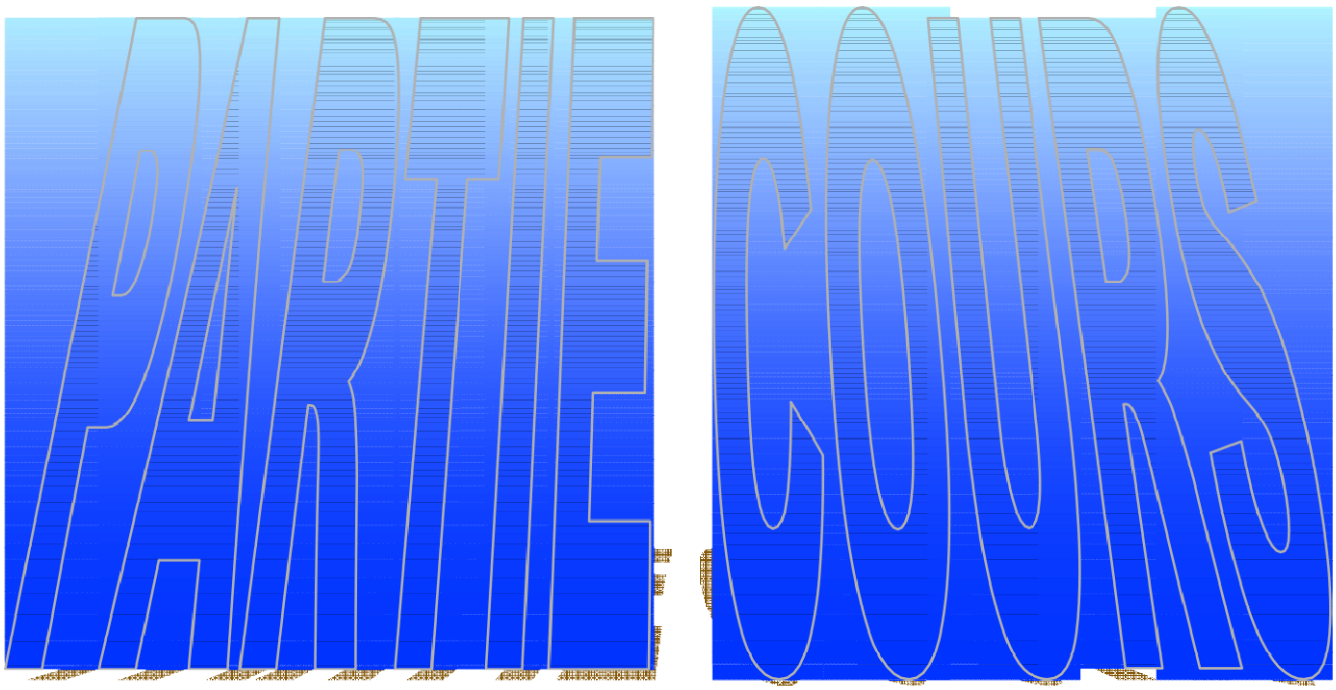
SOMMAIRE

Liste des figures


PARTIE COURS

Introduction Générale

Jusqu'aux années 60 les bases de données sont organisées d'une façon classique en fichiers, à la fin des années 60 les premiers SGBD (Système de gestion de file d'attente) sont apparu, **les système réseaux** et **hiérarchiques**.

À partir de 1970 la deuxième génération de SGBD est apparue, **les systèmes relationnels**. Au début des années 80 on a eu la troisième génération de SGBD **les systèmes orientés objet**.

L'organisation en fichier présente des limites :

- Particularité de la saisie et des traitements en fonction des fichiers ➔ Un ou plusieurs programmes par fichier.
- Contrôle en différé des données ➔ Augmentation des délais et du risque d'erreur.
- Particularisation des fichiers en fonction du traitement ➔ Grande redondance des données.

L'organisation base de données est meilleure :

- Uniformisation de la saisie et standardisation des traitements.
- Contrôle immédiat de la validité des données.
- Partage de données entre plusieurs traitements ➔ limitation de la redondance des données.

Chapitre 1 : Introduction aux bases de données

Introduction :

Les bases de données sont actuellement au coeur du système d'information des entreprises. Les systèmes de gestion de bases de données, initialement disponibles uniquement sur des "mainframes", peuvent maintenant être installés sur tous les types d'ordinateurs y compris les ordinateurs personnels. Ce chapitre répond au questions : Qu'est-ce qu'une base de données? Que peut-on attendre d'un système de gestion de bases de données?

I- Généralité sur les bases de données (BD):

I.1 Définition :

Il est difficile de données une définition exacte de la notion de base de données. Une définition très générale pourrait être :

Définition I.1 (Base de données) **:** Une Base de données est Un ensemble organisé d'informations avec un objectif commun.

Ici le support pour rassembler et stocker les données n'est pas important ((papier, fichiers, etc.), dès lors que des données sont rassemblées et stockées d'une manière organisée dans un but spécifique, on parle de base de données.

Plus précisément, on appelle base de données un ensemble structuré et organisé qui permet de stocker de grandes quantités d'informations afin d'en faciliter l'exploitation (ajout, mise à jour, recherche de données). Bien entendu, dans le cadre de ce cours, nous nous intéressons aux bases de données informatisées.

Définition I.2 (Base de données informatisée) **:** Une base de données est un ensemble structuré de données enregistrées sur des supports accessible par l'ordinateur représentant des informations du monde réel et pouvant être interrogées et mises à jour par une communauté d'utilisateurs.

I.2 Différentes BD :

- **Base de données personnelle :** sa taille est entre 10 KO et 100 KO elle peut être implémenter par un SGBD tel que Ms Access.

- **Base de données professionnelle typiques :** sa taille est entre 100 KO et 100 GO et peut être implémenter par un SGBD tel que SQL Server.
- **Base de données professionnelle très grande :** c'est une VLDB (Very Large Data Base) sa taille est très grande elle dépasse 100 GO et implémenter par un SGBD tel que Oracle.

I.3 Composantes d'une BD :

On à deux types de composantes d'une BD :

- Logiciel :
  - SGBD : ensemble de logiciels gérant une BD.
  - Les outils frontaux (4-GL).
    - ✓ Générateurs : de formes, de rapports, des applications ;
    - ✓ Interfaces WEB : HTML, XML…
    - ✓ Etc.
  - Utilitaires :
    - ✓ Chargement ;
    - ✓ Statistiques ;
    - ✓ Aide à la conception.
- Matériel :
  - Ordinateur : avec son CPU, RAM, disque pour la base, bandes pour la sauvegarde.

  Ou bien

  - Machine spécialisée :
    - ✓ Ne supporte que la BD ;
    - ✓ En général multiprocesseur ;
    - ✓ Les applications sont sur d'autres ordinateurs ;
    - ✓ Liaison par LAN (Local Area Network).

II- Les Système de gestion d'une base de données :

II.1 Définition :

Un système de gestion de base de données (SGBD) est un logiciel assurant la structuration, stockage, mise à jour et consultation des données d'une BD (voir figure I.1).

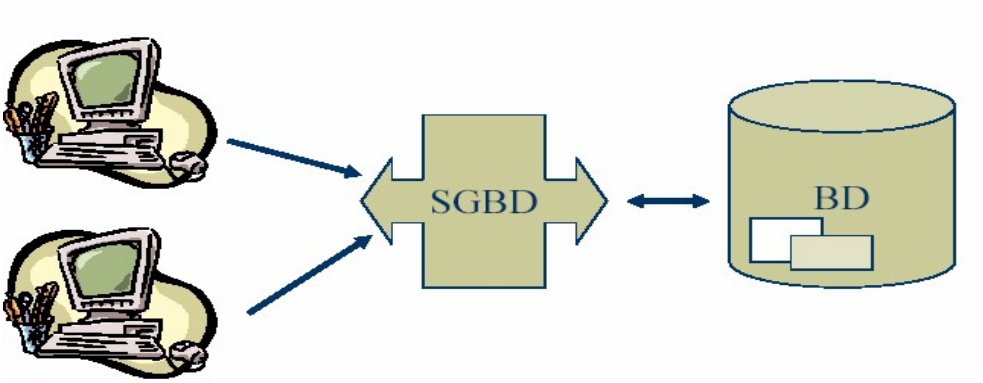


Figure I-1 : SGBD

II.2 Différents types de SGBD:

- SGBD Hiérarchique :
  - ✓ Les données sont représentées dans la base sous forme d'arbre.
- SGBD Réseau :
  - ✓ Les données sont représentées dans la base sous forme d'un graphe quelconque.
- SGBD Relationnel :
  - ✓ Fondée sue la théorie mathématique des relations ;
  - ✓ Représentation très simple des données ;
  - ✓ Manipulation d'un langage non procédural, puissant et simple d'emploi. (SQL est un standard parmi ces langages).
  - ➔ Les SGBDR dominent le marché des SGBD.
- SGBD Objet :
  - ✓ Enregistre les données sous forme d'objets.

II.3 Les utilisateurs des SGBD:

On a 3 types d'utilisateurs :

- **Utilisateurs Interactifs :**
  - ✓ Cherche les données sans connaître la BD ;

- ✓ Il a une interface visuelle pour consulter et manipuler les données.

Exemple : Secrétaire, caissière…

- **Concepteur et programmeurs d'application :**
  - ✓ Construisent les interfaces pour les utilisateurs interactifs ;
  - ✓ Spécialistes de SQL.

Exemple : Un informaticien qui connaît bien les SGBD et un au plusieurs langage de programmation.

- **Administrateur : DBA**
  - ✓ Définit et maintient la BD ;
  - ✓ A la priorité sur tous les autres utilisateurs.

Exemple : Un ingénieur technicien en informatique.

II.4 Les fonctions des SGBD :

- Description des données : langage de définition de données (LDD);
- Recherche des données;
- Mise à jour des données;
- Transformation des données;

} Langage de manipulation de données (LMD)

- Contrôle de l'intégrité des données ➔ Respect des contraintes d'intégrité;
- Gestion des accès concurrents;
- Sécurité et confidentialité des données;
- Gestion des pannes;
- Optimisation d'accès aux données;
- Outils d'administration de la BD (sauvegarde et restauration des données…);
- …

II.5 Architecture d'un SGFA :

II.5.1 Architecture fonctionnelle :

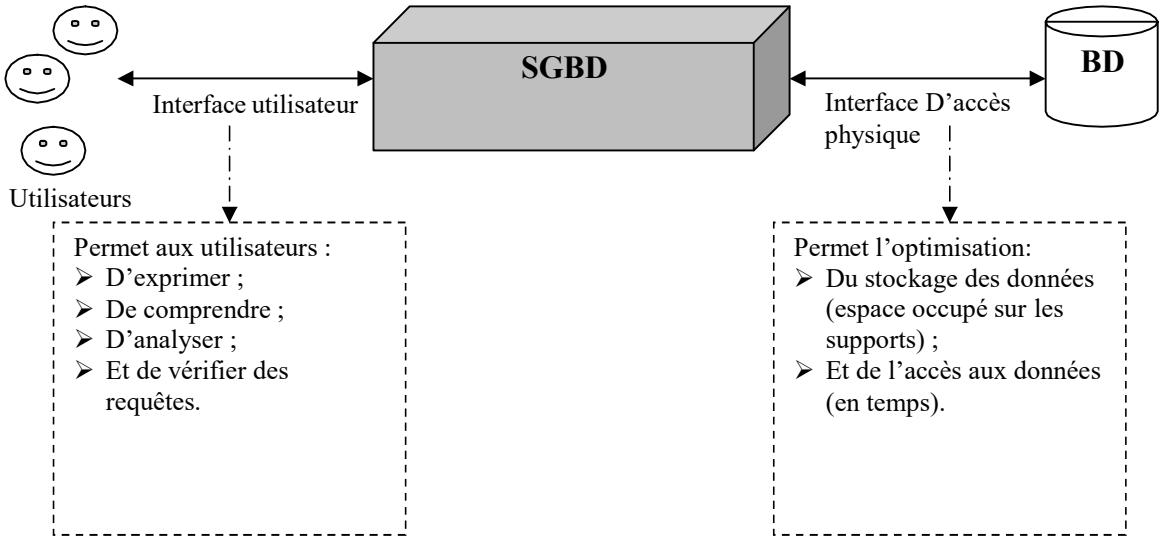


Figure I-2 : Architecture fonctionnelle

II.5.2 Architecture ANSI/SPARC (voir figure I.3):

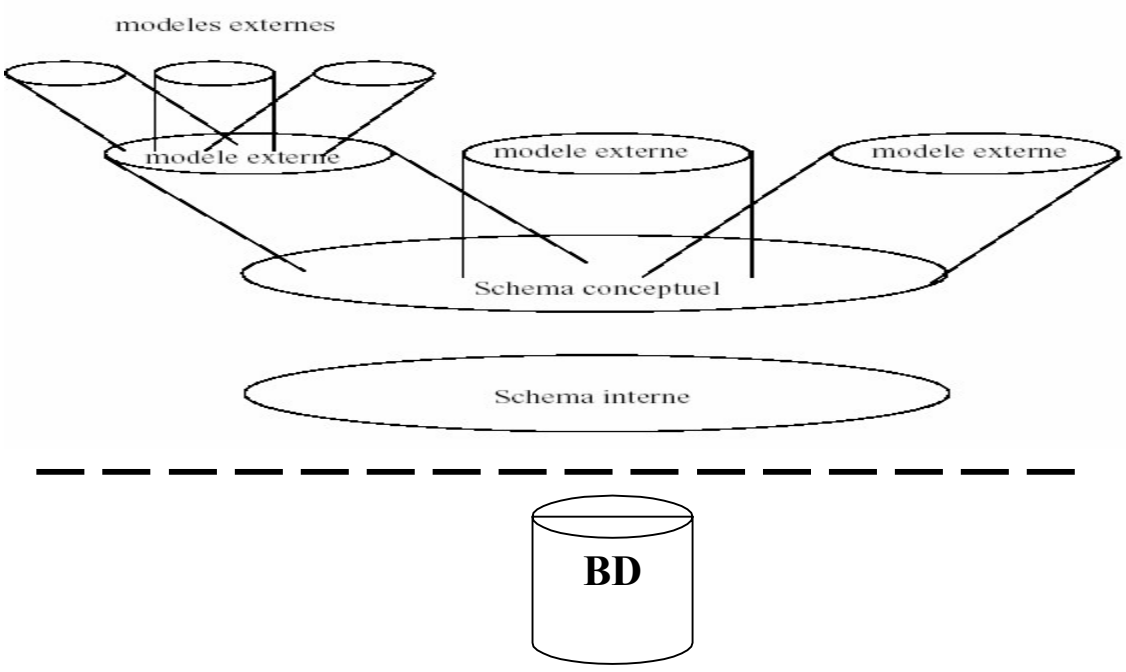


Figure I-3 : Architecture ANSI/SPARC

- **Schéma conceptuel :**(couche logique) c'est le niveau centrale, il assure les fonctions de contrôle global :
  - ✓ Optimisation des requêtes ;
  - ✓ Gestion des conflits d'accès simultanés ;
  - ✓ Contrôle général de la cohérence de l'ensemble ;
  - ✓ …

➔ L'analyse du réel est le domaine des méthodes de conception de la BD : c'est la définition logique de la BD.

