Base de données

Chap1 Introduction aux bases de données

1.1. Notion de base de données

Les bases de données ont pris une place importante en informatique, et particulièrement dans le domaine de la gestion. L'étude des bases de données à conduit au développement de concepts, méthodes et algorithmes spécifiques, notamment pour gérer les données en mémoire secondaire.

Dans un premier temps nous pouvons dire qu'une base de données est *un ensemble organisé d'informations avec un objectif commun.*

Peu importe le support utilisé pour rassembler et stocker les données (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.

Cependant dans le cadre informatique nous pouvons donner la définition suivante.

1.1.1. Définition

Une base de données (informatisée) 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.

Cette collection de données interdépendantes, stockées sans redondance inutile est mise à la disposition de plusieurs utilisateurs ou programmeurs, organisées indépendamment des programmeurs.

Elle constitue le cœur du système d'information.

1.1.2. Les modèles de bases de données (différents types de SGBD)

Les modèles de base de données sont liés aux Systèmes de Gestion de Base de Données (SGBD).

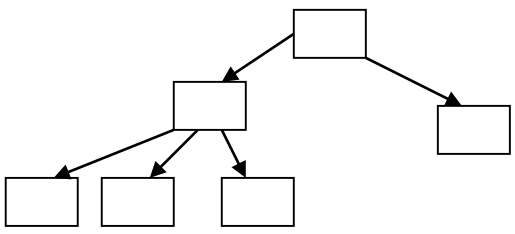
Un SGDB est un logiciel qui permet à l'utilisateur d'interagir avec une base de données.

Il permet d'organiser les données sur les périphériques et fournit les procédures de sélections et de recherches de ces mêmes données.

Il existe actuellement 5 grands modèles de bases de données ou types de SGBD.

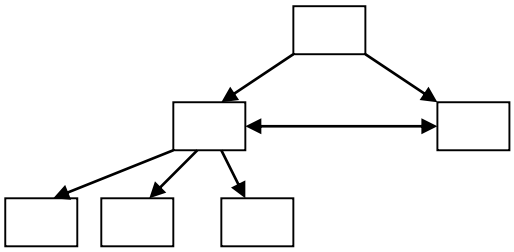
1.1.2.1. Le modèle hiérarchique

Il s'agit des premiers SGBD apparus avec *IMS d'IBM*. Les données sont classées hiérarchiquement selon une arborescence descendante.



1.1.2.2. Le modèle réseau ou codasyl (1960)

Il est organisé sur le même principe que le modèle hiérarchique. Toutefois la structure n'est pas forcement arborescent dans le sens descendant. Il gère aussi les bases de données navigationnelles qui utilisent les pointeurs vers des enregistrements.



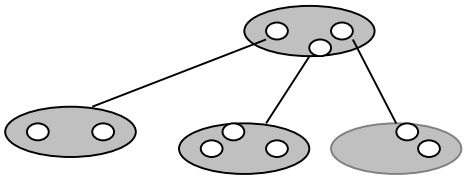
1.1.2.3. Le modèle relationnel (1970)

A l'heure actuelle les plus utilisés. Les données sont enregistrées dans des tableaux à 2 dimensions (lignes et colonnes). Les bases relationnelles sont basées sur l'algèbre relationnelle et un langage déclaratif (généralement SQL).

	C1	C2
L1		
L2		
L3		
L4		

1.1.2.4. Le modèle objet (1990)

Les données sont représentées en tant qu'instances des classes hiérarchisées. Chaque champ est un objet. De ce fait chaque donnée est active et possède ses propres méthodes d'interrogation et d'affectation.

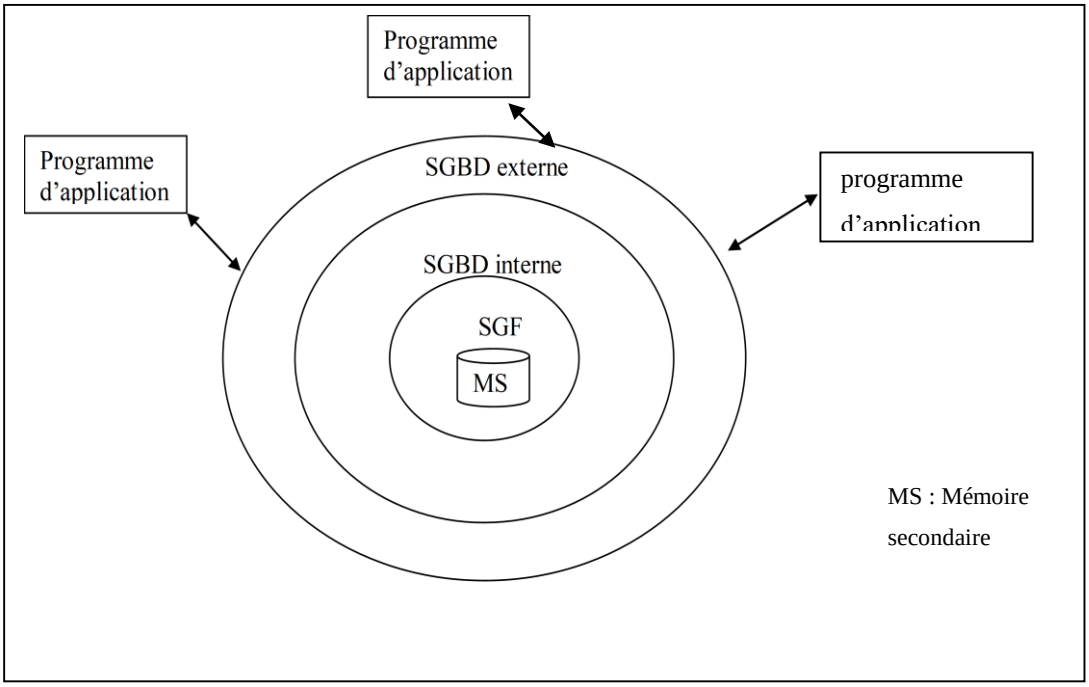


1.1.3. Architecture et évolution des SGBD relationnels

*1.1.3.1.*architecture d'un système de gestion d'une base de données

Un SGBD est organisé en trois couches :

- Le système de gestion de fichiers (SGF) ;
- Le système d'accès aux données ou SGBD interne ;
- La présentation des données aux utilisateurs ou SGBD externe



Structure canonique d'un SGBD

L'architecture à trois niveaux définie (par le standard ANSI/SPARC((*American National Standard Institute* /) permet d'avoir une indépendance entre les données et les traitements.

1.1.4. les caractéristiques d'un SGBD

On peut se demander quels sont les objectifs et avantages de l'approche SGBD par rapport aux fichiers classiques. Les principales caractéristiques d'un SGBD sont :

- Indépendance physique des données

A ce niveau les aspects physiques (disques, bandes) et organisationnels sont transparents pour l'utilisateur. Les modifications des éléments du niveau physique n'interfèrent pas sur les applications.

- Indépendance logique des données

Toutes modifications faites au niveau logique n'entraînent ni de modifications au niveau physique ni dans les applications non concernées.

- Manipulabilité

La possibilité pour les utilisateurs non informaticiens d'accéder aux données sans faire référence aux éléments techniques.

- Efficacité et rapidité d'accès

Le système fournit des réponses rapides aux requêtes en recherchant les meilleurs chemins d'accès par le biais d'algorithmes sophistiqués.

- Cohérence des données

Les données sont soumises à des règles d'intégrité, lesquelles règles sont définies au niveau conceptuel du système d'information. Les applications doivent respecter ces règles lors de la modification des données pour que soit assurée la cohérence de celle-ci.

- Limitation de la redondance

Elle permet une meilleure gestion de l'espace mémoire, facilite les mises à jour de la base de données et évite les erreurs dues aux mises à jour multiples.

- Partageabilité des données

Il s'agit de permettre à plusieurs utilisateurs d'accéder aux données de la base en même temps. Il s'agit donc de gérer les accès simultanés.

- La sécurité des données

Il s'agit de protéger les données en limitant leur accès par l'utilisation de mécanisme adéquat permettant d'autoriser, de contrôler, de supprimer les droits d'accès de n'importe quel utilisateur à l'ensemble des données.

Le système doit garantir la restauration de la base dans son dernier état cohérent en cas de reprise après une panne

Résumé

La répartition du parc des SGBD n'est pas équitable. En effet 75% sont relationnels, 20%de types réseaux et les 5% restants entre les autres bases de données. Ces chiffres risquent d'évoluer quelques années et la frontière entre les bases relationnelles et objets peut être éliminée par l'introduction d'une couche objet sur les bases relationnel.

Chap2 Le modèle Entité- Association (entity-relationship)

2.1. introduction

La phase de conception nécessite souvent de nombreux choix qui auront parfois des répercussions importantes par la suite. La conception de base de données (BD) ne fait pas exception à la règle. Les théoriciens de l'information ont donc proposé des méthodes permettant de structurer la pensée, de présenter de manière abstraite le travail que l'on souhaite réaliser. Ces méthodes ont donné naissance à une discipline : l'analyste (*l'analyste est une discipline qui étudie et présente de manière abstraite le travail à effectuer)*. La phase d'analyse est très importante puisque c'est elle qui sera validée par les utilisateurs avant la mise en œuvre du système concret. Il existe de nombreuse méthodes d'analyse (fonctionnelle, systémique, objet) dont merise. Elle permet de réaliser différents modèles conceptuels, organisationnels et physiques. Celui qui nous intéresse particulièrement ici est le MCD.

2.2. concepts

2.2.1. Une entité

Une entité représente un objet du Système d'information (acteur, document, concept, …). On met les informations nécessaires et suffisantes pour caractériser cette entité.

Les entités sont formalisées par des rectangles.

2.2.2. propriété

La propriété peut être définie comme une donnée élémentaire ou atomique. les propriétés servent à décrire les entités et les associations. Les propriétés sont collectées lors de l'établissement du dictionnaire des données.

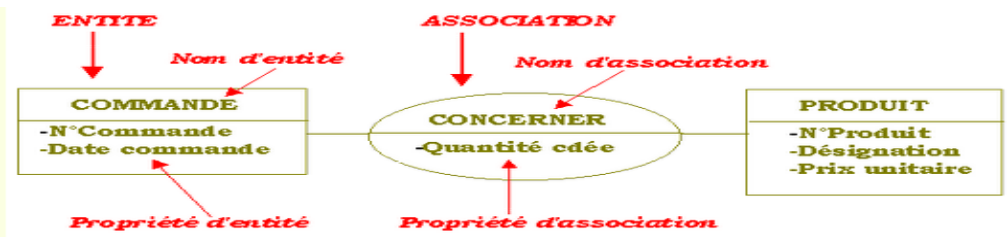
2.2.3. Identifiant

Une propriété particulière, appelée **identifiant** permet de distinguer sans ambiguïté toutes les occurrences de l'entité. L'identifiant est toujours *souligné*. Il est une propriété qui ne peut pas changer au cours du temps pour une occurrence

2.2.4. Association

L'association est un lien entre deux entités (ou plus). On doit lui donner un nom, souvent un verbe, qui caractérise le type de relation entre les entités. Une association possède parfois des propriétés.

