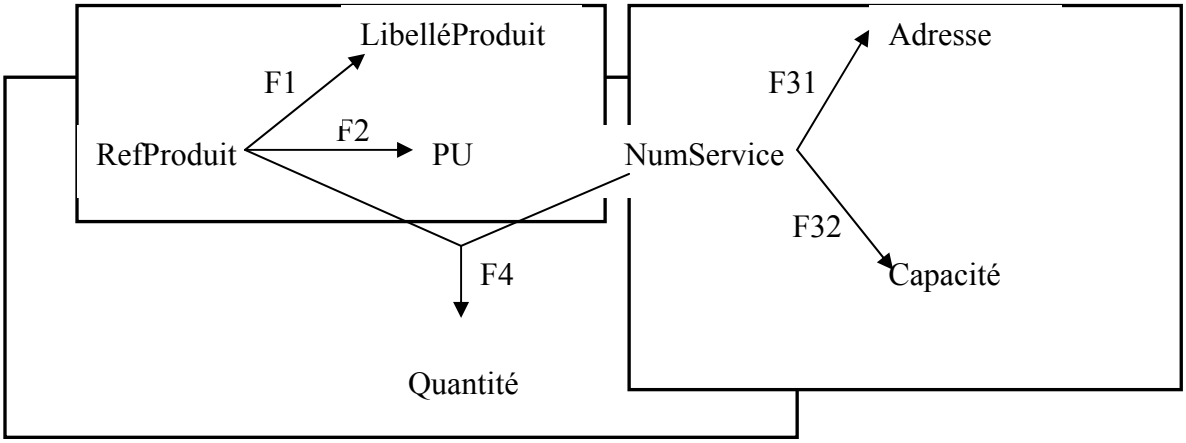
Avion (NoAvion, #Type, Propriétaire) avec les dépendances fonctionnelles F1 et F4
Type-Avion(Type, Capacité, Constructeur) avec les dépendances fonctionnelles F2 et F3

- D'après l'étape 3 nous allons construire d'abord la relation **Avion (NoAvion, type, propriétaire)** et nous éliminons les dépendances fonctionnelles F4 et F1
- NoAvion et Propriétaire vont devenir des attributs isolés, il faut donc les éliminer
- Nous construisons maintenant la relation **Type-Avion(Type, Capacité, Constructeur)** et nous éliminons les dépendances fonctionnelles F2 et F3 :
- Type, Capacité et Constructeur sont des attributs isolés ils sont alors éliminés.
- Le graphe est maintenant vide.

Exemple 2

PRODUIT (RefProduit, LibelleProduit, PU, Quantité, NumService, Adresse, Capacité)
Avec les dépendances fonctionnelles suivantes :
F1 : RefProduit → LibelleProduit
F2 : RefProduit → PU
F3 : NumService → Adresse, Capacité
F4 : RefProduit, NumService → Quantité
La dépendance fonctionnelle F3 n'est pas canonique, il faut donc la décomposer en deux dépendances fonctionnelles F31 et F32 :
F31 : NumService → Adresse
F32 : NumService → Capacité

Le graphe des dépendances est le suivant :



Produit (RefProduit, LibelleProduit, PU) avec les dépendances fonctionnelles F1 et F2
Service (NumService, Adresse, Capacité) avec les dépendances fonctionnelles F31 et F32
Produit-Service (#RefProduit, #NumService, Quantité)avec la dépendance fonctionnelle F4

- D'après l'étape 3 nous allons construire d'abord la relation **Produit (RefProduit, LibelleProduit, PU)** et nous éliminons les dépendances fonctionnelles F1 et F2
- LibelleProduit et PU vont devenir des attributs isolés, il faut donc les éliminer
- Nous construisons maintenant la relation **Service (NumService, Adresse, Capacité)** et nous éliminons les dépendances fonctionnelles F31 et F32 :
- Adresse et Capacité sont des attributs isolés ils sont alors éliminés.
- Nous construisons maintenant la relation **Produit-Service (RefProduit, NumService, Quantité)** et nous éliminons la dépendance fonctionnelle F4.
- RefProduit, NumService, Quantité sont des attributs isolés ils sont alors éliminés.
- Le graphe est maintenant vide.

V. Conclusion

Ce chapitre a été consacré à l'étude du processus de normalisation et à la présentation des différents types de dépendances fonctionnelles et de leurs types. Ces dépendances fonctionnelles sont vues comme des contraintes d'intégrité. Elles servent à réduire la redondance des données et à éviter des risques d'anomalies à travers des schémas relationnels « corrects ».
Généralement, la démarche de la mise en place d'une base de données relationnelle est la suivante :

- Concevoir un schéma de données exprimé à l'aide d'un modèle de représentation « plus abstrait » que le modèle relationnel, tel que le modèle entité association
- Transformer ce schéma en un schéma relationnel
- S'assurer que chaque relation est au mois en troisième forme normale
- Normaliser les relations qui ne le seraient pas.

Dans le chapitre suivant nous allons étudier les possibilités d'interrogation et de création de schémas relationnels normalisés à travers le langage SQL.

Exercices

Exercice 1

(i) La relation suivante est elle en 1FN?

VOITURE (no-voiture, type(modèle, marque), couleur, puissance)

Il s'agit d'une relation correspondant à un ensemble de voitures, pour laquelle les attributs sont

- no-voiture : est le numéro permettant d'identifier une voiture
- type : permettant d'indiquer le type de la voiture son modèle et sa marque
- couleur
- puissance : en chevaux fiscaux.

Si elle n'est pas en 1FN, proposer une transformation de cette relation en 1FN

(ii) La relation suivante est elle en 2FN?

LOCATION (date-location, no-client, prix-location, durée, nom, prénom, profession)

Il s'agit d'une relation correspondant à des locations de voitures à des clients

- Date-location : date à laquelle la location est faite
- No-client : numéro permettant d'identifier le client
- Prix-location : prix de la location
- Durée : durée de la location
- Nom : nom du client
- Prénom : prénom du client
- Profession : profession du client

Si ce n'est pas le cas, proposer une décomposition de cette relation en 2FN.

(iii) La relation suivante est elle en 3FN?

LOCATION (no-location, no-voiture, no-client, durée, puissance-voiture, année-voiture)

Il s'agit d'une relation correspondant à des locations de voitures à des clients

- No-location : numéro de l'opération de location
- No-voiture : numéro permettant d'identifier la voiture
- No-client : numéro permettant d'identifier le client
- Durée : durée de la location
- Puissance-voiture : puissance fiscale de la voiture
- Année-voiture : année d'acquisition de la voiture par la société de location de voitures.

Si ce n'est pas le cas, proposer une décomposition de cette relation en 3FN.

Exercice 2

Une société décide de vendre des livres par correspondance. Elle constitue pour cela une base de données comportant notamment les détails des différentes commandes qu'elle reçoit. Parmi les tables considérées figure ainsi la table COMMANDE, correspondant à des commandes de livres par des clients, pour laquelle deux structures sont étudiées :

(i) Soit la relation suivante :

COMMANDE1 (date-commande, no-client, nom, prénom, profession, délai)

- **date-commande** : date à laquelle la commande a été passée par le client
- **no-client** : numéro permettant d'identifier le client
- **nom** : nom du client
- **prénom** : prénom du client
- **profession** : profession du client

- **délai** : délai, à partir de la date-commande, au bout duquel le livre est envoyé au client.

Expliquer pourquoi cette relation n'est pas en 2FN et proposer une décomposition de cette relation en 2FN.

(ii) Soit la relation suivante :
COMMANDE2 (no-commande, date-commande, no-livre, no-client, nombre-pages, éditeur, titre, délai)

- **no-commande** : numéro de la commande
- **date-commande** : date à laquelle la commande a été passée par le client
- **no-livre** : numéro permettant d'identifier le livre
- **no-client** : numéro permettant d'identifier le client
- **nombre-pages** : nombre de pages du livre
- **éditeur** : éditeur du livre
- **titre** : titre du livre
- **délai** : délai, à partir de la date-commande, au bout duquel le livre est envoyé au client.

Expliquer pourquoi cette relation n'est pas en 3FN et proposer une décomposition de cette relation en 3FN.

Exercice 3

Pour une table R = {A, B, C, D, E} avec les dépendances fonctionnelles suivantes:

A, B ==> C, D, E
A ==> D, E
D ==> E

(i) Selon la définition de la 2e forme normale, expliquer pourquoi cette table n'est pas en 2FN.

(ii) Décomposer R pour régler le problème de la 2FN. Assurez-vous de préserver les dépendances.

Exercice 4

Avec les tables et les dépendances fonctionnelles produites dans la question 1.ii,

(i) Vérifier si chacune des tables décomposées est en 3FN.

(ii) Décomposer les tables pour obtenir au moins la 3e FN.

Exercice 5

Pour une table R = {A, B, C, D, E} avec les dépendances fonctionnelles suivantes:

A ==> B, C, D, E
CD ==> E
EC ==> B

(i) Indiquer la forme normale de R, en donnant la justification.

(ii) Décomposer R pour obtenir la forme normale la plus élevée que possible. Assurez-vous de préserver les dépendances.

Exercice 6

Avec une table R = {A, B, C} et les dépendances fonctionnelles suivantes:

A, B ==> C
C ==> B

(i) Indiquer la forme normale de R, en donnant la justification.
(ii) Décomposer R pour obtenir la forme normale la plus élevée que possible. Assurez-vous de préserver les dépendances.

Exercice 7

Pour une table R = {A, B, C, D, E, F, G, H, I, J} avec les dépendances fonctionnelles suivantes:

A, B ==> C
A ==> D, E
B ==> F
F ==> G, H
D ==> I, J

(i) Quelle est la clé de R?
(ii) Décomposer progressivement la table R jusqu'aux formes normales optimales. Indiquer clairement les tables finales. Assurez-vous de préserver les dépendances.

