Ministère de l'Enseignement Supérieur de la Recherche Scientifique
Direction Générale des Etudes Technologiques

SUPPORT DE COURS

Base de données

Natija BOUZIDI
Assistante technologue à ISET Sidi Bouzid

AU : 2009-2010
(Dernière mise à jour : Février 2016)

Avant Propos

Ce support de cours intitulé « Base de données » est à l'intention des étudiants de la deuxième année en Licence Appliqués en Technologies de l'Informatique spécialité Multimédia et Développement Web (MDW) ou Réseaux et Services Informatiques (RSI) de l'Institut Supérieur des Etudes Technologiques de Sidi Bouzid.

Le cours comporte six chapitres qui sont reparties comme suit :

Chapitre n°1 : Introduction aux bases de données

Chapitre n°2 : Modèle Entité/Association

Chapitre n°3 : Modèle Relationnel

Chapitre n°4 : Normalisation d'une base de données relationnelle

Chapitre n°5 : L'Algèbre Relationnelle

Chapitre n°6 : Langage SQL

L'objectif principal est de faire apprendre aux étudiants à concevoir une base de données. J'introduirai dans le Chapitre n°1 une introduction sur les systèmes à fichier, des notions de base d'une base de données et d'un système de gestion d'une base de données ainsi que les différentes phases de conception d'une base de données relationnelles.

Dans un 2^{éme} chapitre j'ai introduirai une démarche de construction d'un modèle Entité/Association, puis dans le chapitre n°3 les concepts de base d'une relation ainsi que les règles de passage d'un modèle Entité/Association à un modèle relationnel.

Le chapitre n°4 sera consacré à la normalisation et les formes normales d'une base de données relationnelle.

Par la suite, on a un chapitre n°5 qui étudie les opérateurs de base de l'algèbre relationnel.

Je terminerai à la fin par le chapitre n°6 qui sera consacré pour le langage SQL.

Chaque cours sera suivi par un TD, les TD de ce module sont préparés dans un fascicule séparé.

Enfin, J'espérai que le présent support aura le mérite d'être un bon support pédagogique pour l'enseignant et un document permettant une concrétisation expérimentale pour l'étudiant.

L'auteur

Natija BOUZIDI

Fiche matière

PRE REQUIS

- Notion de fichier et Notion de variables

OBJECTIFS GENERAUX

A la fin de ce module, l'étudiant doit être capable de :

- Découvrir et comprendre l'ensemble des concepts sous-jacents aux bases de données.
- Approfondir les concepts de modélisation, de conception et d'implémentation de BD.
- Analyser une étude de cas donné afin de dégager le modèle entités/associations et le modèle relationnel associé.
- Concevoir une base de données en respectant les règles et les normes des modèles de données.
- Etudier l'algèbre relationnel à travers différents opérateurs (spécifiques et ensemblistes)
- Appliquer les opérations de l'algèbre relationnelle pour interroger une base.
- Apprendre à utiliser un langage normalisé d'accès aux données (SQL).

POPULATION

- Profil : Licence
- Spécialité : Technologies de l'informatique.
- Niveau : L2(S3)
- Option : MDW (Multimédia et Développement Web) et RSI (Réseaux et Services Informatiques)

DEROULEMENT

- Volume horaire : 1h 30 de cours intégré /semaine
- Durée : 15 semaines

EVALUATION

- Tests
- Interrogation Orale
- Devoir de contrôle et devoir de synthèse

MOYEN PEDAGOGIQUE

- Tableau
- Polycopiés de Travaux Dirigés

Table des matières

Liste des figures

Chapitre n° 1 : Introduction aux bases de données

Objectifs spécifiques

A la fin de ce chapitre, l'étudiant doit être capable de :

- Présenter les différents besoins ainsi que les différents moyens pour étudier les données.
- Définir une base de données.
- Expliquer l'intérêt de l'approche base de données par rapport à celle à fichiers.
- Définir un Système de Gestion de Bases de Données (SGBD)
- Enumérer les fonctions d'un SGBD
- Différencier entre les niveaux d'abstraction lors de la conception d'une base de données.

Plan du chapitre

I. Introduction

II. Les besoins

III. Les moyens

IV. Approche classique

V. Introduction aux bases de données

VI. Introduction aux SGBD

VII. Architecture d'un SGBD

Volume horaire :

3 heures

Introduction

La prise de décision est une part importante de la vie d'une entreprise (établissements d'enseignement, ministères, banques, agences de voyages, transport, santé, …). Elle nécessite d'être bien informé sur la situation et donc d'avoir des informations à jour et disponibles immédiatement.

Avant les bases de données, chaque application écrite pour un organisme travaillait avec ses propres fichiers : une même information, par exemple le numéro de téléphone d'un client, est alors enregistré dans plusieurs fichiers différents. Ceci cause des délais de mise à jour, peut amener les diverses applications à travaillait sur des données contradictoire, et multiple les possibilités de mises à jour.

Au contraire, quand la gestion des données est centralisée, chaque donnée n'est enregistrée qu'en un seul endroit de la base. Ce qui diminue :

- Les risques d'erreurs de mise à jour,
- Supprime le problème d'avoir des informations contradictoires sur une même donnée dans des fichiers différents.

Les SGBD (Système de Gestion des Bases de Données) évolués offrent des instructions très puissantes pour traiter les données. Les SGBD récents permettent l'interrogation de la base de données par des utilisateurs non informaticiens dans des langages non procéduraux.

I. Les systèmes à fichiers

I.1. Définition

Un fichier est un « récipient d'information, constituant une mémoire secondaire idéale, permettant d'écrire des programmes d'application indépendantes des mémoires secondaires »[1]. Qui se caractérise par un nom, un créateur, une date de création, un contenu, un emplacement en mémoire secondaire et une organisation (séquentielle, indexée, séquentielle indexée, …).

I.2. Différents traitements des fichiers

Le fichier est manipulé à travers un ensemble de commandes :

- Créer : le fichier est créé sans données.
- Supprimer : le fichier inutile est supprimé pour libérer de l'espace sur le support.
- Ouvrir : un fichier doit être ouvert avant qu'un programme puisse l'utiliser.

[1] Selon G. GARDARIN

- Fermer : lorsqu'il y a plus d'accès au fichier, il doit être fermé.
- Lire : les données sont lues à partir du fichier. En général, les données sont lues à partir de la position courante.
- Ecrire : les données sont écrites dans le fichier à partir de position courante.
- Ajouter : ajouté des données à la fin du fichier.
- Positionner : il faut indiquer la position des données à lire ou à écrire.

Exemple :

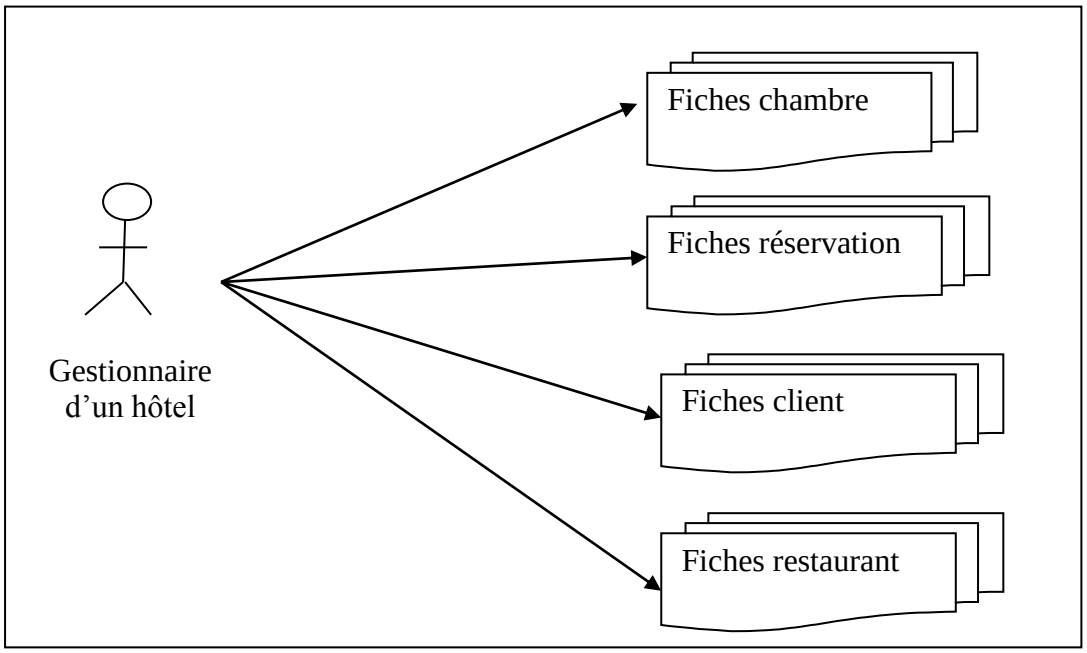


Figure 1.1 : Organisation des données en fichiers

I.3. Limites de l'utilisation des fichiers

Les systèmes à fichiers soulèvent certaines limites à savoir :

- Ecrire des programmes pour pouvoir manipuler ces informations.
- Les données et les traitements sont logiquement et physiquement liés.
- Pour des applications nouvelles, l'utilisateur devra obligatoirement écrire de nouveaux programmes et pourra être amené à créer de nouveaux fichiers qui contiendront peut-être des informations déjà présentes dans d'autres fichiers.
- Les applications manipulées sont aussi rigides, contraignantes, longues et coûteuses à mettre en œuvre.
- Particularisation de la saisie et des traitements en fonction des fichiers => un ou plusieurs programmes par fichier.

Exemple :

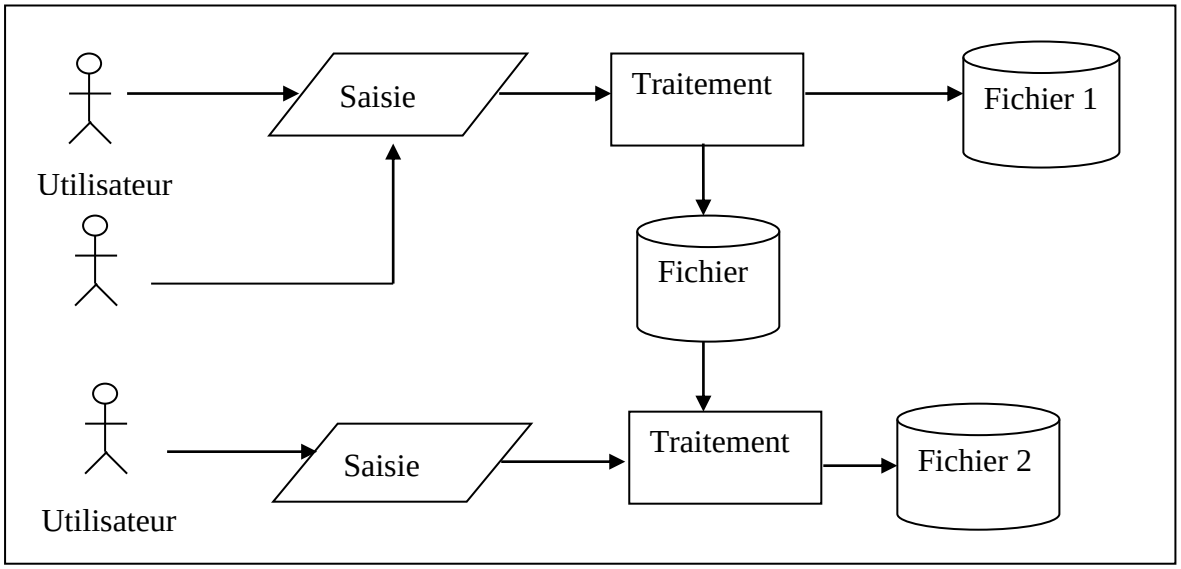


Figure 1.2 : Exemple d'une utilisation de fichiers

II. Les bases de données

III.1. Définition

Une base de données est un ensemble organisé d'informations avec un objectif commun.

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

C'est une collection de données cohérentes entre elles, généralement de taille importante.

Donc, une base de données nécessite :

- ✓ Un espace de stockage.
- ✓ Une structuration de relations entres les données (sémantique).
- ✓ Un logiciel permettant l'accès aux données stockées pour la recherche et la mise à jour de l'information.

Exemple :

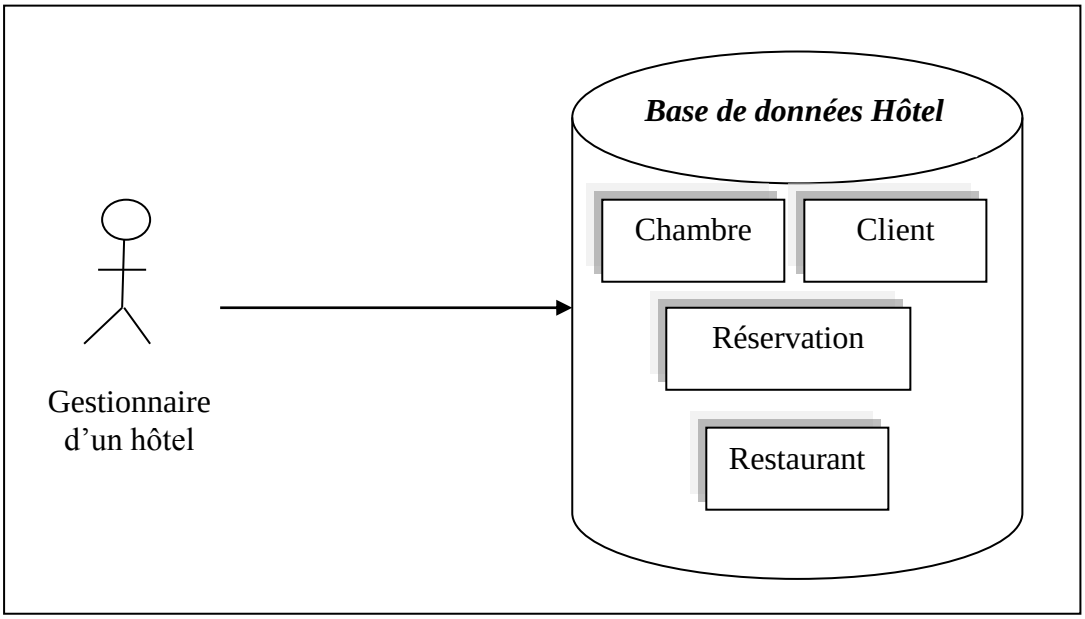


Figure 1.3 : Organisation des données en base de données

III.2. Modèles de base de données

III.2.1. Hiérarchique

Une base de données hiérarchique est une forme de SGBD[2] qui lie des enregistrements dans une structure arborescente de façon à ce que chaque enregistrement n'ait qu'un seul possesseur (par exemple, une paire de chaussures n'appartient qu'à une seule personne). Ce modèle utilise des pointeurs entre les différents enregistrements. Il s'agit du premier modèle de SGBD.

III.2.2. Réseau

Comme le modèle hiérarchique ce modèle utilise des pointeurs vers des enregistrements. Toutefois la structure n'est plus forcément arborescente dans le sens descendant.

Est en mesure de lever de nombreuses difficultés du modèle hiérarchique grâce à la possibilité d'établir des liaisons de type n-n, les liens entre objets pouvant exister sans restriction.

Pour retrouver une donnée dans une telle modélisation, il faut connaître le chemin d'accès (les liens) ce qui rend les programmes dépendants de la structure de données.

[2] SGBD : Système de Gestion de Bases de Données

III.2.3. Relationnel

Est une base de données structurée suivant les principes de l'algèbre relationnel. Le père de bases de données relationnelles (BDR) est Edgar Frank Codd[3].

Un premier prototype de SGBDR (SGBD Relationnel) a été construit dans les laboratoires d'IBM.

Depuis 1980, cette technologie a mûri et a été adoptée par l'industrie. En 1987, le langage SQL, qui étend l'algèbre relationnel, a été standardisé.

Les données sont enregistrées dans des tableaux à deux dimensions (lignes et colonnes). La manipulation de ces données se fait selon la théorie mathématique des relations.

III.2.4. Objet

Les données sont stockées sous forme d'objets, c'est-à-dire de structures appelées classes présentant des données membres. Les champs sont des instances de ces classes.

La notion de base de données Objet ou relationnel – Objet est plus récente et encore en phase de recherche et de développement.

III.3. Historiques des bases de données

- Années 1960 : approche fichiers de données
- Années 1970 : premiers SGBD commerciaux (modèles hiérarchique, réseau)
  - ✓ couplage encore assez fort entre données et programmes
  - ✓ difficulté d'accès aux données
- Années 1980 : premiers SGBD relationnels
  - ✓ modèle fondé sur une théorie mathématique
  - ✓ langage déclaratif d'accès aux données, "simple" à utiliser
- Années 1990 : premiers SGBD orientés-objets (OO)
  - ✓ intégration de types de données plus divers
- Années 2000
  - ✓ Intégration au relationnel des points forts de l'OO, gestion de données complexes, devenant une réalité
- Aujourd'hui Données plus variées (textes, sons, images, parole, ..),
  - ✓ Bases de Données réparties,
  - ✓ Bases de Données orientées objets,

[3] Edgar Frank Codd : chercheur chez IBM à la fin des années 1960

- ✓ Bases de Connaissances et Systèmes Experts,
- ✓ Bases de données déductives,
- ✓ Génie Logiciel et SGBD,
- ✓ Accès intelligent multimodal et naturel (langage naturel écrit, graphique, parole, etc.).

III.4. Organisation base de données

Exemple :

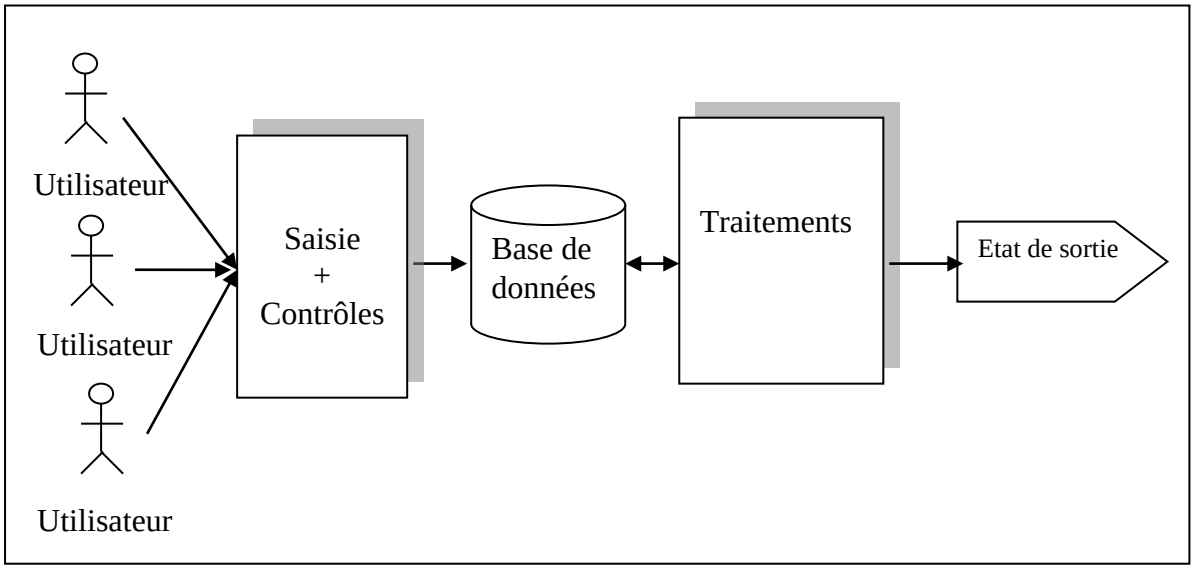


Figure 1.4 : Exemple d'une utilisation de base de données

- Uniformisation de la saisie et standardisation des traitements (ex. tous les résultats de consultation sous forme de listes et de tableaux)
- Contrôle immédiat de la validité des données.
- Partage de données entre plusieurs traitements => limitation de la redondance des données.