Exemple :

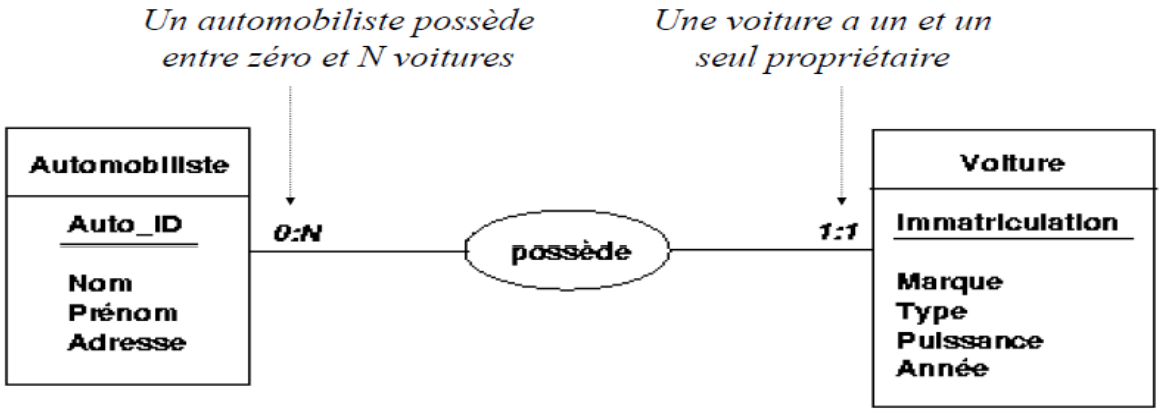


2.2.5. Cardinalités

Ce sont des expressions qui permettent d'indiquer combien de fois au minimum et au maximum le lien entre 2 entités peut se produire. Pour une association de 2 entités, il y a 4 cardinalités à indiquer et trois valeurs typiques : **0**, **1** et **N** (plusieurs).

Les cardinalités traduisent des règles de gestion. Ce *sont des règles propres au* SI(système d'information) *étudié* qui expriment des **contraintes** sur le modèle.

Exemple :



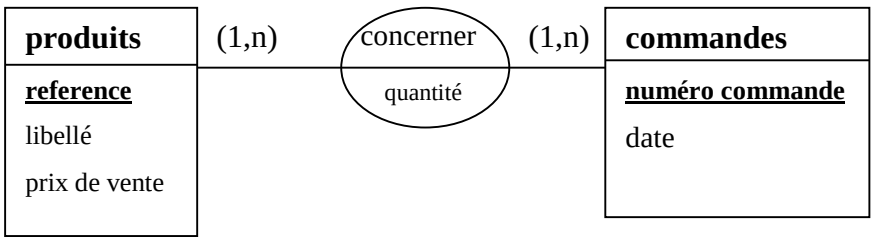
2.2.6. occurrence

Une *entité* est une *famille* d'objets ayant les mêmes caractéristiques, appelées propriétés. Une entité représente un ensemble d'occurrences. Un *membre* de la famille est appelé **occurrence** d'entité.

Une propriété est une information élémentaire qui permet de décrire une entité ou une association. Une propriété peut prendre une **valeur** (On peut dire qu'une *valeur* est une *occurrence* de *propriété*)

De même, une *association* est un *lien* entre 2 entités ou plus, et une **occurrence d'association** est un lien entre *2* ***occurrences d'entités***.

Exemple : considérons le schéma suivant



Les propriétés prennent des valeurs pour chaque occurrence d'une entité.

Exemples d'occurrences

propriété	**valeur de propriété**
Référence :	**456**
libellé :	Manteau
Prix de vente :	1000
entité	**occurrence d'entite**
PRODUIT	Manteau de référence 456 à 1000
COMMANDE	numéro **123** du 08/10/02

association **occurrence d'association**

concerner est le lien entre la commande **123** et le manteau **456** pour une quantité de **3** unités

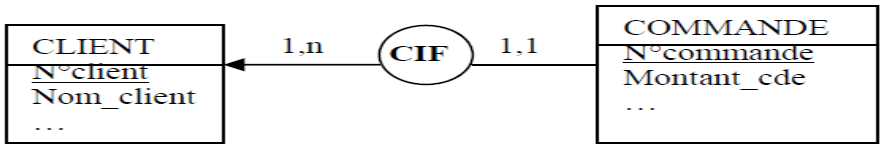
2.2.7. Contrainte d'intégrité fonctionnelle (CIF)

Une contrainte d'intégrité est une condition qui doit constamment vérifier le contenu de base de données.

on parle de Contrainte d'identité fonctionnelle (CIF) entre entités quand les entités sont reliées par une relation de cardinalité 0,1 ou 1,1, et 0,n ou 1,n.

Elle indique que l'une des entités est totalement déterminée par la connaissance de l'autre.

Par exemple si on connaît une commande bien précise, on connaît un client bien précis.



2.3. La méthodologie de conception d'une base de données.

La conception d'une base de données relationnelles (BDR) est un processus complexe consistant à définir à partir d'observation sur le monde réel, à modéliser un schéma-relation *normalisé.* Il s'agit d'organiser les données de la base en identifiant *les domaines, les attributs, les relations et les contraintes d'intégrités*

Pour concevoir une base de données, nous allons suivre les *étapes* suivantes :

2.3.1. Relever les informations qui sont manipulées et les règles qui sont appliquées lors de la gestion de l'entreprise que l'on se propose d'informatiser.

L'analyse des besoins consiste à identifier les besoins de l'entreprise et à les convertir en besoins d'un système. C'est une phase au cours de laquelle on rassemble des informations qui serviront à la modélisation des données. Cette démarche est basée d'une part sur la discussion avec les

utilisateurs réels et d'autre part, sur l'examen des documents existants (fichiers manuels, formulaires, ...). On pourra ainsi dresser une sorte de dictionnaires des informations

2.3.2. Modéliser les données

La modélisation des données comporte :

- ✓ L'expression de la sémantique de la représentation du monde réel par des données et les liens entre ces données;
- ✓ La traduction du schéma conceptuel dans un modèle de données de type hiérarchique, en réseau, relationnel …;
- ✓ La normalisation consistant à représenter les objets du modèle de données sous forme de tables normalisées afin de réduire la quantité de données redondantes présentes et les anomalies dans la base de données.

2.3.3. Les méthodes de conception d'une base

De nombreuses méthodes de conception de bases de données relationnelles ont été proposées. Il existe deux approches complémentaires importantes dont l'approche synthétique (inductive) et l'approche analytique (déductive).

2.3.3.1. Approche synthétique (entité association)

L'approche synthétique a pour objectif d'obtenir un schéma relationnel à partir d'un ensemble d'entités types modélisant au mieux le monde réel. La méthode basée sur le modèle entité-association (E/A) de Chen et la méthode synthétique de Codd.

Ses concepts sont l'entité, l'association, la cardinalité.

Dans cette approche, on part des objets visibles et on arrive au modèle entité relation selon les étapes suivantes :

- ✓ Modéliser les entités qui apparaissent le plus naturellement possible puis ajouter les relations ;
- ✓ Affecter un identifiant et les propriétés descriptives aux objets sans penser au fonctionnement mais respecter la règle de non répétitivité.
- ✓ S'assurer que toutes les entités types participent au moins à une relation.
- ✓ Préciser les cardinalités
- ✓ Rechercher les éventuelles dépendances fonctionnelles (ou CIF) et procédersi possible aux décompositions

2.3.3.2. *Approche analytique* (normalisation)

L'objectif de la normalisation est d'obtenir un schéma relationnel tel que la redondance des données et les anomalies de stockage soient évitées.

La normalisation réorganise les relations obtenues par la méthode synthétique en éliminant les attributs multivalés.

On distingue 6 formes normales de relation : 1ere forme normale, 2ere forme normale, 3ere forme normale, forme normale de BOYCE-CODD, 4ere forme normale et 5ere forme normale.

2.4. Description d'un schéma entité/association

La modélisation conceptuelle que nous proposons pour un univers ou une entreprise dont on veut stocker les données, conduit à l'élaboration d'un schéma entité/association.

2.4.1. la Détermination du diagramme entité/association

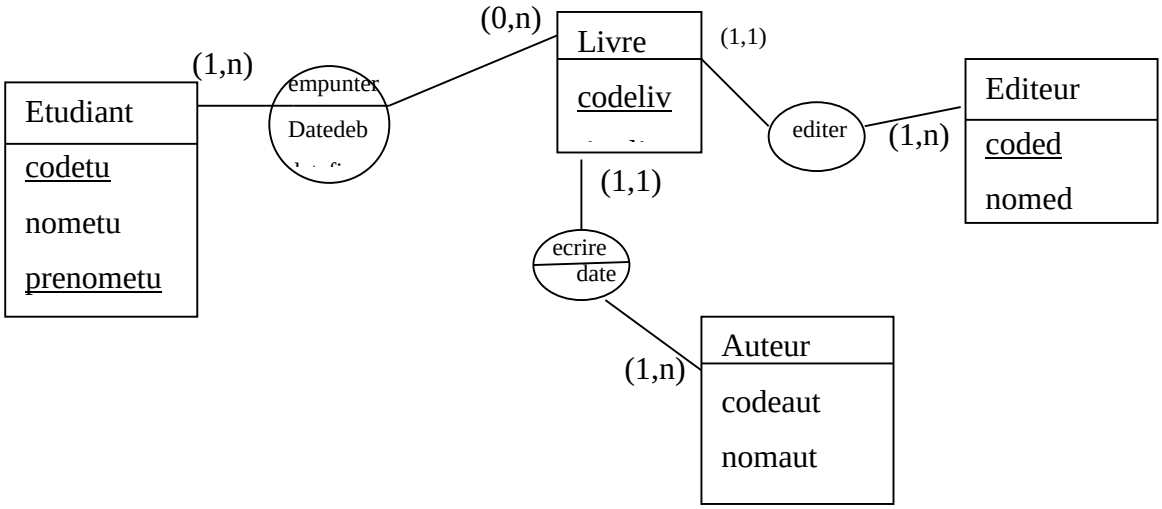
On détermine

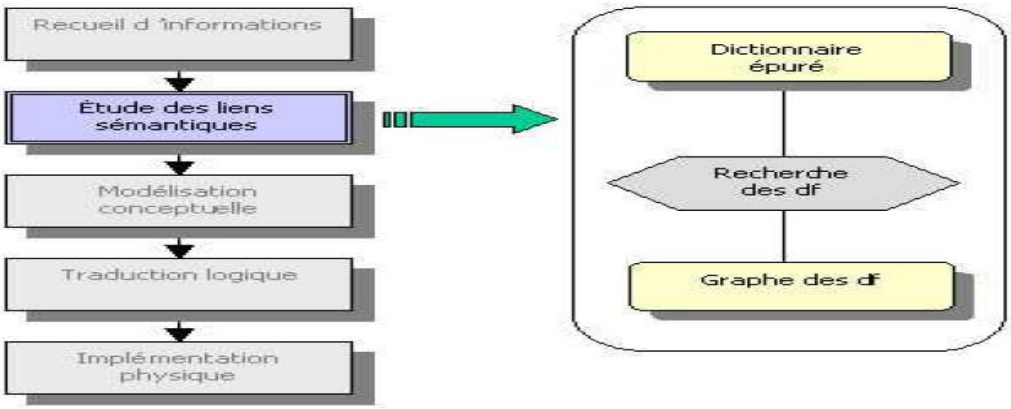
- ✓ Les entités et leur identifiant ;
- ✓ Les associations qui représentent les liens (relations) entre entités;
- ✓ Les cardinalités des associations ;
- ✓ On représente le tout par un graphique appelé diagramme entité association de CHEN.