Exercice 8

Voici deux ensembles de dépendances fonctionnelles. Avec les axiomes d'Armstrong, démontrer qu'il est possible d'obtenir l'ensemble G à partir de l'ensemble F.

F = {A ==> C, AC ==> D, E ==> AD, E ==> H}
G = {A ==> CD, E ==> AH}

Exercice 9

Que pensez vous du schéma relationnel obtenu à l'exercice 3, chapitre modèle relationnel. Améliorez ce schéma par les règles de normalisation.

Exercice 10

On envisage de créer un système centralisé de réservation de chambres d'hôtel, dans une région de vacances qui englobe plusieurs stations ayant chacune plusieurs hôtels.
Tout demandeur peut téléphoner au système, afin de réserver des chambres. L'opérateur chargé de la réservation lui demande plusieurs renseignements: nom, adresse, numéro de téléphone et diverses indications sur sa demande: période de réservation, le nombre de chambres, la catégorie d'hôtel, la station demandée.
L'opérateur affecte à chaque demande un numéro d'ordre. Pour deux périodes distinctes, on considérera qu'il y a deux demandes distinctes.

L’opérateur est chargé de vérifier si la demande peut erre satisfaite ; s’il n’y a pas de possibilité, la demande peut être mise en attente. S’il est possible de satisfaire la demande, il y a confirmation par l’opérateur, avec création d’une réservation qui comporte tous les renseignements qui vont permettre sa réalisation.
Une réservation est associée à un client. Un client est une personne qui a, ou a eu, au moins une réservation établie par le système.
Une demande est relative à un seul demandeur et la réservation à un seul client.
Chaque hôtel est décrit par son nom, son numéro, la station à laquelle il appartient, sa catégorie, le nombre de chambres disponibles, le prix unitaire de chaque chambre et les périodes de disponibilité pour chaque chambre. Chaque station est décrite par son nom et son numéro.
Chaque période est repérée par une date début et une date fin.

Questions :
1) Établir le graphe de dépendances fonctionnelles
2) Chercher la collection des relations en troisième forme normale

Chapitre 7
Le Langage SQL

7-1- Introduction

SQL (**S**tructured **Q**uery **L**anguage, traduit Langage de requêtes structuré ou langage d'interrogation structuré) est un langage de quatrième génération (**L4G**), non procédural, conçu par IBM dans les années 70. **SQL** est basée sur l'algèbre relationnelle (opérations ensemblistes et relationnelles). SQL a été normalisé dès 1986 mais les premières normes, trop incomplètes, ont été ignorées par les éditeurs de SGBD. La norme actuelle SQL-2 (appelée aussi SQL-92) date de 1992. Elle est acceptée par tous les SGBD relationnels. Ce langage permet l'accès aux données et se compose de quatre sous-ensembles :

❶ Le **L**angage d'**I**nterrogation de **D**onnées : **LID** : Ce langage permet de rechercher des informations utiles en interrogeant la base de données. Certains considèrent ce langage comme étant une partie du LMD.

❷ Le **L**angage de **M**anipulation de **D**onnées : **LMD** (**D**ata **M**anipulation **L**anguage **DML**) : Ce langage permet de manipuler les données de la base et de les mettre à jour.

❸ Le **L**angage de **D**éfinition de **D**onnées : **LDD** (**D**ata **D**efinition **L**anguage **DDL**) : Ce langage permet la définition et la mise à jour de la structure de la base de données (tables, attributs, vues, index, ...).

❹ Le **L**angage de **C**ontrôle de **D**onnées : **LCD** (**D**ata **C**ontrol **L**anguage **DCL**) : Ce langage permet de définir les droits d'accès pour les différents utilisateurs de la base de données, donc il permet de gérer la sécurité de la base et de confirmer et d'annuler les transactions.

SQL[1]			
LDD	**LMD**	**LID**	**LCD**
Create Alter Drop	Insert Update Delete	Select	Grant Revoke

Exemple : Soit la base de données relationnelle (BD commerciale) suivante :

Produit (NP, LibP, Coul, Poids, PU, Qtes) - *Désigne l'ensemble des produits.*
Client (NCl, NomCl, AdrCl) - *Désigne l'ensemble des clients.*
Commande (NCmd, DateCmd, #NCl) - *Désigne l'ensemble des commandes.*
Ligne_Cmd (#NCmd, #NP, Qte) - *Désigne l'ensemble des lignes de commandes.*

Client

NCl	**NomCl**	**AdrCl**
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

Produit

NP	**LibP**	**Coul**	**Poids**	**PU**	**Qtes**
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

[1] *SQL ne fait pas la différence entre majuscules et minuscules*

Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

Toutes les manipulations concernant les commandes SQL seront faites sur cette base de données.

7-2- Langage d'Interrogation de Données

7-2-1- Syntaxe générale

Le **SQL** est à la fois un langage de manipulation de données et un langage de définition de données. Toutefois, la définition de données est l'oeuvre de l'administrateur de la base de données, c'est pourquoi la plupart des personnes qui utilisent le langage SQL ne se servent que du langage de manipulation de données et plus précisément du langage d'interrogation des données, permettant de sélectionner les données qui les intéressent.
La principale commande du langage d'interrogation de données est la commande **SELECT**.
La commande **SELECT** est basée sur l'algèbre relationnelle, en effectuant des opérations de sélection de données sur plusieurs tables relationnelles par projection. Sa syntaxe générale est la suivante :

SELECT[2] **[ALL] | [DISTINCT]** <*liste des attributs*> | ***
FROM[3] <*liste des tables*>
[WHERE <*condition*>**]**
[GROUP BY <*liste des attributs*>**]**
[HAVING <*condition de groupement* >**]**
[ORDER BY <*liste des attributs de tri*>**]** **;**

- […] : le contenu entre crochet est facultatif.
- L'option **ALL** est, par opposition à l'option **DISTINCT**, l'option par défaut. Elle permet de sélectionner l'ensemble des lignes satisfaisant à la condition logique.
- L'option **DISTINCT** permet de ne conserver que des lignes distinctes, en éliminant les doublons.
- La *liste des attributs* indique la liste des attributs choisis, séparés par des virgules. Lorsque l'on désire sélectionner l'ensemble des colonnes d'une table il n'est pas nécessaire de saisir la liste de ses attributs, l'option * permet de réaliser cette tâche.
- La *liste des tables* indique l'ensemble des tables (séparées par des virgules) sur lesquelles on opère.
- La *condition* permet d'exprimer des qualifications complexes à l'aide d'opérateurs logiques et de comparateurs arithmétiques.

[2] Permet d'indiquer quelles colonnes ou quelles expressions doivent être retournées par l'interrogation.
[3] Spécifie les tables participant à l'interrogation.

7-2-2- Projection

Une projection est une instruction permettant de sélectionner un ensemble d'attributs dans une table.

Syntaxe :
SELECT [ALL] | [DISTINCT] <*liste des attributs*> | ***
FROM <*la table***>** ;

Applications :

✓ Donner la liste des clients.

SELECT *
FROM Client ;

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

✓ Donner l'ensemble des produits qui ont été commandés (NP seulement).

SELECT NP
FROM Ligne_Cmd;

SELECT DISTINCT NP
FROM Ligne_Cmd;

Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

NP
P001
P004
P006
P002
P007
P001
P002
P004
P005
P008
P001
P002

NP
P001
P004
P006
P002
P007
P005
P008

DISTINCT est utilisé pour éliminer les duplications, par défaut on obtient tous les tuples (**ALL**).

7-2-3- Restrictions

Une restriction consiste à sélectionner les lignes satisfaisant à une condition logique effectuée sur leurs attributs. En SQL, les restrictions s'expriment à l'aide de la clause **WHERE** suivie d'une condition logique exprimée à l'aide d'opérateurs logiques (**AND**, **OR**, **NOT**), d'opérateurs

arithmétiques (+, -, *, /, %), de comparateurs arithmétiques (=, !=, >, <, >=, <=) et des prédicats (**NULL**, **IN**, **BETWEEN**, **LIKE**, **ALL**, **SOME**, **ANY**, **EXISTS**). Ces opérateurs s'appliquent aux valeurs numériques, aux chaînes de caractères et aux dates[4].

Syntaxe :

SELECT [ALL] | [DISTINCT] <*liste des attributs*> | *
FROM <*la table*>
WHERE <*condition*> **;**

Au niveau de la clause **WHERE**, les prédicats sont les suivants :

WHERE exp1 = exp2	Condition est vraie si les deux expressions exp1 et exp2 sont égales.
WHERE exp1 != exp2	Condition est vraie si les deux expressions exp1 et exp2 sont différentes.
WHERE exp1 < exp2	Condition est vraie si exp1 est inférieure à exp2.
WHERE exp1 > exp2	Condition est vraie si exp1 est supérieure à exp2.
WHERE exp1 <= exp2	Condition est vraie si exp1 est inférieure ou égale à exp2.
WHERE exp1 >= exp2	Condition est vraie si exp1 est supérieure ou égale à exp2.
WHERE exp1 **BETWEEN** exp2 **AND** exp3	Condition est vraie si exp1 est comprise entre exp2 et exp3, bornes incluses.
WHERE exp1 **LIKE** exp2	Condition est vraie si la sous–chaîne exp2 est présente dans exp1.
WHERE exp1 **NOT LIKE** exp2	Condition est vraie si la sous–chaîne exp2 n'est pas présente dans exp1.
WHERE exp1 **IN** (exp2, exp3,…)	Condition est vraie si exp1 appartient à l'ensemble (exp2, exp3, …).
WHERE exp1 **NOT IN** (exp2, exp3,…)	Condition est vraie si exp1 n'appartient pas à l'ensemble (exp2, exp3, …).
WHERE exp1 **IS NULL**	Condition est vraie si exp1 est nulle.
WHERE exp1 **IS NOT NULL**	Condition est vraie si exp1 n'est pas nulle.
WHERE exp1 Op **ALL** (exp2, exp3,…)	Op représente un opérateur de comparaison (<, >, =, !=, <=, >=) Condition est vraie si la comparaison de l'exp1 est vraie avec toutes les valeurs de la liste (exp2, exp3, …). Si la liste est vide, le résultat est vrai.
WHERE exp1 Op **ANY** (exp2, exp3,…) **WHERE** exp1 Op **SOME** (exp2, exp3,…)	Op représente un opérateur de comparaison (<, >, =, !=, <=, >=) Condition est vraie si la comparaison de l'exp1 est vraie avec au moins une valeur de la liste (exp2, exp3, …). Si la liste est vide, le résultat est faux.
WHERE EXISTS sous-requête	Condition est vraie si le résultat de la sous-requête n'est pas vide.