III.5. Qui intervient sur une base de données

On distingue essentiellement cinq types différents d'utilisateurs bases de données, qui interviennent de compétences très diverses, à savoir :

III.5.1. Administrateurs de bases de données

L'administrateur a la responsabilité du fonctionnement général du SGBD. Il crée les ressources (bases, comptes) à la demande et attribue les droits d'accès. Ils vérifient régulièrement que les ressources sont suffisantes (taille du disque, puissance machine), repère

les failles de sécurité, etc. Il gère les accès à la base de données, les droits des utilisateurs, les sauvegardes, les restaurations, etc.

III.5.2. Les consultants/analystes

Ils prennent en charge la première étape qui consiste en l'analyse des activités et des flux d'information mis en jeu dans le monde réel à modéliser.

III.5.3. Concepteurs de bases de données

Ce sont les personnes qui s'occupent de traduire le modèle conceptuel en un modèle logique exploitable par le SGBD. Le concepteur est un spécialiste des bases de données qui préparent les structures de données, les vues, les schémas d'accès, etc.

III.5.4. Développeurs d'application

Les développeurs définissent les schémas externes et développent les programmes qui alimentent ou exploitent la base de données en vue d'applications particulières. Ils utilisent pour cela le langage de bases de données du SGBD (SQL, PL/SQL, …) éventuellement couplés avec un langage de programmation classique (java, C#, PHP, …).

III.5.5. Utilisateurs finaux

Ils manipulent la base de données. Il est possible de distinguer des familles d'utilisateurs avec des droits différents vis-à-vis de l'accès à la base. On suppose qu'ils n'ont aucune connaissance sur les bases de données.

III. Les systèmes de gestion des bases de données (SGBD)

IV.1. Définition

Un Système de Gestion de Bases de Données (SGBD) : est un logiciel assurant structuration, stockage, maintenance, mise à jour et consultation des données d'une BD.
C'est un logiciel qui permet à des utilisateurs de définir, créer, mettre à jour une base de données et d'en contrôler l'accès.

C'est un outil qui agit comme interface entre la base de données et l'utilisateur. Il fournit les moyens pour définir, contrôler, mémoriser, manipuler et traiter les données tout en

assurant la sécurité, l'intégrité et la confidentialité indispensables dans un environnement multi-utilisateurs.

Actuellement, la plupart des SGBD fonctionne selon un mode Client/serveur.

On distingue essentiellement trois fonctions des SGBD à savoir :

- Description des données : Langage de Définition de Données (LDD).
- Recherche, manipulation et mise à jour de données : Langage de Manipulation de Données (LMD).
- Contrôle de l'intégrité et sécurité de données : Langage de Contrôle de Données (LCD).

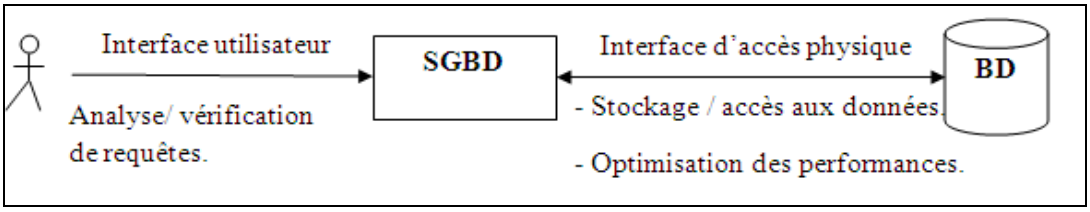


Figure 1.5 : Structure de fonctionnement d'un SGBD

IV.2. Objectifs des SGBD

Les bases de données et les systèmes de gestion de bases de données ont été créés pour répondre à un certain nombre de besoins et pour résoudre un certain nombre de problèmes. Des objectifs principaux ont été fixés aux systèmes de gestion de bases de données dès l'origine de ceux-ci et ce, afin de résoudre les problèmes causés par la démarche classique. Ces objectifs sont les suivants:

- **Indépendance physique:** La façon dont les données sont définies doit être indépendante des structures de stockages utilisées.
- **Indépendance logique:** Un même ensemble de données peut être vu différemment par des utilisateurs différents. Toutes ces visions personnelles des données doivent être intégrées dans une vision globale.
- **Manipulations des données par des langages non procéduraux :** Des utilisateurs non informaticiens doivent pouvoir manipuler simplement les données, c'est-à-dire les interroger et les mettre à jour sans préciser d'algorithme d'accès.
- **Efficacité des accès aux données:** Ces langages doivent permettre d'obtenir des réponses aux interrogations en un temps «raisonnable». Ils doivent donc être optimisés et, entre autres, il faut un mécanisme permettant de minimiser le nombre d'accès disques. Tout ceci, biensûr, de façon complètement transparente pour l'utilisateur.
- **Non redondance des données:** la redondance d'informations pose différents

problèmes (coût en temps, coût en volume et risque d'incohérence entre les différentes copies). Un des objectifs des bases de données est de contrôler cette redondance, voire de la supprimer, en offrant une gestion unifiée des informations complétée par différentes vues pour des classes d'utilisateurs différents.

- **Cohérence des données:** Un schéma de base de données se compose d'une description des données et de leurs relations ainsi que d'un ensemble de contraintes d'intégrité. Une contrainte d'intégrité est une propriété de l'application à modéliser qui renforce la connaissance que l'on en a. On peut classifier les contraintes d'intégrité, en contraintes structurelles (un employé a un chef et un seul par exemple) et contraintes dynamiques (un salaire ne peut diminuer). Les SGBD commerciaux supportent automatiquement un certain nombre de contraintes structurelles, mais ne prennent pas en compte les contraintes dynamiques (elles doivent être codées dans les programmes d'application).
- **Partage des données:** Il s'agit de permettre à plusieurs utilisateurs d'accéder aux mêmes données au même moment. Si ce problème est simple à résoudre quand il s'agit uniquement d'interrogations et quand on est dans un contexte mono utilisateur, cela n'est plus le cas quand il s'agit de modifications dans un contexte mutli-utilisateurs. Il s'agit alors de pouvoir:
  - ✓ permettre à deux (ou plus) utilisateurs de modifier la même donnée "en même temps"
  - ✓ assurer un résultat d'interrogation cohérent pour un utilisateur consultant une table pendant qu'un autre la modifie.
- **Sécurité des données:** Les données doivent pouvoir être protégées contre les accès non autorisés. Pour cela, il faut pouvoir associer à chaque utilisateur des droits d'accès aux données.
- **Résistance aux pannes:** Que se passe-t-il si une panne survient au milieu d'une modification, si certains fichiers contenant les données deviennent illisibles? Les pannes, bien qu'étant assez rares, se produisent quand même de temps en temps. Il faut pouvoir, lorsque l'une d'elles arrive, récupérer une base dans un état "sain". Ainsi, après une panne intervenant au milieu d'une modification deux solutions sont possibles:
  - ✓ soit récupérer les données dans l'état dans lequel elles étaient avant la modification,
  - ✓ soit terminer l'opération interrompue.

IV. Niveaux d'abstraction

Pour assurer ces objectifs (surtout les deux premiers), trois niveaux de description des données ont été définis par la norme ANSI/SPARC : niveau externe, niveau conceptuel, niveau interne.

L'architecture ANSI/SPARC, datant de 1975, définit des niveaux d'abstraction pour un système de gestion de bases de données :

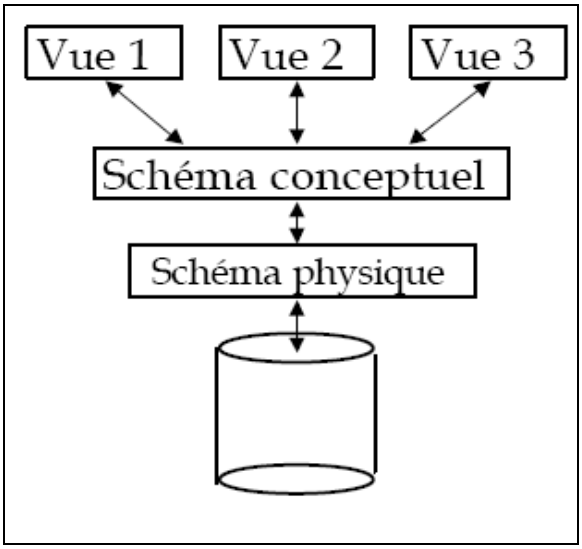


Figure 1.6 : Niveaux d'abstraction : Architecture ANSI/SPARC

V.1. Niveau externe

Ce niveau présente une vue de la description pour chaque utilisateur de sa perception des données.

En effet, il prend en charge le problème du dialogue avec les utilisateurs, c'est-à-dire l'analyse des demandes de l'utilisateur, le contrôle des droits d'accès de l'utilisateur, la présentation des résultats. On appelle cette description le schéma externe ou vue.

V.2. Niveau conceptuel

Il s'agit d'une description de la structure de toutes les données qui existent dans la base, description de leurs propriétés (relations qui existent entre elles) c'est-à-dire de leur sémantique inhérente, sans soucis d'implémentation physique ni de la façon dont chaque groupe de travail voudra s'en servir. On appelle cette description le schéma conceptuel.

V.3. Niveau interne

Ce niveau s'occupe du stockage des données au niveau des unités de stockage, des fichiers,…. On appelle cette description le schéma interne.

V. Fonctions des SGBD

- **Description des données :** Aux niveaux externes, internes, conceptuels par les administrateurs grâce à un Langage de Définition de Données (LDD)
- **Recherche de données :** Interrogation.
- **Mise à jour des données :** Insertion, modification, suppression grâce à un Langage de Manipulation de Données (LMD).
- **Transformation des données :** Ex. Changement de format : date sur 2 chiffres → date sur 4 chiffres.
- **Contrôle de l'intégrité des données :** Exactitude des données stockées dans la base (respect des contraintes d'intégrité).
- **Gestion de transactions :** S'assurer qu'un groupe de mises à jour est totalement exécuté ou pas du tout (atomicité des transactions) ;
- **Sécurité :** Personnalisation des accès à la base (accès par mots de passe).

VI. Processus de conception d'une base de données

- **Analyse du monde réel:** Le monde réel est perçu comme un système qui se traduit par: des classes d'entités, des propriétés sur ces classes et des liaisons entre ces classes.
- **Modélisation conceptuelle:** Les principes généraux à respecter :
  - le schéma conceptuel doit être libre de toute considération non significative du système (organisation physique des données, aspects particuliers à un usager tels que des formats de messages…) ;
  - Tous les aspects du système doivent être décrits dans le schéma conceptuel ; aucun d'eux ne doit intervenir ailleurs en particulier dans des programmes d'application indépendants du schéma conceptuel.
  - ***Caractéristiques :***
    - ✓ prend en compte les aspects statiques et dynamiques du système;
    - ✓ fournit un langage pour communiquer avec un système informatique et avec diverses catégories d'usagers ;
    - ✓ permet de prendre en compte des évolutions ;
    - ✓ est indépendante de tout SGBD.
- **Modélisation logique :** Il traduit le modèle conceptuel dans le modèle du SGBD. Il existe différents types de modèles logiques de SGBD : hiérarchiques, réseau,

relationnels, orientés objet. Certains modèles peuvent être spécifiques à un SGBD.

- **Modélisation physique :** représente les structures de stockage internes et détaille l'organisation des fichiers;

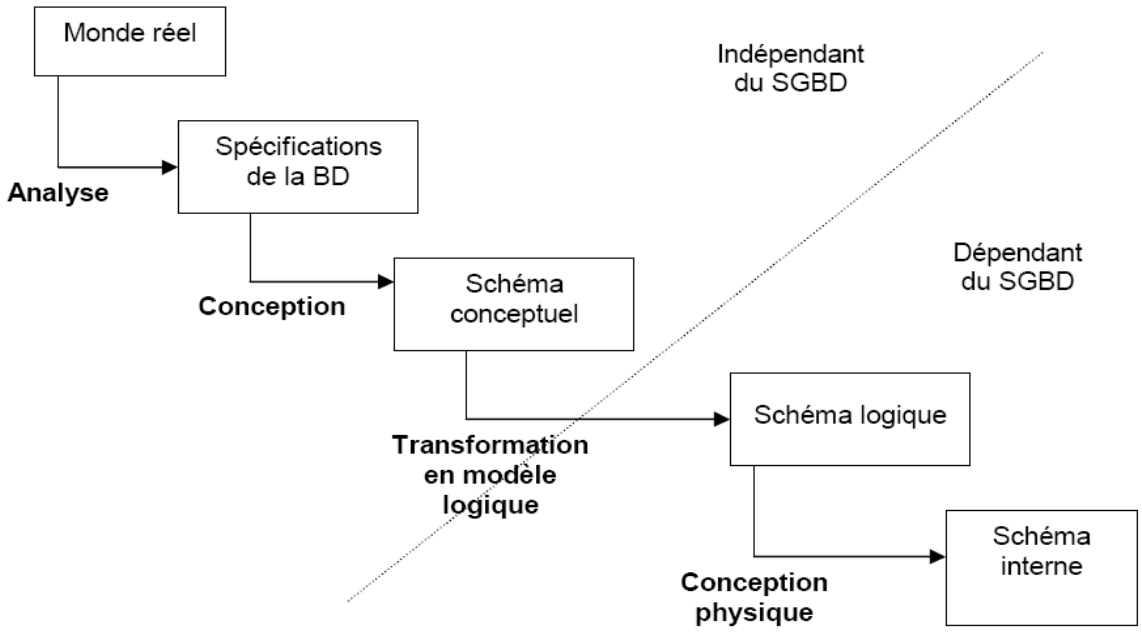


Figure 1.7 : Processus de conception d'une base de données

Chapitre n°2 : Modèle Entités / Associations

Objectifs spécifiques

A la fin de ce chapitre, l'étudiant doit être capable de :

- Assimiler la sémantique du modèle Entités/Associations
- Utiliser le formalisme du modèle Entités/Associations
- Distinguer entre les différents types d'attributs
- Analyser une étude de cas donné
- Modéliser en Entités/Associations

Plan du chapitre

I. Généralités

II. Concepts de base (Entité, Association, cardinalité)

III. Démarche à suivre pour produire un schéma E/A

Volume horaire :

3 heures

I. Généralités

Quand nous construisons directement les tables d'une base de données dans un logiciel de base de données (Oracle, SQLServer, DB2, MS-Access, MySQL, PostGree,…), nous sommes exposés à deux types de problèmes :

- Nous ne savons pas toujours dans quelle table placer certaines colonnes (par exemple : l'adresse de livraison se met dans la table des clients ou dans la table des commandes ?).
- Nous avons du mal à prévoir les tables de jonction intermédiaires (par exemple : la table des interprétations qui est indispensable entre la table des films et la table des acteurs ?).

Il est donc nécessaire de recourir à une étape préliminaire de conception des bases de données en présentant le modèle Entité/Association (E/A) qui est utilisé à peu près universellement pour la modélisation et la conception de bases de données (relationnelles principalement).

Le modèle E/A est un modèle conceptuel conçu en 1976 et qui résulte des travaux de BACHMAN, CHEN, TARDIEU.

Il est essentiellement utilisé pour la phase de conception initiale. Il utilise une représentation graphique.

Il est à la base de la plupart des méthodes de conception. La syntaxe employée ici est celle de la méthode UML, reprise à peu près à l'identique de celle de la méthode OMT.

Il existe beaucoup d'autres notations, dont celle de la méthode MERISE principalement utilisée en France.

Ces notations sont globalement équivalentes. Dans tous les cas la conception repose sur deux concepts complémentaires, entité et association.

La mise en œuvre de la base de données : transformation du schéma E/A en un schéma logique de SGBD.

II. Concepts de base : attribut, entité, association, cardinalité

II.1. Attribut

Un attribut est défini comme étant le champ ou la plus petite unité de données possédant un nom.

Exemples : CIN, nom, prenom, age, matricule, adresse, sexe, date_naissance,…

Propriétés :

On distingue plusieurs types de propriétés spécifiques pour les attributs à savoir :

- *Attribut atomique (simple)* : attribut non décomposable en d'autres attributs.
  Exemples : num_inscrit, nom, CIN, num_telephone, matricule.
- *Attribut composé (complexe)* : la valeur de l'attribut est une concaténation des valeurs de plusieurs attributs simples.
  Exemple : Adresse (Rue, Code postal, Ville).
- *Attribut dérivé* : la valeur de l'attribut est calculée ou déduite à partir des valeurs des autres attributs.
  Exemples : Moyenne, Durée, Age
- *Valeur nulle (NULL)* : pour un attribut, c'est une valeur non définie.
  Exemple : Pour une personne dont on ne connaît pas sa date de naissance, l'attribut date_naissance prend la valeur NULL.

Type d'attribut : Entier, Réel, Date, Chaîne de caractères, …

Domaine d'attribut : Ensemble de valeurs admissibles pour un ou plusieurs attributs.

Exemple : Si le prix des produits est compris entre 15DT et 30DT, alors le domaine de l'attribut prix est [15..30].

II.2. Entité

On désigne par entité tout objet identifiable et pertinent pour l'application. Une entité est similaire à la notion d'objet, elle représente les choses du monde réel.

Un type d'entité permet de définir de façon conceptuelle une entité dont tous les membres partagent les mêmes caractéristiques.

Exemples :

Les produits ou les articles vendus par une entreprise peuvent être regroupés dans une même entité « Articles », car d'un article à l'autre, les informations ne changent pas de nature (à chaque fois, il s'agit de la désignation, du prix unitaire).

Par contre les articles et les clients ne peuvent pas être regroupés : leurs informations ne sont pas homogènes (un article ne possède pas d'adresse et un client ne possède pas de prix unitaire). Il faut donc leur réserver deux entités distinctes : l'entité articles et l'entité clients.

- Une occurrence d'entité est constituée par l'ensemble des valeurs de chacune des propriétés d'un type d'entité.
  Exemple : Type d'entité : Personne, avec les attributs : cin, nom, prenom
  Occurrences:

07427819	Sami	Ben Salem
08216728	Aymen	Jbali
03987432	Ali	Ayari

Identifiant d'une entité :

C'est un ensemble de propriété dont la valeur ne doit pas appartenir à plus d'une entité c.à.d. caractérise de façon unique les occurrences d'un type d'entité.

Exemple :

L'attribut CIN de l'entité Personne : Toute personne a un seul numéro de carte d'identité nationale qui le distingue des autres.

Notation :

Chaque entité est représentée par un rectangle et doit avoir un identifiant qui doit être fixé pour chaque type d'entité pour le distinguer des autres attributs, donc on va le souligner dans le diagramme.