2.4.2. Le formalisme

En 1976 CHEN a défini un outil de représentation graphique: le modèle entité/association

Exemple :





Chap3 Le Modèle relationnel et le passage du modèle Entité - Association en modèle relationnel

Le modèle relationnel est basé sur une organisation des données sous la forme de tables.

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

3.1. Modèle de bases de données relationnelles.

3.1.1. Introduction

En 1970, Codd, mathématicien au centre de recherches d'IBM San-José (Carlifornie), formalisa le modèle relationnel qui est un système à base de tables ou relations. Ce modèle conceptuel constitue un progrès important car il repose sur une représentation unifiée de l'information sous *forme de tables*. Il dispose d'un fondement mathématique solide avec l'algèbre relationnelle. Il permet une plus grande indépendance entre les applications, les données et le support physique. Il n'apparait commercialement qu'au début des années 80. En effet à partir de 1980, on l'apparition des SGBD relationnels sur le marché (Oracle, Ingres, Informix, Sybase, DB2 …)

Le succès du modèle relationnel auprès des chercheurs, concepteurs et utilisateurs est dû à la puissance et à la simplicité de ses concepts. En outre, contrairement à certains autres modèles, il repose sur des bases théoriques solides, notamment la théorie des ensembles et la logique des prédicats du premier ordre. Ainsi, toutes les données d'un système de gestion de bases de données relationnelles sont représentées sous forme de tableaux de valeurs appelés relations.

3.1.2. Les objectifs

Les objectifs du modèle relationnel sont :

- ✓ proposer des schémas de données faciles à utiliser ;
- ✓ améliorer l'indépendance logique et physique ;
- ✓ mettre à la disposition des utilisateurs des langages de haut niveau ;
- ✓ optimiser les accès à la base de données ;
- ✓ améliorer l'intégrité et la confidentialité ;
- ✓ fournir une approche méthodologique dans la construction des schémas.

3.1.3. Les concepts de base

3.1.3.1. Le domaine

Le domaine est l'ensemble fini ou infini de valeurs. Cet ensemble de valeurs est caractérisé par un nom. Il s'agit de l'ensemble des valeurs admissibles pour un attribut d'une relation.

Exemples :

D1 = {entiers} ; D1 est l'ensemble des entiers.

D2 = {chaînes de caractères} ; D2 est l'ensemble des chaines de caractères.

nometu a pour domaine l'ensemble de chaîne de caractères ou, plus simplement, chaîne).

couleurs = {rosé, blanc, rouge}

3.1.3.2. L'attribut

Un attribut est un identificateur décrivant une information stockée dans une base. Il prend ses valeurs dans un domaine. Autrement dit un attribut est une colonne d'une table.

Exemple : nometu désigne le nom de l'étudiant.

Nometu peut prendre la valeur '*Koffi'* dans l'ensemble des chaine de caractères.

3.1.3.3. La relation

Une relation est un sous-ensemble du produit cartésien de n domaines d'attributs (n > 0), caractérisée par un nom unique.

Exemple :

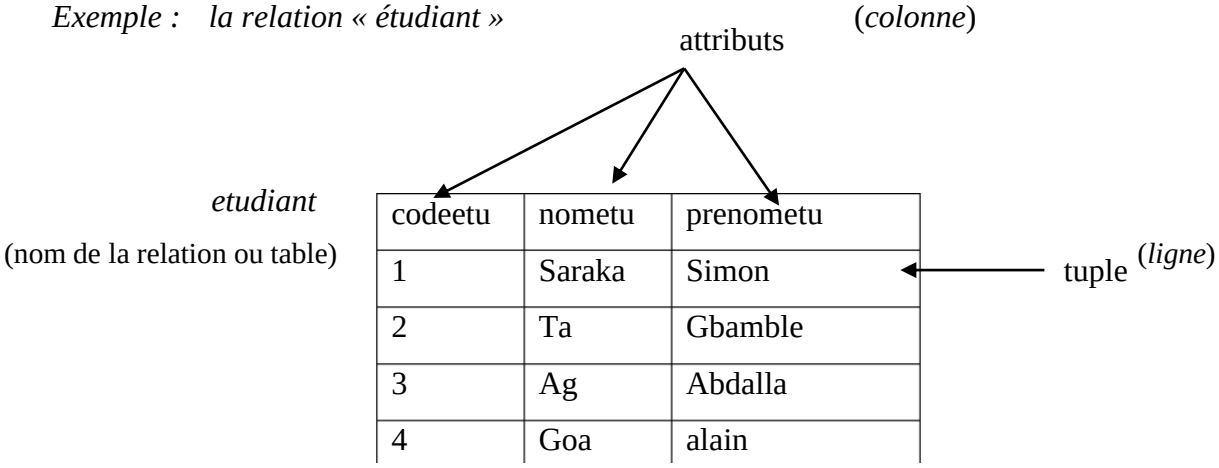
3.1.3.4.Le degré d'une relation

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

3.1.3.5. La cardinalité d'une relation

La cardinalité d'une relation est le nombre de tuples(*le* nombre d'occurrences) qui la composent.

Exemple : la relation « étudiant »



La représentation d'une relation

*La relation « **etudiant** « contient **3** attributs correspondant au code de l'étudiant, nom de l'étudiant et au prénom de l'étudiant, **4** tuples. Donc le degré de la relation est **3** et sa cardinalité est **4**.*

La relation etudiant est définie de la manière suivante :

Etudiant ={(1, Saraka, Simon) , (2, Ta, Gbamble), (3, Ag, Abdalla), (4, Goa, Alain)}

3.1.3.6. Le schéma d'une relation

Le schéma d'une relation est défini par la *liste de ses attributs*.

Exemple:

Le schema de relation " **auteurs**(codeaut, nomaut)" représente le fait que chaque occurrence d'entité **auteurs** a un *matricule* et *nom*.

Le schema de la relation etudiant notée *Etudiants* est défini par :

Etudiants (codetu ,nometu, prenometu, photoetu , adretu)

3.1.3.7. Le tuple

Un ***tuple*** est une ligne d'une relation ou table.

Les données sont organisées sous forme de tables à deux dimensions, les lignes sont appelées n-uplet ou *tuple* en anglais ;

3.1.3.8. La clé

Une clé est constituée de 1 ou plusieurs attributs. Elle peut être clé primaire ou clé étrangère de la relation.

3.1.3.9. La clé primaire d'une relation

La clé primaire d'une relation est un attribut *(l'identifiant de l'entité)*. Elle est choisie comme étant la clé *unique* d'accès aux tuples de la relation.

3.1.3.10. La clé étrangère d'une relation.

La clé étrangère d'une relation est un attribut qui permet d'identifier de façon unique un tuple faisant *référence à une clé primaire* appartenant à une autre table

3.1.3.11. La clé candidate

La clé candidate est l'ensemble *minimum d'attributs* susceptibles de *jouer le rôle* de la clé primaire.

3.1. Traduction des entités

Toute entité est traduite selon les trois règles suivantes :

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

Exemple **:**

Etudiant
codetu
nometu
prenometu
photoetu
adretu

devient La relation **Etudiant** (codetu, nometu , prenometu, photoetu, adretu) représente le fait que chaque occurrence d'entité etudiant a un matricule un nom, un prenom , une photo et une adresse**.**

L'entité etudiant qui devient *une relation Etudiant ou table Etudiant* dans laquelle les attributs deviennet les colonnes et, l'identifiant codetu constitue alors la clé primaire de cette relation.

3.2. Traduction des associations

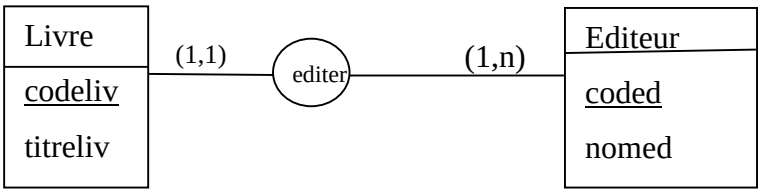
La traduction d'une association s'effectue selon les cardinalités relatives aux entités participant à l'association. Plusieurs cas peuvent se présenter.

3.2.1. Associations de type un à plusieurs

Une association un à plusieurs est une association liant une entité A de cardinalité 0,N ou 1,N et d'entité B de cardinalité 0,1 ou 1,1. On dit qu'on a une association père-fils ou maitre –esclave.