Remarques :

- L'opérateur **IN** est équivalent à **= ANY**
- L'opérateur **NOT IN** est équivalent à **!= ANY**

[4] Les constantes de type chaîne de caractères et date doivent être encadrées par apostrophes ' … ' contrairement aux nombres.

Applications :

✓ Donner le numéro et le nom des clients de la ville de Sousse.

SELECT NCl, NomCl
FROM Client
WHERE AdrCl = 'Sousse';

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

NCl	NomCl
CL03	AMS
CL04	GLOULOU
CL06	ELECTRON
CL07	SBATIM

✓ Donner la liste des commandes dont la date est supérieure à '01/01/2004'.

SELECT *
FROM Commandes
WHERE DateCmd > '01/01/2004';

Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

NCmd	DateCmd	NCl
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C005	11/03/2004	CL03

✓ Donner la liste des produits dont le prix est compris entre 20 et 50.

SELECT *
FROM Produit
WHERE PU **BETWEEN** 20 **AND** 50;

Ou bien

SELECT *
FROM Produit
WHERE (PU >= 20) **AND** (PU <= 50);

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

NP	LibP	Coul	Poids	PU	Qtes
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P006	Serrure	Jaune	2	47.000	1250

✓ Donner la liste des clients dont les noms commencent par 'B'.

SELECT *
FROM Client
WHERE NomCl **LIKE** 'B%';

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL12	BATFER	Tunis

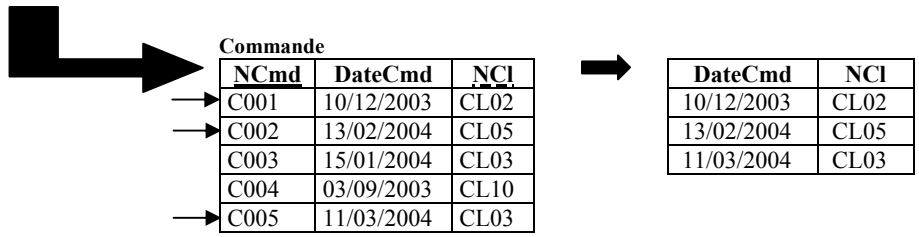
Le prédicat **LIKE** permet de faire des comparaisons sur des chaînes grâce à des caractères, appelés caractères *jokers* :

- Le caractère % permet de remplacer une séquence de caractères (éventuellement nulle).
- Le caractère _ permet de remplacer un caractère.
- Les caractères [-] permettent de définir un intervalle de caractères (par exemple [J-M]).

La sélection des clients dont les noms ont un E en deuxième position se fait par l'instruction : **WHERE** NomCl **LIKE** "_E%"

✓ Donner les numéros des clients dont les dates de leurs commandes se trouvent parmi les dates suivantes : (‘10-12-03’, ‘10-12-04’,’13-02-04’,’11-03-04’).

SELECT DateCmd, NCl
FROM Commande
WHERE DateCmd **IN** (‘10-12-2003’,’10-12-2004’,’13-02-2004’,’11-03-2004’);



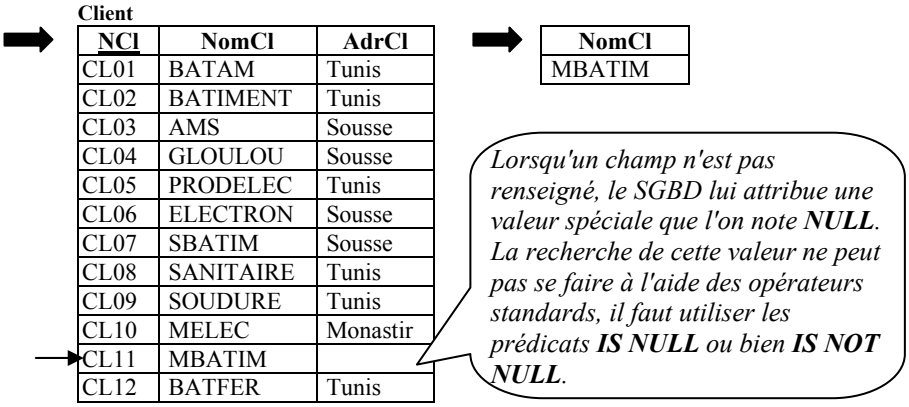
Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

DateCmd	NCl
10/12/2003	CL02
13/02/2004	CL05
11/03/2004	CL03

✓ Donner les noms des clients qui n’ont pas d’adresse.

SELECT NomCl
FROM Client
WHERE AdrCl **IS NULL;**



Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

NomCl
MBATIM

*Lorsqu'un champ n'est pas renseigné, le SGBD lui attribue une valeur spéciale que l'on note **NULL**. La recherche de cette valeur ne peut pas se faire à l'aide des opérateurs standards, il faut utiliser les prédicats **IS NULL** ou bien **IS NOT NULL**.*

7-2-4- Expressions, alias et fonctions

a. Expression

Les expressions acceptées par SQL portent sur des attributs, des constantes et des fonctions. Ces trois types d’éléments peuvent être reliés par des opérateurs arithmétiques (+, -, *, /). Les expressions peuvent figurer :

✓ En tant que colonne résultat d’un ordre **SELECT**.
✓ Dans une clause **WHERE**.
✓ Dans une clause **ORDER BY** (cf. Tri).
✓ Dans les ordres de manipulation des données (**INSERT**, **UPDATE**, **DELETE** cf. LMD).

b. Alias

Les alias permettent de renommer des attributs ou des tables.

SELECT attr1 **AS** aliasa1, attr2 **AS** aliasa2, …
FROM table1 aliast1, table2 aliast2… ;

Pour les attributs, l’alias correspond aux titres des colonnes affichées dans le résultat de la requête. Il est souvent utilisé lorsqu’il s’agit d’attributs calculés (expression).

NB : Le mot clé **AS** est optionnel.

c. Fonction

Le tableau suivant donne le nom des principales fonctions prédéfinies :

Nom de la fonction	Rôle de la fonction
AVG	Moyenne
SUM	Somme
MIN	Minimum
MAX	Maximum
COUNT (*)	Nombre de lignes
COUNT (Attr)	Nombre de valeurs non nulles de l'attribut
COUNT([DISTINCT] Attr)	Nombre de valeurs non nulles différentes de l'attribut

Applications :

✓ Donner la valeur des produits en stock.

SELECT NP, (Qtes * PU) **AS** "Valeur Totale"
FROM Produit ;

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

NP	Valeur Totale
P001	21600
P002	1500
P003	37500
P004	29700
P005	15600
P006	58750
P007	11000
P008	72000

On peut exprimer cette requête sans spécifier le mot clé **AS** :

SELECT NP, (Qtes* PU) "Valeur Totale"
FROM Produit ;

On ne met les noms des colonnes résultats entre guillemets que s'ils contiennent des espaces.

SELECT NP, (Qtes * PU) Valeur
FROM Produit ;

✓ Donner la moyenne des prix unitaires des produits.

SELECT AVG(PU)
FROM Produit;

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

AVG(PU)
29

7-2-5- Sélection avec jointure

Il s'agit ici de sélectionner les données provenant de plusieurs tables ayant un ou plusieurs attributs communs. Cette jointure sera assurée grâce aux conditions spécifiées dans la clause **WHERE**.

Syntaxe :

SELECT [ALL] | [DISTINCT] <*liste des attributs*> | *
FROM <*liste des tables*>
WHERE Nom_Table1.Attrj = Nom_Table2.Attrj **AND** …
AND <*condition*> ;

Applications :

✓ Donner les libellés des produits de la commande numéro 'C002'.

```
SELECT DISTINCT LibP
FROM Produit, Ligne_Cmd
WHERE Produit.NP = Ligne_Cmd.NP
      AND NCmd='C002';
```

Ou bien

```
SELECT DISTINCT LibP
FROM Produit P, Ligne_Cmd L
WHERE P.NP = L.NP
      AND NCmd='C002';
```

Pour ne pas spécifier à chaque fois le nom complet des tables Produit et Ligne_Cmd dans la condition, on peut renommer ces tables.

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

LibP
Prise
Verrou

On peut exprimer cette jointure autrement (cf. Sous-requêtes) :

```
SELECT LibP
FROM Produit
WHERE NP IN      (SELECT NP
                  FROM Ligne_Cmd
                  WHERE NCmd = 'C002');
```

✓ Donner les produits (toutes les informations) commandés au cours de l'année 2003 et qui sont vendus aux clients de Tunis.

```
SELECT DISTINCT P.NP, LibP, Coul, Poids, PU, Qtes
FROM Produit P, Commande C, Client Cl, Ligne_Cmd L
WHERE P.NP = L.NP
      AND C.NCmd = L.NCmd
      AND Cl.NCl = C.NCl
      AND Cl.AdrCl ='Tunis'
      AND DateCmd BETWEEN '01/01/2003' AND '31/12/2003';
```

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18	1200
P004	Peinture	Blanc	25	33	900
P006	Serrure	Jaune	2	47	1250

✓ Donner les libellés des produits qui ont un prix unitaire supérieur à celui du produit 'Robinet'.

```
SELECT P.LibP
FROM Produit P, Produit P1
WHERE P.PU > P1.PU
        AND P1.LibP ='Robinet';
```

Dans certaines requêtes, on est obligé de renommer soit des tables, soit des attributs (ici les tables).