Diagramme E/A :

On prend ici l'exemple d'une entité Personne avec les attributs : cin, nom, prenom

Le digramme entité / association (E/A) associé est :

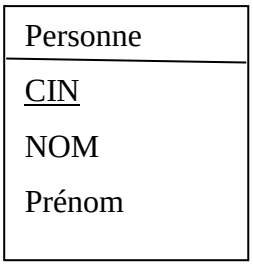


Figure 2.8 : Exemple d'un digramme E/A

II.3. Association

Définition :

Une association est une liaison perçue entre plusieurs entités. Elle présente un lien où chaque entité liée joue un rôle bien déterminé.

C'est une représentation d'un lien non orienté entre plusieurs entités (qui jouent chacune un rôle déterminé).

Exemple : Une personne achète une maison.

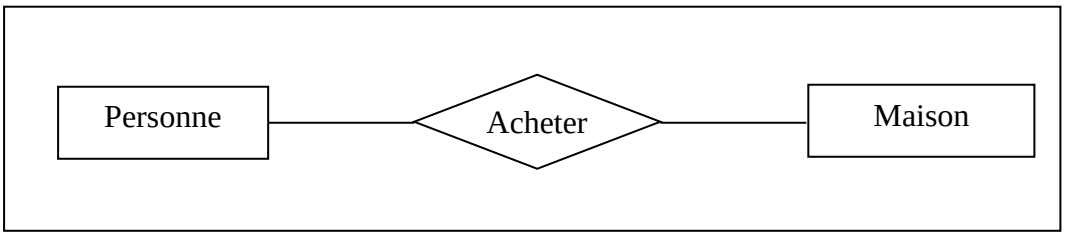


Figure 2.9 : Exemple d'association entre deux entités

Les entités Personne et Maison sont dites participantes à la relation Acheter.

De même on peut trouver des diagrammes de plus de deux entités comme illustre l'exemple suivant :

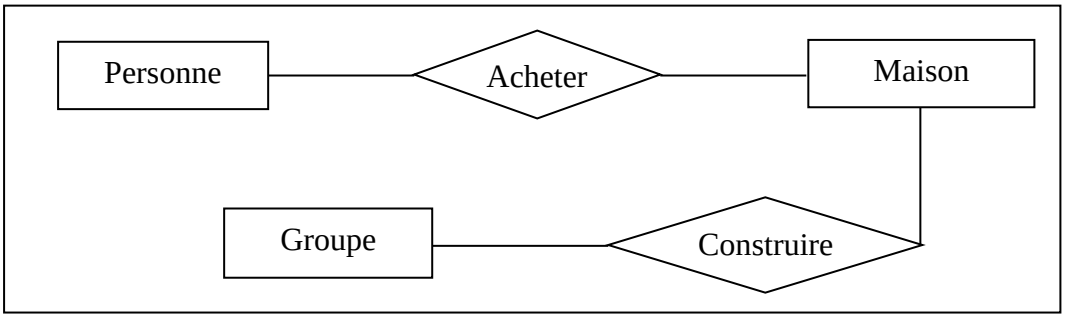


Figure 2.10 : Exemple de diagrammes E/A de trois entités

Remarque : Dans ce schéma, les entités Personne et Groupe ne sont pas liées directement, mais indirectement, via l'entité Maison.

Type d'association :

Un type d'association définie des liens entre des types d'entité, sa valeur peut être une table composé de :

- Une colonne par type d'entité participante à l'association.
- Une ligne pour chaque combinaison d'entité participante à une association.

Exemple :

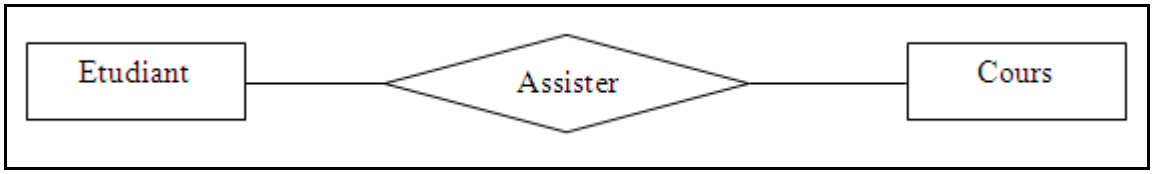


Figure 2.11 : Exemple d'association

La valeur de l'association Assister est :

Etudiant : 1ére entité	***Cours : 2éme entité***
05675432	Algorithmique
03456098	C++
02123987	C++
03654098	Base de données
03456098	Algorithmique

Remarque :

- Une entité possède au moins un attribut (son identifiant).
- Au contraire, une association peut être dépourvue d'attribut (ne contient aucun attribut).

II.4. Cardinalité

Définition :

La cardinalité (ou multiplicité) d'un lien entre une entité et une association précise le minimum et le maximum de fois qu'un individu de l'entité peut être concerné par l'association.

Elle est représentée sous la forme M-N et elle est attachée à une entité et indique les nombres minimum et maximum d'instance d'association pour une instance de cette entité.

Remarque : une cardinalité se lit dans le sens entité vers association.

Types de cardinalités :

Notation :

1 : 1 (une seule fois)
N : Plusieurs fois (0 à N fois)
0-1 : Optionnel (0 ou 1)
1-N : Obligatoire (1 ou plus)
M-N : Limité (entre M et N)

Exemple :

- Association 1-1 : Un client donné ne commande qu'un seul article. Un article donné n'est commandé que par un seul client.

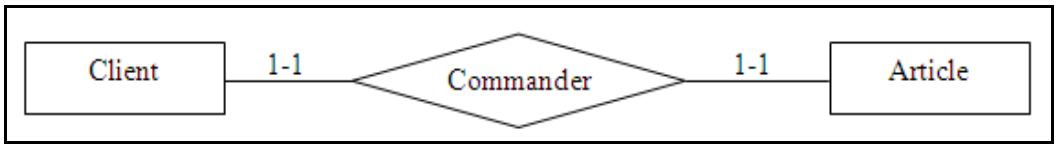


Figure 2.12 : *Exemple d'association de type 1-1*

- Association 0 ou 1-N : Un client donné commande plusieurs articles. Un article donné n'est commandé que par un seul client.

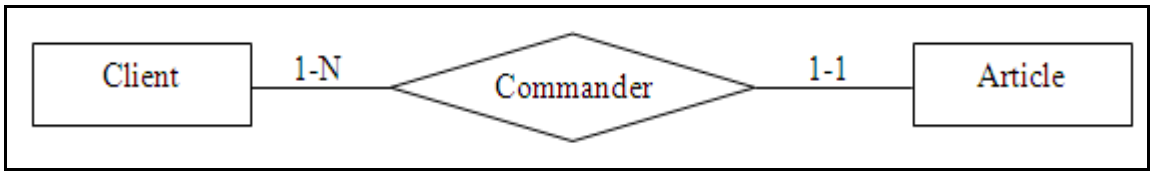


Figure 2.13 : *Exemple d'association de type 1-N*

Remarque : La cardinalité « un à plusieurs » (1-N) peut être aussi « zéro à plusieurs » (0-N) dans le cas où un client existe mais peut ne pas commander d'article.

- Association M-N : Un client donné commande plusieurs articles. Un article a donné est commandé par un ou plusieurs clients.

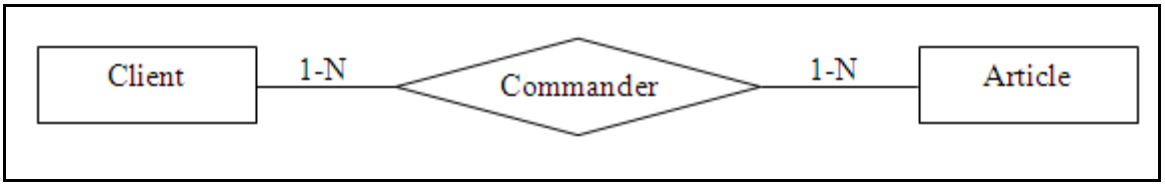


Figure 2.14 : *Exemple d'association de type M-N*

II.5. Association ternaire

Les plus souvent l'association binaire suffisante, mais des associations entre trois types d'entités ou plus peuvent être nécessaire.

Exemple :

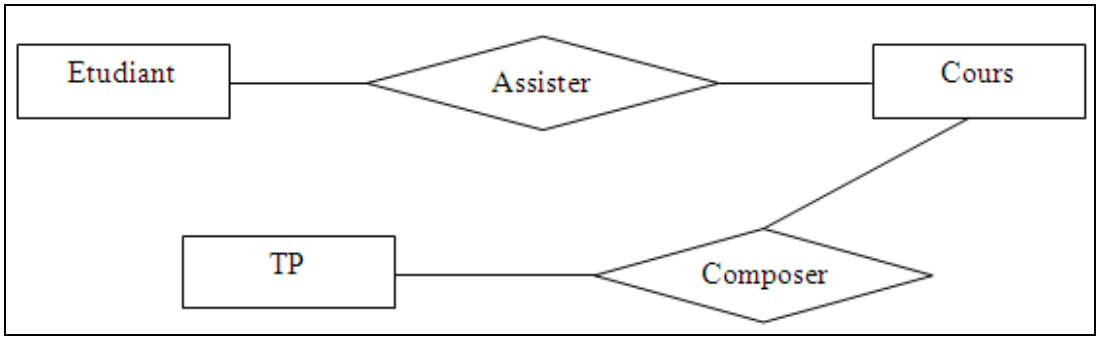


Figure 2.15 : *Exemple de deux associations binaires*

Cette représentation est correcte si chaque étudiant est tenu de participer à toutes les TP de chaque cours.

Question : Comment faire lorsque les étudiants assistent à des TP et ne suivent pas l'intégralité de ces cours.

Réponse : on a besoin d'une association ternaire entre étudiant, cours et TP.

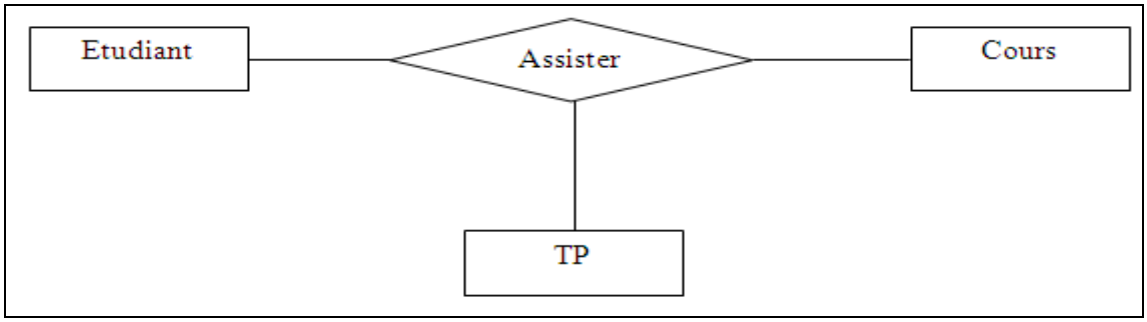


Figure 2.16 : *Exemple d'association ternaire*

II.6. Association réflexive

Il est permis à une association d'être branchée plusieurs fois à la même entité, comme par exemple l'association binaire suivante :

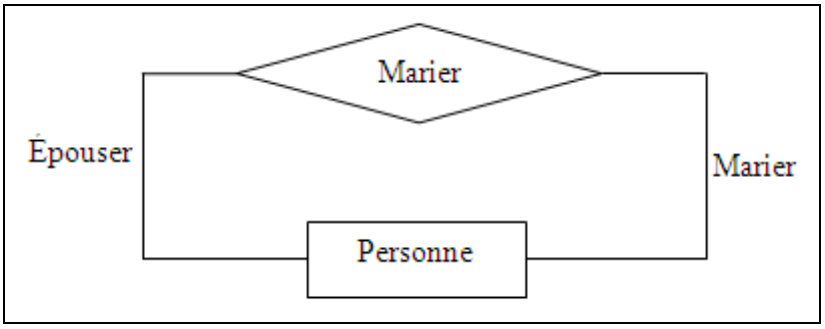


Figure 2.17 : *Exemple d'association réflexive*

Remarque : Dans ce cas on a besoin d'associer des rôles différents pour la même association.

III. Démarche à suivre pour produire un schéma E/A

III.1. Démarche

Afin de pouvoir produire un schéma E/A relatif aux spécifications d'une étude de cas, on procède comme suit :

i- Recueil des besoins et identification des différents attributs.

ii- Déterminer les types d'entité.

iii-Regrouper les attributs par entités.

iv- Identifier les associations entre les entités ainsi que les attributs y associés.

v- Evaluer les cardinalités des associations.

vi- Dessiner le diagramme E/A.

III.2. Conseils divers

Concernant le choix des noms :

- Pour les types entités, choisissez un nom commun décrivant le type entité (ex : Étudiant, Enseignant, Matière). Certain préfèrent mettre le nom au pluriel (ex : Étudiants, Enseignants, Matières). Restez cependant cohérents, soit tous les noms de type entité sont au pluriel, soit ils sont tous au singulier.
- Pour les type association, choisissez un verbe à l'infinitif, éventuellement à la forme passive ou accompagné d'un adverbe (ex : Enseigner, Avoir lieu dans, …).
- Pour les attributs, utilisez un nom commun au singulier éventuellement accompagné du nom du type entité ou du type association dans lequel il se trouve (ex : nom de client, numéro d'article, date de la commande).

Concernant le choix des identifiants des types entités :

- Évitez les identifiants composés de plusieurs attributs (comme, par exemple, un identifiant formé par les attributs *nom* et *prénom* d'un type entité *Personne*) car ils dégradent les performances du SGBD,
- Évitez les identifiants susceptibles de changer au cours du temps (comme numéro du téléphone, email)
- Évitez les identifiants du type chaîne de caractère.

En fait, il est souvent préférable de choisir un identifiant arbitraire de type entier pour les types entités. Cet identifiant deviendra une clé primaire dans le schéma relationnel et le SGBD l'incrémentera automatiquement lors de la création de nouvelles instances.

IV. Exercice d'application : Gestion simplifié de stock

Les clients sont caractérisés par un numéro de client, un nom, un prénom, une date de naissance et une adresse postale (rue, code postal et ville). Ils commandent une quantité donnée des articles à une date donnée.

Les articles sont caractérisés par un numéro d'article, une désignation et un prix unitaire. Chaque article est fourni par un fournisseur unique (mais un fournisseur peut fournir plusieurs articles).

Les fournisseurs sont caractérisés par un numéro de fournisseur, une raison sociale, une adresse email et une adresse postale.

Donnez le modèle E/A correspondant à cette étude de cas.

Correction

Pour construire le modèle entité/Association, on a suivi les différentes étapes présentées par la suite :

1. Dictionnaire de données : Les différents attributs associés à ces spécifications peuvent être résumés comme suit :

N°	***Nom de l'attribut***	***Désignation de l'attribut***	***Type***
1	num_cl	Numéro du client	entier
2	nom_cl	Nom du client	Chaine de caractère
3	prenom_cl	Prénom du client	Chaine de caractère
4	date_nais_cl	Date naissance du client	date
5	adres_cl	Adresse du client	Chaine de caractère
6	qte_art_cmd	Quantité des articles commandés	réel
7	num_art	Numéro de l'article	entier
8	date_cmd	Date de la commande	date
9	design_art	Désignation de l'article	Chaine de caractère
10	prix_unit_art	Prix unitaire de l'article	réel
11	num_frs	Numéro du fournisseur	entier
12	rais_soc_frs	Raison sociale du fournisseur	Chaine de caractère
13	mail_frs	Adresse email du fournisseur	Chaine de caractère
14	adres_frs	Adresse du fournisseur	Chaine de caractère

2. Les différents types d'entité sont : Client, Article, Fournisseur
3. Les entités avec leurs attributs et son identifiant :

Nom de l'entité	***Attribut de l'entité***	***Identifiant de l'entité***
Client	1, 2, 3, 4, 5	1
Article	7, 9, 10	7
Fournisseur	11,12, 13, 14	11

4. Les associations entre les entités :

Nom de l'association	***Entité participante***	***Attributs associés***
Commander	Client et Article	6, 8
Fournir	Fournisseur et Article	

5. Cardinalités :

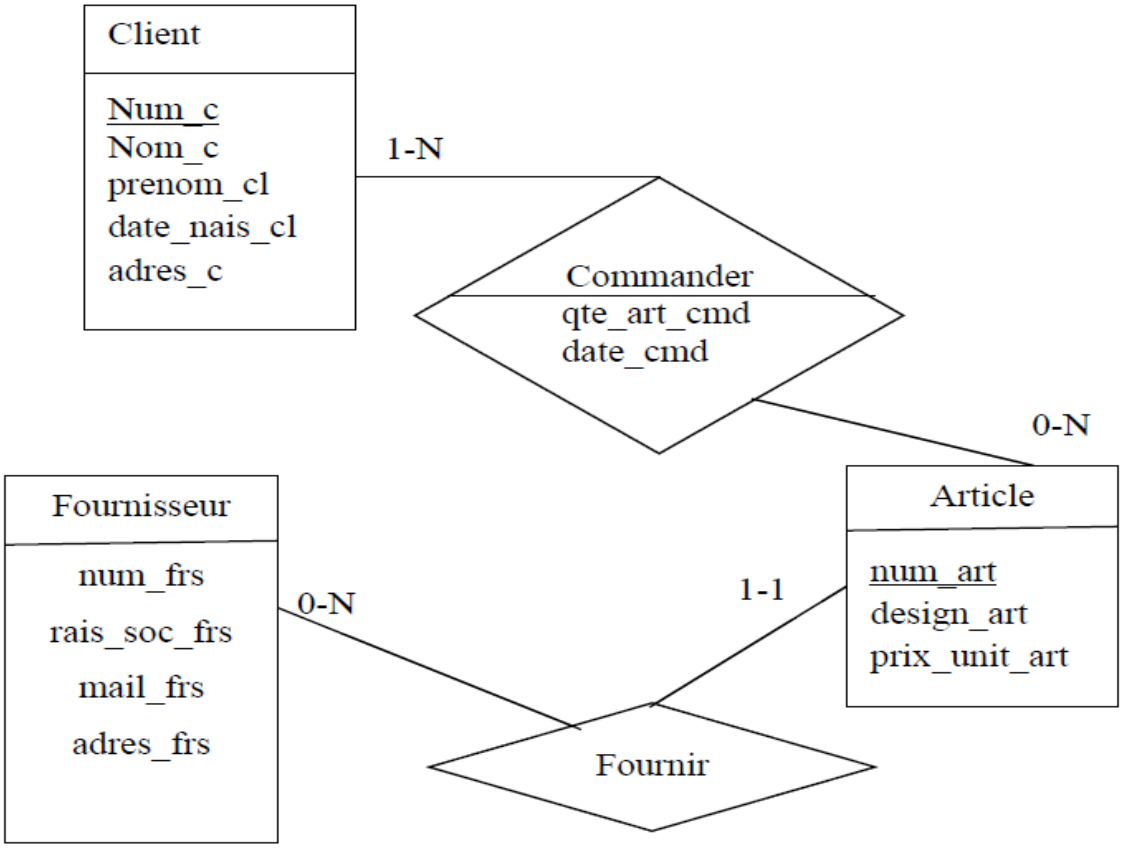
   Client – commander : 1 - N

   Article – commander : 0 - N

   Fournisseur – Fournir : 0 - N

   Article – Fournir : 1 - 1

6. Enfin, le modèle E/A se présente comme suit :



Chapitre n°3 : Modèle relationnel

Objectifs spécifiques

A la fin de ce chapitre, l'étudiant doit être capable de :

- Apprendre les notions de base du modèle relationnel
- Identifier les correspondances avec le modèle E/A
- Traduire un modèle E/A en un modèle relationnel
- Dégager les dépendances fonctionnelles
- Normaliser une relation

Plan du chapitre

I. Généralités

II. Notions de base

III. Traduction E/A – relationnel

Volume horaire

3 heures

I. Généralités

- *Définition :*

Le modèle relationnel est associé aux SGBD relationnels, très simple développé par Codd 1970 de IBM Labs.