Exemple de schéma Conceptuel :

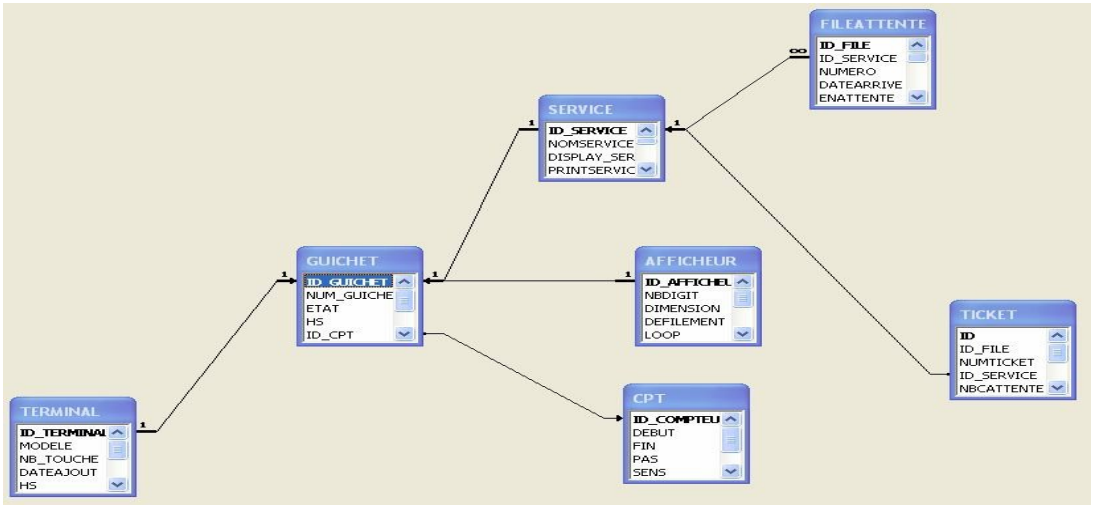


Figure I-4 : Schéma conceptuel

- **Schéma Interne :**(Couche interne) Définit la représentation interne de la BD, il s'occupe :
  - Du stockage des données dans les supports physique (Les disques,..) ;
  - Et la gestion des structures de mémorisation (fichiers) et accès (gestion des index, des clés,…) ;
  - Nécessite au préalable le choix d'un SGBD.
- **Schéma externe :(**couche 1 : externe) il prennent 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.

Exemple de schéma externe:

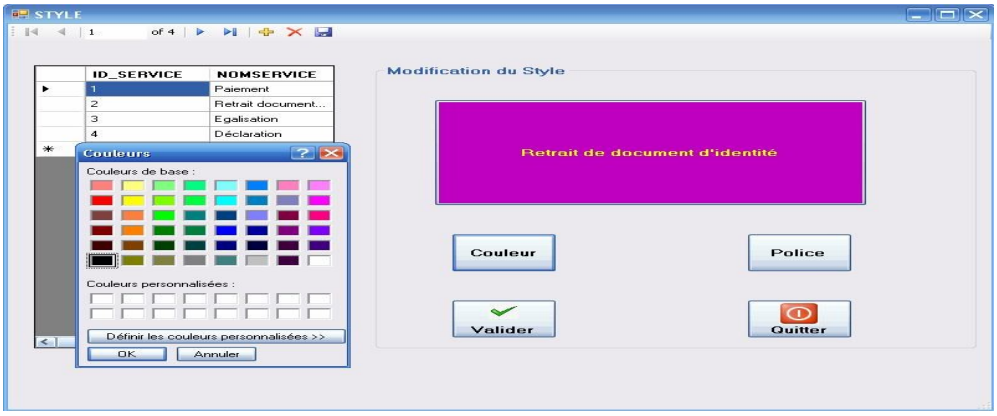


Figure I-5 : Schéma externe

II.5.3 Architecture Opérationnelle :

On a 2 architectures opérationnelles :

- Architecture Client/Serveur : c'est une architecture hiérarchisée mettant en jeu d'une part un serveur de données et d'autre part des clients supportant les applications et la présentation, et dans laquelle les clients dialoguent avec le serveur via un réseau ;
- Architecture Réparties (Distribués) : elle est composée de plusieurs serveurs coopérant à la gestion de bases de données composées de plusieurs sous bases gérées par un seul serveur, mais apparaissant comme des bases uniques centralisées pour l'utilisateur.

III- Processus de conception de base de données :

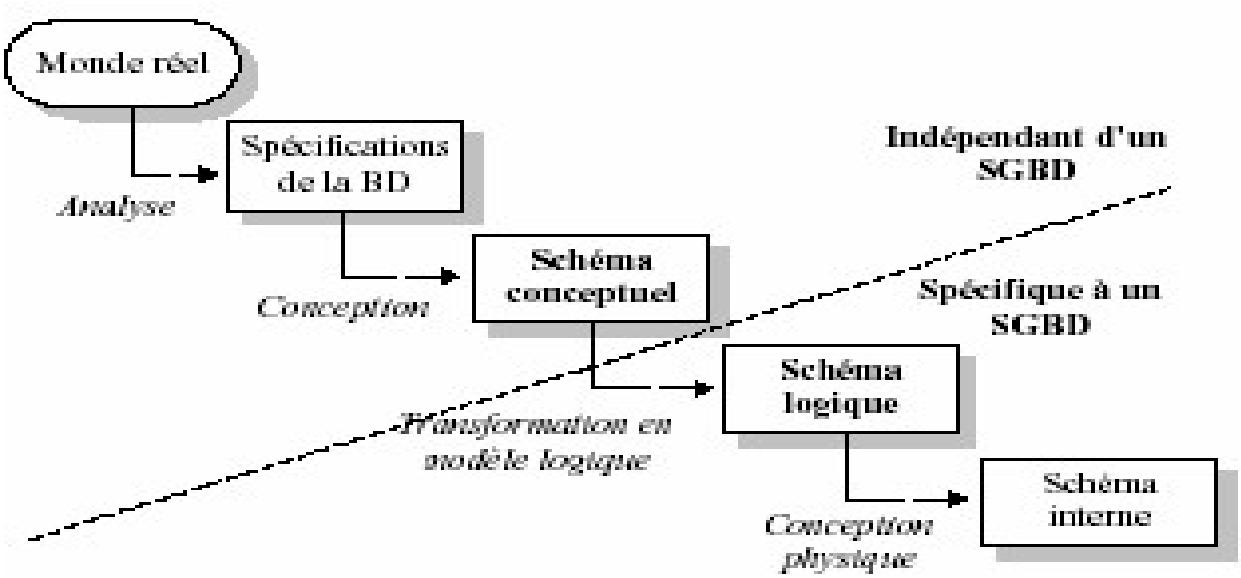


Figure I-6 : Processus de conception d'une BD.

IV- Cycle de Vie d'une BD :

IV.1 Conception :

Phase d'analyse qui permet de sélectionner des données qui seront stockées dans la future BD.

IV.2 Implantation :

Cette phase consiste à choisir le SGBD, Traduire la description de la BD élaborée dans la phase de conception en une description compréhensible par le SGBD (Compilateur), puis l'écrire dans le SGBD.

IV.3 Exploitation :

Traduire la description élaborée dans la phase d'implantation en une description physique en terme de fichiers, d'index…

Chapitre 2 : Le Modèle Entité-Relation

Introduction :

Ce chapitre présente le modèle Entité/Association (E/A) qui est utilisé à peu près universellement pour la ***conception*** de bases de données (relationnelles principalement). La conception d'un schéma correct est primordiale pour le développement d'une application sûr et correcte. Dans la mesure où la base de données est la base de tout le système, une erreur de conception est difficilement récupérable par la suite. Le modèle E/A a pour caractéristiques d'être simple et suffisamment puissant pour représenter des structures relationnelles.

I- Eléments constitutifs du modèle Entité Association :

Le modèle entités-associations est constitué de 3 concepts :

- L'objet ou entité ;
- L'association ;
- La propriété.

L'objet est une entité ayant une existence propre. L'association est un lien ou relation entre objets sans existence propre. La propriété est la plus petite donnée d'information décrivant un objet ou une association.

I.1 Entité :

Il est difficile de donner une définition très précise des entités. On va donner alors une approche.

I.1.1 Définition (entité):

Une entité est un objet, une chose concrète ou abstraite qui peut être reconnue distinctement et qui est caractérisée par son unicité.

Exemples d'entité : Marouene kefi, Bertrand Marchand, le plat sur la table, etc.

I.1.2 Définition (type-entité):

Un type-entité désigne un ensemble d'entités qui possèdent une sémantique et des propriétés communes.

I.2 Attribut ou propriété, valeur :

I.2.1 Définition (attribut, propriété):

Un attribut (ou une propriété) est une caractéristique associée à un type-entité ou à un type-association.

Exemples d'attribut : le nom d'une personne, le titre d'une livre, la couleur d'une voiture.

I.2.2 Définition (Valeur):

C'est un domaine qui définit l'ensemble des valeurs possibles qui peuvent être prise par chaque attribut d'un type-entité ou d'un type-association, pour l'entité chaque attribut possède une valeur compatible avec son domaine.

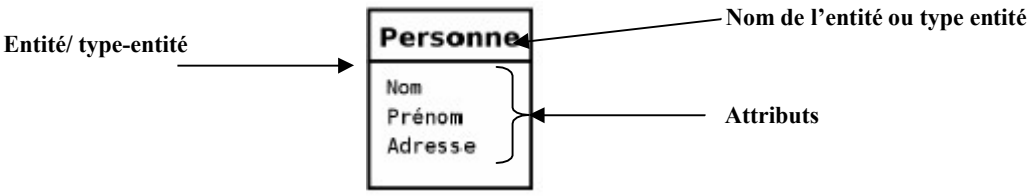


Figure II-1 : Représentation graphique d'un exemple de type-entité comportant 3 attributs.

I.2.3 Règles associées :

1ére règle : Un attribut ne doit jamais être partagé par plusieurs type-entité ou type-association.

2éme règle : Un attribut est une donnée élémentaire, donc pas de donnée calculée ou dérivée.

3éme règle : Un type-entité et ses attributs doivent être cohérents entre eux (i.e. ne traiter que d'un seul sujet).

Par exemple, si le modèle doit comporter des informations relatives à des articles et à leur fournisseur, ces informations ne doivent pas coexister au sein d'un même type-entité. Il est préférable de mettre les informations relatives aux articles dans un type-entité ***Article*** et les informations relatives aux fournisseurs dans un type-entité ***Fournisseur***. Ces deux type-entités seront probablement ensuite reliés par un type-association.

I.3 Identifiant ou clé:

I.3.1 Définition (identifiant, clé):

Un identifiant (ou clé) d'un type-entité ou d'un type-association est constitué par un ou plusieurs de ses attributs qui doivent avoir une valeur unique pour chaque entité ou association de ce type.

➔ Il est donc impossible que les attributs constituant l'identifiant d'un type-entité (respectivement type-association) prennent la même valeur pour deux entités (respectivement deux associations) distinctes. Exemples d'identifiant : le numéro de la carte d'identité national pour une personne, le numéro d'immatriculation pour une voiture, le code ISBN d'un livre pour un livre.

I.3.2 Règle associée :

Chaque type-entité possède au moins un identifiant, éventuellement formé de plusieurs attributs.

Ainsi, chaque type-entité possède au moins un attribut qui, s'il est seul, est donc forcément l'identifiant.

I.4 Association ou relation:

I.4.1 Définition (Association, relation):

Une association (ou une relation) est un lien entre plusieurs entités.

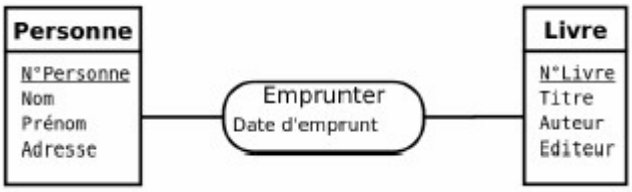


Figure II-2 : Représentation graphique d'un exemple de type-association liant 2 type-entité.

I.4.2 Définition (type-association):

Un type association (ou un type-relation) désigne un ensemble de relations qui possèdent les mêmes caractéristiques. Le type-association décrit un lien entre plusieurs type-entités. Les associations de ce type-association lient des entités de ces type-entités.

I.4.3 Définition (participant):

Les types entités intervenant dans un type-association sont appelés les participants de ce type-association.

I.4.4 Définition (collection):

L'ensemble des participants d'un type-association est appelé la collection de ce type-association.

I.4.5 Définition (dimension d'un type-association):

La dimension d'un type-association est le nombre de type-entités contenu dans la collection.