LibP
Câble
Peinture
Serrure
Fer

7-2-6- Groupement

Il est possible de grouper des lignes de données ayant une valeur commune à l'aide de la clause **GROUP BY** et des fonctions de groupe (cf. Fonction).

Syntaxe :

```
SELECT Attr1, Attr2,…, Fonction_Groupe
FROM Nom_Table1, Nom_Table2,…
WHERE Liste_Condition
GROUP BY Liste_Groupe
HAVING Condition;
```

- La clause **GROUP BY**, suivie du nom de chaque attribut sur laquelle on veut effectuer des regroupements.
- La clause **HAVING** va de pair avec la clause **GROUP BY**, elle permet d'appliquer une restriction sur les groupes créés grâce à la clause **GROUP BY**.

NB : Les fonctions d'agrégat, utilisées seules dans un **SELECT** (sans la clause **GROUP BY**) fonctionnent sur la totalité des tuples sélectionnés comme s'il n'y avait qu'un groupe.

Applications :

✓ Donner le nombre de commandes par client.

```
SELECT NCl, COUNT((NCmd) NbCmd
FROM Commande
GROUP BY NCl ;
```

… Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

NCl	NbCmd
CL02	1
CL05	1
CL03	2
CL10	1

✓ Donner la quantité totale commandée par produit.

```
SELECT NP, SUM(Qte) Som
FROM Ligne_Cmd
GROUP BY NP;
```

… Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

NP	Som
P001	950
P002	400
P004	450
P005	70
P006	100
P007	550
P008	90

✓ Donner le nombre de produits par commande.

```
SELECT NCmd, COUNT(NP) NbProd
FROM Ligne_Cmd
GROUP BY NCmd;
```

… Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

NCmd	NbProd
C001	3
C002	2
C003	1
C004	4
C005	2

✓ Donner les commandes dont le nombre de produits dépasse 2.

```
SELECT NCmd, COUNT(NP) NbProd
FROM Ligne_Cmd
GROUP BY NCmd
HAVING COUNT(NP) > 2;
```

… Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

NCmd	NbProd
C001	3
C004	4

✓ Donner le total des montants par commande.

```
SELECT NCmd, SUM(PU*Qte) TotMontant
FROM Produit P, Ligne_Cmd L
WHERE L.NP = P.NP
GROUP BY NCmd;
```

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

NCmd	TotMontant
C001	19100
C002	3325
C003	900
C004	14040
C005	11850

7-2-7- Tri

Les lignes constituant le résultat d'un **SELECT** sont obtenues dans un ordre quelconque. La clause **ORDER BY** précise l'ordre dans lequel la liste des lignes sélectionnées sera donnée.

Syntaxe :
SELECT Attr1, Attr2,…, Attrn
FROM Nom_Table1, Nom_Table2, …
WHERE Liste_Condition
ORDER BY Attr1 [**ASC**| **DESC**], Attr2 [**ASC**| **DESC**], …;

NB : *L'ordre de tri par défaut est croissant (ASC).*

Application :
✓ Donner les noms des clients suivant l'ordre décroissant.

```
SELECT NomCl
FROM Client
ORDER BY NomCl DESC ;
```

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

NomCl
SOUDURE
SBATIM
SANITAIRE
PRODELEC
MELEC
MBATIM
GLOULOU
ELECTRON
BATIMENT
BATFER
BATAM
AMS

✓ Donner le nombre de produits et la quantité totale par commande suivant l'ordre décroissant du nombre de produits et l'ordre croissant de la quantité totale.

```
SELECT NCmd,
COUNT DISTINCT(NP) NbProd,
SUM(Qte) SomQte
FROM Ligne_Cmd
GROUP BY NCmd
ORDER BY NbProd DESC,
         SomQte ASC ;
```

… **Ligne_Cmd**

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

NCmd	NbProd	SomQte
C004	4	410
C001	3	650
C005	2	750
C002	2	750
C003	1	50

7-2-8- Sous requêtes

Effectuer une sous-requête consiste à effectuer une requête à l'intérieur d'une autre, ou en d'autres termes d'utiliser une requête afin d'en réaliser une autre (on entend parfois le terme de requêtes en cascade).
Une sous-requête doit être placée à la suite d'une clause **WHERE** ou **HAVING**, et doit remplacer une constante ou un groupe de constantes qui permettraient en temps normal d'exprimer la qualification.

- lorsque la sous-requête remplace une constante utilisée avec des opérateurs classiques, elle doit obligatoirement renvoyer une seule réponse (une table d'une ligne et une colonne). Par exemple :
  SELECT … **FROM** …
  WHERE … < (**SELECT** … **FROM** …) ;

- lorsque la sous-requête remplace une constante utilisée dans une expression mettant en jeu les opérateurs **IN**, **ALL** ou **ANY**, elle doit obligatoirement renvoyer une seule colonne.
  SELECT … **FROM** …
  WHERE … **IN** (**SELECT** … **FROM** …) ;

- lorsque la sous-requête remplace une constante utilisée dans une expression mettant en jeu l'opérateur **EXISTS**, elle peut renvoyer une table de n colonnes et m lignes.
  SELECT … **FROM** …
  WHERE … **EXISTS** (**SELECT** … **FROM** …) ;

Applications :

✓ Donner les produits dont les prix unitaires dépassent la moyenne des prix.

SELECT *
FROM Produit
WHERE PU > (**SELECT AVG**(PU) **FROM** Produit);

Le résultat de la sous-requête :

AVG(PU)
29

… **Produit**

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

NP	LibP	Coul	Poids	PU	Qtes
P004	Peinture	Blanc	25	33.000	900
P006	Serrure	Jaune	2	47.000	1250
P008	Fer	Noir	50	90.000	800

✓ Donner les commandes qui ont une date inférieure à chacune des commandes du client 'CL03'.

SELECT *
FROM Commande
WHERE DateCmd < **ALL**
(**SELECT** DateCmd
FROM Commande
WHERE NCl = 'CL03');

Le résultat de la sous-requête :

DateCmd
15/01/2004
11/03/2004

… **Commande**

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C004	03/09/2003	CL10

✓ Donner les commandes qui ont une date inférieure à au moins une des commandes du client 'CL03'.

SELECT *
FROM Commande
WHERE DateCmd < **ANY**
(**SELECT** DateCmd
FROM Commande
WHERE NCl = 'CL03');

Le résultat de la sous-requête :

DateCmd
15/01/2004
11/03/2004

… **Commande**

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10

✓ Donner les clients qui ont passé au moins une commande.

SELECT NomCl
FROM Client Cl
WHERE EXIST (**SELECT** * **FROM** Commande C **WHERE** Cl.NCl =C.NCl);

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

➡

NomCl
BATIMENT
AMS
PRODELEC
MELEC

✓ Donner des clients qui n'ont passé aucune commande.

SELECT NomCl
FROM Client Cl
WHERE NOT EXIST (**SELECT** * **FROM** Commande C **WHERE** Cl.NCl =C.NCl);

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

➡

NomCl
BATAM
GLOULOU
ELECTRON
SBATIM
SANITAIRE
SOUDURE
MBATIM
BATFER

7-2-9- Opérateurs ensemblistes

❶ *Union* : L'opérateur **UNION** permet de fusionner deux sélections de tables pour obtenir un ensemble de lignes égal à la réunion des lignes des deux sélections. Les lignes communes n'apparaîtront qu'une seule fois.

Syntaxe :
Requête1
UNION
Requête2 ;

NB :

- Requête1 et Requête2 doivent avoir la même structure.
- Par défaut les doublons sont automatiquement éliminés. Pour conserver les doublons, il est possible d'utiliser une clause **UNION ALL**.

Application :

✓ Donner l'ensemble des clients de Tunis et de Sousse qui ont des commandes.

```
SELECT Cl.NCl, NomCl, AdrCl
FROM Client Cl, Commande C
WHERE Cl.NCl=C.NCl
        AND J.AdrCl='Tunis'
UNION
SELECT Cl.NCl, NomCl, AdrCl
FROM Client Cl, Commande C
WHERE Cl.NCl=C.NCl
        AND J.AdrCl='Sousse';
```

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse
CL04	GLOULOU	Sousse
CL05	PRODELEC	Tunis
CL06	ELECTRON	Sousse
CL07	SBATIM	Sousse
CL08	SANITAIRE	Tunis
CL09	SOUDURE	Tunis
CL10	MELEC	Monastir
CL11	MBATIM	
CL12	BATFER	Tunis

Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03

R1

NCl	NomCl	AdrCl
CL02	BATIMENT	Tunis
CL05	PRODELEC	Tunis

R2

NCl	NomCl	AdrCl
CL03	AMS	Sousse

NCl	NomCl	AdrCl
CL02	BATIMENT	Tunis
CL05	PRODELEC	Tunis
CL03	AMS	Sousse

❷ *Intersection* : L'opérateur **INTERSECT** permet d'obtenir l'ensemble des lignes communes à deux requêtes.

Syntaxe :
Requête1
INTERSECT
Requête2 ;

NB:
- Requête1 et Requête2 doivent avoir la même structure.
- Il est possible de remplacer l'opérateur **INTERSECT** par des commandes usuelles :

```
SELECT a, b
FROM table1
WHERE EXISTS (   SELECT c, d FROM table2
                 WHERE a=c AND b=d) ;
```

Application :

✓ Donner l'ensemble des produits communs aux commandes C001 et C005.

```
SELECT P.NP, LibP
FROM Produit P, Ligne_Cmd L
WHERE P.NP=L.NP
        AND NCmd='C001'
INTERSECT
SELECT P.NP, LibP
FROM Produit P, Ligne_Cmd L
WHERE P.NP=L.NP
        AND NCmd='C005' ;
```

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

R1

NP	LibP
P001	Robinet
P004	Peinture
P006	Serrure

R2

NP	LibP
P001	Robinet
P002	Prise

NP	LibP
P001	Robinet

❸ *Différence* : L'opérateur **MINUS** permet d'obtenir les lignes de la première requête et qui ne figurent pas dans la deuxième.