C'est un modèle de structuration des informations respecté par les systèmes de gestion de bases de données relationnelles (SGBDR). Dans les SGBDR, les informations sont rangées dans des tables.

Exemples de SGBD: Oracle, DB2, SQLServer, Access, Dbase, …

Une base de données relationnelle est un ensemble de tables relationnelles.

- *Objectifs du modèle relationnel :*
  - ✓ Définir le nombre et la structure des tables de la base de données, en limitant la redondance d'informations et en ayant un schéma de table le plus stable possible.
  - ✓ Indépendance physique : indépendance entre programmes d'application et représentation interne de données.
  - ✓ Développement des LMD non procéduraux : modélisation et manipulation simples de données, langages faciles à utiliser.
  - ✓ Devenir un standard.

II. Notions de base

II.1. Relation

Définition d'une relation:

Une relation est définie par :

- ✓ son nom
- ✓ liste de couples (nom d'attribut : domaine)
- ✓ son (ses) identifiant(s)
- ✓ sa définition (phrase en français)

Les trois premières informations: nom de la relation, liste des couples (attribut : domaine) et identifiant(s) constituent le schéma de la relation.

On appelle schéma d'une base de données relationnelle l'ensemble des schémas de ses relations.

Une relation $\mathbf{R_i}$ est un ensemble d'attributs : **(A1:d1, A2:d2,…, An:dn)** ou plus simplement **(A1, A2,…, An)**.

Avec A : attribut

d : domaine des valeurs des attributs.

R : relation

Notion de domaine :

Un domaine est un ensemble de valeurs que peut prendre un attribut; c'est le domaine définition d'un ou plusieurs attributs.

Exemple de domaines:

Dnom : chaînes de caractères de longueur maximale 30

Dnum : entiers compris entre 0 et 99999

Dcouleur : {"bleu", "vert", "jaune"}

Dâge : entiers compris entre 18 et 27

Dâge : entier > 1

Exemple :

Schéma de la relation Etudiant :

Etudiant (Num : Dnum, Nom : Dnom, Prénom : Dnom, Age : Dâge)

Schéma de la relation Produit:

Produit (Nump, nomp, prixUni)

On peut ajouter que prixUni appartient à]0; 5000] d'où domaine(prixUni) =]0; 5000]

II.2. Tuple

Définition d'un tuple:

Un tuple est un ensemble de valeurs t=<V1, V2, …, Vn> où Vi appartient au domaine di. De même Vi peut aussi prendre la valeur nulle.

Exemple : <2, 'Prod1', 300>

⇨ La population d'une relation est constituée de l'ensemble des tuples de la relation. C'est un ensemble; il n'y a donc ni doubles, ni ordre (les nouveaux tuples sont rajoutés à la fin de la relation).

On appelle schéma d'une base de données relationnelle l'ensemble des schémas de ses relations. On appelle base de données relationnelle, la population de toutes ses relations.

II.3. Règles généraux

Règles de Structures :

- ✓ Un attribut admet une valeur pour chaque tuple selon leur type.
- ✓ On peut trouver un attribut n'admettant pas de valeur pour un tuple : on dit alors qu'il a une valeur nulle.

Exemple : Si on ne connait pas la date de naissance d'un étudiant donc on met la valeur NULL.

Règles d'identification :

- ✓ Toute relation possède un identifiant (une clé)
  - ➔ Il ne peut pas y avoir deux tuples identiques dans la même relation.
- ✓ L'identifiant n'admet pas de valeur nulle.
- ✓ Une relation peut avoir plusieurs identifiants.

  Définition : l'identifiant d'une relation est un ensemble minimum d'attribut de la même relations tel qu'il n'existe pas deux tuples ayant même valeur pour cette identifiant.
- ✓ Tous les attributs de tout identifiant doivent toujours avoir une valeur connue (valeur non nulle).

Représentation d'attribut complexe :

- ✓ Les notions d'attributs complexes n'existent pas dans le modèle relationnel, il faut donc les modéliser autrement : Il faut choisir entre le composé ou le composant.
- ✓ Soit adresse : nom de Rue, Numéro, ville, cp :

  Solution 1 : un attribut par composant :

  Client (num_cl, nom_cl, nom_rue, num_rue, ville, cp)

  Client (1, 'A', 'Rue de Gabes', 5, 'Sfax', 3052)

  Solution 2 : un attribut composé:

  Client (num_cl, nom_cl, adress_cl)

  Client (1, 'A', 'Rue de Gabes 5 Sfax 3052')

Identifiant externe :

Certains attributs référencent des tuples d'une autre relation; c'est à dire que leur valeur est toujours égale à celle de l'identifiant d'un tuple existant dans l'autre relation. On les appelle identifiants externes (ou clés externes ou clés étrangères).

Exemple: Soient les relations suivantes Cours (NomC, …), Etudiant (Num_etud, …) et Suit (#NomC, #Num_etud) (les attributs précédés par le caractère « # » représentent les identifiants externes).

La relation « Suit » possède un identifiant interne représenté par le couple (NomC + Num_etud), et deux identifiants externes : NomC et Num_etud. En effet, NomC "référence" un Cours, c.à.d. que si une valeur NomC existe dans Suit, alors il doit nécessairement exister un cours de ce nom là dans la relation Cours. De même, Num_etud "référence" un Etudiant.

Le schéma d'une relation comprend donc, en plus de la définition du nom de la relation et de ses attributs et domaines associés, la définition de son (ses) identifiant, et celle de ses identifiants externes, s'il en existe.

II.4. Contraintes d'intégrité

Définition :

Une contrainte d'intégrité est une clause permettant de contraindre la modification de tables, faite par l'intermédiaire de requêtes d'utilisateurs, afin que les données saisies dans la base soient conformes aux données attendues.

Les contraintes d'intégrité sont de trois types :

- ✓ Contraintes de la clé primaire.
- ✓ Contraintes référentiels.
- ✓ Contraintes sur les domaines.

- Clé primaire : ensemble d'attributs dont les valeurs permettent de distinguer les tuples les uns des autres (identifiant).

Notation : Pour signaler la clé primaire, ses attributs sont généralement soulignés.

Exemple : Soit la relation Produit (numPd, designPd, puPd) ; L'attribut numPd représente la clé primaire de la relation Produit.

- Contrainte référentiel (Clé étrangère) : sert à la vérification de l'intégrité référentielle. Les identifiants externes désignent nécessairement des tuples existants.
  La clé étrangère est un attribut qui est clé primaire d'une autre relation.

Notation : La clé étrangère doit être précédée par #.

Exemple : Soient le schéma relationnel représenté par les deux relations suivantes : Frs(unmFr, nomFr) et Produit (numPd, designPd, puPd, # numFr)

On considère les deux tuples Frs(F1, Fournisseur1) et Frs(F2, Fournisseur2).

Le tuple Produit (P1, « Produit1 », 100, F2) signifie que le produit de numéro P1 a comme fournisseur numéro F2.

Le tuple Produit (P2, « Produit1 », 100, F1) signifie que le produit de numéro P2 a comme fournisseur numéro F1.

Le tuple Produit (P3, « Produit1 », 100, F3) signifie que le produit de numéro P3 a comme fournisseur numéro F3. Ce tuple n'est pas accepté car la valeur « F3 » de l'identifiant externe numFr n'est pas une valeur de l'identifiant interne numFr.

- Contraintes de domaine : une contrainte de domaine impose qu'une colonne de relation doit comporter des valeurs vérifiant une condition logique.

Exemple : La date de naissance doit être inférieure à la date de système.

Le prix d'un produit doit être entre 100dt et 350dt

L'âge d'une personne doit être supérieur à 1.

….

Récapitulation :

Un schéma relationnel se décompose pour chaque relation par le nom de la relation, les attributs + domaines d'attributs de la relation, identifiant(s), éventuellement les identifiants externes + contraintes d'intégrité associées à cette relation.

III. Traduction E/A - relationnel

La modélisation E/A des données étant effectuée, il faut implanter la structure obtenue en machine, par exemple sous forme d'un SGBD relationnel.

Nous donnons ci-après quatre règles (de R1 à R4) pour traduire un schéma conceptuel entité/association en un schéma relationnel équivalent. Il existe d'autres solutions de transformation, mais ces règles sont les plus simples et les plus opérationnelles.

III.1. Transformation des entités

Règle 1 : Chaque entité devient une relation. L'identifiant de l'entité devient clé primaire pour la relation.
Si aucun attribut ne convient en tant qu'identifiant, il faut en ajouter un de telle sorte que la relation dispose d'une clé primaire (les outils proposent l'ajout de tels attributs).

Exemple :

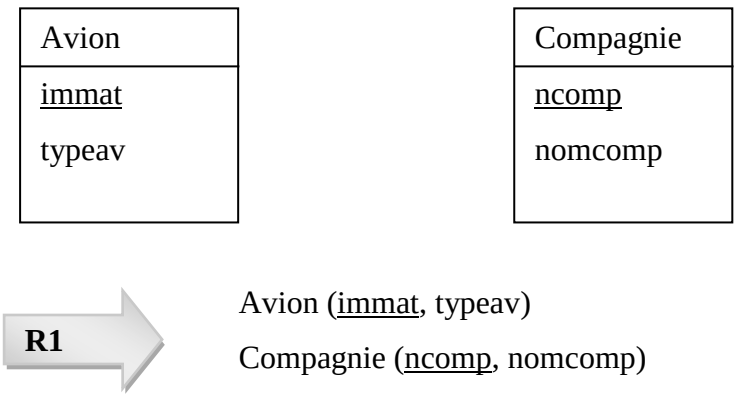


Figure 3.18 : *transformation des Entités*

III.2. Transformation des associations

Les règles de transformation que nous allons voir dépendent des cardinalités/multiplicités maximales des associations. Nous distinguons trois familles d'associations :

- ✓ un-à-plusieurs ;
- ✓ plusieurs-à-plusieurs ou classes-associations, et n-aires ;
- ✓ un-à-un.

III.2.1. Associations un-à-plusieurs

Règle 2 : Il faut ajouter un attribut de type clé étrangère dans la relation *fils* de l'association. L'attribut porte le nom de la clé primaire de la relation *père* de l'association.
On peut se rappeler cette règle de la manière suivante : *la clé de la relation père migre dans la relation fils.*

Les règles R1 et R2 ont été appliquées à l'exemple suivant. La règle R2 fait apparaître la clé étrangère ncomp dans la relation *fils* Avion qui a migré de la relation *père* Compagnie.

Exemple :

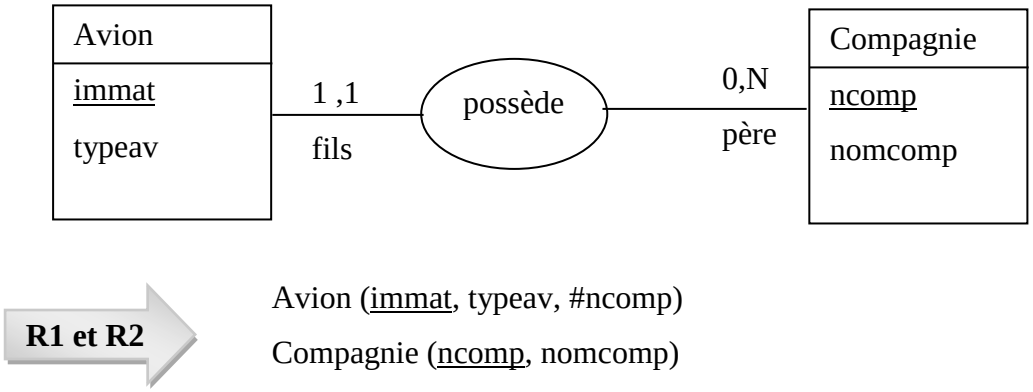


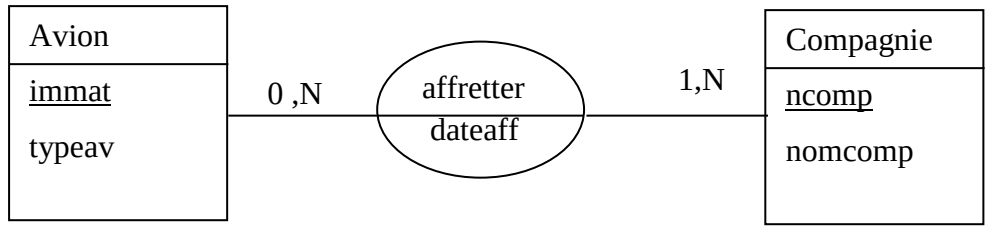
Figure 3.19 : *transformation d'une association un à plusieurs*

III.2.2. Associations plusieurs-à-plusieurs et n-aires

Règle 3 : L'association devient une relation dont la clé primaire est composée par la concaténation des identifiants des entités connectés à l'association. Chaque attribut devient clé étrangère si l'entité connectée dont il devient une relation en vertu de la règle R1.
Les attributs de l'association doivent être ajoutés à la nouvelle relation.
Ces attributs ne sont ni clé primaire, ni clé étrangère.

Les règles R1 et R3 ont été appliquées à l'exemple suivant. La règle R3 crée la relation Affreter dont la clé primaire est composée de deux clés étrangères. L'attribut dateaff de l'association est ajouté à la nouvelle relation.

Exemple :



R1 et R3

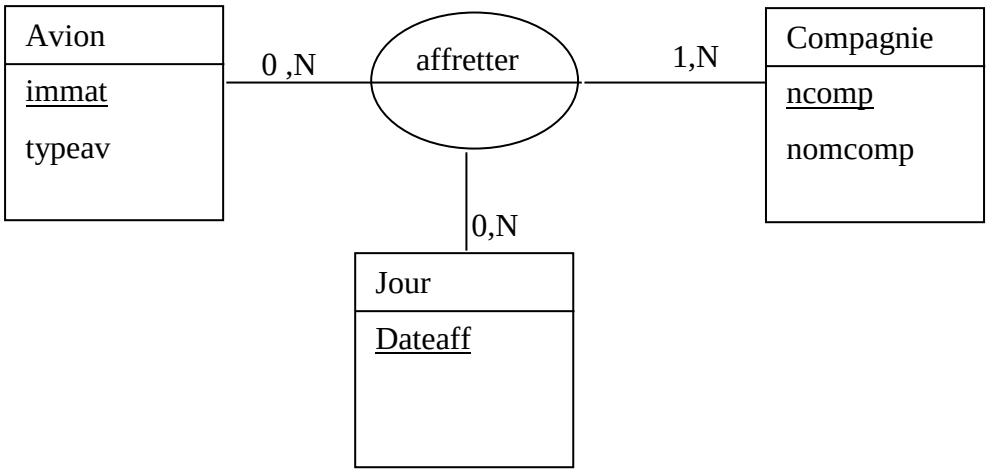
Avion (immat, typeav, #ncomp)
Affretter (#immat, #ncomp, dateaff)
Compagnie (ncomp, nomcomp)

Figure 3.20 : *transformation d'une association plusieurs à plusieurs*

Les règles R1 et R3 sont appliquées à l'association *3*-aire Affreter. On ne dérive pas la relation

Jour car c'est une entité temporelle. En conséquence, l'attribut dateaff ne devient pas une clé étrangère, mais il est nécessaire pour composer la clé primaire de la relation modélisant les affrètements.

Exemple :



R1 et R3

Avion (immat, typeav, #ncomp)
Affretter (#immat, #ncomp, dateaff)
Compagnie (ncomp, nomcomp)

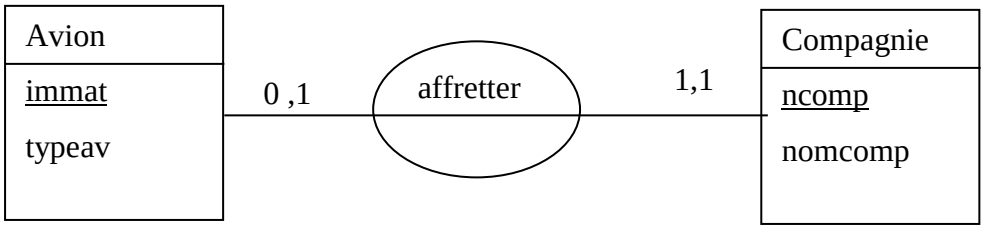
Figure 3.21 : *transformation d'une association 3-aire*

III.2.3. Associations un-à-un

La règle est la suivante, elle permet d'éviter les valeurs NULL dans la base de données.

Règle 4 : Il faut ajouter un attribut clé étrangère dans la relation dérivée de l'entité ayant la cardinalité minimale égale à zéro. L'attribut porte le nom de la clé primaire de la relation dérivée de l'entité connectée à l'association.
Si les deux cardinalités minimales sont à zéro, le choix est donné entre les deux relations dérivées de la règle R1. Si les deux cardinalités minimales sont à un, il est sans doute préférable de fusionner les deux entités en une seule.

Exemple :



R1 et R4

Avion (immat, typeav)
Compagnie (ncomp, nomcomp, #immat)

Figure 3.22 : *transformation d'une association un à un*

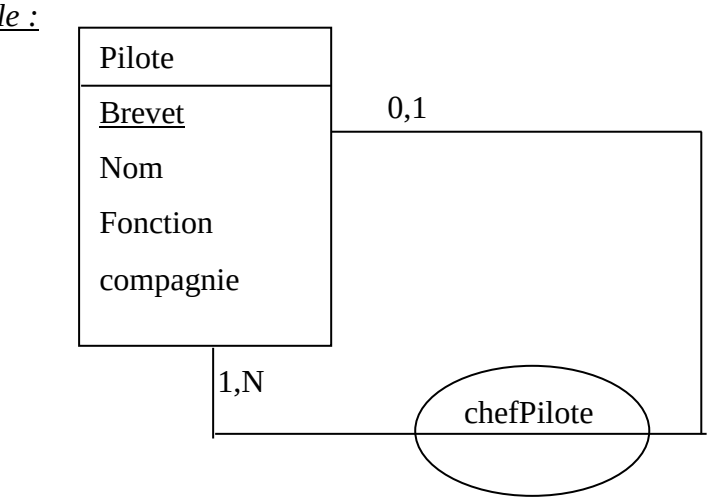
III.3. Associations réflexives

Les associations réflexives sont des associations binaires (*un-à-un*, *un-à-plusieurs*, *plusieurs-à plusieurs*) ou *n*-aires. Les transformations sont analogues aux associations non réflexives.

III.3.1. Un-à-plusieurs

Les règles R1 et R2 sont appliquées à l'association réflexive *un-à-plusieurs* de l'exemple suivant.
La clé étrangère contiendra le code du chef pilote pour chaque pilote. Pour tout chef, cette clé étrangère ne contiendra pas de valeur (NULL).