I.4.6 Règles Associées:

1ére règle **:** Un attribut peut être placé dans un type-association uniquement lorsqu'il dépend de toutes les entités liées par le type-association.

2éme règle **:** La concaténation des identifiants des type-entités liés à un type-association constitue un identifiant de ce type-association et cet identifiant n'est pas mentionné sur le modèle (il est implicite).

I.4.7 Différents types d'associations :

- **Association plurielle** : deux mêmes entités peuvent être plusieurs fois en association.

Figure II-3 : Représentation graphique d'un exemple d'association plurielle.

- **Association réflexive** : Un type-association est réflexif lorsqu'il relie un type-entité et lui-même.

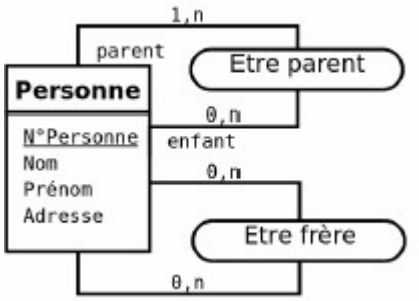


Figure II-4 : Représentation graphique d'un exemple d'association réflexive.

➢ **Association n-aire :** Ce type-association met en relation n type-entités.

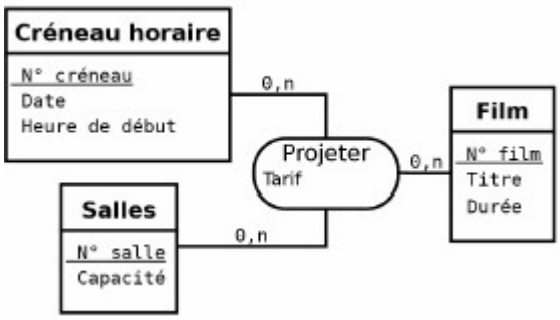


Figure II-5 : Représentation graphique d'un exemple d'association n-aire.

I.5 Cardinalité:

I.5.1 Définition (cardinalité) :

La cardinalité d'une patte reliant un type-association et un type-entité précise le nombre de fois minimal et maximal d'interventions d'une entité du type-entité dans une association du type-association. La cardinalité minimale doit être inférieure ou égale à la cardinalité maximale.

Exemple : une personne peut être l'auteur de 0 à n livre, mais un livre ne peut être écrit que par une personne.

Figure II-6 : Représentation graphique des cardinalités d'un type-association.

I.5.2 Règles associées :

1ére règle : L'expression de la cardinalité est obligatoire pour chaque patte d'un type-association.

2éme règle : Une cardinalité minimal est toujours 0 ou 1 et une cardinalité maximale est toujours 1 ou n.

Ainsi, si une cardinalité maximale est connue et vaut 2, 3 ou plus, alors nous considérons qu'elle est indéterminée et vaut n. En effet, si nous connaissons n au moment de la conception, il se peut que cette valeur évolue au cours du temps. Il vaut donc mieux considérer n comme inconnue dès le départ. De la même manière, on ne modélise pas des cardinalités minimales qui valent plus de 1 car ces valeurs sont également susceptibles d'évoluer. Enfin, une

cardinalité maximale de 0 n'a pas de sens car elle rendrait le type-association inutile.

Les seuls cardinalités admises sont donc :

0,1 : une occurrence du type-entité peut exister tout en étant impliquée dans aucune association et peut être impliquée dans au maximum une association.

0, n : c'est la cardinalité la plus ouverte ; une occurrence du type-entité peut exister tout en étant impliquée dans aucune association et peut être impliquée, sans limitation, dans plusieurs associations.

1,1 : une occurrence du type-entité ne peut exister que si elle est impliquée dans exactement (au moins et au plus) une association.

1, n : une occurrence du type-entité ne peut exister que si elle est impliquée dans au moins une association.

II- Normalisation des type-entités et type-associations :

Les formes normales permettent de minimiser les redondances pour améliorer les performances de la BD. La normalisation peut être aussi bien effectuée sur un modèle entités-associations, où elle s'applique sur les type-entités et type-associations, que sur un modèle relationnel.

Il existe 5 formes normales principales et deux extensions. Plus le niveau de normalisation est élevé, plus le modèle est contient moins de redondances. Un type-entité ou un type-association en forme normale de niveau n est automatiquement en forme normale de niveau n-1. Une modélisation rigoureuse permet généralement d'aboutir directement à des type-entités et type-associations en forme normale de Boyce-Codd.

II.1 1ére forme normale (1FN):

Un type-entité ou un type-association est en première forme normale si tous ses attributs sont élémentaires, c'est-à-dire non décomposables.

Exemple :

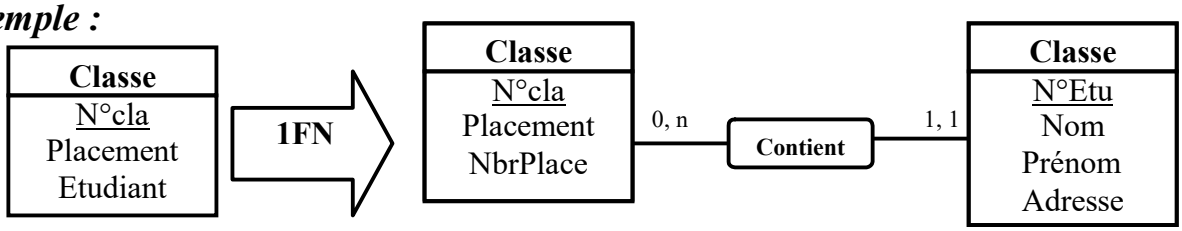


Figure II-7 : Exemple de normalisation en 1FN.

II.2 2éme forme normale (2FN):

Un type-entité ou un type-association est en deuxième forme normale si, et seulement si, il est en première forme normale et si tout attribut n'appartenant pas à la clé dépend de la totalité de cette clé.

Exemple : On suppose dans cet exemple qu'un même fournisseur peut fournir plusieurs produits et qu'un même produit peut être fourni par différents fournisseurs.

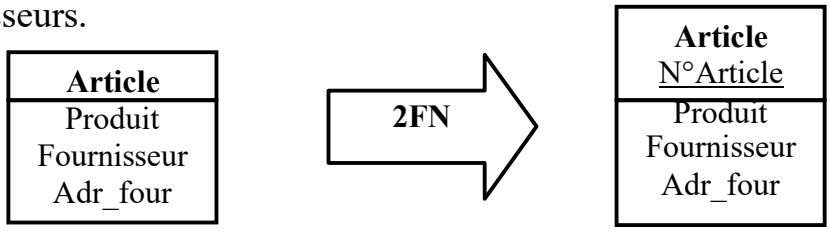


Figure II-7 : Exemple de normalisation en 2FN.

II.3 3éme forme normale (3FN):

Un type-entité ou un type-association est en troisième forme normale si, et seulement si, il est en deuxième forme normale et si tous ses attributs dépendent directement de sa clé et pas d'autres attributs.

Exemple : Dans cet exemple, l'attribut Adresse fournisseur dépend de l'attribut Fournisseur.

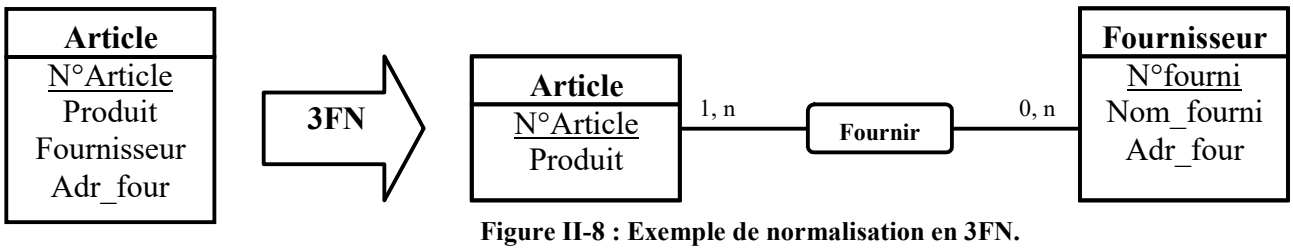


Figure II-8 : Exemple de normalisation en 3FN.

II.4- Forme normale de boyce-Codd (BCNF):

Un type-entité ou un type-association est en forme normale de Boyce-Codd si, et seulement si, il est en troisième forme normale et si aucun attribut faisant partie de la clé dépend d'un attribut ne faisant pas partie de la clé.

Exemple : on suppose ici qu'un institut délivre un seul diplôme, donc l'attribut institut détermine l'attribut diplôme et par suite il faut la changer comme suit.

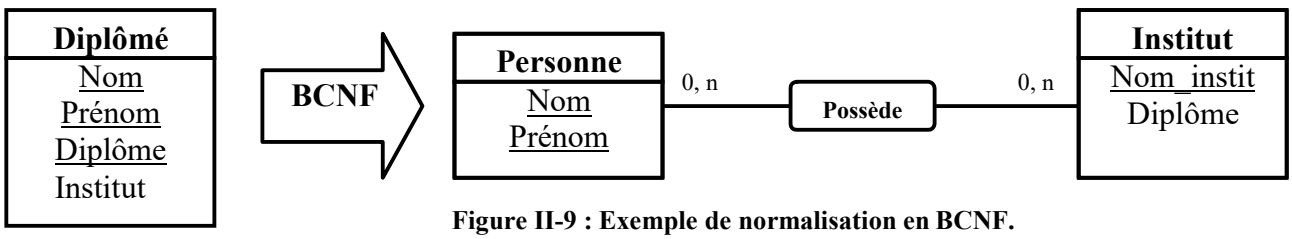


Figure II-9 : Exemple de normalisation en BCNF.

Chapitre 3 : Le Modèle Relationnel

Introduction :

Une base de données relationnelle est une base de données structurée suivant les principes de l'algèbre relationnelle. Le modèle relationnel est un modèle logique associé aux SGBD relationnels, Dans ce modèle, les données sont représentées par des tables, sans préjuger de la façon dont les informations sont stockées dans la machine.

I- Eléments du modèle Relationnel :

I.1- Attribut :

Les attributs nomment les colonnes d'une relation. Ils servent à la fois à indiquer le contenu de cette colonne, et à la référencer quand on effectue des opérations. Un attribut est toujours associé à un domaine. Le nom d'un attribut peut apparaître dans plusieurs schémas de relations.

Exemple : Le nom d'une personne, le numéro de la CIN, couleur d'une voiture…

I.2- Domaine :

Un domaine est un ensemble de valeurs atomiques que peut prendre un attribut.

Exemple :

- Dnom : chaîne de caractères de longueur maximale 30.
- Dnum : entier compris entre 0 et 99999.
- Dcouleur : {« bleu », « vert », « jaune »}
- Dâge : entiers compris entre 16 et 65.

I.3- Relation :

Une relation est un ensemble d'attributs, chaque attribut Ai prend ses valeurs dans un domaine Dom (Ai).

Exemple :

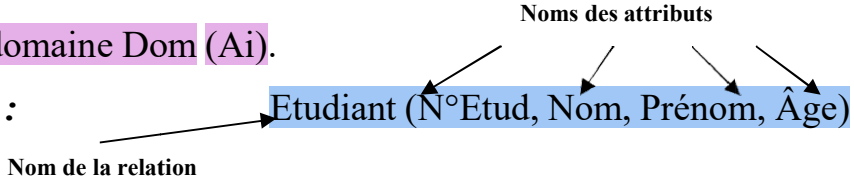


Figure III-1 : Exemple de Relation.

I.4- Schéma de relation :

Un schéma de relation est simplement un nom suivi de la liste des attributs, chaque attribut étant associé à son domaine. La syntaxe est donc :

R(A1 : D1, A2 : D2, …, An : Dn)

Où les Ai sont les noms des attributs et les Di les domaines. L'arité d'une relation est le nombre de ses attributs.

Remarque : On peut trouver dans un schéma de relation plusieurs fois le même domaine, mais une seule fois un nom d'attribut. Le domaine peut être omis en phase de définition.

Exemple : Etudiant (N°Etud : Entier, Nom : Chaîne, Prénom : Chaîne, Âge : Entier).

I.4.1 Degré :

Le degré d'une relation est son nombre d'attributs.

I.4.2 Occurrence ou n-uplets ou tuples :

Une occurrence, ou n-uplets, ou tuples, est un élément de l'ensemble figuré par une relation. Autrement dit, une occurrence est une ligne du tableau qui représente la relation.

Exemple : Etudiant (N°Etud, Nom, Prénom, Âge)

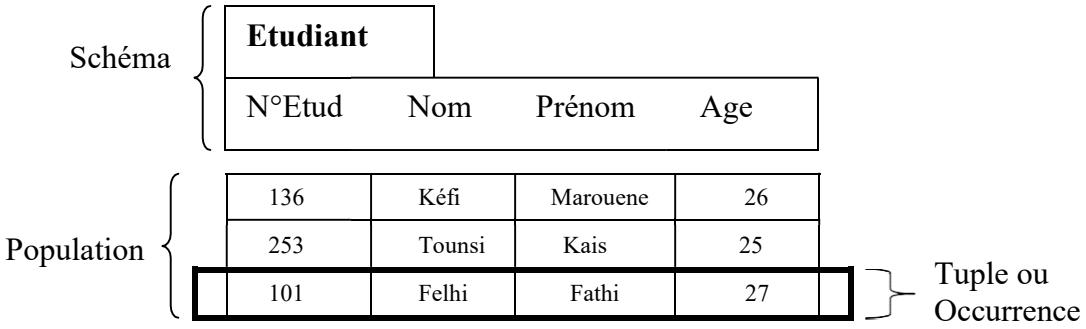


Figure III-2 : Exemple de schéma, population et d'occurrence.

I.5- Clé primaire :

La clé primaire d'une relation est le plus petit sous-ensemble des attributs qui permet d'identifier chaque ligne de manière unique. Comme on a vu que deux lignes sont toujours différentes, l'ensemble de tous les attributs est lui-même une clé mais on peut pratiquement toujours trouver un sous-ensemble qui satisfait la condition.