Syntaxe :
Requête1
MINUS (ou **EXCEPT)**
Requête2 ;
Avec Requête1 et Requête2 de même structure.

Application :
✓ Donner l'ensemble des produits qui n'ont pas été commandés.

```
SELECT NP, LibP
FROM Produit
MINUS
SELECT P.NP, LibP
FROM Produit P, Ligne_Cmd L
WHERE P.NP=L.NP;
```

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000
P003	Câble	Blanc	2	25.000	1500
P004	Peinture	Blanc	25	33.000	900
P005	Poignée	Gris	3	12.000	1300
P006	Serrure	Jaune	2	47.000	1250
P007	Verrou	Gris	1.7	5.500	2000
P008	Fer	Noir	50	90.000	800

Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

Tous les produits

Les produits commandés

R1

NP	LibP
P001	Robinet
P002	Prise
P003	Câble
P004	Peinture
P005	Poignée
P006	Serrure
P007	Verrou
P008	Fer

R2

NP	LibP
P001	Robinet
P002	Prise
P004	Peinture
P005	Poignée
P006	Serrure
P007	Verrou
P008	Fer

NP	LibP
P003	Câble

❸ *Produit cartésien* : Le produit cartésien est appliqué sur deux tables n'ayant pas nécessairement la même structure pour obtenir une autre table.
Syntaxe :
SELECT *
FROM table1, table2, … **;**

Application :
✓ Faire le produit cartésien entre la table Client et la table Produit.

```
SELECT *
FROM Client, Produit;
```

Client

NCl	NomCl	AdrCl
CL01	BATAM	Tunis
CL02	BATIMENT	Tunis
CL03	AMS	Sousse

Produit

NP	LibP	Coul	Poids	PU	Qtes
P001	Robinet	Gris	5	18.000	1200
P002	Prise	Blanc	1.2	1.500	1000

NCl	NomCl	AdrCl	NP	LibP	Coul	Poids	PU	Qtes
CL01	BATAM	Tunis	P001	Robinet	Gris	5	18.000	1200
CL01	BATAM	Tunis	P002	Prise	Blanc	1.2	1.500	1000
CL02	BATIMENT	Tunis	P001	Robinet	Gris	5	18.000	1200
CL02	BATIMENT	Tunis	P002	Prise	Blanc	1.2	1.500	1000
CL03	AMS	Sousse	P001	Robinet	Gris	5	18.000	1200
CL03	AMS	Sousse	P002	Prise	Blanc	1.2	1.500	1000

7-2-10- Langage algébrique versus Langage SQL

Sélection	Clause **WHERE** dans **SELECT**
Projection	Liste des attributs de **SELECT**
Jointure	Présence de plusieurs tables dans la clause **FROM** (avec la condition de jointure) Ex : **SELECT * FROM** T1, T2 **WHERE** T1.Attri=T2.Attrj ;
Produit cartésien	Présence de plusieurs tables dans la clause **FROM** (sans la condition de jointure) Ex : **SELECT * FROM** T1, T2 ;
Union	**UNION**
Différence	**MINUS** ou **EXCEPT**
Intersection	**INTERSECT**
Division	Il n'existe pas en SQL d'équivalent direct à la division. Cependant, il est toujours possible de trouver une autre solution notamment par l'intermédiaire des opérations de calcul et de regroupement.

7-2-11- Exemple récapitulatif

Donner le nombre de commandes et la somme des quantités du produit P001 commandé par client en ne gardant que les clients ayant un nombre de commandes >= à 2. Le résultat doit être trié selon l'ordre croissant des sommes des quantités.

Étape	Requête
4	**SELECT** NCl, **COUNT**(NCmd) NbCmd, **SUM**(Qte) SomQte
1	**FROM** Commande C, Ligne_Cmd L
2	**WHERE** C.No = L.NoCommande
3	**AND** NP = 'P001'
4	**GROUP BY** NCl
5	**HAVING** NbCmd >= 2
6	**ORDER BY** SomQte ;

1ère étape : Sélection des deux tables Commande et Ligne_Cmd.

Commande

NCmd	DateCmd	NCl
C001	10/12/2003	CL02
C002	13/02/2004	CL05
C003	15/01/2004	CL03
C004	03/09/2003	CL10
C005	11/03/2004	CL03
C006	01/03/2004	CL11
C007	01/01/2004	CL02

Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100
C006	P001	20
C007	P001	40
C007	P002	30

2ème étape : Application de la condition de jointure.

NCmd	DateCmd	NCl	NP	Qte
C001	10/12/2003	CL02	P001	250
C001	10/12/2003	CL02	P004	300
C001	10/12/2003	CL02	P006	100
C002	13/02/2004	CL05	P002	200
C002	13/02/2004	CL05	P007	550
C003	15/01/2004	CL03	P001	50
C004	03/09/2003	CL10	P002	100
C004	03/09/2003	CL10	P004	150
C004	03/09/2003	CL10	P005	70
C004	03/09/2003	CL10	P008	90
C005	11/03/2004	CL03	P001	650
C005	11/03/2004	CL03	P002	100
C006	01/03/2004	CL11	P001	20
C007	01/01/2004	CL02	P001	40
C007	01/01/2004	CL02	P002	30

3ème étape : Sélection des commandes du produit P001 uniquement.

NCmd	DateCmd	NCl	NP	Qte
C001	10/12/2003	CL02	P001	250
C003	15/01/2004	CL03	P001	50
C005	11/03/2004	CL03	P001	650
C006	01/03/2004	CL11	P001	20
C007	01/01/2004	CL02	P001	40

4ème étape : Groupement par NCl en calculant le nombre de commandes et la somme des quantités.

NCl	NbCmd	SomQte
CL02	2	290
CL03	2	700
CL11	1	20

5ème étape : Sélection des groupes ayant un nombre de commandes >= 2.

NCl	NbCmd	SomQte
CL02	2	290
CL03	2	700

6ème étape : Tri selon la somme des quantités.

NCl	NbCmd	SomQte
CL03	2	700
CL02	2	290

7-3- Langage de Manipulation de Données

Le LMD est un ensemble de commandes permettant la consultation et la mise à jour des objets créés. La mise à jour englobe l'insertion de nouvelles données, la modification et la suppression de données existantes.

7-3-1- Insertion de données

Syntaxe :

INSERT INTO Nom_Table [(Attr1, Attr2,…, Attrn)]
VALUES (Val1, Val2,…, Valn);

Permet d'insérer un tuple à la fois.

Ou

INSERT INTO Nom_Table (Attr1, Attr2, …, Attrn)
SELECT…;

Permet d'insérer plusieurs tuples à partir d'une ou plusieurs autres tables.

L'ordre **INSERT** attend la clause **INTO**, suivie du nom de la table, ainsi que du nom de chacun des attributs entre parenthèses (les attributs omis prendront la valeur **NULL** par défaut). Les

données sont affectées aux attributs (colonnes) dans l'ordre dans lequel les attributs ont été déclarés dans la clause **INTO**.

Les valeurs à insérer peuvent être précisées de deux façons :

- avec la clause **VALUES** : une seule ligne est insérée, elle contient comme valeurs, l'ensemble des valeurs passées en paramètre dans la parenthèse qui suit la clause **VALUES**.
  INSERT INTO Nom_table (Attr1, Attr2, ...)
  VALUES (Valeur1, Valeur2, ...) ;
  Lorsque chaque attribut de la table est modifié, l'énumération de l'ensemble des attributs est facultative. Lorsque les valeurs sont des chaînes de caractères, il ne faut pas omettre de les délimiter par des guillemets.

- avec la clause **SELECT** : plusieurs lignes peuvent être insérées, elle contiennent comme valeurs, l'ensemble des valeurs découlant de la sélection.
  INSERT INTO Nom_table (Attr1, Attr2, ...)
  SELECT Attr1, Attr2, ...
  FROM Nom_table2
  WHERE condition ;
  Lorsque l'on remplace un nom d'attribut suivant la clause **SELECT** par une constante, sa valeur est affectée par défaut aux tuples. Il n'est pas possible de sélectionner des tuples dans la table dans laquelle on insère des lignes (en d'autres termes Nom_table doit être différent de Nom_table2).

Applications :

✓ Remplissage de la table **Client**.

INSERT INTO Client **VALUES**('CL01','BATAM','Tunis');
INSERT INTO Client **VALUES**('CL02','BATIMENT','Tunis');
INSERT INTO Client **VALUES**('CL03','AMS','Sousse');
INSERT INTO Client **VALUES**('CL04','GLOULOU','Sousse');
INSERT INTO Client **VALUES**('CL05','PRODELEC','Tunis');
INSERT INTO Client **VALUES**('CL06','ELECTRON','Sousse');
INSERT INTO Client **VALUES**('CL07','SBATIM','Sousse');
INSERT INTO Client **VALUES**('CL08','SANITAIRE','Tunis');
INSERT INTO Client **VALUES**('CL09','SOUDURE','Tunis');
INSERT INTO Client **VALUES**('CL10','MELEC','Monastir');
INSERT INTO Client **VALUES**('CL11','MBATIM','');
INSERT INTO Client **VALUES**('CL12','BATFER','Tunis');

✓ Remplissage de la table **Produit**.