Exemple :



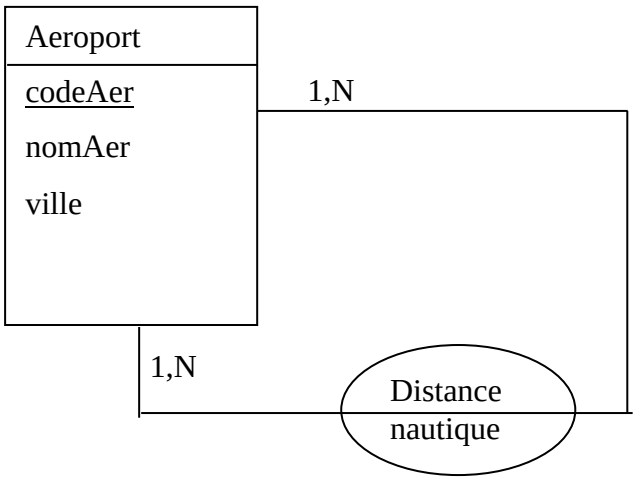
R1 et R2 Pilote (Brevet, nom, fonction, compagnie, #chef)

Figure 3.23 : *transformation d'une association réflexive un à plusieurs*

III.3.2. Plusieurs-à-plusieurs

Les règles R1 et R3 sont appliquées à l'association réflexive *plusieurs-à-plusieurs* de l'exemple suivant (modélisation de la distance entre deux aéroports).

Exemple :



R1 et R3

Aeroport (codeAer, nomAer, ville)

Distance (#codeAer1, #codeAer2, nautique)

Figure 3.24 : *transformation d'une association réflexive plusieurs à plusieurs*

IV. Exercice d'application

L'exemple du modèle Entités/Associations élaboré dans le chapitre précédent (Exercice d'application) se traduit en modèle relationnel comme suit :

CLIENT (numCl, nomCl, prenomCl, datenaisCl, adrCl)

PRODUIT (numPd, designPd, puPd, #numFr)

FOURNISSEUR (numFr, rsFr, emailFr, adrFr)

COMMANDE (#numCl, #numPd, dateCd, qtePc)

Chapitre n°4 : Normalisation d'une base de données

Objectifs spécifiques

A la fin de ce chapitre, l'étudiant doit être capable de :

- Apprendre les propriétés des dépendances fonctionnelles
- Dégager les dépendances fonctionnelles
- Connaitre les différentes formes normales
- Normaliser une relation

Plan du chapitre

I. Normalisation

II. Dépendance fonctionnelle (DF)

III. Les formes Normale

Volume horaire

3 heures

I. Normalisation

I.1. Définition

La normalisation est un processus de transformation d'une relation posant des problèmes lors de la mise à jour en relation ne posant pas de problème.

On mesure la qualité d'une relation par son degré de normalisation N, la valeur de N est présenté en fonction des formes normales.

I.2. Objectifs de la normalisation

Exemple : Soit la relation COMMANDE_PRODUIT (NumProd, Quantite, NumFour, Adresse).

NumProd	Quantite	NumFour	Adresse
101	300	901	Av. Hbib Bourguiba
104	1000	902	Av. 7 Novembre
112	78	904	Rue 20 mars
103	250	901	Av. Hbib Bourguiba

Cette relation présente différentes anomalies.

- *Anomalies de modification* : Si l'on souhaite mettre à jour l'adresse d'un fournisseur, il faut le faire pour tous les tuples concernés.
- *Anomalies d'insertion* : Pour ajouter un nouveau fournisseur, il faut obligatoirement fournir des valeurs pour NumProd et Quantité.
- *Anomalies de suppression* : La suppression du produit 104 fait perdre toutes les informations concernant le fournisseur 902.

➔ Pour faire face à ce genre de problèmes, on a recours à la normalisation.

Objectifs de la normalisation :

- ✓ Suppression des problèmes de mise à jour
- ✓ Minimisation de l'espace de stockage (élimination des redondances).
- ✓ Incohérence

II. Dépendance fonctionnelle

La normalisation est basée sur la notion de dépendance fonctionnelle (DF) introduite par CODD en 1972 afin de caractériser des relations pouvant être décomposées sans pertes d'information.

II.1. Définition

Soit R (X, Y, Z) une relation où X, Y, et Z sont des ensembles d'attributs. Z peut être vide. On dit que Y dépend fonctionnellement de X ou X détermine Y noté (X → Y) si étant donné une valeur de X, il lui correspond une valeur unique de Y (quelque soit l'instant considéré).

Exemple : Soit la relation PRODUIT (NumProd, Dési, PrixUni)

NumProd → Dési

Dési → PrixUni

NumProd → PrixUni

Remarque : Il est essentiel de bien remarquer qu'une dépendance fonctionnelle (en abrégé, DF) est une assertion sur toutes les valeurs possibles et non pas sur les valeurs actuelles : elle caractérise une intention et non pas une extension d'une relation.

II.2. Propriétés des dépendances fonctionnelles

Les dépendances fonctionnelles obéissent à certaines propriétés. Les trois propriétés suivantes composent les axiomes des dépendances fonctionnelles et sont connues sous le nom d'axiomes d'Armstrong.

- ✓ **Réflexivité** : Y ⊂ X => X→ Y ; tout ensemble d'attributs détermine lui-même ou une partie de lui-même.
- ✓ **Augmentation** : X → Y => X, Z → Y, Z ; si X détermine Y, les deux ensembles d'attributs peuvent être enrichis par un troisième.
- ✓ **Transitivité** : X → Y et Y → Z => X→ Z ; par exemple, à partir des deux DF NumProd → Dési et Dési → PrixUni on déduit que NumProd → PrixUni

D'autres propriétés se déduisent de ces axiomes :

- ✓ **Union** : X → Y et X→ Z => X→Y, Z
- ✓ **Pseudo-transitivité** : X → Y et Y, W →Z => X, W → Z
- ✓ **Décomposition** : X → Y et Z ⊂ Y => X→ Z

L'intérêt de ces axiomes et des propriétés déduites est de pouvoir construire, à partir d'un premier ensemble de dépendances fonctionnelles, l'ensemble de toutes les dépendances fonctionnelles qu'elles génèrent.

II.3. DF élémentaire

Une DF de la forme X, Y → Z est élémentaire si ni X → Z ni Y → Z ne sont des DF. Autrement dit, tous les attributs à gauche sont nécessaires pour déterminer l'attribut à droite de la flèche.

Exemple :

- La dépendance fonctionnelle NCIN, NV → dateAchat est élémentaire.
- La dépendance fonctionnelle NCIN, NV → marque n'est pas élémentaire car le numéro voiture (NV) seul suffit pour déterminer la marque (NV → marque est une DF).

II.4. DF directe

Une dépendance fonctionnelle X → Z est directe si elle n'est pas déduite par transitivité, c'est-à-dire qu'il n'existe pas un attribut Y tel que X → Y et Y → Z.

Exemple :

- La dépendance fonctionnelle NV → modèle est une DF directe.
- La dépendance fonctionnelle modèle → marque est une DF directe.
- La dépendance fonctionnelle NV → marque est une DF indirecte (elle peut être déduite à partir des deux DF précédentes)

II.5. Graphe de DF

Pour chaque relation, il faut connaître les DF. Il est facile alors de les représenter sous forme de graphe : Graphe des DF.

Dans ce graphe, les sommets sont des attributs et un arc relie l'attribut A à l'attribut B si et seulement si A → B.

Exemple 1:

T(A,B,C,D, E) avec DF= {E → A, E →B, E → C, C → D}

Donc on peut déduire : E → D, E → A, B, C, D

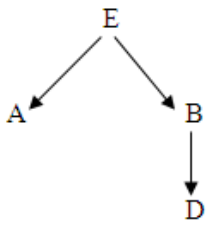


Figure 4.25 : *Exemple 1 : Graphe de DF (1)*

Si on ajoute H avec H → D on ne peut ainsi déduire d'autres DF, donc le graphe de DF est comme suit :

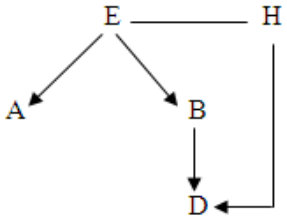


Figure 4.26 : *Exemple 1 : Graphe de DF (2)*

Exemple 2 :

Soit la relation « Commande » ayant le schéma suivant :

Commande (numcde, datecde, montant, numcli, nomcli, adrcli)

Il est évident que l'attribut « numcde » permet d'identifier la date « datecde », le montant total d'une commande ainsi le numéro du client qui a passé la commande. A partir de ce numéro, on peut connaitre les autres informations relatives au client à savoir son nom et son adresse.

Notons F, l'ensemble des dépendances fonctionnelles entre les attributs de la relation « Commande » :

F = {numcde → datecde ; numcde → montant ; numcde → numcli ;
numcli → nomcli; numcli → adrcli}

La figure suivante montre le graphe de DF relatif à cet exemple :

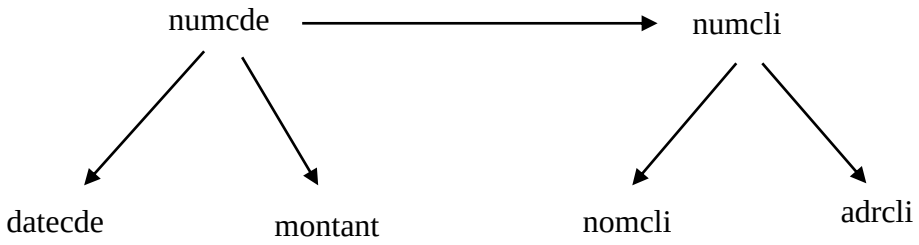


Figure 4.27 : *Exemple 2 : Graphe de DF*

II.6. Fermeture transitive

On appelle fermeture transitive d'un ensemble de dépendance fonctionnelle F, l'ensemble F+ qui est l'union de F et de l'ensemble des dépendances fonctionnelles déduites par transitivité.

Exemple :

Soit l'ensemble de DF :

F = {numcde → datecde ; numcde → montant ; numcde → numcli ;
numcli → nomcli; numcli → adrcli}

On obtient la fermeture transitive suivante :

F+ = F ∪ { numcli → nomcli; numcde → adrcli }

Le schéma suivant illustre le graphe de DF obtenues à partir de F+ :

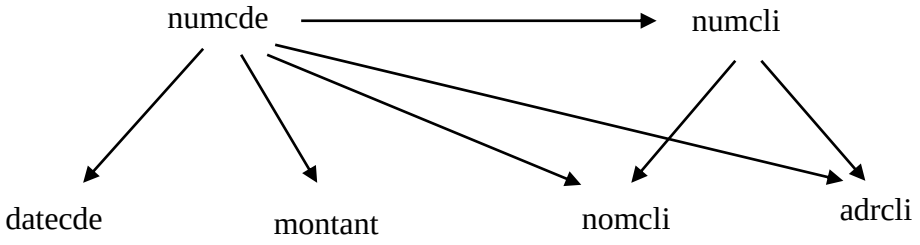


Figure 4.28 : *Exemple de fermeture transitive*

A partir de la notion de fermeture transitive, il est possible de définir l'équivalence de deux ensembles de DF élémentaires.

Définition : Deux ensembles de dépendances fonctionnelles F et F' sont équivalents s'ils ont la même fermeture transitive.

Par la suite, il est intéressant de déterminer un ensemble minimal de DF permettant de générer toutes les autres, c'est la couverture minimale.

II.7. Couverture minimale

La couverture minimale associée à un ensemble d'attributs est un ensemble F de dépendances fonctionnelles vérifiant les deux propriétés suivantes :

- **i.** Aucune dépendance dans F n'est pas redondante, ce qui signifie que pour toute dépendance fonctionnelle f de F, F - {f} n'est pas équivalent à F.
- **ii.** Toute dépendance fonctionnelle élémentaire des attributs est dans la fermeture transitive de F (notée F+).

Exemple :

Soit l'ensemble F définit comme suit :

F = {numcde → datecde ; numcde → montant ; numcde → numcli ;
numcli → nomcli; numcli → adrcli}

F est une couverture minimale pour l'ensemble des attributs de la relation « Commande » définie ci-dessus. La couverture minimale va constituer un élément essentiel pour composer des relations sans pertes d'information directement à partir des attributs.

II.8. Clé d'une relation

Pour une relation R(A1, A2, …, An), un sous-ensemble d'attributs X est une clé si et seulement si :

i. X → A1, A2, …, An

ii. Il n'existe pas de sous-ensemble Y de X tel que Y → A1, A2, …, An

Une clé est donc un ensemble minimal d'attributs qui détermine tous les autres.

III. Les formes normales

Les formes normales correspondent à une décomposition optimale des entités en relations élémentaires correctement construites. Cela évite les redondances d'informations et facilite la maintenance des données. Les formes normales s'appuient sur les dépendances fonctionnelles entre les attributs.

Si le relevé d'information, le dictionnaire des données, le graphe des dépendances fonctionnelles ont été correctement construits, le modèle relationnel qui en découle devrait déjà être en troisième forme normale (3FN), voire en forme normale de Boyce-Codd (FNBC).

III.1. Première forme normale (1FN)

Définition : Une relation est en première forme normale (1FN) si chaque valeur de chaque attribut de chaque tuple d'une relation R est une valeur simple (tous les attributs sont simple et monovalués). C.à.d. si tout attribut est atomique (n'est pas décomposable).

Exemple 1: Soit la relation Personne ayant le schéma suivant :

PERSONNE (cin, nom, prénoms, age)

Cette relation n'est pas en 1FN si l'attribut prénoms peut être composé c.-à-d. de type [Med Ali].

Exemple 2: Soit la relation Fournisseur ayant le schéma suivant :

Fournisseur (numfrs, nomfrs, adresse)

Cette relation n'est pas en 1FN si l'attribut adresse est elle-même composée de la manière suivante (rue, ville, codePostal).

Dans ce cas, le principe de passage en 1FN consiste à créer autant de champs que ceux composant l'adresse, ce qui donne le schéma suivant :

Fournisseur (numfrs, nomfrs, rue, ville, codePostal)

III.2. Deuxième forme normale (2FN)

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

- ✓ elle est en 1FN ;
- ✓ tout attribut n'appartenant pas à la clé primaire ne dépend pas d'une partie de la clé ; c.-à-d. qu'il est en dépendance fonctionnelle élémentaire (totale) avec la clé.

Exemple 1:

La relation Client (NumCli, Nom, DateNaiss, Rue, CP, Ville) est en 2FN.

Car c'est une relation en 1FN car tous les attributs sont atomiques et monovalués.

On a la clé est composée d'un seul attribut (Numcli) donc tout attribut n'appartient pas à la clé ne dépend pas d'une partie de la clé.

⇨ La relation Client est en 2FN

Exemple 2 :

Soit la relation Commande avec une clé composée de 2 attributs (numcde +numcli) ayant le schéma suivant :

Commande (numcde, datecde, montant, numcli, nomcli, adrcli)

Commande					
numcde	**datecde**	**montant**	**numcli**	**nomcli**	**adrcli**
C1	02/05/2016	1200	1	Ali	Sfax
C2	08/05/2016	250	2	Mohamed	Sousse
C3	12/05/2016	340	3	Anas	Tunis
C4	15/05/2016	320	4	Lotfi	Sidi Bouzid
C5	15/05/2016	225	5	Natija	Sidi Bouzid
C6	22/05/2016	60	6	Salah	Bizerte

Cette relation n'est pas en 2FN étant donné qu'elle contient des dépendances fonctionnelles partielles. A titre d'exemple, la DF numcde, numcli → nomcli n'est pas élémentaire ; il suffit en effet de connaître le numéro du client pour en déduire le nom.

Pour faire passer cette relation en 2FN, il faut la décomposer en deux, ce qui donne le résultat suivant :

Commande (numcde, datecde, montant, #numcli) et

Client (numcli, nomcli, adrcli)

Commande			
numcde	**datecde**	**montant**	**#numcli**
C1	02/05/2016	1200	2
C2	08/05/2016	250	1
C3	12/05/2016	340	3
C4	15/05/2016	320	3
C5	15/05/2016	225	2
C6	22/05/2016	60	4

Et

Client		
numcli	**nomcli**	**adrcli**
1	Ali	Sfax
2	Mohamed	Sousse
3	Anas	Tunis
4	Lotfi	Sidi Bouzid
5	Natija	Sidi Bouzid

III.3. Troisième forme normale (3FN)

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

- ✓ Elle est en 2FN ;
- ✓ Tout attribut n'appartenant pas à la clé ne dépend pas d'un autre attribut non clé ; c.-à-d. qu'il n'existe aucune DF transitive entre la clé et les autres attributs.

⇨ Si la profondeur d'un graphe de DF est supérieure à 1 donc la relation n'est pas en 3FN.

Exemple 1:

La relation Compagnie (numVol, numAvion, numPilote) avec les DF :

numVol → numAvion, numAvion → numPilote et numVol → numPilote est en 2FN, mais pas en 3FN.

Anomalies de mise à jour sur la relation Compagnie : Il n'est pas possible d'introduire un nouvel avion sur un nouveau vol sans préciser le pilote correspondant.

La décomposition suivante donne deux relations en 3FN qui permettent de retrouver (par transitivité) toutes les DF : R1 (numVol, #numAvion) ; R2 (numAvion, numPilote).

Exemple 2:

Considérons la relation Fournisseur ayant le schéma suivant :

Fournisseur (numfrs, nomfrs, ville, #codepostal)

Les DF sont les suivantes :

DF = { numfrs → nomfrs, ville, codepostal ; codepostal → ville }

Il est clair que la relation « Fournisseur » est en 2FN (la clé primaire n'est pas composé) mais n'est pas en 3FN car un attribut nom clé (ville) dépond d'un autre attribut non clé (codepostal).

Pour faire passer cette relation en 3FN il faut la décomposer comme suit :

Fournisseur (numfrs, nomfrs, codepostal)

Adresse (codepostal, ville)

Les nouvelles relations Fournisseur et Adresse sont en 3FN.

III.4. Forme normale de Boyce-Codd (BCFN)

Définition : Une relation est en forme normale de Boyce-Codd (BCFN) si et seulement si :

- ✓ Elle est en 3FN ;
- ✓ Le seul déterminant (membre gauche d'une DF) existant dans la relation est la clé primaire. Autrement dit, il n'existe aucune DF entre des parties de la clé ou entre un attribut et une partie de la clé.

Exemple :

Soit la relation ProductionRegion définit comme suit :

ProductionRegion (typeav, pays, region)

ProductionRegion		
typeav	**pays**	**region**
A1	Tunisie	Sousse
A2	Tunisie	Sfax
A3	Algérie	Algérois
A4	Tunisie	Sidi Bouzid
A5	Algérie	Constantinois

Les DF existantes sont :

DF = {typeav, pays → region ; region → pays}

L'existence de la dernière DF permet de statuer que cette relation n'est pas en BCNF.

Pour faire passer cette relation en BCFN il faut la décomposer comme suit :

RegionsPays (region, pays) et Avionregion (typeav, #region)

RegionPays	
region	**pays**
Sousse	Tunisie
Sfax	Tunisie
Algérois	Algérie
Sidi Bouzid	Tunisie
Constantinois	Algérie

AvionRegion	
typeav	**#region**
A1	Sousse
A2	Sfax
A3	Algérois
A4	Sidi Bouzid
A5	Constantinois

IV. Exercice d'application

Considérons l'exemple d'une agence immobilière. Les données sur les agences assurées sont rassemblées dans une seule relation « Agence » illustrée par le tableau suivant :

numcli	nomcli	numapp	villeapp	datedloc	datefloc	montant	numprop	nomprop
C1	Anas	A1	Sfax	01/01/2015	01/01/2016	350	P1	Mohamed
		A2	Sousse	01/09/2015	01/09/2016	250	P2	Ali
C2	Ali	A1	Sfax	01/11/2015	01/11/2016	350	P1	Mohamed
		A3	Tunis	01/02/2016	01/02/2017	400	P2	Ali
		A2	Sousse	01/04/2016	01/04/2017	250	P2	Ali

On va maintenance procéder à la décomposition de cette relation de manière à obtenir un schéma normalisé.