I.6- Clé étrangère :

Une clé étrangère dans une relation est formée d'un ou plusieurs attributs qui constituent une clé primaire dans une autre relation.

I.7- Schéma relationnel :

Un schéma relationnel est constitué par l'ensemble des schémas de relation.

I.8- Base de données relationnelle :

Une base de données relationnelle est constituée par l'ensemble des n-uplets des différentes relations du schéma relationnel.

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

II.1 Règles de passage :

- La normalisation devrait toujours être effectuée avant le passage au modèle relationnel.
- Chaque type-entité donne naissance à une relation. Chaque attribut de ce type-entité devient un attribut de la relation. L'identifiant est conservé en tant que clé de la relation.
- Chaque type-association dont aucune patte n'a pour cardinalité maximale 1 donne naissance à une relation. Chaque attribut de ce type-association devient un attribut de la relation. L'identifiant, s'il est précisé, est conservé en tant que clé de la relation, sinon cette clé est formée par la concaténation des identifiants des type-entités qui interviennent dans le type-association.
- Un type-association dont au moins une patte a une cardinalité maximale à 1 (ce type-association devrait être binaire et n'a généralement pas d'attribut) ne devient pas une relation. Il décrit en effet une dépendance fonctionnelle. La relation correspondant au type-entité dont la patte vers le type-association a une cardinalité maximale valant 1, se voit simplement ajouter comme attribut (et donc comme clé étrangère) l'identifiant de l'autre type-entité.

II.2 Traduction des associations 1-1 :

La traduction d'une telle association dépend des cardinalités minimales de l'association.

❖ **1er cas :**

Les deux cardinalités minimales sont 1.

➔ Dans ce cas On construit une seule relation contenant les attributs des deux entités. La clé est reportée une seul fois si elle est la même.

Exemple :

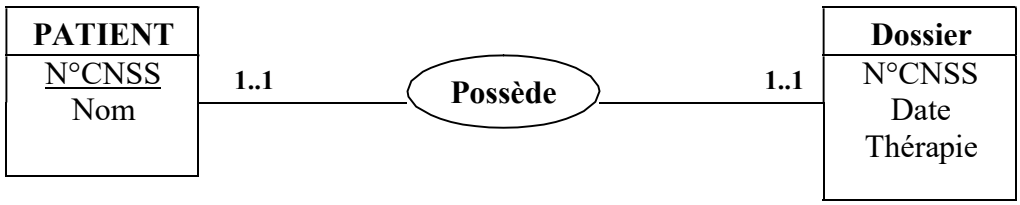


Figure III-3 : Exemple d'association 1-1 et 1-1.

➔ Patient Dossier (N°CNSS, Nom, Date, Thérapie)

❖ **2éme cas :**

Une cardinalité minimale est 0, l'autre est 1.

➔ Dans ce cas, on reporte dans l'entité dont la participation est totale, la clé de l'autre entité.

Exemple :

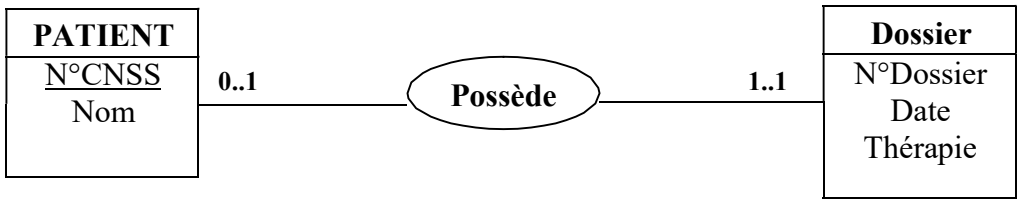


Figure III-4 : Exemple d'association 0-1 et 1-1.

➔ Patient (N°CNSS, Nom)

Dossier (N°Dossier, Date, Thérapie, N°CNSS)

❖ **3éme cas :**

Les deux cardinalités minimales sont 0.

➔ Dans ce cas, on crée une relation spécifique avec les clés des deux entités.

Exemple :

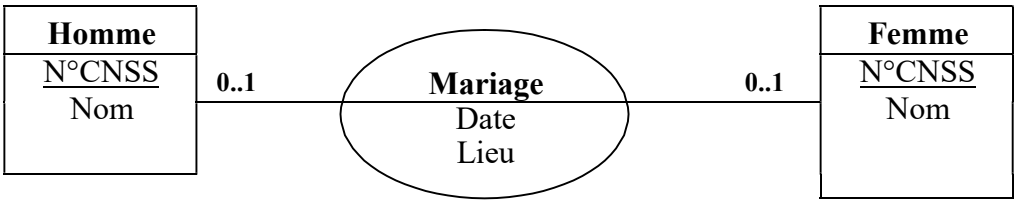


Figure III-5 : Exemple d'association 0-1 et 0-1.

➔ Homme (N°CNSS, Nom)

Femme (N°CNSS, Nom)

Mariage (N°CNSS_Hom, N°CNSS_Fem, date, lieu)

II.3 Traduction des associations 1-N :

Là encore cela dépend des cardinalités minimales.

❖ **1er cas :**

L'entité à participation multiple a une cardinalité minimale de 1.

➔ Dans ce cas, on reporte l'identifiant de l'entité (1, N) dans la relation représentant l'entité (1,1).

Exemple :

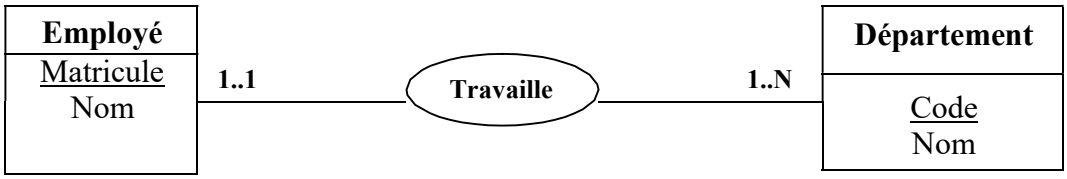


Figure III-6 : Exemple d'association 1-1 et 1-N.

➔ Employé (Matricule, Nom, Code)

Département (Code, Nom)

❖ **2éme cas :**

L'entité à participation multiple a une cardinalité minimale 0.

➔ Dans ce cas, on crée une relation spécifique pour traduire l'association.

Exemple :

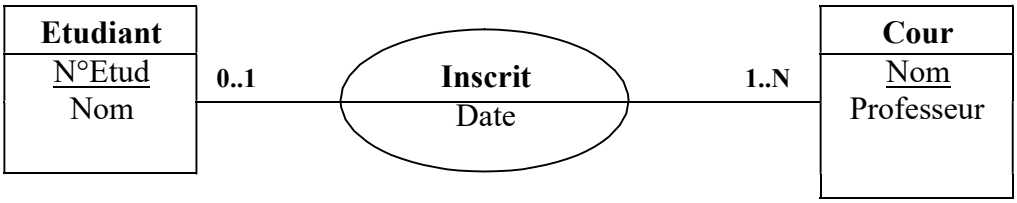


Figure III-7 : Exemple d'association 0-1 et 1-N.

➔ Etudiant (N°Etud, Nom)

Cour (Nom, Professeur)

Etudiant_Cour (N°Etud, Nom_Cour, Date)

II.4 Traduction des associations M-N :

Ici dans tous les cas, on crée une relation spécifique dont la clé sera composée des clés des entités participantes.

Exemple :

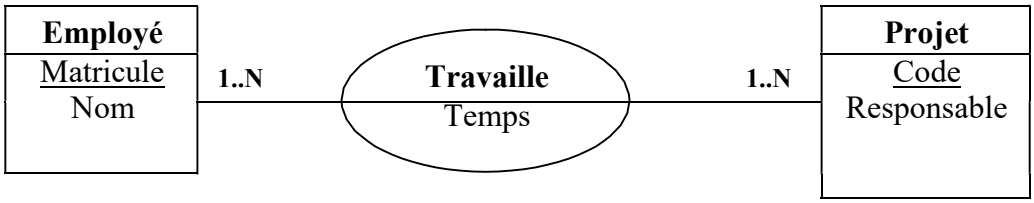


Figure III-8 : Exemple d'association 1-N et 1-N.

➔ Employé (Matricule, Nom)

Projet (Code, Responsable)

Travaille (Matricule, Code, Temps)

II.5 Traduction des associations n-aires :

Elles se traduisent comme les associations M-N, la seule difficulté est de déterminer la clé.

Exemple :

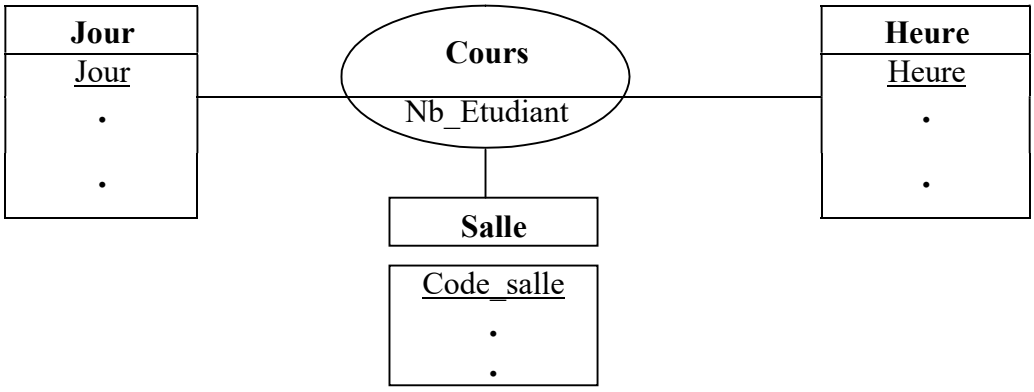


Figure III-9 : Exemple d'association n-aire.

➔ Jour (jour,…)

Heure (Heure,…)

Salle (Code_salle,…)

Cours (Code_salle, jour, heure, Nb_Etudiant)

II.6 Traduction des associations récursives :

On crée ici une relation spécifique représentant l'association. La clé dépend du type (1-1, 1-N ou M-N) de l'association.

Exemple1 :

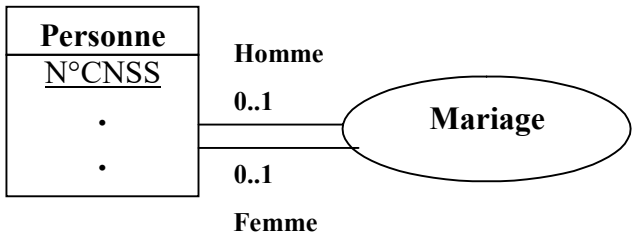


Figure III-10 : Exemple d'association récursive 0-1 et 0-1.

➔ Personne (N°CNSS,…)

Mariage (N°CNSS Homme, N°CNSS Femme)

Exemple2:

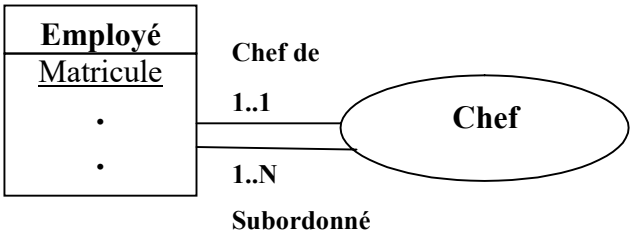


Figure III-11 : Exemple d'association récursive 1-1 et 1-N.

➔ Employé (Matricule,…)

Chef (Matricule_Sub, Matricule_Chef)

Exemple3:

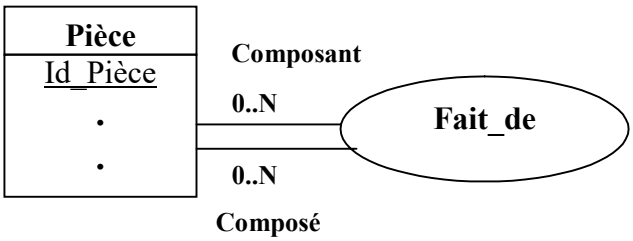


Figure III-12 : Exemple d'association récursive 0-N et 0-N.

➔ Pièce (Id_Pièce,…)

Fait_de (Id_Pièce_Composant, Id_Pièce_Composé)

III- Normalisation:

Les formes normales sont différents stades de qualité qui permettent d'éviter la redondance dans les bases de données relationnelles afin d'éviter ou de limiter :

- Les pertes de données,
- Les incohérences au sein des données,

- La dégradation des performances des traitements.

C'est un processus de transformation d'une relation posant des problèmes lors des MAJ en relations ne posant pas de problèmes.

Contrairement à ce que nous avions fait dans le chapitre précèdent, nous abordons ici la normalisation en nous appuyant sur les notions de dépendance fonctionnelle, dépendance multivaluée et dépendance de jointure.

III.1 Dépendance Fonctionnelle (DF) :

III.1.1 Dépendance Fonctionnelle (DF) :

Il existe une dépendance fonctionnelle entre deux groupes d'attributs A et B d'une relation R si à toute valeur de A on ne peut associer, à tout instant, qu'une et une seule valeur de B.

On dit alors que B dépend (Fonctionnellement) de A (noté A ➔ B). A est la source de la DF et B le But. On dit aussi que A détermine (Fonctionnellement) B.

III.1.2 Dépendance Fonctionnelle élémentaire :

E ➔ F est élémentaire si il n'existe pas un groupe attributs E1 tel que E1 est inclut dans E et E1➔ E.

Autrement dit, une dépendance fonctionnelle est élémentaire si la cible est un attribut unique et si la source ne comporte pas d'attributs superflus. La question sur l'élémentarité d'une dépendance fonctionnelle ne doit donc se poser que lorsque la partie gauche de la dépendance fonctionnelle comporte plusieurs attributs.

III.1.3 Dépendance Fonctionnelle directe :

Une dépendance fonctionnelle A ➔ B est une dépendance fonctionnelle directe s'il n'existe aucun attribut B tel que l'on puisse avoir A ➔ C et C ➔ B.

En d'autres termes, cela signifie que la dépendance entre A et B ne peut pas être obtenue par transitivité.