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

- ✓ L'entité A (Entité père) devient une relation (père).
- ✓ L'entité B (Entité fils) devient une relation (fils).
- ✓ *L'identifiant* de l'entité A (père) devient attribut de la relation fils. Cet attribut est désigné comme *clé étrangère*.
- ✓ Les attributs de l'association migrent vers la relation fils et deviennent ses attributs.



les associations *ecrire* et *editer* ne deviennent pas des relations, leurs identifiants migrent dans la relation **Livre** , on parle de clé étrangère. On obtient donc :

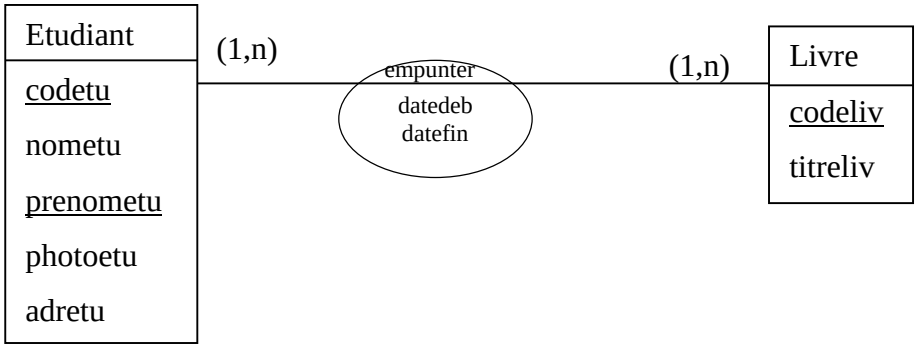
Livre (codeliv, **coded #** ,**codeaut#** , titreliv , date)

3.2.2. Association Plusieurs-à-Plusieurs

Une association de plusieurs à plusieurs est une association reliant une entité A de cardinalité 0, N ou 1, N et une entité B de cardinalité 0, N ou 1, N

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

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



On obtient les relations suivantes:

Etudiant (codetu, nometu , prenometu, photoetu, cel)

Livre (codeliv, **coded #** ,**codeaut#** , titreliv , date)

Emprunter(codetu ,codeliv#,datedeb#, datefin)

3.3. Association un à un

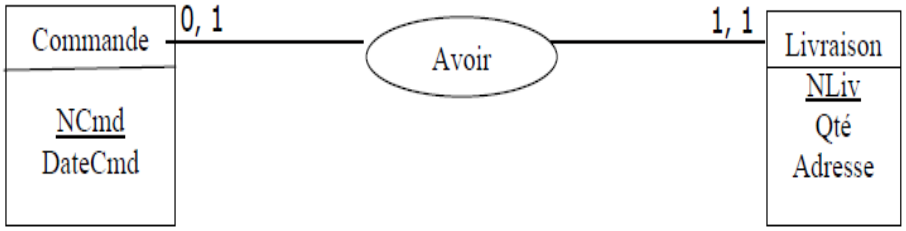
Une association de type un à un est une association reliant une entité A de cardinalité 0, 1 ou 1, 1 et une entité B de cardinalité 0, 1 ou 1, 1

Pour ce type d'association deux traductions sont possibles :

3.3.1. Première solution

Les deux entités seront transformées en deux relations. Une de ces deux relatons sera choisie et étendue par la liste des attributs éventuels de l'association ainsi que de l'identifiant de l'autre entité en tant que clé étrangère. Ce choix se base sur la séquence temporelle de création des entités. L'entité qui sera créée en second lieu aura comme clé étrangère l'identifiant de l'entité créée en premier lieu. Cette solution est la plus adaptée dans le cas où une ou les deux cardinalités minimales sont nulles.

Exemple **:**



Le modèle relationnel correspondant est le suivant :

Commande (NCmd, DateCmd)

Livraison (NLiv, Qté, Adresse, **# NCmd**)

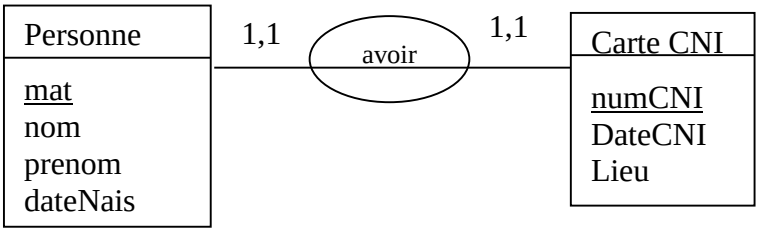
La relation *'Livraison'* a comme clé étrangère l'identifiant de la relation *'Commande'* car la création d'une livraison survient après la création d'une commande.

3.3.2. Deuxième solution

Les deux entités et l'association seront transformées en une seule relation contenant les attributs des deux entités ainsi que les attributs éventuels de l'association, l'identifiant de l'entité A ou de l'entité B sera choisie comme clé primaire de la nouvelle relation.

Cette solution est surtout utilisée dans le cas où les deux entités ont des cardinalités 1,1 qui ne sont pas sujettes à des modifications dans le temps.

Exemple :



Le modèle relationnel correspondant est le suivant :

Personne (mat, Nom, Prenom, DateNais)

CarteCNI (numCNI, DateCNI, Lieu, # mat)

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

On aura donc **Personne** (mat, Nom, Prenom, DateNais, numCNI, DateCNI, Lieu, # mat)

Chap4 Normalisation d'une base de données relationnelles

4.1. Introduction

La normalisation correspond au processus d'organisation des données dans une base de données. Ce processus comprend la création de tables et l'établissement de relations entre celles-ci conformément à des règles conçues à la fois pour protéger les données et pour rendre la base de données plus flexible grâce à l'élimination de la redondance et des dépendances incohérentes.

Les données redondantes entraînent un gaspillage d'espace disque et créent des problèmes de maintenance.

Il existe plusieurs règles de normalisation des bases de données. Ces règles sont connues sous le nom de « formes normales ».

Les formes normales des relations et les mécanismes pour les construire permettent d'obtenir des relations non redondantes.

Le concept de forme normale a été introduit par E.F.Codd. Il faut noter que le seul élément qui est pris en compte par les formes normales est la non redondance d'informations d'un schéma.

4.2. Les règles et les contraintes de modélisation

Cette analyse des dépendances est la deuxième phase de l'analyse des données. Elle a pour but de préparer la phase suivante, à savoir le MCD en recherchant les *liens entre les différentes données* également appelés *les dépendances*. L'analyse de ces liens produira comme document un diagramme ou un graphe des dépendances. Cette étape reste néanmoins importante en phase de démarrage dans l'analyse des données.

4.2.1. Les dépendances fonctionnelles

Le but des dépendances fonctionnelles et de la théorie de la normalisation est de s'assurer que le schéma relationnel défini pour une base de données est correctement construit. Un mauvais schéma relationnel peut en effet entrainer des anomalies lors des manipulations.

.

4.2.1.1. Définition de la dépendance fonctionnelle

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

On notera une telle dépendance fonctionnelle $X \rightarrow Y$ (X détermine Y ou Y dépend fonctionnellement de X).

exemple:

Dans la relation **Etudiant** (codetu, nometu , prenometu, photoetu, cel)

codetu alors on connait le *nom* de manière unique

si on connait le *codetu* alors on connait le *prenom* de manière unique.

on alors que *codetu* détermine les attributs nometu et prenometu et autres

on note : codetu ⟶ nometu et codetu ⟶ prenometu

Un *ensemble d'attributs B dépend fonctionnellement d'un ensemble A* (A ⟶ B) si et seulement si 2 tuples coïncident pour des valeurs sur A, coïncident également sur B.

exemple :

catalogues (fournisseur, produit, prix) ; fournisseur, produit ⟶ prix

concerner (reference, numero commande, quantité) ; reference, numero commande ⟶ quantité

Remarque :

Toutes les propriétés d'une entité dépendent fonctionnellement de l'identifiant.

4.2.1.2. Propriétés des dépendances fonctionnelles

- ***reflexivité***

X ⟶ X X C Y ⟹ Y ⟶ X

- ***transitivité***

X ⟶ Y Y ⟶ Z ⟹ X ⟶ Z

- ***projection***

X ⟶ Y,Z ⟹ X ⟶ Y et X ⟶ Z

- ***augmentation***

X ⟶ Y ⟹ V Z X,Z ⟶ Y

- ***additivité***

X ⟶ Y et X ⟶ Z ⟹ X ⟶ Y, Z

4.2.2. Quelques dépendances fonctionnelles

4.2.2.1.Dépendance fonctionnelle élémentaire

Une dépendance fonctionnelle élémentaire est une dépendance fonctionnelle de la forme X → A, où A est un attribut unique n'appartenant pas à X et où il n'existe pas X′ inclus au sens strict dans X (*i.e.* X′ ⊂ X) tel que X′ → A.

Exemples ***:***

RefProduit → LibProduit est élémentaire

(NumFacture, RefProduit) → QtéFacturée est élémentaire (ni la référence produit seule, ni le numéro de facture seul permettent de déterminer la quantité)

Remarque **:**

(*NumFacture, RefProduit)* → *LibProduit n'est pas élémentaire puisque la référence du produit suffit à déterminer le libellé.*

4.2.2.2. Dépendance fonctionnelle directe

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

Exemple **:**

soient les dépendances fonctionnelles :

NumFacture → Numfourniseur et Numfournisseur → Nomfournisseur

NumFacture → Nomfournisseur n'est pas une dépendance fonctionnelle directe puisqu'elle est obtenue par transitivité.

Cependant *Numfournisseur → Nomfournisseur est une dépendance directe.*

.

4.2.2.3. Dépendances fonctionnelles symétriques

Certaines dépendances fonctionnelles sont symétriques.

Exemple :

NoSérieVéhicule → NoImmatriculation et ***NoImmatriculation → NoSérieVéhicule***

Dans ce cas, il faut choisir de privilégier une des dépendances fonctionnelles, en fonction des règles de gestion. S'il s'agit d'assurer le suivi du véhicule tout au long de sa vie, le no d'Immatriculation pouvant changer, on choisira dépendance fonctionnelle (***NoSérieVéhicule → NoImmatriculation)***.

4.2.2.4. Dépendance fonctionnelle multivaluée

Etant donné une relation R(C) , si à une valeur de A peuvent correspondre plusieurs valeurs de B, on dit qu' il existe une dépendance multivaluée de A vers B et on note A B

Exemple :

assuré(N°assur, Nom,sinistre,vehicule)

N°assu ⟶▸ *véhicule (car un assuré peut posséder plusieurs véhicules)*

4.2.3. Graphe de dépendance fonctionnelle et couverture minimale

4.2.3.1. Graphe de dépendance fonctionnelle

Les dépendances fonctionnelles peuvent être représentées à l'aide d'un graphe dont les *nœuds* sont les *attributs* impliqués dans les dépendances et les *arcs* les *dépendances elles-mêmes.*

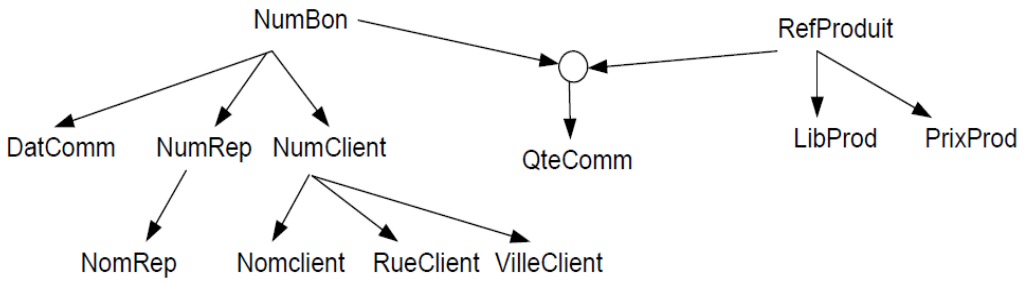
L'origine d'un arc peut être multiple mais sa cible doit être un nœud unique.