INSERT INTO Produit **VALUES**('P001','Robinet','Gris',5,18,1200);
INSERT INTO Produit **VALUES**('P002','Prise','Blanc',1.2,1.5,1000);
INSERT INTO Produit **VALUES**('P003','Câble','Blanc',2,25,1500);
INSERT INTO Produit **VALUES**('P004','Peinture','Blanc',25,33,900);
INSERT INTO Produit **VALUES**('P005','Poignée','Gris',3,12,1300);
INSERT INTO Produit **VALUES**('P006','Serrure','Jaune',2,47,1250);
INSERT INTO Produit **VALUES**('P007','Verrou','Gris',1.7,5.5,2000);
INSERT INTO Produit **VALUES**('P008','Fer','Noir',50,90,800);

✓ Remplissage de la table **Commande**.

INSERT INTO Commande **VALUES**('C001', '10/12/2003', 'CL02');
INSERT INTO Commande **VALUES**('C002', '13/02/2004', 'CL05');
INSERT INTO Commande **VALUES**('C003', '15/01/2004', 'CL03');
INSERT INTO Commande **VALUES**('C004', '03/09/2003', 'CL10');
INSERT INTO Commande **VALUES**('C005', '11/03/2004', 'CL03');

✓ Remplissage de la table **Ligne_Cmd**.

INSERT INTO Ligne_Cmd **VALUES**(‘C001’,’P001’,250);
INSERT INTO Ligne_Cmd **VALUES**(‘C001’,’P004’,300);
INSERT INTO Ligne_Cmd **VALUES**(‘C001’,’P006’,100);
INSERT INTO Ligne_Cmd **VALUES**(‘C002’,’P002’,200);
INSERT INTO Ligne_Cmd **VALUES**(‘C002’,’P007’,550);
INSERT INTO Ligne_Cmd **VALUES**(‘C003’,’P001’,50);
INSERT INTO Ligne_Cmd **VALUES**(‘C004’,’P002’,100);
INSERT INTO Ligne_Cmd **VALUES**(‘C004’,’P004’,150);
INSERT INTO Ligne_Cmd **VALUES**(‘C004’,’P005’,70);
INSERT INTO Ligne_Cmd **VALUES**(‘C004’,’P008’,90);
INSERT INTO Ligne_Cmd **VALUES**(‘C005’,’P001’,650);
INSERT INTO Ligne_Cmd **VALUES**(‘C005’,’P002’,100);

7-3-2- Modification de données

En utilisant la commande **UPDATE**, on peut modifier les valeurs d’un ou plusieurs champs, dans une ou plusieurs lignes existantes d’une table.

Syntaxe :
UPDATE Nom_Table
SET Attr1 = Expr1, Attr2 = Expr2, …
WHERE Condition;

Ou

UPDATE Nom_Table
SET (Attr1, Attr2, …) = (**SELECT** …)
WHERE Condition;

La modification à effectuer est précisée après la clause **SET**. Il s'agit d'une affectation d'une valeur à un attribut grâce à l'opérateur = suivi d'une expression algébrique, d'une constante ou du résultat provenant d'une clause **SELECT**. La clause **WHERE** permet de préciser les tuples sur lesquels la mise à jour aura lieu.

Application :

✓ Modifier le Poids du produit numéro P002 à 1.

```
UPDATE Produit
SET Poids = 1
WHERE NP = ‘P002’;
```

✓ Augmenter la quantité en stock des différents produits de 10%.

```
UPDATE Produit
SET Qtes = 1.1 * Qtes;
```

7-3-3- Suppression de données

L’ordre **DELETE** permet de supprimer des lignes d’une table.

Syntaxe :
DELETE FROM Nom_Table
WHERE Condition;

Ici la clause **FROM** précise la table sur laquelle la suppression s'effectue et la clause **WHERE** décrit la qualification, c'est-à-dire l'ensemble des lignes qui seront supprimées.

NB : L'ordre **DELETE** est à utiliser avec précaution car l'opération de suppression est irréversible. Il faudra donc s'assurer dans un premier temps que les lignes sélectionnées sont bien les lignes que l'on désire supprimer!

7-4- Langage de Définition de Données

7-4-1- Types de données

Pour chaque attribut que l'on crée, il faut préciser le type de données que le champ va contenir. Celui-ci peut être un des types suivants :

Type de donnée	**Syntaxe**	**Description**
Type alphanumérique	CHAR(n)	Chaîne de caractères de longueur fixe n (n<16383) Ajout de blancs pour garder la longueur fixe
Type alphanumérique	VARCHAR(n)	Chaîne de caractères de longueur variable de n caractères maximum (n<16383)
Type numérique	SMALLINT	Entier signé de 16 bits (-32768 à 32757)
Type numérique	INTEGER	Entier signé de 32 bits (-2E31 à 2E31-1)
Type numérique	NUMBER(n,[d])	Nombre de n chiffres [optionnellement d après la virgule]
Type numérique	NUMERIC(n, d) DECIMAL(n, d)	Nombres décimaux à nombre fixe de décimales
Type numérique	FLOAT DOUBLE PRECISION	Nombre à virgule flottante (double précision avec au moins 15 chiffres significatifs)
Type numérique	REAL	Nombre à virgule flottante (simple précision avec 7 chiffres significatifs)
Type horaire	DATE	Date sous la forme 16/07/99
Type horaire	TIME	Heure sous la forme 12:54:24.85
Type horaire	TIMESTAMP	Date et Heure

7-4-2- Valeur NULL

Un attribut qui n'est pas renseigné, et donc vide, est dit contenir la valeur '**NULL**'. Cette valeur n'est pas zéro, c'est une absence de valeur.

7-4-3- Contraintes d'intégrité

Une contrainte d'intégrité est une règle qui permet de contrôler la validité et la cohérence des valeurs entrées dans les différentes tables de la base. Elle peut être définie sous deux formes :

- ✓ Dans les commandes de création des tables.
- ✓ Au moment de la modification de la structure de la table.

Il existe des contraintes :

- ✓ *Sur un attribut* : La contrainte porte sur un seul attribut. Elle suit la définition de l'attribut. Ces contraintes sont :
  - **NOT NULL** : Spécifie que pour toute occurrence, l'attribut doit avoir une valeur (la saisie de ce champ est obligatoire).
  - **UNIQUE** : Toutes les valeurs de l'attribut sont distinctes.
  - **PRIMARY KEY** : L'attribut est une clé primaire pour la table et elle peut être remplacée par **UNIQUE** et **NOT NULL**.
  - **REFERENCES** table (attribut) : Il s'agit d'une contrainte d'intégrité fonctionnelle par rapport à une clé ; chaque valeur de l'attribut doit exister dans

la table dont l'attribut est référencée. On utilise cette contrainte pour les clés étrangères.

- **CHECK** : C'est une contrainte associée à une condition qui doit être vérifiée par toutes les valeurs de l'attribut (domaine des valeurs de l'attribut).

✓ *Sur une table* : La contrainte porte sur un ensemble d'attributs d'une même table, une virgule sépare la définition d'une contrainte de table des définitions des attributs. Ces contraintes sont :

- **UNIQUE** (attri, attrj,…) : L'unicité porte sur le n-uplet des valeurs.
- **PRIMARY KEY** (attri, attrj,…) : Clé primaire de la table (clé composée).
- **FOREIGN KEY** (attri, attrj, …) **REFERENCES** table (attrm, attrn, …) [**ON DELETE CASCADE**] : Désigne une clé étrangère sur plusieurs attributs. L'option **ON DELETE CASCADE** indique que la suppression d'une ligne de la table de référence va entraîner automatiquement la suppression des lignes référencées.

Il est possible de donner un nom à une contrainte grâce au mot clé **CONSTRAINT** suivi du nom que l'on donne à la contrainte, de telle manière à ce que le nom donné s'affiche en cas de non respect de l'intégrité, c'est-à-dire lorsque la clause que l'on a spécifiée n'est pas validée.

7-4-4- Création d'une table

La création de tables se fait à l'aide du couple de mots-clés **CREATE TABLE**.

Syntaxe :
CREATE TABLE nom_table
(Attribut1 Type [Contrainte d'attribut],
Attribut2 Type [Contrainte d'attribut],
…
Attributn Type [Contrainte d'attribut],
[Contrainte de relation], …);

Application :

✓ Création de la table **Client** en donnant à sa clé **NCl** les propriétés **NOT NULL** et **UNIQUE**.

```
CREATE TABLE Client
(NCl VARCHAR2(4) NOT NULL UNIQUE,
NomCl VARCHAR2(15),
AdrCl VARCHAR2(10));
```

✓ Création de la table **Produit** en définissant la contrainte **pk_NP** de clé primaire équivalente à l'attribution des propriétés **NOT NULL** et **UNIQUE**. Ne pas définir l'attribut Qtes ceci sera défini ultérieurement par l'instruction **ALTER TABLE**.

```
CREATE TABLE Produit
(NP VARCHAR2(4) CONSTRAINT pk_NP PRIMARY KEY,
 LibP VARCHAR2(15),
 Coul VARCHAR2(10),
 Poids NUMBER(6,3),
 PU NUMBER(6,3) );
```

✓ Création de la table **Commande** sans définir de contrainte ni de propriétés sur la clé, ceci sera défini ultérieurement par l'instruction **ALTER TABLE**.

```
CREATE TABLE Commande
(NCmd VARCHAR2(4),
 DateCmd DATE,
 NCl VARCHAR2(4));
```

✓ Création de la table **Ligne_Cmd** en spécifiant la contrainte de clé primaire et l'une des contraintes de clé étrangère.

```
CREATE TABLE FPJ
(NCmd VARCHAR2(4),
 NP VARCHAR2(4),
 Qte NUMBER(5),
 CONSTRAINT pk_LCMD PRIMARY KEY (NCmd, NP),
 CONSTRAINT fk_NP FOREIGN KEY (NP) REFERENCES Produit(NP));
```

7-4-5- Insertion de lignes à la création

Il est possible de créer une table en insérant directement des lignes lors de la création. Les lignes à insérer peuvent être alors récupérées d'une table existante grâce au prédicat **AS SELECT**.