III.2 Dépendance Multivaluée (DM) :

III.2.1 Dépendance Multivaluée (DM) :

Une dépendance multivaluée (DM) caractérise une indépendance entre deux ensembles d'attributs (Y et Z) corrélés par une même troisième X

Et on la note : X $\twoheadrightarrow$ Y et on lit X multidétermine Y.

III.2.2 Dépendance Multivaluée élémentaire :

C'est une dépendance multivaluée X $\twoheadrightarrow$ Y d'une relation R telle que :

- ✓ Y n'est pas vide est disjoint de X.
- ✓ R ne contient pas une autre DM de type X1 $\twoheadrightarrow$ Y1 telle que X1 inclut dans X et Y1 inclut dans y.

III.3 Les Formes Normale :

III.3.1 1ére Forme normale (1FN) :

Une relation est en première forme normale (1FN) si elle possède une clé et que tous ses attributs sont atomiques.

Remarque : Un attribut atomique est un attribut n'ayant à un instant donné qu'une seule valeur ou ne regroupant pas un ensemble de valeur.

Exemple:

La relation : ***Personne (num-personne, nom, prénom, rue-et-ville)***

N'est pas en 1FN, car le champ rue-et-ville est décomposable, il faut alors la décomposé :

Personne (num-personne, nom, prénom, rue, ville).

III.3.2 2éme Forme normale (2FN) :

Une relation est en deuxième forme normale (2FN) si elle est en 1FN et que toute DF entre la clé et les autres attributs est élémentaire.

Autrement dit, une relation est en 2FN normale si, et seulement si, elle est en 1FN et si tout attribut n'appartenant pas à la clé ne dépend pas que d'une partie de la clé.

Exemple:

Soit le schéma de relation suivant :

CommandeLivre (Num-Commande, Num-Client, Titre, Auteur, Quantité, Prix).

Cette relation indique qu'un client (identifié par ***Num-Client***) a passé une commande (identifiée par ***Num-Commande***) de livre. Elle est bien en première forme normale. Par contre, les attributs ***Titre***, ***Auteur***, ***Quantité*** et ***Prix*** ne dépendent que de ***Num-Commande*** (c'est à dire d'une partie de la clé), et pas de ***Num-Client***. Cette relation n'est donc pas en deuxième forme normale. Une solution simple pour la normaliser est de la remplacer par :

CommandeLivre (Num-Commande, Num-Client, Titre, Auteur, Quantité, Prix).

Remarque : il existe une petite astuce ici pour connaître si une relation R est en 2FN : si R est en 1FN et que sa **clé primaire** est composé d'un **seul attribut** alors elle est en 2FN.

III.3.3 3éme Forme normale (3FN) :

Une relation est en troisième forme normale (3FN) si elle est en 2FN et que toute DF entre la clé et les autres attributs est direct.

Autrement dit, une relation est en 3FN si, et seulement si, elle est en 2FN et si tout attribut n'appartenant pas à la clé ne dépend pas d'un attribut non-clé.

Exemple:

La relation ***CommandeLivre (Num-Commande, Num-Client, Titre, Auteur, Quantité, Prix)***. Est en 2FN, mais, les attributs ***Auteur*** et ***Prix*** dépendent de l'attribut ***Titre***. La relation n'est donc pas en troisième forme normale. Pour la normaliser, il faut la décomposer de la manière suivante :

CommandeLivre (Num-Commande, Num-Client, Num-Livre, Quantité)

Livre (Num-Livre, Titre, Auteur, Prix)

III.3.4 Forme de Boyce-Codd (BCNF) :

Une relation est en BCNF si, et seulement si, elle est en 3FN et les seules dépendances fonctionnelles élémentaires sont celles dans lesquelles une clé entière détermine un attribut.

Exemple:

Soit le schéma relationnel décrivant l'enseignement d'une matière donnée à une classe par un enseignant :

Matière (nom-matière)

Classe (num-classe)

Enseignant (nom-enseignant)

Enseignement (nom-enseignant, num-classe, nom-matière)

Dans la relation Enseignement on a les dépendances fonctionnelles suivantes :

Nom-matière, num-classe ➔ nom-enseignant

Nom-enseignant ➔ nom-matière

Donc nom-enseignant est déterminer par une partie de la clé et pas la clé entière, de même pour nom-matière.

Pour normaliser la relation ***Enseignement***, il faut la décomposer pour aboutir au schéma relationnel suivant :

Matière (nom-matière)

Classe (num-classe)

Enseignant (nom-enseignant, nom-matière)

Enseigner (nom-enseignant, num-classe)

III.3.5 4éme Forme normale (4FN) :

Une relation est en 4FN si, et seulement si, elle est en BCNF et les seules dépendances multivaluées élémentaires sont celles dans lesquelles une super clé détermine un attribut.

En autre terme, une relation R n'est pas en 4FN si l'on peut trouver une dépendance de la forme X $\twoheadrightarrow$ Y où X n'inclut pas une clé de R.

Exemple:

Enseigner (nom-enseignant, num-classe)

IV- L'algèbre relationnelle:

L'algèbre relationnelle est un support mathématique cohérent sur lequel repose le modèle relationnel.

On peut distinguer trois familles d'opérateurs relationnels :

- **Les opérateurs unaires (Sélection, Projection) :** ce sont les opérateurs les plus simples, ils permettent de produire une nouvelle table à partir d'une autre table.
- **Les opérateurs binaires ensemblistes (Union, Intersection et Différence) :** ces opérateurs permettent de produire une nouvelle relation à partir de deux **relations de même degré et de même domaine.**
- **Les opérateurs binaires ou n-aires (Produit cartésien, Jointure, Division) :** ils permettent de produire une nouvelle table à partir de deux ou plusieurs autres tables.

IV.1 Les opérateurs unaires:

VI.1.1 Sélection :

La sélection (parfois appelée restriction) génère une relation regroupant exclusivement toutes les occurrences de la relation R qui satisfont l'expression logique E, on la note :

$$\sigma_{(E)}R$$

Il s'agit d'une opération unaire essentielle dont la signature est :

Relation* * *expression logique* ➔ *Relation

En d'autres termes, la sélection permet de choisir (i.e. sélectionner) des lignes dans le tableau. Le résultat de la sélection est donc une nouvelle relation qui a les mêmes attributs que R. Si R est vide (i.e. ne contient aucune occurrence), la relation qui résulte de la sélection est vide.

VI.1.2 Projection :

La projection consiste à supprimer les attributs autres que A1;…An d'une relation et à éliminer les n-uplets en double apparaissant dans la nouvelle relation ; on la note :

$$\Pi_{(A_1,\ldots A_n)}R.$$

Il s'agit d'une opération unaire essentielle dont la signature est :

Relation * ***Liste d'attributs*** ➔ ***Relation***

IV.2 Les opérateurs binaires ensemblistes:

VI.2.1 Union :

L'union est une opération portant sur deux relations R1 et R2 ayant le même schéma et construisant une troisième relation constituée des n-uplets appartenant à chacune des deux relations R1 et R2 sans doublon, on la note :

$R_1 \cup R_2$

Il s'agit une opération binaire ensembliste commutative essentielle dont la signature est :

Relation * ***Relation*** ➔ ***Relation***

VI.2.2 Intersection :

L'intersection est une opération portant sur deux relations R1 et R2 ayant le même schéma et construisant une troisième relation dont les n-uplets sont constitués de ceux appartenant aux deux relations, on la note :

$R_1 \cap R_2$

Il s'agit une opération binaire ensembliste commutative identique dont la signature est :

Relation * ***Relation*** ➔ ***Relation***

VI.2.3 Différence :

La différence est une opération portant sur deux relations R1 et R2 ayant le même schéma et construisant une troisième relation dont les n-uplets sont constitués de ceux ne se trouvant que dans la relation R1 , on la note :

$R_1 - R_2$

Il s'agit une opération binaire ensembliste non commutative essentielle dont la signature est :

Relation * ***Relation*** ➔ ***Relation***

IV.3 Les opérateurs binaires ou aires:

VI.3.1 Produit cartésien :

Le produit cartésien est une opération portant sur deux relations R1 et R2 et qui construit une troisième relation regroupant exclusivement toutes les possibilités de combinaison des occurrences des relations R1 et R2, on la note :

Il s'agit une opération binaire commutative essentielle dont la signature est :

Relation* * *Relation* ➔ *Relation

VI.3.2 Jointure :

La jointure est une opération portant sur deux relations R1 et R2 qui construit une troisième relation regroupant exclusivement toutes les possibilités de combinaison des occurrences des relations R1 et R2 qui satisfont l'expression logique E, elle est noté :

Il s'agit d'une opération binaire commutative dont la signature est :

Relation* * *Relation* * *expréssion logique* ➔ *Relation

VI.3.3 Division :

La division est une opération portant sur deux relations R1 et R2, telles que le schéma de R2 est strictement inclus dans celui de R1, qui génère une troisième relation regroupant toutes les parties d'occurrences de la relation R1 qui sont associées à toutes les occurrences de la relation R2, on la note :

Il s'agit d'une opération binaire non commutative dont la signature est :

Relation* * *Relation* ➔ *Relation

Chapitre 4 : Le Langage SQL

Introduction :

SQL signifie « Structured Query Language » c'est-à-dire « Langage d'interrogation structuré ». En fait SQL est un langage complet de gestion de bases de données relationnelles. Il a été conçu par IBM dans les années 70. Il est devenu le langage standard des systèmes de gestion de bases de données (SGBD) relationnelles (SGBDR). Il est utilisé par les principaux SGBDR : DB2, Oracle, Informix, Ingres, RDB,... Chacun de ces SGBDR a cependant sa propre variante du langage.

I- Catégories d'instructions:

Les instructions SQL sont regroupées en catégories en fonction de leur utilité et des entités manipulées. Nous pouvons distinguer cinq catégories, qui permettent :

- La définition des éléments d'une base de données (tables, colonnes, clefs, index, contraintes, . . .),
- La manipulation des données (insertion, suppression, modification, extraction, . . .),
- La gestion des droits d'accès aux données (acquisition et révocation des droits),
- La gestion des transactions,
- Le SQL intégré.

I.1 Langage de définition de données (LDD):

C'est un langage orienté au niveau de la structure de la base de données. Le LDD permet de créer, modifier, supprimer des objets. Il permet également de définir le domaine des données (nombre, chaîne de caractères, date, booléen, . . .) et d'ajouter des contraintes de valeur sur les données. Il permet enfin d'autoriser ou d'interdire l'accès aux données et d'activer ou de désactiver l'audit pour un utilisateur donné.

Les instructions du LDD sont :

- CREATE,
- ALTER,
- DROP,
- AUDIT,
- NOAUDIT,
- ANALYZE,
- RENAME,
- TRUNCATE.

I.2 Langage de manipulation de données (LMD):

C'est l'ensemble des commandes concernant la manipulation des données dans une base de données. Le LMD permet l'ajout, la suppression et la modification de lignes, la visualisation du contenu des tables et leur verrouillage.

Les instructions du LMD sont :

- INSERT,
- UPDATE,
- DELETE,
- SELECT,
- EXPLAIN,
- PLAN,
- LOCK TABLE.

Ces éléments doivent être validés par une transaction pour qu'ils soient pris en compte.

I.3 Langage de protection d'accès (DCL):

Il s'occupe de gérer les droits d'accès aux tables.

Les instructions du DCL sont :

- GRANT,
- REVOKE.

I.4 Langage de contrôle de transaction (TCL):

Il gère les modifications faites par le LMD, c'est-à-dire les caractéristiques des transactions, la validation et l'annulation des modifications.

Les instructions du TCL sont :

- COMMIT,
- SAVEPOINT,
- ROLLBACK,
- SET TRANSACTION.

I.5 SQL intégré :

Il permet d'utiliser SQL dans un langage de troisième génération (C, Java, Cobol, etc.) :

- Déclaration d'objets ou d'instructions ;
- Exécution d'instructions ;
- Gestion des variables et des curseurs ;
- Traitement des erreurs.

Les instructions du SQL intégré sont :

- DECLARE,
- TYPE,
- DESCRIBE,
- VAR,
- CONNECT,
- PREPARE,
- EXECUTE,
- OPEN,
- FETCH,
- CLOSE,
- WHENEVER.

II- Définir une base (LDD):

II.1 Contraintes d'intégrité :

Soit le schéma relationnel minimaliste suivant :

Acteur (Num-Act, Nom, Prénom)

Jouer (Num-Act, Num-Film)

Film (Num-Film, Titre, Année)

II.1.1 Contrainte d'intégrité de domaine :

Toute comparaison d'attributs n'est acceptée que si ces attributs sont définis sur le même domaine. Le SGBD doit donc constamment s'assurer de la validité des valeurs d'un attribut.

C'est pourquoi la commande de création de table doit préciser, en plus du nom, le type de chaque colonne.

Par exemple, pour la table ***Film***, on précisera que le ***Titre*** est une chaîne de caractères et l'***Année*** une date. Lors de l'insertion de n-uplets dans cette table, le système s'assurera que les différents champs du n-uplet satisfont les contraintes d'intégrité de domaine des attributs précisées lors de la création de la base. Si les contraintes ne sont pas satisfaites, le n-uplet n'est, tout simplement, pas inséré dans la table.

II.1.2 Contrainte d'intégrité de Table (ou d'entité ou de relation) :

Lors de l'insertion de n-uplets dans une table, il arrive qu'un attribut soit inconnu ou non défini. On introduit alors une valeur conventionnelle notée NULL et appelée valeur nulle.

Mais dans le cas d'une clé primaire, on ne doit pas avoir une valeur nulle, en plus cette clé doit être unique, cette contrainte forte est appelée contrainte d'intégrité de table.

➔ Tout SGBD relationnel doit vérifier l'unicité et le caractère défini (NOT NULL) des valeurs de la clé primaire.