Exemple **:**

On a les dépendances fonctionnelles suivantes :

NumBon → DatComm, NumClient, NumRep
NumClient → Nomclient, RueClient, Villeclient
NumRep → NomRep
RefProduit → LibProd, PrixProd
NumBon, RefProd → QteComm

On obtient ***:***



4.2.3.2. Couverture minimale

A partir d'un ensemble F de dépendances fonctionnelles élémentaires, on peut composer par transitivité d'autres DF élémentaires. On aboutit à une fermeture transitive l'ensemble des DF élémentaires. Ainsi on obtient la couverture minimale ou la *Structure d' Accès Théorique*(SAT) en supprimant toutes les fermetures.

transitives.

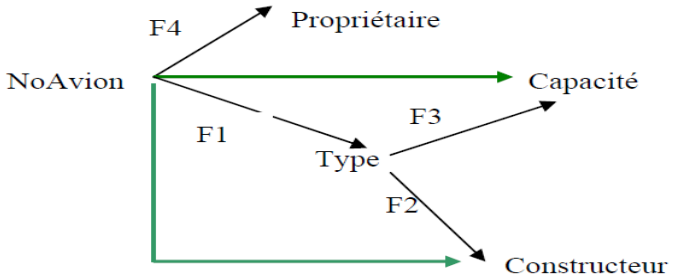
Exemple :

Avion(NoAvion, Type, Constructeur, Capacité, Propriété). Avec les dépendances fonctionnelles suivantes :

F1 : NoAvion → Type *F4 : NoAvion → Propriétaire*
F2 : Type → Constructeur *F3 : Type → Capacité*

On a F= {F1, F2, F3, F4}

Le graphe de dépendance donne :



4.3. Normalisation des Bases de Données relationnelles

La théorie de la normalisation est liée au modèle relationnel de données .

4.3.1. Les objectifs

Les objectifs de la normalisation sont :

- ✓ Eliminer les incohérences au sein des données ;
- ✓ Eliminer les redondances, réduisant du même coup l'espace de stockage ;
- ✓ Limiter les pertes de données ;
- ✓ Améliorer les performances des traitements.

4.3.2. Les formes normales

L'un des principaux travaux de mise en place d'une base de données relationnelles va consister à définir des relations ou tables en formes normales de plus haut degré possible.

Ces formes normales ont été définies pour permettre la décomposition des relations sans perte d'informations en utilisant la notion de dépendance fonctionnelle.

Chaque forme normale marque une étape supplémentaire de progression vers des relations présentant de moins en moins de redondance. Cette normalisation peut être effectuée, et c'est préférable, pendant la phase de conception sur le modèle entités-associations.

4.3.2.1.1^{ere} forme normale

Une relation est en 1ere *forme normale* (1FN) si chacun de ses attributs contient une *valeur atomique*.

Codeliv	titreliv	nomaut	genre
1	L' enfant noir	Camara Laye	roman
2	Pagne noire	Bernard dadié	Conte
3	Afrique Debout	Bernard dadié	poesie

en 1FN

Codeliv	titreliv	nomaut	genre
1	L' enfant noir	Camara Laye	roman
2	**Pagne noire, Afrique Debout**	Bernard dadié	Conte

Pas en 1FN car livre n°2 possède 2 titres différents

4.3.2.2. 2^{ere} forme normale

Une relation est en *2eme forme normale* (2FN) si elle est en 1FN et si les es attributs n'appartenant pas à la clé primaire (lorsque celle-ci est constituée d'un groupe d'attributs) ne dépendent pas fonctionnellement d'une partie de la clé.

Toute dépendance fonctionnelle de la clé vers un attribut est élémentaire.

vendre(N°four, N°produit, nomfour,prix)

N°four	N°produit	nomfour	prix
F1	P2	Saraka	1000
F2	P2	Goa	80
F3	P3	Goa	120

La relation en 1 FN mais pas en 2 FN car la connaissance de N°four *détermine le nom du fournisseur or la clé est (*N°four, N°produit*).*

Solution : on décompose l'entité ***vendre*** *en supprimant nomfour et en créant 2 schémas de relation.*

Vendre(*N°produit,prix) ;* ***fournisseur****(N°four,nomfour)*

4.3.2.3. 3eme forme normale

Une relation est en *3eme forme normale* (3FN) si elle est en 2FN et si de plus, tout attribut non clé ne dépend pas fonctionnellement d'un autre attribut non clé. Toute dépendance fonctionnelle élémentaire de la clé est directe.

Exemple :

LIVRE (N°Liv, Titre, Auteur, Pays, Genre) pas en 3FN

car Auteur → Pays or auteur ne compose pas la clé de la relation.

On décompose en supprimant l'attribut dépendant ***Pays*** *de la table et en créant une table contenant les attributs de la dépendance.*

LIVRE (N°Liv, Titre, Auteur, Genre) et AUTEUR (Auteur, Pays)

Remarque :

La 3ème forme normale permettra d'éliminer les redondances dues aux dépendances transitives.

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

4.3.2.4. Forme normale BOYCE-CODD

Une relation est en forme normale de Boyce-Codd (BCNF) si et seulement si les seules dépendances fonctionnelles élémentaires sont celles dans lesquelles une clé détermine un attribut non-clé.

Exemple :

ADRESSE (Code_post, Ville, Rue) en 3FN mais pas en BCFN car « Ville » membre de la clé dépend fonctionnellement de « Code_post » non membre de la clé.

On décompose en supprimant Code_post de la relation ADRESSE et en créant une relation

CODE (Code_post, Ville).

Remarque :

Dans la pratique, la plupart des problèmes de conception peuvent être résolus en appliquant les concepts de troisième forme normale et de forme normale de BOYCE-CODD.

4.3.3. Normalisation par rapport à d'autres contraintes

Dans certains cas de modélisation, les dépendances fonctionnelles ne suffisent pas à traduire toute la sémantique de l'application. Il faut donc introduire de nouvelles contraintes et les rajouter dans le processus de normalisation.

Remarque :

- *La dépendance fonctionnelle permet de définir les premières formes normales jusqu'à la forme normale de Boyce-Codd (1FN, 2FN, 3FN et BCNF).*
- *La dépendance multivaluée permet de définir la quatrième forme normale (4FN) et la dépendance de jointure la cinquième forme normale (5FN).*

4.3.3.1.4ème forme normale (4FN)

Une relation est en quatrième forme normale (4FN) si, et seulement si, elle est en forme normale de BOYCE-CODD et si elle ne possède pas de dépendance multivaluée ou si X ↠ Y étant la dépendance multivaluée, il existe une propriété A telle que X → A.

4.3.3.2. 5ème forme normale (5FN)

*Une relation R est en cinquième forme normale (5FN) si, pour toute dépendance de jointure non triviale *{X1, X2, …, Xn} dans R, chacun des Xi contient une clé candidate de R.*

Exemple :

Exemple : Soit le schéma de relation **VINS**(Buveur,Cru,Producteur) modélisant des vins bus par des buveurs d'un cru donné et commandés à un producteur produisant ce cru et une de ces extensions :

BUVEUR	CRU	PRODUCTEUR
Jacques	Chablis	Pierre
Jacques	Chablis	Nicolas
Jacques	Volnay	Nicolas
Louis	Chablis	Nicolas

Cette extension présente encore des redondances.

Si la contrainte d'intégrité : " tout buveur ayant bu un cru et ayant commandé à un producteur produisant ce cru, a aussi commandé ce cru à ce producteur" est déclarée, le schéma relationnel VINS obéit à la dépendance de jointure :

* [Buveur Cru, Buveur Producteur, Cru Producteur]

et peut être décomposé en trois schémas (composantes de jointure).

BUVEUR	CRU
Jacques	Chablis
Jacques	Volnay
Louis	Chablis

BUVEUR	PRODUCTEUR
Jacques	Pierre
Jacques	Nicolas
Louis	Nicolas

CRU	PRODUCTEUR
Chablis	Pierre
Chablis	Nicolas
Volnay	Nicolas

Dans le processus de normalisation vu précédemment, on rajoute les dépendances de jointure(DJ) en introduisant chaque ensemble d'attributs associé à chaque composante de jointure (CJ).

Chap5 l'algèbre relationnelle

5.1. introduction

Afin de manipuler ces relations, il a été défini un ensemble d'opérateurs constituant une algèbre.

Le langage de base de données est issu de l'algèbre relationnelle qui est un sous ensemble de la théorie des ensembles. Cette théorie manipule des ensembles typés dont tous les éléments sont de même type.

Exemple : $\{(a,b);(a,c)\}$ et non de type $\{(a);(a,c)\}$

Elle a été inventé en1970 par E.F CODD et est constituée d'un ensemble d'opérations formelles sur les relations .les opérations relationnelles permettent de créer une nouvelle relation (table) à partir d'opérations élémentaires sur d'autres tables (union, intersection, différence)

5.2. les opérations de bases

Une opération de base est définie par le fait qu'elle ne peut être réalisée par combinaison d'autres opérations. Il existe 5 opérations élémentaires pouvant être classées en 2 catégories.

- **Les opérations unaires**
- **Les opérations ensemblistes**

5.2.1. Les opérations unaires

a) La projection ($\prod$)

La projection est une coupure verticale de la relation (table). Elle consiste à créer une relation à partir d'une autre en ne gardant que les colonnes spécifiées dans la projection.

On note proj $_{x1,x2,\ldots,xn}$(R) où $x_1,x_2,\ldots x_n$ représentent les colonnes que l' on garde .

Exemple : Etudiant (codetu, nometu , prenometu, photoetu, cel)

on désire la liste des nom et prenoms des étudiants .

$\prod_{\text{Nometu,prenometu}}$ *etudiant* ou proj(nometu, prenometu)

b) La sélection (δ)ou restriction

La sélection est une coupure horizontale de la relation (table) . Elle consiste à créer une relation à partir d'une autre en ne gardant que les lignes pour lesquelles une colonne vérifie certaines propriétés

On note select $_Q$(R) où $_Q$ représente la qualification (la condition à réaliser).

Exemple : dans la table auteurs **Auteur** (codeaut ,nomaut, prenomaut) on désire les informations relatives à Dadié

δ_{Auteurs} ou select(nomaut= 'Dadié')

Auteurs

Nomaut='Dadié'

Les comparateurs sont { = , $\neq, \leq, \geq, \ldots$}

.

5.2.2. Les opérations ensemblistes

Ceux sont l'union ($\cup$), la différence (- ou \ ) et le produit cartésien (X) . Pour les appliquer il faudrait que la condition d'union-compatibilité soit vérifiée c'est à dire

$R \cup S, R \cap SouR - S$ n'a de sens que si les relations R et S sont de même parité et les attributs de même type.

a) L'union

L'union de 2 relations (tables) est une relation contenant l'ensemble des tuples appartenant à l'une ou l'autre des relations (ou les deux).