Syntaxe :

```
CREATE TABLE nom_table
 (Attr1 Type [Définition de Contrainte],
  Attr2 Type [Définition de Contrainte],
 ...)
AS SELECT Attr1, Attr2, ...
FROM nom_table2
WHERE Condition;
```

7-4-6- Création d'index

La création d'un index permet d'accélérer les recherches d'informations dans la base. La ligne est retrouvée instantanément si la recherche peut utiliser un index, sinon la recherche se fait séquentiellement. Une autre utilité de la création d'index est de garantir l'unicité de la clé en utilisant l'option **UNIQUE**.

Syntaxe :

```
CREATE [UNIQUE] INDEX nom_index
ON nom_table (Attr1[ASC/DESC], Attr2[ASC/DESC], …);
```

- L'option **UNIQUE** permet de définir la présence ou non de doublons pour les valeurs de l'attribut.
- Les options **ASC/DESC** permettent de définir un ordre de classement des valeurs présentes dans l'attribut.

7-4-7- Modification de la structure d'une table

La clause **ALTER** permet l'ajout de nouveaux attributs, la modification des attributs ou la suppression des attributs d'une table.

❶ *Ajout d'un attribut* : Permet d'ajouter un attribut à la structure initiale de la table.

Syntaxe :

```
ALTER TABLE Nom_Table
ADD Attribut Type;
```

Application :

✓ Ajout de l'attribut **Qtes** à la table **Produit**.

```
ALTER TABLE Produit
ADD Qtes NUMBER(5);
```

❷ *Modification d'un attribut* **:** Associée avec la clause **MODIFY**, la clause **ALTER** permet la modification du type de données d'un attribut. On ne peut qu'agrandir la taille d'un attribut.

Syntaxe :

ALTER TABLE Nom_Table
MODIFY Attribut Nouveau_Type;

Application : Modification de la taille de l'attribut LibP à **VARCHAR**(20).
ALTER TABLE Produit
MODIFY LibP **VARCHAR**(20);

❸ *Suppression d'un attribut* : Permet de supprimer un attribut d'une table.

Syntaxe :
ALTER TABLE Nom_Table
DROP COLUMN Attribut ;

Il faut noter que la suppression d'attributs (colonnes) n'est possible que dans le cas où :

- L'attribut ne fait pas partie d'une vue,
- L'attribut ne fait pas partie d'un index,
- L'attribut n'est pas l'objet d'une contrainte d'intégrité.

❹ *Ajout de contrainte* : Permet d'ajouter une contrainte au niveau d'une table.

Syntaxe :
ALTER TABLE Nom_Table
ADD CONSTRAINT Nom_Contrainte Définition_Contrainte;

Applications :

✓ Ajout de la contrainte de **PRIMARY KEY** sur l'attribut **NCmd** de la table **Commande**.
ALTER TABLE Commande
ADD CONSTRAINT pk_NCMD **PRIMARY KEY** (NCmd);

✓ Ajout des contraintes de clés étrangères sur la table **Commande**.
ALTER TABLE Commande
ADD CONSTRAINT fk_NCL **FOREIGN KEY** (NCl) **REFERENCES** Client(NCl);

✓ Ajout des contraintes de clés étrangères sur la table **Ligne_Cmd**.
ALTER TABLE Ligne_Cmd
ADD CONSTRAINT fk_NCMD **FOREIGN KEY** (NCmd) **REFERENCES** Commande(NCmd);

✓ Ajout de la contrainte **CHECK** sur l'attribut **Qte** (**Qte** > 0) de la table **Ligne_Cmd**.
ALTER TABLE Ligne_Cmd
ADD CONSTRAINT ck_QTE **CHECK** (Qte > 0);

❺ *Suppression de contrainte* : Permet de supprimer une contrainte.

Syntaxe :
ALTER TABLE Nom_Table
DROP CONSTRAINT Nom_Contrainte;

❻ *Désactivation d'une contrainte* : Permet de désactiver une contrainte, elle est par défaut active (au moment de sa création).

Syntaxe :
ALTER TABLE Nom_Table
DISABLE CONSTRAINT Nom_Contrainte;

❼ *Activation d'une contrainte* : Permet d'activer une contrainte désactivée.

Syntaxe :
ALTER TABLE Nom_Table
ENABLE CONSTRAINT Nom_Contrainte;

7-4-8- Suppression d'une table / un index

La clause **DROP** permet d'éliminer des vues, des index et même des tables. Cette clause est toutefois à utiliser avec précaution dans la mesure où elle est irréversible.

Syntaxe :

- Pour supprimer un index : **DROP INDEX** Nom_Index [**ON** Nom_Table];
- Pour supprimer une table : **DROP TABLE** Nom_table ;

NB **:**

- Le nom de la table est obligatoire si on veut supprimer un index d'une table d'un autre utilisateur.
- Un index est automatiquement supprimé dès qu'on supprime la table à laquelle il appartient.

7-5- Langage de Contrôle de Données

Plusieurs personnes peuvent travailler simultanément sur une base de données, toutefois ces personnes n'ont pas forcément les mêmes besoins : certaines peuvent par exemple nécessiter de modifier des données dans la table, tandis que les autres ne l'utiliseront que pour la consulter. Ainsi, il est possible de définir des permissions pour chaque personne en leur octroyant un mot de passe. Cette tâche incombe à l'administrateur de la base de données (en anglais *DBA*, *DataBase Administrator*). Il doit dans un premier temps définir les besoins de chacun, puis les appliquer à la base de donnée sous forme de permissions. Le langage SQL permet d'effectuer ces opérations grâce à deux clauses :

- **GRANT** permet d'accorder des droits à un (parfois plusieurs sur certains SGBD) utilisateur
- **REVOKE** permet de retirer des droits à un (ou plusieurs sur certains SGBD) utilisateur

Les permissions (appelées aussi droits ou privilèges) peuvent être définies pour chaque (un grand nombre) clause. D'autre part il est aussi possible de définir des rôles c'est-à-dire de permettre à d'autres utilisateurs d'accorder des permissions.

❶ *GRANT* : Permet au propriétaire d'une table ou vue de donner à d'autres utilisateurs des droits d'accès à celles ci.

Syntaxe :
GRANT Liste_Privilège **ON** Table/ Vue **TO** Utilisateur [**WITH GRANT OPTION**];

Les privilèges sont :

SELECT	Droit de lecture
INSERT	Droit d'insertion de lignes
UPDATE	Droit de modification de lignes
UPDATE (Attr1**,** Attr2**, …)**	Droit de modification de lignes limité à certains attributs
DELETE	Droit de suppression de lignes
ALTER	Droit de modification de la structure de la table
INDEX	Droit de création d'index
ALL	Tous les droits

Application :
GRANT SELECT ON Produit **TO** User1;

❷ *REVOKE* : Un utilisateur ayant accordé un privilège peut l'annuler à l'aide de la commande **REVOKE**.

Syntaxe :
REVOKE Liste_Privilège **ON** Table/Vue **FROM** Utilisateur;
Application :
REVOKE SELECT ON Produit **FROM** User1;

NB **:** Si on enlève un privilège à un utilisateur, ce même privilège est automatiquement retiré à tout autre utilisateur à qui il l'aurait accordé.

Exercices

Exercice 1 : Questions à choix multiples

1. Quelle clause SQL permet d'effectuer un tri sur les données sélectionnées ?

A	WHERE
B	ORDER BY
C	GROUP BY

2. A quoi sert une clé primaire ?

A	Rendre un champ non modifiable
B	Identifier chaque enregistrement de manière unique
C	Verrouiller la base de données

3. Dans une requête SQL, quel est le sens de l'expression **WHERE Nom LIKE 'Be_'** ?

A	Tous les noms qui commencent par Be
B	Tous les noms qui ne commencent pas par Be
C	Tous les noms de 3 caractères qui commencent par Be

4. La fonction **COUNT** permet-elle de réaliser le total des valeurs d'un champ numérique ?

A	OUI
B	NON

5. Dans une requête SQL **CHECK**

A	permet de rechercher un attribut ayant une valeur donnée.
B	est une contrainte associée à une condition qui doit être vérifiée par toutes les valeurs de l'attribut.
C	permet de rechercher un attribut clé ayant une valeur donnée.

6. Pami les requêtes SQL suivantes, quelles sont celles qui donnent le nombre de tuples dans la relation R (AC, A2, A3) ?

A	SELECT COUNT (*) FROM R ;
B	SELECT SUM (*) FROM R ;
C	SELECT COUNT (AC) FROM R ;

7. Pami les requêtes SQL suivantes, quelles sont celles qui donnent la somme des valeurs de l'attribut A2 de la relation R (AC, A2, A3) ?

A	SELECT AC , SOMME (*) FROM R GROUP BY AC ;
B	SELECT SUM (*) FROM R ;
C	SELECT SUM (A2) FROM R ;

8. La requête suivante appliquée sur la relation R (AC, A2, A3) :

   SELECT COUNT (*) FROM R GROUB BY AC;

A	donne le nombre de tuples
B	donne la somme des tuples
C	ne donne rien

9. Si on applique la requête suivante sur la relation Ligne_Cmd (NCmd, NP, Qte) avec les données ci-dessous quel sera le résultat :

   SELECT NCmd, COUNT (*) FROM Ligne_Cmd WHERE Qte > 100 GROUB BY NCmd Having SUM (Qte) > 600;

Ligne_Cmd

NCmd	NP	Qte
C001	P001	250
C001	P004	300
C001	P006	100
C002	P002	200
C002	P007	550
C003	P001	50
C004	P002	100
C004	P004	150
C004	P005	70
C004	P008	90
C005	P001	650
C005	P002	100

A

NCmd	COUNT
C001	3
C002	2
C005	2

B

NCmd	COUNT
C002	2
C005	1

C

NCmd	COUNT
C001	3
C002	2
C004	2
C005	2

10. Si on applique la requête suivante sur la relation avec les données de la question précédente, quel sera le résultat :

    SELECT COUNT (*) FROM Ligne_Cmd WHERE Qte > 100;

A

COUNT
12

B

COUNT
9

C

COUNT
6

Exercice 2 :

1. Quel est le résultat de la requête suivante :

   SELECT NoDep, SUM (Salaire) FROM Employes WHERE (Annee > 1980) GROUP BY NoDep HAVING SUM (Salaire) > 1000.000 ORDER BY 2 ;

2. Détecter l'erreur qui existe dans la requête suivante :

   ALTER TABLE Employes CHECK (Annee > 1950);

Exercice 3 :

La table CDA repréesente un Calendrier de Dates d'Anniversaire. La table MARIAGES stocke les mariages en vigueur; un homme ou une femme ne peut avoir plusieurs conjoints en même temps.