- **Première Forme Normale (1FN)**

La relation « Agence » n'est en 1FN, il faut l'aplatir de manière à supprimer tous les attributs multivalués. Le résultat est la relation « Agence1 » suivante en 1FN :

numcli	nomcli	numapp	villeapp	datedloc	datefloc	montant	numprop	nomprop
C1	Anas	A1	Sfax	01/01/2015	01/01/2016	350	P1	Mohamed
C1	Anas	A2	Sousse	01/09/2015	01/09/2016	250	P2	Ali
C2	Ali	A1	Sfax	01/11/2015	01/11/2016	350	P1	Mohamed
C2	Ali	A3	Tunis	01/02/2016	01/02/2017	400	P2	Ali
C2	Ali	A2	Sousse	01/04/2016	01/04/2017	250	P2	Ali

- ***Dépendances fonctionnelles et clé***
  - ✓ Les dépendances fonctionnelles sont représentées par l'ensemble DF suivante :

DF = {numcli → nomcli ; numcli, numapp → datedloc, datefloc ;
numapp → villeapp, montant, numprop, nomprop ; numprop → nomprop }

- ✓ La clé de cette relation sera la concaténation des attributs (numcli + numapp) puisqu'à partir de cette combinaison on peut déterminer tous les autres attributs.

- **Deuxième Forme Normale (2FN)**

Il faut donc décomposer R pour obtenir un schéma en 2ème forme normale : les 3 relations déduites des DF sont toutes en 2FN.

Client	
numcli	**nomcli**
C1	Anas
C2	Ali

Appartement				
numapp	**villeapp**	**montant**	**numprop**	**nomprop**
A1	Sfax	350	P1	Mohamed
A2	Sousse	250	P2	Ali
A3	Tunis	400	P2	Ali

Location			
# numcli	**#numapp**	**datedloc**	**datefloc**
C1	A1	01/01/2015	01/01/2016
C1	A2	01/09/2015	01/09/2016
C2	A1	01/11/2015	01/11/2016
C2	A3	01/02/2016	01/02/2017
C2	A2	01/04/2016	01/04/2017

- **Troisième Forme Normale (3FN) et BCFN**

D'après les DF numapp → numprop et numprop → nomprop on déduit que la DF nomapp → numprop est indirecte car elle est issue d'une transitivité. Pour ramener la table « Appartement **»** vers la 3FN, il faut supprimer cette DF, ce qui revient à définir une nouvelle relation « Propriétaire ».

Finalement, on obtient une base de données formée de quatre tables qui sont toutes en BCFN.

Client	
numcli	**nomcli**
C1	Anas
C2	Ali

Appartement			
numapp	**villeapp**	**montant**	**#numprop**
A1	Sfax	350	P1
A2	Sousse	250	P2
A3	Tunis	400	P2

Appartement	
numprop	**nomprop**
P1	Mohamed
P2	Ali

Location			
# numcli	**#numapp**	**datedloc**	**datefloc**
C1	A1	01/01/2015	01/01/2016
C1	A2	01/09/2015	01/09/2016
C2	A1	01/11/2015	01/11/2016
C2	A3	01/02/2016	01/02/2017
C2	A2	01/04/2016	01/04/2017

Chapitre n°5 : Algèbre relationnelle

Objectifs spécifiques

A la fin de ce chapitre, l'étudiant doit être capable de :

- Reconnaître l'utilité des opérateurs ensemblistes et spécifiques
- Analyser des requêtes plus ou moins complexes
- Appliquer les opérateurs appropriés dans l'expression des requêtes

Plan du chapitre

I. Définition

II. Opérateurs ensemblistes

III. Opérateurs spécifiques

IV. Exercice d'application

Volume horaire

3 heures

I. Introduction

L'algèbre relationnelle est un langage de requêtes d'interrogation des données. C'est un langage théorique comme l'algorithmique : il ne peut pas être compris directement par les SGBDR. Il faut le traduire dans un langage supporté par le SGBD tel que SQL (Structred Query Langage : langage standard pour tous les SGBDR).

Pour mieux comprendre SQL et pour mieux construire des requêtes SQL, il est nécessaire d'étudier l'algèbre relationnelle. L'algèbre relationnelle est un ensemble d'opérateurs permettant donc de manipuler les données des tables d'une base de données à l'aide de requêtes. Elle prépare la conception de requêtes qui seront traduite en SQL.

Les limites de l'algèbre relationnelle : il n'est pas possible de faire des tris sur les relations.

Résultat obtenu : une nouvelle relation qui peut à son tour être manipulée.

L'algèbre relationnelle permet d'effectuer des recherches dans les relations. Elle est composée de :

- Opérateurs ensemblistes : sont les opérateurs binaires d'union, de différence et de produit cartésien ainsi que des opérateurs dérivés comme les opérateurs d'intersection, de division.
- Opérateurs spécifiques : sont les opérateurs unaires de projection et de restriction et les binaires de jointure (théta-jointure) et de jointures naturelles.

II. Opérateurs ensemblistes

II.1. Union

Définition :

T=union(R, S) contient tous les tuples appartenant à R ou S ou aux deux relations (en évitant le doublon de tuple).

Représentation symbolique : **T= R U S**

Précondition : les deux relations R et S doivent avoir même schéma (les mêmes attributs et ceux-ci ont le même domaine).

Représentation graphique :

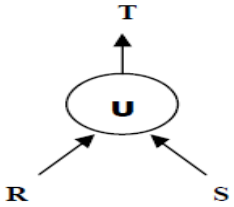


Figure 5.29 : représentation graphique de l'opérateur Union

Exemple : R et S sont représentés par les relations R1 et R2 et T par R1 U R2 :

R_1	**Ncin**	**Nom**	**Prénom**	**Age**
	03164571	Ben salah	Ali	20
	03264571	Haddad	Saber	19
	03364572	Issa	Ines	18

R_2	**Ncin**	**Nom**	**Prénom**	**Age**
	03164571	Ben salah	Ali	20
	03464571	Abid	Fadhila	22
	06064571	Issa	Amor	25

Alors

R1 U R2	**Ncin**	**Nom**	**Prénom**	**Age**
	03164571	Ben salah	Ali	20
	03264571	Haddad	Saber	19
	03364572	Issa	Ines	18
	03464571	Abid	Fadhila	22
	06064571	Issa	Amor	25

II.2. Différence

Définition :

T=munis(R, S) permet de retirer les tuples de la relation S existant dans la relation R (c.à.d. T ayant pour tuples ceux appartenant à R et n'appartenant pas à S).

Représentation symbolique : **T=R - S**

Précondition : les deux relations R et S doivent avoir même schéma (les mêmes attributs et ceux-ci ont le même domaine).

Représentation graphique :

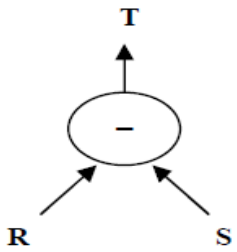


Figure 5.30 : représentation graphique de l'opérateur Différence

Exemple : R et S sont représentés par les relations R1 et R2 et T par R1 - R2 :

R_1	Ncin	Nom	Prénom	Age
	03164571	Ben salah	Ali	20
	03264571	Haddad	Saber	19
	03364572	Issa	Ines	18

R_2	Ncin	Nom	Prénom	Age
	03164571	Ben salah	Ali	20
	03464571	Abid	Fadhila	22
	06064571	Issa	Amor	25

Alors

R_1 - R_2	Ncin	Nom	Prénom	Age
	03264571	Haddad	Saber	19
	03364572	Issa	Ines	18

II.3. Intersection

Définition :

T=intersect(R, S) contient les tuples qui appartiennent à la fois aux deux relations R et S.

Représentation symbolique : **T=R ∩ S**

Précondition : les deux relations R et S doivent avoir même schéma (les mêmes attributs et ceux-ci ont le même domaine).

Représentation graphique :

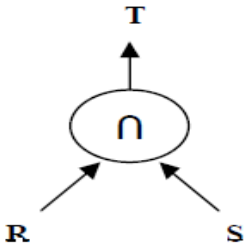


Figure 5.31 : représentation graphique de l'opérateur Intersection

Exemple : R et S sont représentés par les relations R1 et R2 et T par R1 ∩ R2 :

R_1	Ncin	Nom	Prénom	Age
	03164571	Ben salah	Ali	20
	03264571	Haddad	Saber	19
	03364572	Issa	Ines	18

R_2	Ncin	Nom	Prénom	Age
	03164571	Ben salah	Ali	20
	03464571	Abid	Fadhila	22
	06064571	Issa	Amor	25

Alors

$R_1 \cap R_2$	Ncin	Nom	Prénom	Age
	03164571	Ben salah	Ali	20

II.4. Produit cartésien

Définition :

T=product(R, S) permet de construire une relation T ayant pour schéma la concaténation des deux relations R et S et pour tuples toutes les combinaisons des tuples de deux relations (c.à.d. permet d'associer chaque tuple de R à chaque tuple de S).

Représentation symbolique : **T=R X S**

Précondition : les deux relations n'ont pas le même schéma (c.à.d. n'ont pas les mêmes attributs).

Représentation graphique :

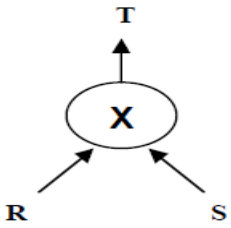


Figure 5.32 : représentation graphique de l'opérateur Produit Cartésien

Exemple : R et S sont représentés par les relations R1 et R2 et T par R1 X R2 :

R_1	Ncin	Nom	Prénom	Age
	03164571	Ben salah	Ali	20
	03264571	Haddad	Saber	19
	03364572	Issa	Ines	18

R_2	Ville
	Sidi bouzid
	Gafsa
	Tunis

Alors

R1XR2	Ncin	Nom	Prénom	Age	Ville
	03164571	Ben salah	Ali	20	Sidi bouzid
	03264571	Haddad	Saber	19	Sidi bouzid
	03364572	Issa	Ines	18	Sidi bouzid
	03164571	Ben salah	Ali	20	Gafsa
	03264571	Haddad	Saber	19	Gafsa
	03364572	Issa	Ines	18	Gafsa
	03164571	Ben salah	Ali	20	Tunis
	03264571	Haddad	Saber	19	Tunis
	03364572	Issa	Ines	18	Tunis

III. Opérateurs spécifiques

III.1. Projection

Définition :

T= project(R / A, B, C), T ne contient que les attributs A, B et C de R.

Le but de l'opérateur « projection » est de ne retenir que certain attribut dans une relation.

Remarque : Effet de bord de la projection : une projection qui ne conserve pas la clé de la relation peut générer dans le résultat deux tuples identiques, le résultat ne garde que les tuples différents (c.à.d. pas de doublon de tuples).

Représentation symbolique : **T= $\pi_{<A, B, C>}$ (R)**

Représentation graphique :

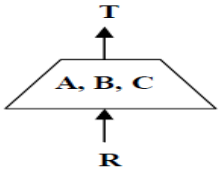


Figure 5.33 : représentation graphique de l'opérateur Projection

Exemple :

R	Ncin	Nom	Prénom	Age
	03164571	Ben salah	Ali	20
	03264571	Haddad	Saber	19
	03364572	Issa	Ines	18
	03364565	Issa	Ines	26

Alors

π <nom, prénom> (R)	Nom	Prénom
	Ben salah	Ali
	Haddad	Saber
	Issa	Ines

III.2. Restriction (Sélection)

Définition :

T=restrict(R / C) permet de construire une relation T de même schéma que R, mais comportant les seuls tuples qui vérifient la condition C en argument (ou Prédicat).

Représentation symbolique : **T=$\sigma_{<C>}$ (R)**

Représentation graphique :

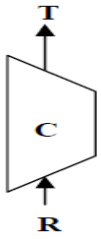


Figure 5.34 : représentation graphique de l'opérateur Restriction

Exemple :

R	Ncin	Nom	Prénom	Age	Ville
	03164571	Ben salah	Ali	20	Sidi bouzid
	03264571	Haddad	Saber	19	Gafsa
	03464571	Abid	Fadhila	22	Tunis
	03364572	Issa	Ines	18	Tunis

Alors

σ < ville=Tunis> (R)	Ncin	Nom	Prénom	Age	Ville
	03464571	Abid	Fadhila	22	Tunis
	03364572	Issa	Ines	18	Tunis

III.3. Jointure Naturelle

Définition :

T=join(R, S), Produit cartésien R X S et Restriction A=B sur les attributs A de R et B de S. (la jointure naturelle permet de créer toute les combinaisons significatifs entre tuples de deux relations). Elle porte la même valeur pour les attributs de même nom.

Représentation symbolique : **T=R >< S**

Précondition : les deux relations ont au moins un attribut de même nom.

Représentation graphique :

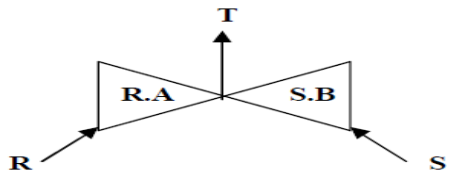


Figure 5.35 : représentation graphique de l'opérateur Jointure Naturelle

Exemple :

R et S sont représentés par les relations R1 et R2 et T par R1 >< R2 :

R_1 (fournisseur)	Ncinf	Nom	Prénom	Ville
	03364572	Briki	Amer	Sousse
	03264571	Fkhi	Anas	Gabes

R_2 (facture)	Cod_fact	Date_fact	Mont_fact	Ncinf
	F001	01/01/2003	1000	03364572
	F002	15/01/2004	1600	03264571

Alors

R1 >< R2	Ncinf	Nom	Prénom	Ville	Cod_fact	Date_fact	Mont_fact
	03364572	Briki	Amer	Sousse	F001	01/01/2003	1000
	03264571	Fkhi	Anas	Gabes	F002	15/01/2004	1600

III.4. Théta-jointure

Définition :

T=join$_{[P]}$ (R, S), Produit cartésien R X S et Restriction A=B sur les attributs A de R et B de S selon le prédicat P (la théta-jointure permet de créer toute les combinaison significatifs entre tuples de deux relations).

C'est un critère de combinaison explicitement définie (par le prédicat) en paramètre de l'opération c.à.d. P.

Représentation symbolique : **T=R ><$_{[P]}$ S**

Précondition : les deux relations non pas d'attribut de même nom.

Représentation graphique :

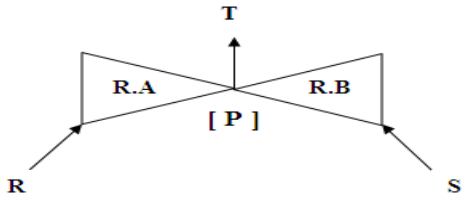


Figure 5.36 : représentation graphique de l'opérateur Théta-Jointure

Exemple :

R et S sont représentés par les relations R1 et R2 et T par R1 >_[P] R2 :

R_1 (client)	Ncinc	Nom	Prénom	Ville
	03164571	Ben salah	Ali	Sidi bouzid
	03264571	Haddad	Saber	Gafsa
	03364572	Issa	Ines	Tunis

R_2 (facture)	Cod_fact	Date_fact	Mont_fact	Ncinf
	F001	01/01/2003	1000	03364572
	F002	01/02/2004	1200	03364572
	F003	15/01/2004	1600	03264571

Alors

R1 >< [Ncinc=Ncinf] R2	Ncinc	Nom	Prénom	Ville	Cod_fact	Date_fact	Mont_fact
	03364572	Issa	Ines	Tunis	F001	01/01/2003	1000
	03364572	Issa	Ines	Tunis	F002	01/02/2004	1200
	03264571	Haddad	Saber	Gafsa	F003	15/01/2004	1600

II.1. Division

Définition :

T = division(R, S)

Où R(A_1, A_2, …,A_p, A_{p+1},.., A_n) et S(A_{p+1}, … , A_n)

T(A_1, A_2, … , A_p) contient tous les attributs tels que la concaténation à chacun des tuples de S donne toujours un tuple de R.

Représentation symbolique : **T = R ÷ S**

Précondition : les deux relations n'ont pas le même schéma (c.à.d. n'ont pas les mêmes attributs).

Représentation graphique :

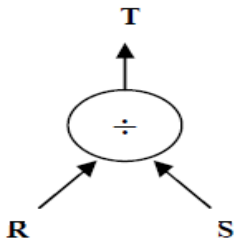


Figure 5.37 : représentation graphique de l'opérateur Division

Exemple : R et S sont représentés par les relations R1 et R2 et T par R1 ÷ R2 :

R_1 (fournir)	NF	NP
	F001	P001
	F002	P001
	F003	P003
	F001	P002
	F001	P003

R_2 (produit)	NP
	P001
	P002

Alors

$R_1 \div R_2$	NF
	F003
	F001

IV. Exercice d'application

Soit le modèle relationnel suivant :

CLIENT (numCl, nomCl, prenomCl, datenaisCl, adrCl)

PRODUIT (numPd, designPd, puPd, #numFr)

FOURNISSEUR (numFr, rsFr, emailFr, adrFr)

COMMANDE (#numCl, #numPd, dateCd, qtePc)

Proposer, en algèbre relationnelle, une formulation des requêtes suivantes.

1. Déterminer les noms des clients de numéro C1.
2. Déterminer les noms et les emails des fournisseurs.
3. Déterminer les produits ayant un prix unitaire entre 200 et 500.
4. Déterminer les clients qui ont commandé des produits.
5. Déterminer les produits qui ne sont pas commandés.
6. Déterminer les noms des fournisseurs de Sfax.
7. Déterminer les noms des fournisseurs des produits commandés le 12/05/2014.
8. Déterminer les numéros des clients de même adresse.
9. Déterminer les numéros des Produits des fournisseurs de Sousse.
10. Déterminer les commandes de quantité supérieure à 3.

Chapitre n°6 : Langage SQL

Objectifs spécifiques

A la fin de ce chapitre, l'étudiant doit être capable de :

- Apprendre à créer une base de données en tenant compte des contraintes d'intégrité
- Savoir ajouter, modifier, supprimer des enregistrements d'une table
- Construire des requêtes d'interrogations correspondant à des critères plus ou moins complexes
- Appliquer des droits d'accès à une base de données

Plan du chapitre

I. Présentation de SQL

II. Définition de données

III. Manipulation de données

IV. Interrogation de données

V. Contrôle de données

Volume horaire

6 heures

I. Présentation de SQL

SQL signifie Structured Query Language est le langage des bases de données relationnelles répandant à la fois aux problématiques de création des objets de base de données, de manipulation des données, de gestion de la sécurité, …

Il s'agit d'un langage non procédural qui a été conçu par IBM dans les années 70. Il est devenu le langage standard des systèmes de gestion de bases de données relationnelles (SGBDR) depuis 1986. SQL est le standard pour la définition, la manipulation et le contrôle de BD relationnelles.