On note union(R,S) ou $R \bigcup S$

$R(A,B) \bigcup S(C,D) = \{ (x,y)/(x,y) \in R \text{ ou } (x,y) \in S \}$

b) La différence

La différence entre 2 relations (tables) est une relation contenant l'ensemble des tuples appartenant à la première relation mais pas seconde.

On note minus(R,S) ou $R - S$ ou R\S

$R(A,B) - S(C,D) = \{ (x,y)/(x,y) \in R \text{ et } (x,y) \notin S \}$

c) Le produit cartésien

Le produit cartésien de 2 relations (tables) est une relation contenant la concaténation de l'ensemble des tuples d'une ligne d' une relation à ceux de l'autre des relations et c pour chaque).

On note product(R,S) ou D=RXS

D (A,B) = $R(A)XS(B) = \{ z = (x,y)/ x \in R \text{ et } \forall x' \in R, y' \in S, \exists z' = (x',y') \}$

5.3. les opérations dérivées

Les opérations élémentaires dites opérations de base vont permettre de déterminer d'autres opérations appelées opérations dérivées telles que la différence, la jointure …

5.3.1. L'intersection

L'intersection de 2 relations (tables) est une relation contenant l'ensemble des tuples appartenant aux 2 relations à la fois.

On note inter(R,S) ou $R \cap S$

$R(A,B) \cap S(C,D) = \{ (x,y) / (x,y) \in R \text{ et } (x,y) \in S \}$

5.3.2.La jointure (∞)

La jointure (∞) est un produit cartésien auquel on applique une condition de sélection de la forme attribut$_1$ (-) attribut$_2$ où attribut$_1$ et attribut$_2$ sont de meme type et sont dans des relations différentes.

Exemple : on donne

Etudiant (codetu, nometu , prenometu, photoetu, adretu)
Livre (codeliv, **coded #** ,**codeaut#** , titreliv , date)
Emprunter(codetu ,codeliv#,datedeb#, datefin)

Etudiant ∞ Emprunter (codetu, nometu , prenometu, photoetu, codeliv#,datedeb#, datefin)
Etudiant.Codeetu = emprunter.codeetu

5.3.3. La division

Une relation D= R÷S si et seulement si

D (A)= R(A,B)÷S(B)= $\{x / \forall y \in S, (x,y) \in R\}$

5.4. Quelques requêtes avec l'algèbre relationnelle

Exo1 : on donne les tables suivantes :

article (N°Ex, Titre, Page_deb, Page_fin)

auteur (Titre, Auteur)

mots-clefs (Titre, Mot-clef)

On désire :

1. Titre des articles de Lopez
2. Mots clefs associés au titre ′Base de données′
3. Liste des auteurs intéressés par le mot-clef ′programmation logique′
4. Les pages début et fin de l'article écrit par Kouadio dans le n°27
5. Quels sont les auteurs qui ont écrit dans le n°204 ?
6. Liste des articles (Titre) traitant de logique (mot-clef) et postérieur à la publication de ′logique formelle′ (titre)
7. Liste des articles anonymes
8. Quels articles ont exactement un auteur ?

Chap5 Langage SQL (rappels)

5.1. La normalisation de SQL

SQL (Structured Query Langage = langage de requêtes structurés) est un langage de création et d'interrogation de bases de données. Avec l'invention du modèle relationnel en 1970, de nombreux langages ont fait leur apparition dont ***IBM Sequel*** (Structured English Query Language) en 1977, ***IBM Sequel/2 , IBM System/R, IBM DB2.***

Ce sont ces langages qui ont donné naissance au standard SQL, normalisé en 1986 par l'**ANSI** (American National Standard Instute) dite norme ***ISO 9075*** pour donner SQL/86. Puis en 1989 la version SQL/89 a été approuvée. La norme SQL/92 est appelée désormais ***SQL2***.

La norme **SQL 3** aussi référencée par l'ISO sous le nom de code SQL/**1999** possède de nombreux atout pour faire de SQL un langage d'une puissance sans équivalente dans le monde du *relationnel objet.*

La norme SQL **2003** introduit des fonctions pour la manipulation XML, « window functions », ordres standardisés et colonnes avec valeurs auto-produites (y compris colonnes d'identité).

Enfin la norme SQL **2008** avec l'ajout de quelques fonctions de fenêtrage (ntile, lead, lag, first value, last value, …), limitation du nombre de ligne (OFFSET / FETCH), amélioration mineure sur les types distincts, curseurs et mécanismes d'auto incréments.

5.2. Le langage SQL2

Aujourd'hui le langage SQL est un élément indispensable pour les bases de données. Il est utilisé par l'ensemble des SGBR (oracle, SQLserver, informix, Acces …).

Il est composé de 2 grandes parties :

- **Le langage de description de données (LDD)** permet de d'écrire les fonctions d'administration d'une base de données telle la création d'une table, d'une vue, la gestion des utilisateurs, la spécification des contraintes d'intégrité et les outils d'optimisation.
- **Le langage de manipulation de données (LMD)** qui définit l'ensemble des opérations de manipulation des tables comme la sélection, la modification ou la suppression en fonction de critère donné.

5.2.1. Le langage de manipulation de données (LMD)

SQL est un langage de manipulation de données signifie qu'il permet de sélectionner, d'insérer, modifier ou supprimer des données dans une table d'une base de données relationnelles .

Chaque table contient une ou plusieurs colonnes dites *attributs* et 0 ou plusieurs lignes appelées t*uples.*

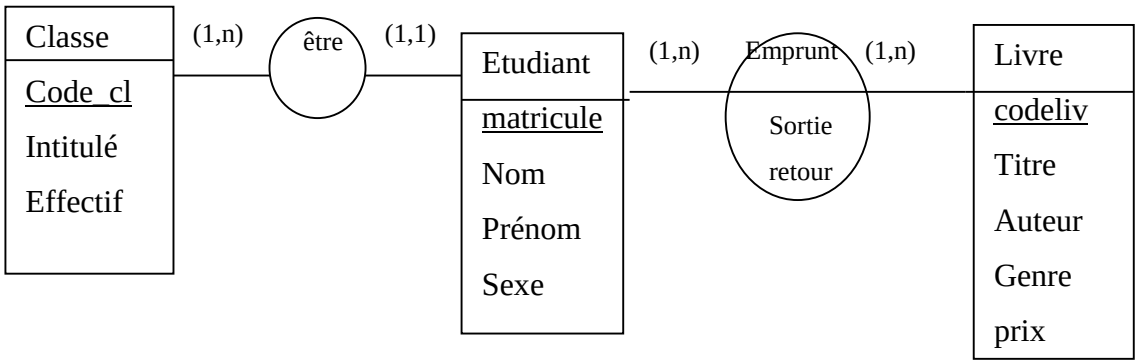
Pour manipuler la Base de Données c'est-à-dire les tables ou les vues on a les commandes suivantes :

- ✓ **Select** : Pour la recherche de données ;
- ✓ **Insert** : Pour l'insertion de n-uplet (ligne) dans une table ;
- ✓ **Delete** : Pour la suppression de n-uplet dans une table ;
- ✓ **Update** : Pour la modification de n-uplet dans une table

A partir d'un modèle conceptuel de données nous allons appliquer les règles de transformation afin d'obtenir un modèle logique de données puis créer les différentes tables avec le langage SQL.

Exemple :

Soit le modèle conceptuel de données ci-dessous :



On obtient le modèle logique de données suivant :

Etudiant (matricule, nom , prenom, sexe, *code_cl*)
Livre (code_liv, titre ,auteur , genre , prix)
Emprunter(*matricule* ,*codeliv*,sortie ,retour)

5.2.1.1.Sélection de tuples

La principale commande du langage de manipulation des données est la commande **select**, basée sur l'algèbre relationnelle elle effectue des opérations de sélection de données sur plusieurs tables relationnelles par projection.

Syntaxe generale:

Select <clause d'unicité>

<liste de colonnes>

From <listes de tables>

Where <critère de sélection>

Group by <liste de colonnes>

Having <critère de sélection>

Order by <critère d' ordre>

En fait l'ordre SQL `SELECT` est composé de 6 clauses dont 4 sont optionnelles.

5.2.1.1.1. Les requêtes simples

Chaque requête de sélection contient au minimum les clauses **select** et **from**.

Select permet de spécifier les attributs que l'on désire voir apparaître dans le résultat de la requête. *From* spécifie les tables sur lesquelles porte la requête.

Exemple ***:***

Afficher le matricule, le nom, le prénom des étudiants

Select matricule, nom, prenom

From Etudiant ;

Le caractère *étoile* (`*`) permet de récupérer automatiquement tous les attributs de la table générée par la clause `FROM` de la requête.

Exemple :

Nous voulons afficher toutes les informations relatives aux étudiants :

Select matricule, nom, prenom, sexe

From Etudiant ;

Ou encore

Select *

From Etudiant ;

- ✓ *Les opérateurs Distinct ou ALL*

  La **<clause d'unité**> définie au moyen du mot clé DISTINCT permet d'afficher un seul exemplaire des tuples . Sinon le SGBDR rapatrie toutes les lignes qui satisfont la requête, généralement dans l'ordre ou il les trouve, même si ces dernières sont en double pour l'opérateur ALL défini par défaut.

Exemple :

Afficher le nom , le prenom de tous les étudiants

Select distinct nom, prenom
From Etudiant ;

5.2.1.1.2. Les requêtes avec la clause WHERE

La clause **WHERE** permet d'inclure une condition à la sélection, on l'appelle aussi **restriction**.

Sa syntaxe est :

Select < liste des attributs (colonnes) >
From < liste des tables >
Where < expression logique > ;

Exemples :

Nous voulons afficher les étudiants de nom Koffi.

Select *
From Etudiant
Where nom = ' Koffi ';

Par exemple, pour sélectionner les romans *policiers*, on utilise la requête :
Select *
From Livre
Where genre='Policier';

5.2.1.1.3. Requêtes multi-relations ou plusieurs tables

Les requêtes multi-relations représentent en fait les **jointures**. Une jointure sert à lier des champs de différentes tables, mettre une relation entre eux. Il faut que les champs que l'on souhaite lier soient du même type.

La spécification d'un critère de jointure est nécessaire dans la clause **where** lorsqu'on a plus de 2 tables dans la <liste de tables>.

Elle s'effectue par une comparaison d'attributs de différentes tables.

Chaque nom de table peut être associé à un synonyme ou alias (*exemple «* Etudiant **E**, Livre **L »**).

S'il y a plusieurs tables dans la <liste des tables> penser à une jointure en définissant le critère de jointure dans la clause WHERE.

Exemple :

Nous voulons savoir les étudiants qui ont emprunté les livres de genre base de données.