II.1.3 Contrainte d'intégrité de référence :

Dans tout schéma relationnel, il existe deux types de relation :

- Les relations qui représentent des entités de l'univers modélisé ; elles sont qualifiées de statiques, ou d'indépendantes ; les relations ***Acteur*** et ***Film*** en sont des exemples ;
- Les relations dont l'existence des n-uplets dépend des valeurs d'attributs situées dans d'autres relations ; il s'agit de relations dynamiques ou dépendantes ; la relation ***Jouer*** en est un exemple.

Lors de l'insertion d'un n-uplet dans la relation ***Jouer***, le SGBD doit vérifier que les valeurs ***Num-Act*** et ***Num-Film*** correspondent bien, respectivement, à

une valeur de ***Num-Act*** existant dans la relation ***Acteur*** et une valeur ***Num-Film*** existant dans la relation Film.

Lors de la suppression d'un n-uplet dans la relation ***Acteur***, le SGBD doit vérifier qu'aucun n-uplet de la relation ***Jouer*** ne fait référence, par l'intermédiaire de l'attribut ***Num-Act***, au n-uplet que l'on cherche à supprimer.

II.2 Création d'une table (CREATE TABLE) :

Une table est un ensemble de lignes et de colonnes. La création consiste à définir le nom de ces colonnes, leur type, la valeur par défaut à la création de la ligne (DEFAULT) et les règles de gestion s'appliquant à la colonne (CONSTRAINT).

II.2.1 Création simple :

La commande de création de table la plus simple ne comportera que le nom et le type de chaque colonne de la table. A la création, la table sera vide, mais un certain espace lui sera alloué. La syntaxe est la suivante :

CREATE TABLE nom_table ***(***
Nom_colone1 type1,
Nom_colone2 type2,
…
Nom_coloneN typeN
)

Les types de données sont :

- **INTEGER** : Ce type permet de stocker des entiers signés codés sur 4 octets.
- **BIGINT :** Ce type permet de stocker des entiers signés codés sur 8 octets.
- **REAL :** Ce type permet de stocker des réels comportant 6 chiffres significatifs codés sur 4 octets.
- **DOUBLE PRECISION :** Ce type permet de stocker des réels comportant 15 chiffres significatifs codés sur 8 octets.
- **NUMERIC [(précision, [longueur])] :** Ce type de données permet de stocker des données numériques à la fois entières et réelles avec une

précision de 1000 chiffres significatifs. ***Longueur*** précise le nombre maximum de chiffres significatifs stockés et ***précision*** donne le nombre maximum de chiffres après la virgule.

- **CHAR(longueur) :** Ce type de données permet de stocker des chaînes de caractères de longueur fixe. ***Longueur*** doit être inférieur à 255, sa valeur par défaut est 1.
- **VARCHAR (longueur) :** Ce type de données permet de stocker des chaînes de caractères de longueur variable. ***Longueur*** doit être inférieur à 2000, il n'y a pas de valeur par défaut.
- **DATE :** Ce type de données permet de stocker des données constituées d'une date.
- **TIMESTAMP :** Ce type de données permet de stocker des données constituées d'une date et d'une heure.
- **BOOLEAN :** Ce type de données permet de stocker des valeurs Booléenne (Vrai ou Faux).
- **MONEY :** Ce type de données permet de stocker des valeurs monétaires.
- **TEXT :** Ce type de données permet des stocker des chaînes de caractères de longueur variable.

II.2.2 Création avec insertion de données :

On peut insérer les données en même temps qu'on la crée, la commande est la suivante :

CREATE TABLE nom_table [(nom_col1, nom_col2, ...)] AS SELECT ...

II.2.3 Contraintes d'intégrité:

A la création d'une table, les contraintes d'intégrité se déclarent de la façon suivante :

CREATE TABLE nom_table ***(***
nom_col_1 type_1 [***CONSTRAINT*** nom_1_1] contrainte_de_colonne_1_1
[***CONSTRAINT*** nom_1_2] contrainte_de_colonne_1_2
... ...
[***CONSTRAINT*** nom_1_m] contrainte_de_colonne_2_m,
nom_col_2 type_2 [***CONSTRAINT*** nom_2_1] contrainte_de_colonne_2_1
[***CONSTRAINT*** nom_2_2] contrainte_de_colonne_2_2

... ...
[*CONSTRAINT* nom_2_m] contrainte_de_colonne_2_m,
nom_col_n type_n [*CONSTRAINT* nom_n_1] contrainte_de_colonne_n_1
[*CONSTRAINT* nom_n_2] contrainte_de_colonne_n_2
... ...
[*CONSTRAINT* nom_n_m] contrainte_de_colonne_n_m,
[CONSTRAINT nom_1] contrainte_de_table_1,
[CONSTRAINT nom_2] contrainte_de_table_2,
... ...
[CONSTRAINT nom_p] contrainte_de_table_p
)

❖ Les contraintes de colonne sont :

- **NOT NULL ou NULL :** Interdit (***NOT NULL***) ou autorise (***NULL***) l'insertion de valeur NULL pour cet attribut.
- **UNIQUE :** Désigne l'attribut comme clé secondaire de la table. Deux n-uplets ne peuvent recevoir des valeurs identiques pour cet attribut, mais l'insertion de valeur ***NULL*** est toutefois autorisée. Cette contrainte peut apparaître plusieurs fois dans l'instruction.
- **PRIMARY KEY :** Sésigne l'attribut comme la clé primaire de la table , cette contrainte ne doit apparaître qu'une seul fois dans une table donnée.
- **REFERENCES table [(colonne)] [ON DELETE CASCADE] :** Contrainte d'intégrité référentielle pour l'attribut de la table en cours de définition. Les valeurs prises par cet attribut doivent exister dans l'attribut colonne qui possède une contrainte PRIMARY KEY ou UNIQUE dans la table ***table***. En l'absence de précision d'attribut colonne, l'attribut retenu est celui correspondant à la clé primaire de la table ***table*** spécifiée.
- **CHECK (condition) :** Vérifie lors de l'insertion de n-uplets que l'attribut réalise la condition c***ondition***.
- **DEFAULT valeur_par_défaut :** Permet de spécifier la valeur par défaut de l'attribut.

❖ Les contraintes de colonne sont :

- **PRIMARY KEY (colonne, ...) :** c'est quand une table contient plusqu'un attribut comme clé.
- **UNIQUE (colonne, ...) :** utiliser si on veut déclarer plusieurs clés secondaires dans une table.

- **FOREIGN KEY (colonne, ...) REFERENCES table [(colonne, ...)] [ON DELETE CASCADE | SET NULL] :** c'est la déclarartion de clé étrangére qui pointe (ou refére) à la clé primaire de la table ***table***, pour que les données de la colonne de la table en cours de définition et ceux de la table ***table*** soient les mêmes.
- **CHECK (condition) :** Cette contrainte permet d'exprimer une condition qui doit exister entre plusieurs attributs de la ligne.
- **ON DELETE CASCADE :** Demande la suppression des n-uplets dépendants, dans la table en cours de définition, quand le n-uplet contenant la clé primaire référencée est supprimé dans la table maître.
- **ON DELETE SET NULL :** Demande la mise à ***NULL*** des attributs constituant la clé étrangère qui font référence au n-uplet supprimé dans la table maître.

Exemple :

CREATE TABLE affectation_activite(
refelv number(4),
nomact varchar(10),
nbheure number(1),
constraint A ***primary key(***refelv,nomact***),***
constraint B ***foreign key(***refelv***) references*** eleves***(***refelv***),***
constraint C ***foreign key(***nomact***) references*** activite***(***nomact***)***
);

Clé étrangère qui réfère sur la table élèves

Pas de virgule pour la dernière ligne.

II.3 Suppression d'une table (DROP TABLE) :

Supprimer une table revient à éliminer sa structure et toutes les données qu'elle contient. Les index associés sont également supprimés.

La syntaxe est la suivante :

DROP TABLE nom_table

Remarque : Si une autre table pointe (ou référe) à la table a supprimer, on a deux solution :

- ❖ Supprimer la deuxième table avant ;
- ❖ Supprimer la clé étrangére de la deuxième table.

II.4 Modifier une table (ALTER TABLE):

- Pour ajouter ou modifier une colonne, la syntaxe est la suivante :

ALTER TABLE nom_table

{***ADD/MODIFY***} ([nom_colonne type [contrainte], ...])

exemple : pour ajouter une colonne nommé niveau à la table classes, qui peut être soit premier, deuxième ou troisiéme on fait :

ALTER TABLE classes

MODIFY (niveau varchar(10) check(niveau in ('premiere', 'deuxième', 'troisiéme'))

);

- Pour ajouter une contrainte sur la table, la syntaxe est la suivante :

ALTER TABLE nom_table

ADD [***CONSTRAINT*** nom_contrainte] contrainte

exemple : pour ajouter une clé étrangère dans la table Activité qui réfère à la table station, on fait :

ALTER TABLE Activité

ADD (CONSTRAINT FK_ACT_ST ***FOREIGN KEY*** (Nom_station) ***REFERENCES*** STATION (Nom_station)***) ;***

- Pour renommer une colonne, la syntaxe est :

ALTER TABLE nom_table ***RENAME COLUMN*** ancien_nom ***TO*** nouveau_nom ;

- Pour renommer une table, la syntaxe est :

ALTER TABLE nom_table ***RENAME TO*** nouveau_nom

III- Modifier une base (LMD):

III.1 Insérer des n-uplets (INSERT INTO):

La commande ***INSERT INTO*** permet d'insérer une ligne de données dans une table, mais il faut spécifier les champs qui vont être remplie.

La syntaxe est la suivante :

INSERT INTO nom_table (nom_col_1, nom_col_2, ...)

VALUES (val_1, val_2, ...)

La liste des noms de colonne est optionnelle (c'est-à-dire on peut mettre que quelques colonnes ou rien du tout). Si on ne met rien alors tous les champs de la table seront pris, si une colonne n'est pas spécifié dans ***VALUE*** alors elle prend la valeur NULL.

Exemple :

INSERT INTO classes (nomcla, niveau)

VALUES ('1math 1','première');

On peut aussi insérer des données provenant d'une autre table, la syntaxe est la suivante :

INSERT INTO nom_table (nom_col1, nom_col2, ...)

SELECT ...

Exemple :

INSERT INTO activite1 (nomact, désignation, jour)

SELECT ACT, NOM, DATE ***FROM*** activité2

WHERE ACT = 'Ski' ;

III.2 Modifier des n-uplets (UPDATE):

Cette commande permet de changer les valeurs d'une ou de plusieurs colonnes, la syntaxe est la suivante :

UPDATE nom_table

SET Nom_col_1 = {expression_1 | (SELECT ...)},

Nom_col_2 = {expression_2 | (SELECT ...)},

...

Nom_col_n = {expression_n | (SELECT ...)}

WHERE conditions

Les valeurs des colonnes Nom_col_1, Nom_col_2, ..., Nom_col_n sont modifiées dans toutes les lignes qui satisfont le prédicat condition. En l'absence d'une clause WHERE, toutes les lignes sont mises à jour. Les expressions expression_1, expression_2, ..., expression_n peuvent faire référence aux anciennes valeurs de la ligne.

Exemple :

UPDATE DOSSIER

SET DateVisite = DateVisite + '01:00'

WHERE N°Patient = 2025;

III.3 Supprimer des n-uplets (DELETE):

Avec la commande ***DELETE*** on peut supprimer des lignes de la table, la syntaxe est la suivante :

DELETE FROM nom_table

WHERE condition ;

Exemple :

DELETE FROM DOSSIER

WHERE N°Dossier = 0080 ;

IV- Interroger une base (LMD):

IV.1 Sélection (SELECT):

La commande SELECT constitue, à elle seule, le langage permettant d'interroger une base de données. Elle permet de :

- Sélectionner certaines colonnes d'une table (projection) ;
- Sélectionner certaines lignes d'une table en fonction de leur contenu (sélection) ;
- Combiner des informations venant de plusieurs tables (jointure, union, intersection, différence et division) ;
- Combiner entre elles ces différentes opérations.

La syntaxe la plus simple pour une sélection est :

SELECT [ALL | DISTINCT] {* | attribut [, ...]}

FROM nom_table [, ...]

[***WHERE*** condition]

- La clause ***SELECT*** permet de spécifier les attributs que l'on désire voir apparaître dans le résultat de la requête ; le caractère étoile (*) récupère tous les attributs de la table générée par la clause ***FROM*** de la requête ;
- La clause ***FROM*** spécifie les tables sur lesquelles portes la requête ;

- La clause ***WHERE***, qui est facultative, énonce une condition que doivent respecter les n-uplets sélectionnés.

Exemple :

SELECT nomAct, Date_Act ***FROM*** activité

Where N°Act = 102 ;

Remarque importante :

Lorsque on utilise un nom de table ou d'attributs qui est un mot clef de SQL, il faut l'entourer d'apostrophe double.

Exemple :

SELECT ''Date'' ***FROM*** cours

WHERE N°cour = 10 ;

IV.2 Traduction des opérateurs de projection, sélection et produit cartésien de l'algèbre relationnelle :

- L'opération de projection vue dans le chapitre précédent se traduit par :

  SELECT DISTINCT A_1, ..., A_n FROM relation ;

  DISTINCT permet de ne retenir qu'une occurrence de n-uplet dans le cas où une requête produit plusieurs n-uplets identiques.

- L'opération de sélection se traduit par :

  SELECT * FROM relation ***WHERE*** condition ;

- L'opération de produit cartésien se traduit par :

  SELECT * FROM relation_1, relation_2

IV.3 Syntaxe générale de la commande SELECT:

La syntaxe complète de la commande ***SELECT*** est la suivante :

SELECT [ALL | DISTINCT] {* | expression [***AS*** nom_affiché]} [, ...]

FROM nom_table [[***AS***] alias] [, ...]

[***WHERE*** condition1 [***AND/OR***] conditon2]

[***GROUP BY*** expression [, ...]]

[***HAVING*** condition3 [, ...]]

[{UNION | INTERSECT | EXCEPT [ALL]} requête]

[***ORDER BY*** expression [***ASC | DESC***] [, ...]]