On maintient l'anniversaire et l'adresse de tous les mariés. On apprend que Tante Odette est née le 27 juin 1936. Elle est mariée avec Oncle Urbain depuis le 1 mai 1950.

CDA

Nom	Anniversaire	Année	Adresse	Ville
Tante Odette	27 juin	1936	17 Rue R. Barre	Mons
Oncle Urbain	27 juin	1927	17 Rue R. Barre	Mons
Mon chat	17 mars	2001	Chez moi	Enghien
Jean Bidon	23 mai	1963	36 Rue d'Egmont	Bruxelles
Anne Lalo	15 mars	1965	35 Rue d'Egmont	Mons
Jean Crevette	12 janvier	1965	23 Place du Parc	Mons

MARIAGES

Femme	Mari	Jour	Année
Tante Odette	Oncle Urbain	1 mai	1950
Anne Lalo	Jean Crevette	14 juillet	1978

1. Soit la requête Q1 suivante :

```
SELECT Femme
FROM   MARIAGES M
WHERE Mari IN (SELECT Nom FROM CDA WEHRE Adresse = M.Adresse);
```

   a. Que donne la requête Q1 ?
   b. Donner le résulat de cette requête.

2. Soit la requête Q2 suivante :

```
SELECT Nom
FROM   CDA
WHERE Nom Not IN (SELECT Femme FROM MARIAGES UNION
                  SELECT Mari     FROM MARIAGES);
```

   a. Que donne la requête Q2 ?
   b. Donner le résulat de cette requête.

Exercice 4 :

Soit la base de données décrite par les trois relations suivantes :

Film (CodeFilm, Titre, Année, PaysProd, Réalisateur, Genre)
Acteur (CodeActeur, Nom, Pays)
Jouer (#CodeActeur, #CodeFilm, Salaire)

Les attributs ont la signification suivante :

CodeFilm	: Numéro d'un film	**Genre**	: Genre du film (aventure, horreur, …)
Titre	: Titre du film	**CodeActeur**	: Numéro d'un acteur
Année	: Année de sortie du film	**Nom**	: Nom de l'acteur
PaysProd	: Pays du producteur du film	**Pays**	: Nationalité de l'acteur
Réalisateur	: Nom du réalisateur du film	**Salaire**	: Salaire perçu par un acteur dans un film

Formuler chacune des requêtes suivantes en SQL :

1) LDD

a) En respectant les contraintes d'intégrités référentielles et les contraintes suivantes, créer la table **Jouer** :
- Le CodeActeur est un entier compris entre 1000 et 9000.

- Le CodeFilm est un nombre sur 3 chiffres qui doit être obligatoirement défini.

b) Dans la table Acteur, ajouter l'attribut Sexe : un caractère qui désigne le sexe de l'acteur ('F' pour Féminin ou 'M' pour Masculin).

2) LMD (les attributs entre crochets représentent les attributs résultats)

a) Retrouver les films [Titre] d'horreur sortis après 1950.

b) Retrouver les noms des acteurs anglo-saxons (anglais et américains).

c) Quelles sont les actrices (sexe féminin) des films d'horreur ?

d) Donner les pays d'origine des acteurs qui ont joué dans des films policiers ou des films de guerre.

e) Donner la liste des films [Titre] joués par les acteurs français.

f) Donner la liste des acteurs [Nom] réalisateurs.

g) Quels sont les acteurs [Nom] qui jouent dans des films joués par l'acteur X ?

h) Donner le salaire maximum des acteurs anglais.

i) Donner le nombre de films tournés par acteur [CodeActeur, Nom, Nombre de films].

j) Retrouver pour chaque film les noms des acteurs ayant joué dans ce film [Titre, Nom].

k) Retrouver les noms des acteurs et des réalisateurs sous la direction desquels ils ont joué [Réalisateur, Nom].

l) Retrouver les noms des acteurs qui ne sont pas français.

m) Quels sont les acteurs [Nom] qui ont joué dans des films produits par des pays autres que les leurs ?

n) Donner le nombre de films tournés par les acteurs américains.

o) Donner le salaire total des acteurs français.

p) Donner les acteurs qui ont participé à des films français et anglais.

Exercice 5 :

Soit la base de données décrite par les relations suivantes :

RESTAURANT	(REST, NOMR, ADR, TEL)
CONSOMMATEUR	(CONS, NOMC)
PLAT	(PLAT, NOMP, PRIX)
FREQUENTE	(#CONS, #REST)
SERVICE	(#REST, #PLAT)
PREFERE	(#CONS, #PLAT)

Les relations RESTAURANT, CONSOMMATEUR et PLAT fournissent respectivement les données relatives à un restaurant, un consommateur et un plat du système étudié.

La relation FREQUENTE indique les restaurants que chaque consommateur visite. L'attribut CONS désigne le numéro d'un consommateur. L'attribut REST désigne le numéro d'un restaurant.

La relation SERVICE fournit les plats servis par chaque restaurant. L'attribut PLAT désigne le numéro d'un plat.

La relation PREFERE donne les plats préférés par un consommateur.

Les clés primaires de chaque relation sont soulignées dans le schéma relationnel de la base fournie ci-dessus.

Formuler chacune des requêtes suivantes en SQL :

1) LDD

a) En respectant les contraintes d'intégrités référentielles et les contraintes suivantes, créer la table FREQUENTE :

- Le numéro d'un consommateur est un entier compris entre 1000 et 9000.
- Le numéro d'un restaurant est un nombre sur deux chiffres qui doit être obligatoirement défini.
- La clé primaire de cette relation est composée des attributs CONS et REST.

b) Ajouter l'attribut NUMJ un entier compris entre 1 et 7 à la relation SERVICE. Le NUMJ décrit le numéro du jour de la semaine où le plat peut être servi par le restaurateur.

2) LMD

a) Donner la liste des numéros des consommateurs qui fréquentent le restaurant 'LES DUNES'.

b) Donner la liste des numéros des consommateurs qui fréquentent plus de 10 restaurants.

c) Donner la liste des numéros des restaurants pour lesquels le nombre de jours de service par semaine est supérieur à quatre.

d) Donner les noms des restaurants (NOMR) qui servent des plats que le consommateur 'HABIB' préfère. Proposer deux solutions : une en utilisant les jointures et une autre en utilisant les requêtes imbriquées.

e) Donner la liste des numéros des consommateurs qui préfèrent au moins un plat que le consommateur numéro 1001 préfère. En utilisant l'opérateur ensembliste IN.

f)

a. Donner la liste des numéros des restaurants qui servent le plat 'PAELLA' tous les jours (sachant que le nombre de jours ouvrables est 6).
b. Donner la liste des numéros des restaurants qui servent le plat 'STEAK FRITES' pendant un jour quelconque.
c. En déduire, la liste des numéros des restaurants qui servent le plat 'PAELLA' tous les jours (sachant que le nombre de jours ouvrables est 6) et des 'STEAK FRITES' pendant un jour quelconque.

g) Donner la liste des numéros des restaurants chez lesquels 'HABIB' peut trouver n'importe quel plat préféré à n'importe quel jour de la semaine.

Exercice 6 :

Soit un ensemble de personnes identifiées par un numéro et caractérisées par un nom et un ensemble de banques identifiées par un numéro.

Les banques sont localisées dans une ville. Une personne peut ouvrir un ou plusieurs comptes dans une banque. On connaît le solde de chaque compte. Chaque banque affecte à ses comptes un numéro unique.

La base de données relationnelle résultat est la suivante :

BANQUE (**NoB**, Ville)

PERSONNE (**NoP**, Nom)

COMPTE (**NoC, #NoB**, Solde, #NoP)

On demande d'exprimer en SQL les requêtes suivantes:

1. La liste des numéros de banques de SousseLa liste des comptes, dont le solde est débiteur de plus de 1000 dinars, de la banque 123
3. Le nombre de banques à SousseLa liste des clients des banques de SousseLa liste des banques dans la même ville que la banque 123

Exercice 7 :

Soit la base de données relationnelle suivante :

DEPOT (**NumD**, Demande)

FABRIQUE (**NumF**, Capacité, PrixV, QtéRemise)

TRANSPORT (#**NumD**, #**NumF**, Distance, Quantité)

- NumD : Numéro du dépôt
- Demande : demande d'un dépôt
- NumF : Numéro de la fabrique
- Capacité : capacité de fabrication de la fabrique
- PrixV : Prix de vente adopté par la fabrique
- QtéRemise : Quantité minimale exigé par la fabrique pour faire une remise
- Distance : Distance parcourue pour transporter les produits d'une fabrique à un dépôt
- Quantité : Quantité des produits transportés d'une fabrique à un dépôt

On demande d'exprimer en SQL les requêtes suivantes :

1. Sélectionner la fabrique qui a la plus grande capacité.
2. Sélectionner la fabrique et le dépôt qui sont les plus proches.
3. Sélectionner les dépôts qui ont été livrés par la fabrique 7.
4. Sélectionner les fabriques qui ont effectué plus de 3 transports à l'entrepôt 5
5. Sélectionner les dépôts dont la demande est supérieure à la somme des quantités que reçoit ce dépôt.
6. Sélectionner les dépôts dont la demande est supérieure à la demande moyenne de tous les dépôts.