Select distinct Etudiant.nom, Etudiant.prenom , Livre.titre
From Etudiant, Livre, Emprunt
Where Etudiant**.**matricule**=**Emprunt.matricule **and** Emprunt.code_liv=Livre.code_liv ;

Ou

Select distinct E.nom, E.prenom , L.titre
From Etudiant E, Livre L , Emprunt F
Where E.matricule=F.matricule **and** F.code_liv=E.code_liv ;

5.2.1.1.4. La clause GROUP BY

Cette clause sert à *grouper* les résultats d'une requête. Cela permet une meilleure lisibilité
La <liste de colonnes> de la clause GROUP BY doit contenir nécessairement des colonnes de la <liste de colonnes> de la clause select.
Exemple :
Nous allons afficher les livres empruntés par les étudiants par classe.

Select E.matricule, E.nom, E.prenom , L.titre, C.code_cl, C.intitulé
From Etudiant E, Livre L , Emprunt F, Classe C
Where E.matricule=**F**.matricule and F.code_liv=E.code_liv and C.codec_cl= E.code_cl
Group By C. code_cl ;

Pour appliquer des restrictions sur ces " sous-tables " on utilise la commande **Having** qui est en gros l'équivalent du **Where** pour les groupes

Le **<critère de sélection>** est une expression booléenne vraie ou fausse. L'évaluation de ce critère conditionne le choix ou le rejet du tuple sur lequel il s'applique. Le <critère de sélection> de la clause HAVING s'applique aux partitions crées par GROUP BY. De ce fait la clause having ne peut exister sans la clause group by.

Exemple :
Nous allons afficher les auteurs ayant écrit au moins de 2 livres.
Select auteur**,** COUNT(*)
From Livre
Group by Auteur
Having COUNT(*)> =2;

5.2.1.1.5. La clause ORDER BY

La clause ORDER BY permet de trier le résultat final de la requête.
Elle est donc la dernière clause de tout ordre SQL.

Le **<critère d'ordre**> est constitué de noms de colonnes de la <liste de colonnes> séparés par des virgules ou une liste de numéro séparés par des virgules. Chaque numéro correspond au rang d'une colonne dans la <liste de colonnes> de la clause SELECT.

A côté de chaque nom de colonne, l'on peut ajouter la mention « DESC » pour un tri dans l'ordre décroissent. Par défaut c'est la mention « ASC » ordre croissant.

syntaxe :

Order by expression [ASC | DESC] [, ...]

Exemple :

Nous voulons Afficher nom, prénoms et le nombre de livres empruntés par tout étudiant qui a emprunté plus de 4 livres puis classer par nombre décroissant de livres empruntés

```
Select  nom, prenoms, count(codeliv)
from   Etudiant E, Emprunt emp
where  E.mat= emp.mat
group by nom , prenoms
having count (codeliv) >= 4
order by 3 desc ;
```

5.2.1.1.6. La clause Where et d'autres opérateurs ou comparateurs de chaînes de caractères

En plus des opérateurs arithmétiques et logiques (+,-,*, /, OR, AND, NOT, =, <=,>=, <,>), les opérateurs particuliers suivantes sont utilisées dans le <critère de sélection> :

- Opérateur d'intervalle BETWEEN

```
Select *
From Livre
where prix between 7000 and 10000;
```

- Opérateur dans IN

```
Select *
 From Livre
 where prix in (7000, 8000, 9000, 10000);
```

- Opérateur de test de valeur nulle IS NULL

```
Select *
 From  Emprunt
 where  Retour  IS NOT NULL;
```

- Opérateur de ressemblance LIKE

Le caractère '_' remplace n' importe quel caractère et ' %' n' importe quelle séquence de caractères.

Select *

From Livre

Whrere titre Like '% les %' ;

Select *

From Livre

Whrere titre Like ''A_ %' ;

5.2.1.1.7. Les requetes avec les fonctions agrégats

Les fonctions agrégats permettent de faire du dénombrement, déterminer un maximum, un minimum, faire des moyennes …

Les fonctions de calcul COUNT(), SUM(), AVG(), MIN() et MAX() peuvent être utilisées dans la clause **Select** et dans la clause **Having**. Elles s'utilisent généralement conjointement avec la clause **Group by**.

Exemples :

Nous voulons afficher le prix des livres du genre roman.

Select *SUM (prix)*
From *Livre*
where *genre ='Roman'*;

*Nous voulons afficher le nombre d'auteurs ayant écrit au moins 2 livres, les prix minimum et maximum des livres***.**

Select auteur**,** COUNT(*), MIN (prix), MAX (prix)

From Livre

Group by Auteur

Having COUNT(*)> =2;

5.2.1.1.8. Les sous requêtes

On parle de requêtes **imbriquées** lorsqu'on a une requête dans la clause " **where** " ou " **having** " c'est à dire lorsque l'on fait une requête à l'intérieur d'une requête. On parle aussi de **sous-requêtes.** L'utilisation de requête SQL dans la clause WHERE (sous requête) est autorisée à condition de mettre la sous requête entre parenthèses. Les sous requêtes sont utilisées dans les cas suivants :

- Avec les opérateurs de comparaison

Nous souhaitons afficher les livres dont le prix est supérieur au prix moyen.

Select *

From Livre

where prix > (**select** AVG (prix) **from** Livre);

- Avec l'opérateur d' ensemble IN

Nous voulons afficher les livres déjà empruntés

Select *
From *Livre*
where *Code_liv* ***in*** *(select code_liv from Emprunt);*

- Avec les contrôles ANY ou ALL

Select *

From Livre

where prix > ANY(**select** prix **from** Livre **where** genre=' roman');

- Avec le test d'existence

Select *

From Livre

where EXISTS (**select *** **from** Emprunt **where** Livre**.**code_liv=Emprunt.code_liv);

5.2.1.2.insertion de tuples

L'insertion de nouvelles données dans une table se fait grâce à l'ordre **insert** suivi de la clause **into**. Il permet d'insérer de nouvelles lignes dans la table.

Syntaxe :

Insert into table (col1,..., coln)
values (val1,...,valn) ;

Insert into vue (col1,..., coln)
values (val1,...,valn) ;

ou

Insert into table (col1,..., coln)
select ... ;

Les lignes à insérer proviennent d'une requête d'interrogation. Là encore la correspondance entre colonnes doit être assurée.

Exemple :

Insert into Livre
Values ('15','tribalique','Lopez','nouvelle',2000) ;

Insert into Etudiant (matricule, nom , prenoms, sexe, *code_cl*)
Values ('5','Kouadio','Marcel','M','G1') ;

Insert into *etudiant2*
select *matricule, nom, prenom, sexe*
from *Etudiant ;*

5.2.1.3. suppression de tuples

La suppression de données dans une table se fait grâce à l'ordre **delete** suivi de la clause **From** qui précise la table sur laquelle la suppression s'effectue puis d'une clause **where** qui décrit l'ensemble des lignes qui seront supprimées. Il permet d'insérer de nouvelles lignes dans la table.

Syntaxe :

Delete From <nom table ou vue>

Where <critère de sélection> ;

Description

La clause Where est optionnelle, lorsqu'elle n'est pas précisée tous les tuples de la table ou de la vue sont détruits.

Exemple:

Delete from Emprunt

Where retour is not NULL;

5.2.1.4. Modification de tuples

La modification de données consiste à modifier des tuples (lignes) dans une table se fait grâce à l'ordre **update** suivi de la clause **set**. La clause **Where** permet de préciser les tuples sur lesquels la mise à jour se fera.

Syntaxe :

Update <nom table ou vue>

Set <col1>= <expression1>, <col2>=<expression2>,…

Where <critère de sélection> ;

Description

Chaque expression peut contenir des fonctions et combiner des opérateurs.

Exemple :

Update Emprunt

Set retour='12/04/09'

Where code_liv='10' and retour IS NUL;

5.2.1.5. Les transactions

Syntaxe :

Commit work;

Rollback work;

Description

La transaction (ensemble de commandes) est une unité logique de manipulation de données. La fin de la transaction spécifiée par « *commit work* » valide toutes les commandes « *rollback. Work* » annule toutes les commandes. De plus pour assurer la cohérence des données de la base, le système gère des verrous d'accès aux données. Les verrous sont levés à la fin de la transaction.

5.2.2. Le langage de description de données (LDD)

Le LDD permet la création et la définition des tables, vues, index, attributs…

Pour définir le schéma de BD on a les commandes suivantes :

- Déclaration de tables
- Déclaration des vues- Déclaration des droits d'accès
- Déclaration de contraintes d'intégrités

Les valeurs d'une colonne sont définies selon un *type de données* choisi lors de la création de la table.

Quelques *types de données* manipulés sont

Syntaxes :

char (*longueur*), varchar (*longueur*), numeric(précision, échelle), integer,date ,…

Tableau de type de données :

Types de données	syntaxe	description
alphanumérique	Char (n)	Chaîne de caractères de longueur fixe (n<16383)
alphanumérique	varchar (n)	Chaîne de caractères de n caractères maximum (n<16383)
Numérique	Number(n[d])	Nombres de n chiffres [optionnellement d après la virgule]
Numérique	Smallint	Entier signé de 16 bits (-32768 à 32757)
Numérique	Integer	Entier signé de 32 bits (--2E31 à 2E31-1)
Numérique	Float	Nombre à virgule flottante
Horaire	Date	Date sous forme jj/mm/aa
Horaire	Time	Heure sous la forme H :m :s.ms (12 :54 :41.35
Horaire	Timestamp	Date et Heure

5.2.2.1. Création de table et de vue

Syntaxes :

Create table <nom de table>
(<attribut$_1$> <type attribut$_1$><contrainte$_1$>)
(<attribut$_2$> <type attribut$_2$><contrainte$_2$>)
…
(<attribut$_n$> <type attribut$_n$><contrainten$_n$>) **;**

create View <nom de view>
AS <requête de sélection>
<option de test> **;**

Description

La *contrainte* introduit la notion d'intégrité de données (spécification des contraintes)

<*option test*> qui s'exprime par « with check option> » est facultative. Quand elle est spécifiée, elle impose les restrictions suivantes :

- Toute modification d'un tuple d'une vue est refusée si elle entraîne une exclusion de la vue du tuple concerné.
- Une insertion est refusée si le tuple introduit dans la vue ne satisfait pas au critère de sélection de celle-ci.

Exemple :

Create table *Etudiant* (matricule char(6) primary key, nom char(20) not null, prenoms char(40)) ;

Nous venons de créer la table *Etudiant*.

Nous allons créer une vue nommée *Roman* contenant le *titre* du livre, *l'auteur* et de genre *gestion.*

Exemple :

Create view *Roman*
AS select titre, auteur, genre
From livre
Where genre ='gestion' ;

5.2.2.2. Destruction de table ou de vue

Nous allons supprimer la table *Etudiant* de la base de données. Aussi supprimer la vue *Roman*.