- ***FROM :*** Cette clause spécifie les tables sur lesquelles portes la requête.

- ***WHERE :*** Cette clause permet de filtrer les n-uplets en imposant une condition à remplir pour qu'ils soient présents dans le résultat de la requête.
- ***GROUP BY :*** Cette clause permet de définir des groupes.
- ***HAVING :*** elle doit être toujours après une clause ***GROUP BY***, c'est une condition de regroupement.
- ***UNION, INTERSECT*** et ***EXCEPT :*** Ces clauses permettent d'effectuer des opérations ensemblistes entre plusieurs résultats de requête, ***UNION*** fait l'union, ***INTERSECT*** fait l'intersection et ***EXCEPT*** fait la différence.
- ***ORDER BY :*** Cette clause permet de trier les n-uplets du résultat soit un trie ascendant on utilise alors avec ***ORDRE BY*** la clause ***ASC*** ou descendant et on utilise alors ***DESC***.
- ***ALL :*** permet d'avoir tout les n-uplets du résultat, même si il y'a répétition.
- ***AS :*** permet de renommer une colonne, ou de nommer une colonne créée dans la requête.
- ***OR/AND :*** si on a plus qu'une condition dans ***WHERE***.

Exemples :

1- Pour avoir les noms de pilotes qui ont Numéro de pilote non NULL et un salaire supérieur à 2000 et regrouper les noms selon l'ordre ascendant :

```
SELECT nom FROM pilote
WHERE N°_Pilote IS NOT NULL
AND salaire >2000
Ordre by nom ASC ;
```

2- Pour avoir les numéros de vols qui partent de Marseille ou et qui arrivent à Londres :

```
SELECT N°Vol FROM Vol
Where VilDep = 'Marseille'
AND
VilArr = 'Londres' ;
```

3- pour avoir la liste des vols qui arrivent à Londres et qui partent soit de Tunis ou de Monastir :

SELECT DISTINCT (N°Vol) ***FROM*** VOL

WHERE VilArr =’Londres’ ***AND*** VilDep =’Tunis’

UNION

SELECT N°Vol ***FROM*** VOL

WHERE VilArr =’Londres’ ***AND*** VilDep =’Monastir’;

IV.4 Les fonctions d’agrégations:

- ***AVG ([DISTINCT | ALL] expression) :*** Calcule la moyenne des valeurs de l’expression ***expression.***
- ***COUNT(* | [DISTINCT | ALL] expression) :*** Dénombre le nombre de lignes du groupe. Si expression est présente, on ne compte que les lignes pour lesquelles cette expression n’est pas ***NULL***.
- ***MAX([DISTINCT | ALL] expression)*** : Retourne la plus grande des valeurs de l’expression ***expression.***
- ***MIN ([DISTINCT | ALL] expression)*** : Retourne la plus petite des valeurs de l’expression ***expression.***
- ***STDDEV ([DISTINCT | ALL] expression)*** : Calcule l’écart type des valeurs de l’expression ***expression.***
- ***SUM([DISTINCT | ALL] expression)*** : Calcule la somme des valeurs de l’expression ***expression.***
- ***VARIANCE ([DISTINCT | ALL] expression) :*** Calcule la variance des valeurs de l’expression ***expression.***

DISTINCT indique à la fonction de groupe de ne prendre en compte que des valeurs distinctes.

ALL indique à la fonction de groupe de prendre en compte toutes les valeurs, c’est la valeur par défaut.

Aucune des fonctions de groupe ne tient compte des valeurs ***NULL*** à l’exception de ***COUNT(*)***.

Ainsi, ***SUM(col)*** est la somme des valeurs non ***NULL*** de la colonne col. De même ***AVG*** est la somme des valeurs non ***NULL*** divisée par le nombre de valeurs non N***ULL***.

Il est tout à fait possible d'utiliser des fonctions d'agrégation sans clause ***GROUP BY***. Dans ce cas, la clause SELECT ne doit comporter que des fonctions d'agrégation et aucun nom de colonne. Le résultat d'une telle requête ne contient qu'une ligne.

Exemples :

- Pour avoir la somme de délai et la moyenne on fait:
  SELECT SUM(Delai), ***AVG*** (Delai) ***FROM*** Course ;
- Pour avoir le nombre de cheval, on compte le nombre de ligne de la colonne Num_cheval :
  SELECT COUNT(Num_cheval) ***FROM*** Cheval ;
- Pour avoir le nombre total de vol ainsi que la moyenne par type d'avion, pour les avions qui ont un nombre de vol moyen supérieur à 1000 :
  SELECT SUM (Nb_Vol), ***AVG*** (Nb_Vol) ***FROM*** avion
  GROUP BY Type_Avion
  HAVING AVG (Nb_Vol)>1000 ;

V- Les vues (LMD):

Les vues sont des tables virtuelles qui « contiennent » le résultat d'une requête ***SELECT***. L'un des intérêts de l'utilisation des vues vient du fait que la vue ne stocke pas les données, mais fait référence à une ou plusieurs tables d'origine à travers une requête ***SELECT***, requête qui est exécutée chaque fois que la vue est référencée. De ce fait, toute modification de données dans les tables d'origine est immédiatement visible dans la vue dès que celle-ci est à nouveau référencée dans une requête.

V.1 Création de vue (CREATE VIEW):

La syntaxe est la suivante :

CREATE [OR REPLACE] VIEW nom_Vue [(Nom_colonne [, ...])] ***AS***

Requête

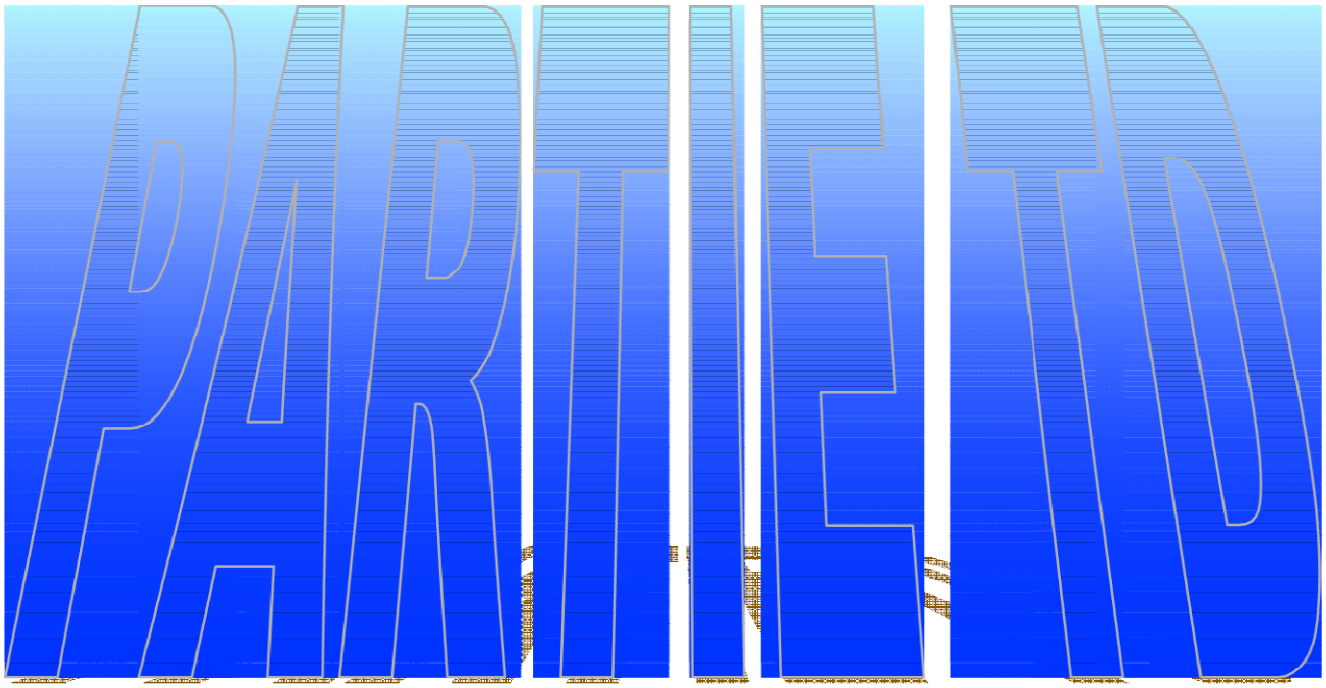
- ***CREATE VIEW :*** définit une nouvelle vue.
- ***CREATE OR REPLACE VIEW :*** définit une nouvelle vue, ou la remplace si il existe une autre vue de même nom. Vous pouvez seulement remplacer une vue avec une nouvelle requête qui génère un ensemble de colonnes identiques.
- ***Nom_colonne :*** Une liste optionnelle de noms à utiliser pour les colonnes de la vue. Si elle n'est pas donnée, le nom des colonnes sera déduit de la requête.
- ***Requête :*** Une requête (c'est-à-dire une instruction ***SELECT***) qui définit les colonnes et les lignes de la vue.

V.2 Suppression de vue (DROP VIEW):

La syntaxe est la suivante :

DROP VIEW nom [, ...] [***CASCADE*** | ***RESTRICT***]

- ***DROP VIEW :*** Supprime une vue existante.
- ***CASCADE :*** Supprime automatiquement les objets qui dépendent de la vue (comme par exemple d'autres vues).
- ***RESTRICT :*** Refuse de supprimer la vue si un objet en dépend. Ceci est la valeur par défaut.


PARTIE II

Série d'exercices N°1

(Réservation de théâtre)

Soit les deux tables relationnelles suivantes :

REPRESENTATION (NREP, NSPEC, NOMSPEC, NTHEATRE, NOMTHEATRE, ADRTHEATRE, TELTHEATRE, DATDEBSPEC, DATFINSPEC, HEURERP, DATREP, CODEPLACE, TYPRHEURE, CODEZONE, TYPEJOUR, PRIXPLACE)

RESERVATION (NRES, DATRES, NOMDEM, ADRDEM, ADRDEM, TELDEM, NREP, CODPLACE, ETATRES, DATETATRES, CODECARTE)

Les données permettent de gérer les différentes représentations des spectacles proposés par les théâtres parisiens et les réservations correspondantes. Les règles suivantes doivent être prise en compte pour normaliser les deux schémas de relations précédentes.

Un théâtre a un numéro unique (NTHEATRE), un nom (NOMTHEATRE), une adresse (ADRTHEATRE) et un téléphone (TELTHEATRE). Un théâtre offre plusieurs spectacles. Chaque spectacle a un numéro unique (NSPEC), un nom (NOMSPEC) ; il se déroule sur une période donnée (DATDEBSPEC, DATFINSPEC) ; il lui correspond « n »représentations. Chaque représentation a un numéro unique (NREP), une heure donnée de début (HEURRERP) à une date donnée (DATREP). Afin de gérer la réservation des places, la base de données connaît tous les numéros de places du théâtre (CODEPLACE). Chaque place correspond à une zone (CODEZONE). Le prix de la place (PRIXPLACE) dépend de la zone, du spectacle, du type de jour (TYPEJOUR) de la représentation pour laquelle la place est louée ainsi que du type d'heure (TYPEHEURE).

La réservation de place se fait par téléphone par une personne qui peut âtre un particulier ou une agence caractérisée par son nom (NOMDEM), son adresse (ADRDEM) et son téléphone (TELDEM). Chaque réservation a un numéro unique (NRES). Elle porte sur une seule représentation et sur un ou plusieurs places. Elle correspond à un prix global (PRIX). La réservation est dite « Pendante »lorsqu'au

moment de l'appel téléphonique la personne n'a pas pu donner un numéro de carte de crédit (CODECARTE) ; elle est « OK » lorsque le numéro de carte a été fourni ; elle est « payée » lorsque le billet a été retiré et le prix payé. On veut garder la trace de l'évolution de l'état des réservations.

Travail demander :

1- De construire le graphe de dépendances fonctionnelles élémentaires et directes sur l'ensemble des attributs.

2- D'en déduire la collection des relations 3FN (Les clés primaires de chaque relation doivent âtre souligné).

3- De décrire cette collection en utilisant SQL et en incorporant les contraintes d'intégrité.

Série d'exercices N°2

(Gestion de prêts)

On demande de structurer l'ensemble des données de gestion des prêts de livres dans une bibliothèque. Universelle est la suivante :
Prêt (NOUV, NEX, NOM, PRENOM, MOT-CLE, NPRET, NABONNE, DATDP, PRIXEX, NED, EDITEUR, DATED, DATRP, TITRE, ETATEX, NBEXPDISPO)

NOUV : numéro de livre (unique) ;
NEX : numéro d'ordre d'un exemplaire pour un livre donné ;
NOM : nom de l'auteur du livre ;
PRENOM : prénom de l'auteur du live ;
MOT-CLE : mot caractéristique du sujet d'un livre ;
NPRET : numéro (unique) de prêt d'un exemplaire de livre ;
NABONNE : numéro (unique) de l'abonné de la bibliothèque ;
DATDP : date de prêt d'un exemplaire de livre ;
DATRP : date de retour d'un prêt d'exemplaire de livre ;
PRIXEX : prix payé pour l'achat d'un exemplaire ;
NED : numéro d'édition d'un exemplaire ;
EDITEUR : nom de l'éditeur d'un livre ;
TITRE : titre d'un livre ;
DATED : date d'édition d'un livre ;
ETATEX : état d'un exemplaire (disponible ou prêté);
NBEXPDISPO : nombre d'exemplaires disponibles d'un livre.

Un ouvrage (ou livre) a un numéro, un titre, des auteurs et plusieurs mots-clé caractéristiques. Il y'a plusieurs exemplaires d'un livre dans la bibliothèque qui sont prêtés aux abonnés. Un prêt est effectué pour une durée délimitée par les deux dates de début du prêt et de retour du prêt. Le prix d'un exemplaire doit être mémorisé dans la base, de même que le numéro d'édition. La base doit être capable de dire quelles sont toutes les éditions d'un livre et les dates correspondantes. Un prêt est fait par un abonné.