Les commandes SQL sont regroupées en trois principales catégories :

- LDD (Langage de Définition des Données) : commandes pour créer (CREATE), modifier (ALTER) ou supprimer (DROP) les objets de la BD.
- LMD (Langage de Manipulation des Données) : commandes pour ajouter (INSERT), modifier (UPDATE), supprimer (DELETE) ou extraire (SELECT) des données et ceux pour verrouiller (LOCK TABLE) les tables.
- LCD (Langage de Contrôle de Données) : commandes pour la validation (COMMIT) et l'annulation (ROLLBACK) des transactions, pour autoriser (GRANT) ou interdire (REVOKE) des accès aux objets d'une base de données.

II. Contrôle de données (LCD)

II.1. Gestion des utilisateurs

Tout accès à la base de données s'effectue par l'intermédiaire de la notion d'utilisateur (compte Oracle).

Chaque utilisateur est défini par :

- ✓ un nom d'utilisateur
- ✓ un mot de passe
- ✓ un ensemble de privilèges

II.1.1. Création d'un utilisateur

Syntaxe : Pour créer un utilisateur, on doit spécifier le nom de l'utilisateur ainsi que le mot de passe via l'instruction :

CREATE USER utilisateur **IDENTIFIED BY** mot_de_passe ;

Exemple :

CREATE USER TI2 **IDENTIFIED BY** Ae3OPd ;

II.1.2. Modification d'un compte utilisateur

Syntaxe : Pour modifier le mot de passe d'un utilisateur, on écrit :

> **ALTER USER** utilisateur **IDENTIFIED BY** nouveau_mot_de_passe ;

Exemple :

> **CREATE USER** TI2 **IDENTIFIED BY** A23ePs ;

II.1.3. Suppression d'un utilisateur

Syntaxe : Pour supprimer un compte utilisateur, on écrit :

> **DROP USER** utilisateur [**CASCADE**] ;

L'utilisation de CASCADE signifie que la suppression de l'utilisateur est accompagnée par la suppression de tous les schémas qu'il a créé.

Exemple :

> **DROP USER** TI2 **CASCADE** ;

II.2. Gestion des privilèges

II.2.1. Attribution de privilèges

Un privilège peut être attribué à un utilisateur par l'ordre GRANT.

Syntaxe :

> **GRANT** privilège
> [**ON** table]
> **TO** utilisateur [**WITH GRANT OPTION**] ;

Remarque : Des droits peuvent être accordés à tous les utilisateurs par un seul ordre GRANT en utilisant le mot réservé PUBLIC à la place du nom d'utilisateur.

Principaux Privilèges :

SELECT : lecture

INSERT : insertion

UPDATE : mise à jour

DELETE : suppression

DBA, ALL : tous les privilèges

Si la clause WITH GRANT OPTION est spécifiée, le bénéficiaire peut à son tour assigner le privilège qu'il a reçu à d'autres utilisateurs.

Exemples :

```
GRANT SELECT
ON Frs
TO PUBLIC ;
GRANT UPADATE, DELETE
ON Frs
TO TI2 WITH GRANT OPTION ;
```

II.2.2. Suppression des privilèges

Un privilège peut être enlevé à un utilisateur par l'ordre REVOKE.

Syntaxe :

```
REVOKE privilège
[ON table]
FROM utilisateur ;
```

Exemples :

```
REVOKE SELECT
ON Frs
FROM TI2 ;
```

III. Définition de données (LDD)

III.1. Création des tables

Syntaxe : Pour créer une table, on fait recours à l'instruction suivante :

```
CREATE TABLE nom_table (
Col1 type1 [NOT NULL],
Col2 type2 [NOT NULL],
...,
Contrainte1,
Contrainte2,
…
) ;
```

L'option **NOT NULL** implique que la valeur nulle est interdite dans cette colonne, on l'utilise généralement pour les colonnes utilisées comme clé dans une table.

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

- NUMBER(N) : Entier à N chiffres
- NUMBER(N , M) : Réel à N chiffres au total (virgule comprise), M après la virgule.
- DATE : Date complète (date et/ou heure) (pour les dates par exemple on a le format : 'JJ-MM-AAAA').
- VARCHAR(N), VARCHAR2(N) : chaîne de N caractères (entre ' ') dont les espaces enfin de la chaîne seront éliminés (longueur variable).
- CHAR(N) : Chaîne de N caractères (longueur fixe).

III.1.1. Contraintes d'intégrité

Définition : Une contrainte d'intégrité est une règle qui permet de contrôler la valeur d'une colonne.

On utilise la clause CONSTRAINT pour décrire les contraintes qui doivent être satisfaites pour que les instructions d'insertion, modification et suppression soient possibles.

Dans la définition d'une table, on peut indiquer des contraintes d'intégrité portant sur une ou plusieurs colonnes.

Les contraintes possibles sont: UNIQUE, PRIMARY KEY, FOREIGN KEY … REFERENCES et CHECK.

Chaque contrainte doit être nommée pour pouvoir la mettre à jour ultérieurement.

On peut utiliser des contraintes pour imposer le respect de l'une des règles suivantes :

- UNIQUE : Imposer que la valeur d'un attribut donné soit unique dans la table.
- PRIMARY KEY : Identifier une ou plusieurs colonnes comme étant une clé primaire.
- FOREIGN KEY : Imposer que la valeur d'une ou de plusieurs colonnes existe dans une autre table pour appliquer la jointure entre eux.
- CHECK : Imposer que la valeur d'une ou de plusieurs colonnes soit conforme à une expression donné. CHECK est un mot associé à une condition d'intégrité référentielle par rapport à une clé unique ou primaire.

Création

- CONSTRAINT nom_contrainte UNIQUE (col1, col2, …) : interdit qu'une colonne, ou la concaténation de plusieurs colonnes, contiennent deux valeurs identiques.
- CONSTRAINT nom_contrainte PRIMARY KEY (col1, col2, …): l'ensemble des colonnes col1, col2, … forment la clé primaire de la table.

- CONSTRAINT nom_contrainte FOREIGN KEY (col_clé_étrangère) REFERENCES nom_table (col_référence) : l'attribut de la table en cours représente la clé étrangère qui fait référence à la clé primaire de la table indiquée.
- CONSTRAINT nom_contrainte CHECK (condition) : contrainte là où on doit obligatoirement satisfaire la condition telle qu'elle est énoncée.

Exemple :

- Création de la table Frs:

```
CREATE TABLE Frs (
num_f NUMBER(5),
nom VARCHAR2(10),
ville VARCHAR2(10),
CONSTRAINT PK_Frs PRIMARY KEY(num_f)
) ;
```

- Création de la table Article:

```
CREATE TABLE Article (
num_a NUMBER(5),
des VARCHAR2(10),
couleur VARCHAR2(10),
num_f NUMBER(5),
CONSTRAINT PK_Article PRIMARY KEY(num_a),
CONSTRAINT U_des UNIQUE(des),
CONSTRAINT FK_Article_Frs FOREIGN KEY(num_f)
REFERENCES Frs (num_f)
) ;
```

III.2. Renommer des tables

Syntaxe : Pour changer le nom d'une table, on fait recours à l'instruction suivante :

```
RENAME Ancien_Nom
TO Nouveau_Nom ;
```

Exemple : Etant donné l'entité Frs, si on souhaite la renommer par Frs1, on écrit :

```
RENAME Frs
TO Frs1 ;
```

III.3. Destruction des tables

Syntaxe : Pour supprimer une table ainsi que son contenu, on utilise l'instruction qui suit :

```
DROP TABLE nom_table ;
```

Remarque : la suppression d'une table engendre la perte des données qu'elle contient.

Exemple : Suppression de la table Frs ainsi que son contenu:

```
DROP TABLE Frs ;
```

III.4. Modification des tables

Il existe plusieurs modifications que l'on peut effectuer sur une table de base de données.

Ajout d'attribut : Après avoir créé la base de données, des tâches de maintenance semblent être parfois nécessaires. D'où l'ajout d'un nouvel attribut :

```
ALTER TABLE nom_table
ADD (col type, …) ;
```

Exemple : Etant donné la table Frs, l'ajout du champ date_naissance à cette table revient à écrire :

```
ALTER TABLE Frs
ADD (date_naissance DATE) ;
```

Modification des attributs : Après avoir créé une table de la base de données, on peut modifier le type d'un attribut en utilisant l'instruction suivante :

```
ALTER TABLE nom_table
MODIFY (col type, …) ;
```

Exemple : Modifier le nombre de chiffres du champ Prix_achat de la table Article nécessite le recours à l'instruction :

```
ALTER TABLE Article
MODIFY (Prix_achat NUMBER(12,3)) ;
```

Ajout de contraintes : Après avoir créé la base de données, on peut ajouter une nouvelle contrainte d'intégrité grâce à l'instruction suivante :

```
ALTER TABLE nom_table
ADD CONSTRAINT nom_contrainte definition_contrainte ;
```

Exemple : Ajouter une contrainte à la table Frs qui permet d'obliger des insertions de nom en majuscule :

```
ALTER TABLE Frs
ADD CONSTRAINT CK_nom CHECK (nom = UPPER(nom)) ;
```

Suppression de contraintes : Pour supprimer une contrainte, on procède comme indique la syntaxe de cette instruction :

ALTER TABLE nom_table
DROP CONSTRAINT nom_contrainte ;

Exemple : Supprimer la contrainte ck_nom de la table Frs

ALTER TABLE Frs
DROP CONSTRAINT CK_nom ;

Désactivation de contraintes : Pour désactiver une contrainte (elle est par défaut active), on procède comme indique la syntaxe de cette instruction :

ALTER TABLE nom_table
DISABLE CONSTRAINT nom_contrainte ;

Exemple : Désactiver la contrainte ck_nom de la table Frs

ALTER TABLE Frs
DISABLE CONSTRAINT CK_nom ;

Activation de contraintes : Pour activer une contrainte, on procède comme indique la syntaxe de cette instruction :

ALTER TABLE nom_table
ENABLE CONSTRAINT nom_contrainte ;

Exemple : Activer la contrainte ck_nom de la table Frs

ALTER TABLE Frs
ENABLE CONSTRAINT CK_nom ;

IV. Manipulation des données (LMD)

IV.1. Ajout de données

Syntaxe : Pour ajouter un tuple dans une table, on procède comme suit :

INSERT INTO nom_table
VALUES (valeur_attribut1, valeur_attribut2, …) ;

Ou on peut utiliser :

INSERT INTO nom_table (attribut1, attribut2, …)
VALUES (valeur_attribut1, valeur_attribut2, …) ;

Exemple : Etant donné la table Frs (num_f, nom, ville). Si on souhaite insérer les informations d'un nouvel fournisseur disposant des informations suivantes (F4, Alyani Sami, Sousse), on écrit :

INSERT INTO ETUDIANT

VALUES (‘F4’, ‘Alyani Smi’, ‘Sousse’) ;

IV.2. Modification de données

Syntaxe : Pour modifier la valeur d’un attribut relatif à un ou plusieurs tuples d’une table, on procède comme suit :

UPDATE nom_table
SET attribut1 = valeur1, attribut2 = valeur2, …
[**WHERE** condition] ;

Exemple : Etant donné la table Frs (num_f, nom, ville). Si le fournisseur Alyani Sami habite maintenant à Tunis, on écrit dans ce cas :

UPDATE Frs
SET Ville=’TUNIS’
WHERE num_f=’F4’ ;

IV.3. Suppression de données

Syntaxe : Il s’agit de supprimer un ou plusieurs tuples d’une table. Pour ce faire, on écrit :

DELETE FROM nom_table
[**WHERE** condition] ;

Exemple : Si on souhaite supprimer le fournisseur de num_f = F4 de la table Frs, on écrit :

DELETE FROM Frs
WHERE num_f=’F4’ ;

V. Interrogation de données

V.1. Généralités

Il s’agit de chercher un ou plusieurs tuples de la base de données.

Syntaxe : L’ordre SELECT possède six clauses différentes, dont seules les deux premières sont obligatoires, Elles sont données ci-dessous dans l’ordre dans lequel elles doivent apparaître quand elles sont utilisées :

SELECT…
FROM…
[WHERE…
GROUP BY…
HAVING…
ORDER BY…**] ;**

V.2. Projection

Tous les attributs d'une table :

```
SELECT *
FROM nom_table ;
```

Remarque : Il est possible de mettre le mot clé facultatif DISTINCT après l'ordre SELECT. Il permet d'éliminer les duplications : si, dans le résultat, plusieurs lignes sont identiques, une seule sera conservée.

Exemple : Liste de tous les fournisseurs

```
SELECT *
FROM Frs ;
```

Quelques colonnes :

```
SELECT col1, col2, …
FROM nom_table ;
```

Exemple : Liste des noms des Frs sans duplication

```
SELECT DISTINCT nom FROM Frs ;
```

V.3. Restriction

Les restrictions se traduisent en SQL à l'aide du prédicat « WHERE » comme suit :

```
SELECT col1, col2, …
FROM nom_table
WHERE predicat ;
```

Un prédicat simple est la comparaison de deux expressions ou plus au moyen d'un opérateur logique. Les trois types d'expressions (arithmétiques, caractères ou dates) peuvent être comparées au moyen des opérateurs d'égalité ou d'ordre (=, !=, <, <=, >, >=) :

- ✓ pour les types date, la relation d'ordre est l'ordre chronologique ;
- ✓ pour les types caractères, la relation d'ordre est l'ordre lexicographique.

Nous résumons les principales formes de restrictions dans ce qui suit :

- **WHERE** exp1 = exp2

  Exemple : Liste des fournisseurs qui s'appellent Alyani Sami

  ```
  SELECT *
  FROM Frs
  WHERE nom = 'Alyani Sami';
  ```

- **WHERE** exp1 != exp2

  Exemple : Liste des fournisseurs qui ne s'appellent pas Alyani Sami

```
SELECT *
FROM Frs
WHERE nom != 'Alyani Sami' ;
```

- **WHERE** exp1 < exp2 ou **WHERE** exp1 <= exp2

Exemple : Liste des articles ayant les Prix_achat inférieurs à 130

```
SELECT *
FROM Article
WHERE Prix_achat <= 130 ;
```

- **WHERE** exp1 > exp2 ou **WHERE** exp1 >= exp2

Exemple : Liste des articles ayant les Prix_achat supérieurs à 130

```
SELECT *
FROM FROM Article
WHERE Prix_achat >= 130 ;
```

- **WHERE** exp1 **BETWEEN** exp2 **AND** exp3

La condition est vrai si exp1 est compris entre exp2 et exp3 (bornes incluses)

Exemple : Liste des articles ayant les Prix_achat compris entre 100 et 200

```
SELECT *
FROM Article
WHERE Prix_achat BETWEEN 100 AND 200 ;
```

- **WHERE** exp1 **LIKE** exp2 :

LIKE teste l'égalité de deux chaînes :

- ✓ « _ » remplace un caractère exactement,
- ✓ « % » remplace une chaîne de caractères de longueur quelconque (y compris de longueur nulle)

Exemple : Liste des fournisseurs ayant les villes commençant par 'A' et contenant au moins 2 caractères.

```
SELECT *
FROM Frs
WHERE ville LIKE 'A_%' ;
```

- **WHERE** exp1 **IN** (exp2, exp3, …)

Le prédicat est vrai si exp1 est égale à l'une des expressions de la liste entre parenthèses.

Exemple : Liste des fournisseurs ayant les villes appartenant à la liste (Sfax, Sousse, Tunis)

```
SELECT *
```

```
FROM Frs
WHERE ville IN ('SFAX', 'SOUSSE', 'TUNIS') ;
```

- **WHERE** exp **IS NULL**

Exemple : Liste des fournisseurs dont les villes sont non définis

```
SELECT *
FROM Frs
WHERE ville IS NULL;
```

Remarque : On peut trouver, de même, les négations des prédicats BETWEEN, NULL, LIKE, IN à savoir : NOT BETWEEN, NOT NULL, NOT LIKE, NOT IN et NOT.

V.4. Expression, alias et fonctions

V.4.1. 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 mathématiques (+, -, *, /). Les expressions peuvent figurer :

- ✓ En tant que colonne résultant d'un ordre SELECT.
- ✓ Dans une clause WHERE.
- ✓ Dans une clause ORDER BY.
- ✓ Dans les ordres de manipulation des données (INSERT, UPDATE, DELETE)

V.4.2. Alias

Les alias permettent de renommer des attributs ou des tables.

Syntaxe : Pour renommer une table dans la clause FROM ainsi que des attributs dans la clause SELECT, on utilise l'instruction qui suit :

```
SELECT attribut1 AS nomalias1, attribut2 AS nomalias2, …
FROM table1 aliastab1, table2 aliasatab2;
```

Exemple :

```
SELECT num_a AS nom article
FROM Article A;
```

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.

V.4.3. Fonction

Le tableau suivant donne des principales fonctions prédéfinies :

Nom de la fonction	Rôle
AVG	La moyenne
SUM	La somme
MIN	Minimum
MAX	Maximum
COUNT(*)	Nombre de ligne
COUNT(attribut)	Nombre de valeurs non nulles de l'attribut
COUNT([DISTANCT] attribut)	Nombre de valeurs non nulles différentes de l'attribut

Exemple : Nombre de Frs:

```
SELECT COUNT(*)
FROM Frs;
```

V.5. Tri

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

```
ORDER BY exp1 [ASC], exp2 [DESC], …
```

L'option facultative **DESC** donne un tri par ordre décroissant. Par défaut, l'ordre est croissant (ASC).

Le tri se fait d'abord selon la première expression, puis les lignes ayant la même valeur pour la première expression sont triées selon la deuxième, …

Remarque : Les valeurs nulles sont toujours en tête quel que soit l'ordre du tri (ascendant ou descendant).

Exemple : Liste des articles ordonnés par ordre croissant des poids et décroissant des couleurs

```
SELECT *
FROM Article
ORDER BY poids, couleur Desc;
```

V.6. Regroupement

V.6.1. La clause GROUP BY

Il est possible de subdiviser une table en groupes, chaque groupe étant l'ensemble de lignes ayant une valeur commune.

Syntaxe :

> **GROUP BY** exp1, exp2, …

Cette clause groupe en une seule ligne toutes les lignes pour lesquelles exp1, exp2, … ont la même valeur.

Remarques :

- Cette clause se place juste après la clause WHERE, ou après la clause FROM si la clause WHERE n'existe pas.
- Des lignes peuvent être éliminées avant que le groupe ne soit formé grâce à la clause WHERE.

Exemples : Liste des fournisseurs ainsi que le nombre de leurs produits

> **SELECT** num_f, **COUNT(*)**
> **FROM** Article
> **GROUP BY** num_f;

Liste des fournisseurs ainsi que le nombre de leurs produits de désignation 'Des2'

> **SELECT** num_f, **COUNT(*)**
> **FROM** Article
> **WHERE** des = 'Des2'
> **GROUP BY** num_f;

V.6.2. La clause HAVING

HAVING sert à préciser quels groupes doivent être sélectionnés. Elle se place après la clause GROUP BY.

Syntaxe :

> **HAVING** predicat

Remarque : Le prédicat suit la même syntaxe que celui de la clause WHERE. Cependant, il ne peut porter que sur des caractéristiques de groupe (fonctions de groupe ou expression figurant dans la clause GROUP BY)

Exemple : Liste des fournisseurs ainsi que le nombre de leurs produits de désignation 'Des2' dont le nombre de produits fournis est supérieur à 2 :

```
SELECT num_f, COUNT(*)
FROM Article
WHERE des = 'Des2'
GROUP BY num_f
HAVING COUNT(*) > 2 ;
```

V.7. Opérateurs ensemblistes

Ces opérateurs sont l'union, la différence et l'intersection. La forme générale est : **REQUETE** *opérateur* **REQUETE**.