Un exemplaire est donc soit prêté soit disponible. L'historique des états des exemplaires de livres doit être stocké dans la base.

Travail demander :

1- Construire le graphe de dépendances fonctionnelles élémentaires et directes.
2- D'en déduire la collection des relations en 3FN.
3- De la décrire dans le langage SQL en introduisant les contraintes associées à chaque base.

Série d'exercices N°3

Exercice1 :

Soit un schéma relationnel constituer d'une seule relation :

R (Id-Cours, Id-Etudiant, Age, Note)

Et des deux dépendances fonctionnelles suivantes :

Id-Cours, Id-Etudiant ➔ Note

Id-Etudiant ➔ Age.

1- Donner quelques exemples de tuples correspondant à la relation R.
2- Indiquer les clés candidates de la relation R.
3- Citer les anomalies et les redondances qui se trouvent dans la relation R.
4- Décomposer la relation R afin de supprimer les anomalies.
5- Vérifier que la décomposition est sans perte de données (le vérifier expérimentalement en faisant une jointure) et sans perte de dépendances.

Exercice2 :

Soit la relation :

Commande (No-Commande, No-Produit, Quantité-Commandée, No-Client, No-Représentant)

Et les dépendances fonctionnelles suivantes :

No-Commande, No-Produit ➔ Quantité-Commandée, No-Client, No-Représentant.

No-Commande ➔ No-Client, No-Représentant.

No-Client ➔ No-Représentant.

1- Donner quelques exemples de tuples correspondant à la relation R.
2- Indiquer les clés candidates de la relation R.
3- Citer les anomalies et les redondances qui se trouvent dans la relation R.
4- Décomposer la relation R afin de supprimer les anomalies.
5- Vérifier que la décomposition est sans perte de données (le vérifier expérimentalement en faisant une jointure) et sans perte de dépendances.

Exercice3 :

Soit la relation :

R (Fournisseur, Adresse, Raison-Sociale, no-Produit, Libellé-Produit, Quantité, Prix, No-Commande, Délai, Date)

Et les dépendances fonctionnelles suivantes :

No-Commande ➔ Fournisseur, Délai, Date.

Fournisseur ➔ Raison-Sociale, Adresse.

No-Commande, no-Produit ➔ Quantité.

No-Produit, Fournisseur ➔ Prix.

No-Produit ➔ Libellé-Produit.

1- Donner quelques exemples de tuples correspondant à la relation R.
2- Indiquer les clés candidates de la relation R.
3- Citer les anomalies et les redondances qui se trouvent dans la relation R.
4- Décomposer la relation R afin de supprimer les anomalies.
5- Vérifier que la décomposition est sans perte de données (le vérifier expérimentalement en faisant une jointure) et sans perte de dépendances.

Série d'exercices N°4

Exercice1 :

Un atelier de confection de vêtements de taille moyenne a différents fournisseurs (1: nom-fournisseur) de files (2: no-fil) de types différents (3: type-fil, 4: libellé-type-fil): coton, soie, invisible, spécial machine, élastique, etc., de différents coloris (5: coloris-fil).

2 ➔ 3, 5

3 ➔ 4

1 ?

Les vêtements sont confectionnés à partir de tissus (6: no-tissu, 7: libellé-tissu, 8: nature-tissu): unis soie, jersey, polyamides, mélanges coton, coton pur, etc.

6 ➔ 7, 8

Et d'accessoires (9: no-accessoire, 10: libellé-accessoire): agrafes, boutons, pression, fermetures éclair, boutons de diverses natures, crochets, rubans, élastiques, etc.

9 ➔ 10

L'atelier n'a retenu qu'un seul fournisseur (11: no-fournisseur) pour chaque accessoire : les prix des accessoires (12: prix-accessoire) ne varient pas beaucoup dans le temps et il est inutile de faire jouer la concurrence.

11 ➔ 1

9 ➔ 10, 11, 12

Par contre, en matière de fil, l'atelier peut choisir entre plusieurs fournisseurs spécialisés: les fils de soie se trouvent chez les fournisseurs 1, 2 et 5, les fils invisibles uniquement chez le fournisseur 5, etc. Le choix, quand il est possible, se fait en fonction de l'éventail des prix (13: prix-fil) du moment, des délais de livraison (14: délai-livraison) qui sont fonction du produit commandé et de l'éloignement du fournisseur (15: distance).

2, 11 ➔ 13, 14

11 ➔ 1, 15 (MAJ de 11 ➔ 1)

Pour chaque collection, un nombre d'exemplaires (16: nb-exemplaire) est associé à un modèle de la collection (17: modèle) dans un tissu et une taille (18: taille) donnés. On confectionnera par exemple 8 exemplaires du modèle 34 dans le tissu n°345 en taille 40, alors qu'on prévoira seulement 3 exemplaires du même modèle dans le même tissu en taille 46.

6, 17, 18 ➔ 16

Pour chaque couple modèle - tissu possible, il y a environ quatre à huit accessoires bien déterminés (élastique d'une couleur donnée, agrafe d'une dimension donnée, etc.) et il faut connaître le nombre nécessaire de chacun de ces accessoires (19: nb-accessoire) pour confectionner le vêtement: il faudra par exemple 6 boutons et 2 agrafes moyennes.

6, 17, 9 ➔ 19

Pour chaque collection (20: saison, 21: année), il y a une trentaine de modèles qui sortent. Chaque modèle n'est présent que dans une seule collection.

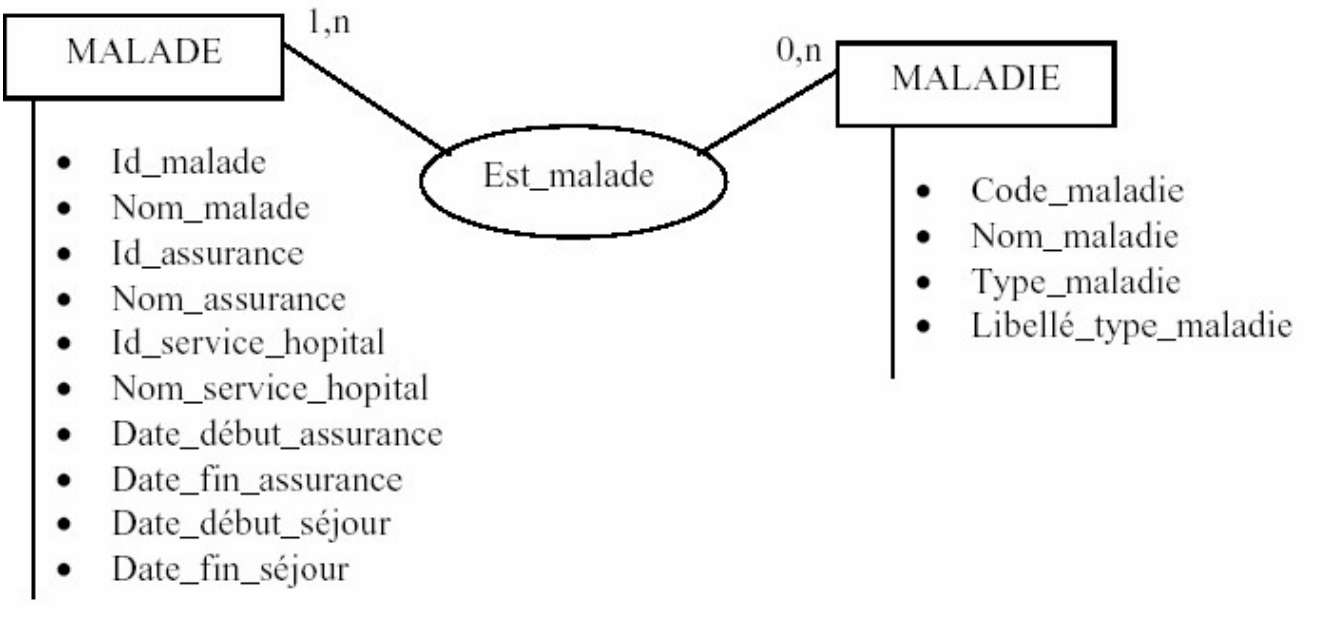
17 ➔ 20, 21

Question :

1- Construire la couverture minimale correspondant au système d'information décrit ci-dessus.
2- Construire le schéma entité-association correspondant la couverture minimale établie en 1.
3- Modifier le diagramme obtenu (sans rajouter de rubriques) pour prendre en compte la contrainte suivante: A chaque tissu correspondent des types de fil. Le fil invisible correspond par exemple à tous les tissus.
4- Traduire le diagramme obtenu en 3. Dans le modèle relationnel.

Exercice2 :

Soit le schéma entité-association suivant :



Dans lequel le type de maladie permet de classifier les maladies en maladies contagieuses et maladies non contagieuses et dans lequel Id-malade est l'identifiant de l'entité Malade et Code-maladie l'identifiant de l'entité Maladie.

Question :

1- Traduire ce diagramme dans le modèle relationnel.

2- Expliquer pourquoi ce schéma n'est pas en BCNF.

3- Modifier le schéma afin d'obtenir un résultat en BCNF.

4- Reporter les modifications sur le diagramme E/A.

Série d'exercices N°5

Exercice1 :

Pour chaque aéroport, on mémorise la ville où il est situé et son nombre de pistes. Il n e peut y avoir deux aéroport avec le même nom, on stocke pour chaque pilote, son numéro, son nom, prénom, sa ville de résidence, son salaire et le prime annuelle qu'il touche. Chaque avion est identifié par son numéro et sa date de mise en service et son type d'avion (Boeing 747, Airbus A340…). Chaque type d'avion est identifié par son nom (A300, B747, A340…) et on mémorise le nom du constructeur (Boeing, Airbus, Dassault…) et le nombre de place du type d'avion.

Chaque vol a un numéro unique, et on mémorise pour chacun le numéro de l'avion qui l'a effectué, le numéro du pilote qui le commande, le numéro du copilote qui l'assiste, l'heure et l'aéroport du départ ainsi que l'heure et l'aéroport d'arrivée.

Chaque pilote peut être pilote ou copilote sur un ou plusieurs vol et un même pilote peut être pilote sur certains vols et copilote sur d'autres.

Question :

1- Donner le dictionnaire de données (Les attributs de toute la base).

2- Définir le schéma Entité / Association de la Base de données.

3- Traduire le schéma Entité / Association en schéma relationnel.

Exercice2 :

Supposons maintenant que la base de données de l'exercice 1 est remplie par des enregistrements.

Question :

1- Modifier l'aéroport de ROISSY auquel a été ajouté une piste d'atterrissage.

2- Modifier le vol numéro 14 pour que l'aéroport de départ soit 'Roissy ' et l'heure d'arrivée soit '15:15'.

3- Les vols qui étaient assuré par l'avion numéro 9 sont désormais assurés par l'avion numéro 10, Rectifier ce ci.

4- Supprimer le vol numéro 18.

5- Créer un nouveau vol 18 avec des caractéristiques identique au vol 2 mais partant à 8:15 et arrivant à 9:30.

6- Donner les noms de tout les pilotes, prénoms et salaires qui résident à paris et à nice.

7- Donner la liste des vols dont l'aéroport de départ est Roissy ou bien Charles de gaule effectué sur un avion de plus de 250 places.

8- Donner la liste des avions qui n'effectuent aucun vol, avec pour chacun son nombre de places et le nom du constructeur.

9- Donner le Salaire minimum et maximum des pilotes.

10- Donner le numéro, le nom et prénom des pilotes qui dirigent plus qu'un vol.

Série d'exercices N°6

Exercice1 :

Un hôpital veut créer une base de données pour conserver une trace de toute transaction faite au sein de l'établissement, pour cela il vous donne les informations suivantes :

- Il nous faut un dossier avec les informations suivantes :
  - Un numéro de dossier, les patients, les médecins qui ont fait le diagnostic, les diagnostics, les traitements et la date de visite.
- Chaque patient a un numéro, un numéro d'association, un nom et un prénom.
- Chaque médecin a un numéro, un nom et un prénom.
- Chaque diagnostic a un numéro et une description.
- Chaque traitement a un numéro et une description.

Question :

1- Donner le dictionnaire de données (Les attributs).
2- Donner le schéma entité/relation de cette base en spécifiant toutes les cardinalités.
3- Traduire ce schéma en schéma relationnel.
4- A l'aide de requêtes SQL faire :
   - Augmenter les dates de visite de une heure.
   - Modifier la date de visite du patient 2025 par 12:00 au lieu de 10:00.
   - Modifier le numéro du médecin pour le dossier 1005 par le numéro 0102.
   - Supprimer le dossier numéro 0080.
   - Donner la liste des médecins qui n'ont aucun dossier.

Exercice2 :

Soit la base de données suivante :

- Cheval (Num_cheval, Date_nais, Nom_cheval)
- Jokey (Num_jockey,Nom_jockey,Adr_jockey,Num_ecurie)
- Ecurie (Num_ecurie, Nom_ecurie, couleur, Num_respon)
- Responsable (Num_resp, Nom_resp, Tel_resp)

- Haras (Nom_haras, Adr_Haras, Tel_haras)
- Course (Num_course, Nom_course, Date_course, type_course, délai, nombre_de_chevaux, NUM_cheval)
- R2 (Num_cheval, Num_course, Num_jockey, Rang)

Question :

1- Donner le nombre total des chevaux.
2- Donner le nombre total de jockey.
3- Donner le nombre total de course.
4- Donner la liste de chevaux qui ont un nom qui contient un « **k** ».
5- Donner le nombre total et moyen d'heure de course.
6- Donner le nombre total et moyen d'heure de course par type de course.
7- Donner les noms des chevaux ayant le plus grand nombre d'heures de course.
8- Donner le type de course qui comporte le plus de chevaux.
9- Donner la liste de chevaux qui n'ont jamais fait de course.
10- Donner la liste de jockeys qui n'ont jamais fait de course.
11- Donner la liste de jockey et de chevaux qui ont fait le plus de course.
12- Supprimer la table Cheval.