Avec *Opérateur* ::= **UNION | INTERSECT | MINUS**

V.7.1. Union

L'opérateur UNION permet de fusionner deux sélections de table pour obtenir un ensemble de lignes égal à la réunion des lignes des deux sélections. Les lignes communes n'apparaîtront qu'une fois.

Exemple : soit la table client représentée par la relation suivante :

Client (matricule, nom, tel)

Donner les noms des clients dont le matricule est cl1 ou cl2 ou cl4 et ceux des clients qui ont un numéro de téléphone :

```
SELECT nom FROM Client WHERE matricule IN ('cl1','cl2','cl4')
UNION
SELECT nom FROM Client WHERE tel IS NOT NULL;
```

V.7.2. Différence

L'opérateur MINUS permet de retirer d'une sélection les lignes obtenues dans une deuxième sélection.

Exemple : soit la table client représentée par la relation suivante :

Client (matricule, nom, tel)

Donner les noms de tous les clients sauf ceux qui n'ont pas un numéro de téléphone :

```
SELECT nom FROM Client
MINUS
```

```
SELECT nom FROM Client WHERE tel IS NULL;
```

V.7.3. Intersection

L'opérateur INTERSECT permet d'obtenir l'ensemble des lignes communes à deux interrogations.

Exemple : soit la table client représentée par la relation suivante :

Client (matricule, nom, tel)

Donner les noms des clients dont le matricule est cl1 ou cl2 ou cl4 et qui ont de numéro de tel :

```
SELECT nom FROM Client WHERE matricule IN ('cl1','cl2','cl4')
INTERSECT
SELECT nom FROM Client WHERE tel IS NOT NULL;
```

V.8. Jointure

V.8.1. Définition

Quand on précise plusieurs tables dans la clause FROM, on obtient le produit cartésien des tables.

Le produit cartésien de deux tables offre en général peu d'intérêt.

Ce qui est normalement souhaité, c'est de joindre les informations de diverses tables, en précisant quelles relations les relient entre elles. C'est la clause WHERE qui permet d'obtenir ce résultat. Elle vient limiter cette sélection en ne conservant que le sous-ensemble du produit cartésien qui satisfait le prédicat.

Exemple : Liste des désignations des articles avec les noms de leurs fournisseurs

```
SELECT des, nom
FROM Article, Frs
WHERE Article.num_f= Frs.num_f ;
```

V.8.2. Jointure d'une table à elle même

Il peut être utile de rassembler des informations venant d'une ligne d'une table avec des informations venant d'une autre ligne de la même table.

Dans ce cas, il faut renommer au moins l'une des deux tables en lui donnant un synonyme, afin de pouvoir préfixer sans ambiguïté chaque nom de colonne.

Exemple : Liste des désignations des articles qui ont le même poids en indiquant pour chaque article le numéro de son fournisseur :

```
SELECT a1.des, a2.num_f
FROM Article a1, Article a2
WHERE a1.poids = a2.poids ;
```

V.9. Requêtes imbriquées

Les requêtes imbriquées facilitent la construction des requêtes. On peut employer des requêtes imbriquées pour joindre deux requêtes.

La construction de la commande **SELECT** supérieure dépend du nombre de valeurs renvoyées par la requête imbriquée.

V.9.1. Requêtes imbriquées renvoyant une valeur unique

En utilisant les opérateurs de comparaison : =, !=, <, <=, >, >=

Attention : la sous-requête ne doit avoir qu'une seule colonne en résultat.

Exemple : Les noms des Frs ayant le même nom que celle du client numéro 'C2'

```
SELECT num_f FROM Frs
WHERE ville = (SELECT ville FROM client WHERE num_c = 'C2') ;
```

V.9.2. Prédicat IN | NOT IN

Exemple : Liste des Frs ayant fournir au moins un article.

```
SELECT nom FROM Frs F, Article A
WHERE F.num_f =A.num_f;
```

Autre solution:

```
SELECT nom FROM Frs F
WHERE F.num_f  IN ( SELECT A.num_f FROM Article A);
```

On recherche les Frs dont le numéro fait partie des numéros de Frs ayant fournir au moins un Article.

Exemple : Les Frs dont le numéro fait partie des numéros de Frs qui n'ont pas fournir aucun article de désignation Des2.

```
SELECT nom FROM Frs
WHERE Frs.num_f
NOT IN ( SELECT Article.num_f FROM Article WHERE des = 'Des2');
```

V.9.3. Le prédicat ALL

<expr> <op> ALL (<requête>)

<op> : est un opérateur de comparaison (= , !=, <, <= , >, >=).

Vrai seulement quand <expr> satisfait la comparaison <op> avec tous les résultats de la sous-requête.

Exemple : Liste des articles qui ont le prix d'achat inférieur à tous les prix de vente de tous les articles :

```
SELECT * FROM Article
WHERE prix_achat <= ALL (SELECT prix_vente FROM Vente);
```

V.10. Fonctions de regroupements

V.10.1. Regroupements

- Les colonnes du **SELECT** doivent toutes apparaître dans le **GROUP BY**.
- Les noms de colonnes dans le **HAVING** doivent aussi être dans le **GROUP BY** (ou être fonction d'agrégat).
- La clause **GROUP BY** va regrouper les lignes qui sont identiques sur les colonnes mentionnées dans le **GROUP BY**.

Exemple : soit la relation Client (nom, prenom, age, ville)

Client	nom	prenom	age	ville
	Dupont	Jean	52	Tours
	Duval	Paul	25	Tours
	Martin	Martine	39	Blois
	Bon	Jean	41	Blois

```
SELECT ville FROM client
GROUP BY ville;
```

Résultat:

Ville
Tours
Blois

```
SELECT ville , COUNT(*) nbcli FROM client
GROUP BY ville;
```

Résultat:

Ville	nbcli
Tours	2

Blois	2

```
SELECT ville , SUM(age) total FROM client
GROUP BY ville;
```

Résultat:

Ville	total
Tours	77
Blois	80

Test sur les groupes : Permet de sélectionner des groupes de la requête de regroupement.

Exemple :

```
SELECT ville , AVG(age) moyenne FROM client
GROUP BY ville
HAVING AVG(age) < 40;
```

Résultat:

Ville	moyenne
Tours	38,5

V.10.2. Différence entre where et having

- **WHERE** sélectionne les lignes de la requête avant de faire les groupes. Il agit donc avant les regroupements.
- **HAVING** sélectionne les groupes une fois qu'ils sont constitués. Il agit donc après les regroupements.

Exemple 1 : Trouver pour chaque ville la moyenne d'âge des personnes de moins de 40 ans :

```
SELECT ville , AVG(age) age FROM client
WHERE age < 40
GROUP BY ville;
```

Exemple 2 : Trouver les villes dont l'âge moyen des habitants est inférieur à 40 ans :

```
SELECT ville , AVG(age) age FROM client
GROUP BY ville
HAVING AVG(age) < 40;
```

VI. Les vues

Une vue est une table virtuelle, c'est-à-dire dont les données ne sont pas stockées dans une table de la base de données, et dans laquelle il est possible de rassembler des informations provenant de plusieurs tables. On parle de "vue" car il s'agit simplement d'une représentation

des données dans le but d'une exploitation visuelle. Les données présentes dans une vue sont définies grâce à une clause select.

Comme le résultat d'un select est lui même une table. Une telle table, qui n'existe pas dans la base mais est créée dynamiquement lors de l'exécution du select, peut être vue comme une table réelle par les utilisateurs. Pour cela, il suffit de cataloguer le select en tant que vue.

Les utilisateurs pourront consulter la base, ou modifier la base à travers la vue, c'est-à-dire manipuler la table résultat du SELECT comme si c'était une table réelle.

VI.1. Création d'une vue

Syntaxe : La commande CREATE VIEW permet de créer une vue en spécifiant le SELECT constituant la définition de la vue : (Une fois créée, une vue s'utilise comme une table.)

```
CREATE VIEW nom_vue
  [(col1 type1,
   col2 type2,
   …….)]
AS SELECT…….
```

Exemple : Vue constituant une restriction de la table EMP aux employés du département 10 :

```
CREATE VIEW EMP10
 AS SELECT * FROM EMP WHERE DEPT = 10;
```

VI.2. Supprimer une vue

Syntaxe : Une vue peut être détruite par la commande

```
DROP VIEW nom_vue;
```

Exemple :

```
DROP VIEW EMP10;
```

VI.3. Renommer une vue

Syntaxe : Une vue peut être renommé par la commande

```
RENAME ancien_nom TO nouveau_nom;
```

Exemple :

```
RENAME EMP10 TO EMP_10;
```

VII. Exercice d'application

Soit le modèle relationnel suivant :

CLIENT (numCl, nomCl, prenomCl, datenaisCl, adrCl)

PRODUIT (numPd, designPd, puPd, #numFr)

FOURNISSEUR (numFr, rsFr, emailFr, adrFr)

COMMANDE (#numCl, #numPd, dateCd, qtePc)

On considère que les types des attributs sont :

- Numcl, nomcl, prenomcl, adrcl, numpd, designpd, numfr, rsfr, emailfr, adrfr : chaines de 30 caractères.
- Datenaiscl, datecd : sont des dates
- pupd, qtepc : réel

A. Ecrire les requêtes SQL nécessaires à la création de la Base de Données décrites ci-dessus, tout en respectant le type et la longueur donnée ci-dessus pour les différents attributs, et en spécifiant les contraintes clés primaires et clés étrangères.

B. Ecrire les commandes nécessaires à l'insertion des extensions de votre choix.

C. Proposer, en langage SQL, une formulation des requêtes suivantes.

1. Déterminer les noms des clients de numéro C1.

2. Déterminer les noms et les emails des fournisseurs.

3. Déterminer les produits ayant un prix unitaire entre 200 et 500.

4. Déterminer les clients qui ont commandé des produits.

5. Déterminer les produits qui ne sont pas commandés.

6. Déterminer les noms des fournisseurs de Sfax.

7. Déterminer les noms des fournisseurs des produits commandés le 12/05/2014.

8. Déterminer les numéros des clients de même adresse.

9. Déterminer les numéros des Produits des fournisseurs de Sousse.

10. Déterminer les commandes de quantité supérieure à 3.

ANNEXE : PRÉSENTATION DE QUELQUES FONCTIONS SQL

Nous allons décrire ci-dessous les principales fonctions disponibles dans Oracle. Il faut remarquer que ces fonctions ne sont pas standardisées et ne sont pas toutes disponibles dans les autres SGBD.

I. Fonctions des chaînes de caractères

Fonctions	Explication
LOWER (col|valeur)	Retourne la colonne ou la valeur en minuscule.
UPPER (col|valeur)	Retourne la colonne ou la valeur en majuscule.
INITCAP (col|valeur)	Met en majuscule la première lettre de chaque mot de la colonne ou de la valeur.
CONCAT (char1, char2)	Retourne char1 concaténée avec char2 (on peut utiliser aussi l'opérateur ||).
LPAD (col|valeur, n, 'caractère')	Retourne la colonne ou la valeur complétée à gauche par une séquence de 'caractère' dont la longueur totale est n (si le caractère n'est pas spécifié, le caractère vide est utilisé).
RPAD (col|valeur, n, 'caractère')	Retourne la colonne ou la valeur complétée à droite par une séquence de 'caractère' dont la longueur totale est n (si le caractère n'est pas spécifié, le caractère vide est utilisé). Si la valeur contenue dans la colonne à un longueur > n, on retourne la position
SUBSTR (col|valeur, pos, n)	Retourne une portion de n caractères de la colonne (valeur) commençant à la position pos.
INSTR (col|valeur, 'string')	Retourne la position de la première occurrence de la chaîne 'string'.
INSTR (col|valeur, 'string', pos, n)	Retourne la position de la n^{éme} occurrence de la chaîne 'string' en commençant à la position pos.
LTRIM (col|valeur, 'string', CHAR(s))	Supprime à partir du gauche les occurrences de tête d'un caractère (ou une combinaison de caractères) spécifié.
RTRIM (col|valeur, 'string', CHAR(s))	Supprime à partir du droite les occurrences de tête d'un caractère (ou une combinaison de caractères) spécifié.

LENGTH (col|valeur)	Retourne la longueur d'une chaîne.
TRANSLATE (col|valeur, 'source', 'cible')	Permet de remplacer les caractères de la chaîne source par ceux de la chaîne cible en respectant l'ordre. TRANSLATE permet de faire un remplacement caractère par caractère. TRANSLATE (ENAME, 'AE', 'A') : élimine tous les 'E' contenues dans les noms des employés. TRANSLATE (ENAME, 'E', ' ') : cette instruction n'est pas permise.
TRANSLATE (col|valeur, 'chaîne1', 'chaîne2')	Permet de remplacer la chaîne de caractères 'chaîne1' par la chaîne de caractères 'chaîne2'. REPLACE permet de faire un remplacement d'une chaîne par une autre. Si on ne spécifie pas la chaîne de remplacement, il y a élimination de la chaîne source.

II. Fonctions des nombres

Fonctions	Explication
ROUND (col|valeur, n)	Arrondit col ou valeur à la précision n (0 par défaut) : ROUND (45.923, 1) retourne 45.9 ROUND (45.923) retourne 46 ROUND (45.4) retourne 45 ROUND (45.323, 1) retourne 45.3 ROUND (42.323, -1) retourne 40 ROUND (1500, -3) retourne 2000 ROUND (1250, -3) retourne 1000
TRUNC (col|valeur, n)	Tronque col ou valeur à la précision n (0 par défaut) : TRUNC (45.923, 1) retourne 45.9 TRUNC (45.923) retourne 45 TRUNC (45.4) retourne 45 TRUNC (45.323, 1) retourne 45.3 TRUNC (42.323, -1) retourne 40 TRUNC (1500, -3) retourne 1000

	TRUNC (1250, -3) retourne 1000
CELL (col|valeur)	Retourne le plus petit entier >= col (valeur)
FLOOR (col|valeur)	Retourne le plus grand entier <= col (valeur)
POWER (col|valeur, n)	Retourne col ou valeur à la puissance de n.
EXP (n)	Retourne la valeur exponentielle de n.
SQRT (col|valeur)	Retourne la racine carrée de col ou valeur.
SIGN (n)	Retourne -1 si n<0, 1 si n>0, 0 si n=0.
ABS (col|valeur)	Retourne la valeur absolue de col ou valeur.
MOD (valeur1, valeur2)	Retourne le reste de la division entière de valeur1 par valeur2.
LOG (m,n)	Retourne le logarithme à base de m de n.
SIN(n), TAN (n), COS (n)	Retourne le sinus, tangente ou cosinus de n.

III. Fonctions sur les dates

Fonctions	Explication
SYSDATE	Retourne la date et l'heure courante.
Date + number	Ajoute un nombre de jours à une date et produit une novelle date.
Date – number	Soustrait un nombre de jours de la date et produit une novelle date.
Date + number/24	Ajoute un nombre d'heures pour une date.
Date1 – Date2	Soustrait une date d'une autre en produisant un nombre de jours.
MONTHS_BETWEEN (date1, date2)	Retourne le nombre de mois entre date1 et date2. Si date1> date2, le résultat est positif, si non le résultat est négatif.
ADD_MONTHS (date, n)	Ajoute n mois à la date.
NEXT-DAY (date1, 'char') NEXT-DAY(date1, n) : 1<= n <= 7	Retourne la date du prochain jour spécifié suivant la date date1.
LAST-DAY (date1)	Retourne la date du dernier jour du mois courant.
ROUND (date, 'MONTH')	ROUND ('16-DEC-89', 'MONTH') retourne '01-JAN-90'.

	ROUND (’18-OCT-04’, ‘MONTH’) retourne ’01-NOV-04’. ROUND (’14-OCT-05’, ‘MONTH’) retourne ’01-OCT-05’.
ROUND (date, ‘YEAR’)	ROUND (’15-MAY-04’, ‘YEAR’) retourne ’01-JAN-04’. ROUND (’15-JUL-04’, ‘YEAR’) retourne ’01-JAN-05’.
TRUNC (date, ‘MONTH’)	Retourne la date du premier jour du mois courant.
TRUNC (date, ‘YEAR’)	Retourne la date du premier jour de l’année courante.

IV. Fonctions de conversions

- **TO_CHAR**(date, [‘format’]) : convertit une date du format par défaut (DD-MMM-YY) à un autre format spécifié.

  TO_CHAR (’05-SEP-89’, ‘DAY, DDTH MONTH YYYY’) retourne TUESDAY, 05^{TH} SEPTEMBER 1989.

  TO_CHAR (’02-OCT-04’, ‘DAY, DD-MM-YYYY’) retourne THURSDAY, 02-10-2004.

  TO_CHAR (‘JUNE 4, 1984’, ‘HH:MI:SS’) retourne 08:17:24.
- **TO_NUMBER(chaîne) :** fonction utilisée pour transformer une chaîne de caractères (quand la chaîne de caractères est composée de caractères numériques) en un type de données numériques : TO_NUMBER(‘1500’) retourne 1500
- **TO_DATE(chaîne, format) :** transforme une chaîne de caractères ‘chaîne’ en donnée de type date en format par défaut (de type date). Le format est identique à celui de la fonction TO_CHAR : TO_DATE (‘JUNE 4, 1984’, ‘MONTH dd, YYYY’) retourne ’04-JUN-84’.

V. Fonctions qui acceptent n’importe quel type de données

- **NVL (col|valeur, val) :** convertit une valeur NULL en val.
- **GREATEST (col|valeur1, col|valeur2, …) :** retourne la plus grande valeur de la liste spécifié : GREATEST (1000, 1200, 2000) retourne 2000.
  - LEAST (col|valeur1, col|valeur2, …) : retourne la plus petite valeur de la liste spécifié : LEAST (1000, 1200, 2000) retourne 1000.

BIBLIOGRAPHIES

1) Baghdadi ZITOUNI « Bases de Données Relationnelles et Objet-Relationnelles – Théorie et Applications sous Oracle », Janvier 2016

2) Chedly FEHRI « Cours Base de données », Ecole Nationale des Ingénieurs de Sfax, 2005/2006.

3) Georges GARDARIN « Bases de données » Eyrolles 2003.

4) Thomas CONNOLLY, Carolyn BEGG « Systèmes de bases de données. Approche pratique de la conception, de l'implémentation et de l'administration (cours et exercices)» Les éditions Reynald Goulet Inc. 2005.

5) Pierre CRESCENZO « Support de cours magistraux de Bases de données » disponible sur le site : http://www.crescenzo.nom.fr/CMBasesDeDonnees

6) Jean-Pierre CHEINEY, Philippe PICOUET, Jean-Marc SAGLIO « Systèmes de Gestion de Bases de Données » disponible sur le site : « http://www.bd.enst.fr/polyv7 ».

7) Frédéric BROUARD, Christian SOUTOU « SQL : Synthèse de cours & exercices corrigés». Collection Synthex. Pearson Education 2005.

8) Roger CHAPUIS « Les bases de données. ORACLE 8i. Développement, administration, optimisation » Dunod, 2001.

9) Richard GRIN « Polycopié Langage SQL » «http://deptinfo.unice.fr/~grin/messupports» visité le 07/12/2006.

10) Mejdi BLAGHGI & Anis ASSÈS, « Support de cours Base de données », ISET Jerba, 2007.