Syntaxe :

Drop table <nom de table> ;

e*xemple :*

Drop table *Etudiant* ;

Syntaxe :

Drop view <nom de vue> **;**

Exemple :

Drop view *Roman* ;

5.2.2.3. Modification du schéma de table

Si la table est créée dans une base de données, nous avons la possibilité de :

- ✓ de modifier ses colonnes ;
- ✓ d'ajouter des colonnes ;
- ✓ de supprimer certaines colonnes.

Description

La clause **alter** permet de modifier des colonnes d'une table.

Associé à la clause **add** , il permet l'ajout de colonnes à une table

Associé à la clause **modify,** il permet la modification de type de données d'une colonne.

Associé à la clause **drop**, il permet supprimer des colonnes d'une table.

Syntaxe:

Alter table <nom de table> **Add** (<col_1><$type_1$><$contrainte_1$>,…) **;**

Exemple :

Alter table *Etudiant* **Add** (sexe char(1)) ;

Syntaxe:

Alter table <nom de table> **modify** (<col_1><$type_1$><$contrainte_1$>,…) **;**

Exemple :

Alter table *Etudiant* **modify** (nom *char*(*40*)) ***;***

Syntaxe:

Alter table <nom de table> **drop** (<col_1><$type_1$><$contrainte_1$>,…) **;**

Exemple :

Alter table *Etudiant* **drop** (matricule char(6) primary key)

5.2.2.4. Spécification des contraintes

Les contraintes qui régissent l'intégrité des données se spécifient à travers la définition des colonnes.

a) Les contraintes

On distingue 4 types d'intégrité :

- De domaine : contrainte DEFAULT
- D'entité : contraintes NOT NULL, UNIQUE, PRIMARY KEY
- De reference : contraintes FOREIGN KEY
- D'application : contraintes : CHECK

- ✓ Contrainte PRIMARY KEY

Cette lause permet d'obtenir l'unicité des lignes de la table et se comporte comme les contraintes not null et unique. Dans le cas de plusieurs colonnes il faut donner un nom à la contrainte à l'aide du mot clé ***constraint****.*

Exemple :

CREATE table Emprunt (matricule char (6) not null, code_liv char(6) not null, sortie Date not null, retour Date, ***constraint*** *emp* PRIMARY KEY **(**matricule,code_liv,sortie**)**);

- ✓ Contrainte DEFAULT

Le langage SQL permet de définir une valeur par défaut lorsqu' un champ de la base n'est pas renseigné grâce la clause ***default ,*** *cela permet notamment de faciliter la création de tables, ainsi que de garantir qu'un champ ne sera pas vide .*

Exemple :

CREATE table *Livre* (code_liv char(6) primary key,…, prix number(9.0) **Default** 1000) ;

- ✓ Contrainte NOT NULL

Oblige de donner une valeur à la colonne.

Exemple :

CREATE table Departement(numdept number(2) **NOT NULL**, …) ;

- ✓ Contrainte UNIQUE

La clause unique permet de vérifier que la valeur saisie pour un champ n'existe pas déjà dans la table.

Exemple :

CREATE table produit (libellé varchar (10) **UNIQUE**,..) ;

- ✓ Contrainte CHECK

Il est possibles de faire un test sur un champ grâce à la clause check() comportant une condition logique portant sur une valeur entre les parenthèses.

Exemples :

CREATE table Etudiant (…, sexe char(1) **CHECK**(sexe in **(**'M','F'**)**)) ;

CREATE table clients(nom char(30) not null, prenoms char (30)not null, age integer **check**(age<100), Emailnot null, **check**(Email like "%@%");

- ✓ Contrainte FOREIGN KEY

Cette clause permet d'établir une relation avec la clé primaire (unique). Elle peut concerner une ou plusieurs tables. On parle alors de ***clé étrangère****. On utilise la clause FOREIGN KEY suivie de la liste de colonnes de la table en cours de définition, séparées par des virgules, entre parenthèses, puis de la clause REFERENCES suivie du nom de la table étrangère et de la liste de ses colonnes correspondantes, séparées par des virgules, entre parenthèses.*

Exemple :

CREATE table Emprunt (…, constraint *N_liv* **Foreign key** (code_Liv) **References** Livre(code_Liv), constraint *mat* **FOREIGN KEY** (matricule) **References** Etudiant(matricule),…);

b) Gestion des contraintes

Comme les colonnes, les contraintes peuvent être ajoutées, modifiées (à travers une colonne) et supprimées .de plus les contraintes peuvent être activées ou désactivées.

Syntaxes :

Alter table <nom de table> **Add (constraint**<nom contrainte><type contrainte> (<liste de colonnes>,…) **;**

Alter table <nom de table> **modify (** <nom colonne**> constraint**<nom contrainte><type contrainte> …) **;**

Alter table <nom de table> **Drop constraint** <nom contrainte>/<type contrainte> [CASCADE] ;

Alter table <nom de table> **Disable constraint** <nom contrainte>/<type contrainte> [CASCADE] ;

Alter table <nom de table> **Enable constraint** <nom contrainte>/<type contrainte> [EXCEPTION INTO <table rejets>] ;

Exemple :

Alter table commande **Enable constraint** paye EXCEPTION INTO rejets

Conclusion

A l'aide des contraintes :

- L'intégrité et la cohérence de données sont garanties
  - Elles sont incluses dans la définition des tables
  - Elles sont appliquées avec des outils de développement (outils, CASE*, SQL*Forms, …)
  - Elles sont incontournables
  - Définitions stockées dans le dictionnaire de données.
- La productivité de développement de l'application est améliorée
  - Pas de programmation spéciale
  - Spécification et maintenance facile
  - Codage réduit

- Une seule définition pour l'utilisation par plusieurs applications
- Les cas de transgression sont signalés à l'utilisateur (le nom de la contrainte apparaît dans le message d'erreur)

➢ Les performances sont améliorées

- Réduction du trafic réseau
- Optimisation du contrôle d'intégrité (pas de contrôle exécuté si aucune colonne n'est modifiée

➢ La gestion est simplifiée

- Compte rendu de chaque transgression dans une table des rejets
- Les contraintes peuvent être inhibées afin de réduire certains de traitements

5.2.2.5. Les privilèges et rôles

Plusieurs personnes peuvent travailler simultanément sur une même base de données, toute fois ces personnes n'ont pas forcement les mêmes besoins. Certains peuvent nécessiter de modifier des données dans une table tandis que les autres ne l'utiliseront que pour la consulter. L'administrateur doit définir des permissions pour chacun d'eux.

Définitions

Un ***privilège*** est un droit attribué aux utilisateurs pour exécuter certaines commandes.

Il existe 2 types de privilèges

- Privilège objet
- Privilège système

Un **rôle** est un ensemble de privilèges donnés à des utilisateurs ou à d'autres rôles permettant de gérer facilement ces droits d'accès aux données.

Un **profil** permet de contrôler la consommation des ressources systèmes (consommation CPU, nombre de sessions simultanées.

a) Le privilège objet

Il donne le droit d'accès à une table, une vue, une séquence, une procédure, une fonction ou un package.

Il peut être redistribué à d'autres utilisateurs avec l'option (WITH GRANT OPTION)

Syntaxes :

La liste des privilèges peut se résumer à **ALL** pour spécifier tous les privilèges et la liste utilisateur à

Grant<privilèges>	**Revoke**<privilèges>
On <nom objet>	**On** <nom objet>
To <liste utilisateurs>	**To** <liste utilisateurs>
[With Grant Option]	

Liste des objets

Objet / Privilèges	Table	Vue	Séquence	Procedure Fonction
ALTER	Oui		Oui	
EXECUTE				Oui
DELETE	Oui	Oui		
INDEX	Oui			
INSERT	Oui	Oui		Oui
REFERENCES	Oui			
SELECT	Oui	Oui	Oui	
UPDATE	Oui	Oui		

Exemples :

GRANT UPDATE (nom)
ON Etudiant
TO ing01;

GRANT UPDATE (nom)
ON Etudiant
TO Koffi, Simon, Cedric
WITH GRANT OPTION;

[GRANT OPTION FOR] UPDATE (nom, prenoms)
ON Etudiant
FROM PUBLIC;

REVOKE ALL ON *Etudiant*
FROM Koffi ;

Remarque : Quand un utilisateur à qui l'on a attribué des privilèges est révoqué, les utilisateurs qui ont bénéficié de ses privilèges sont révoqués également.

b) Le privilège système

Les privilèges systèmes donnent le droit d'exécuter des actions sur un certain type d'objets .

Syntaxes :

Sur un objet appartenant à un utilisateur importe

GRANT CREATE VIEW
TO ing01 ;

Sur tous objets de même type appartenant à n' quel utilisateur

GRANT SELECT ANY TABLE
TO ing04 ;

Sur des opérations d'administrations de base

GRANT AUDIT SYSTEM
CREATE USER
TO ing05

Remarque : Quand un utilisateur à qui l'on a attribué des privilèges systèmes est révoqué, les utilisateurs qui ont bénéficié de ses privilèges ne sont pas révoqués.

c) Les rôles

Les rôles peuvent être composés à la fois de privilèges systèmes et de privilèges objets. Ils ne sont pas la propriété d'un utilisateur. Ils peuvent être activés ou inhibés pour chaque utilisateur et peuvent exiger des mots de passe.

Syntaxes :

Création de rôle

CREATE ROLE chef_de_classe [IDENTIFY BY mot_de_passe] ;

Attribution de privilèges systèmes

GRANT CREATE SESSION, CREATE TABLE, CREATE VIEW
TO chef_de_classe ;

Assignation d'un rôle à un utilisateur

GRANT chef_de_classe
TO ing06 ;

Activation/ désactivation des rôles

SET ROLE chef_de_classe ; *pour activer les privilèges*

SET ROLE ALL ; *pour activer tout*

SET ROLE NONE; *pour inhiber*

SET ROLE ALL EXCEPT liste_de_role :

Conclusion

Le rôle permet de réduire la gestion individuelle des privilèges.

Sa gestion est dynamique :

- Possibilité de modifier les privilèges d' un rôle à chaque fois les possibilités changent au sein de l' entreprise

- Possibilité de changer les privilèges de plusieurs utilisateurs par un changement d'un seul rôle

5.2.2.6. Les performances

Syntaxe :

Create <clause d'unicité> **Index** <nom index>

On <nom de table ou vue >(<col_1><ordre>, <col_2><ordre>, ..);

Description

Les index sont généralement placés sur les colonnes fréquemment utilisées par les requêtes de sélection. Leur création fournit au système un moyen d'accès rapide aux informations de la base . <ordre> indique un tri décroissent (« **desc »**) ou par defaut croissant (« **asc »**)

Exemple :

Create index ind_etu

On Etudiant(nom,prenom) ;

5.3. la spécification des modules appelables (procédures)

SQL est un langage déclaratif (non procédural) qui permet d'exprimer facilement des requêtes cependant il ne contient pas de structure de contrôle telles que les boucles itératives et les contrôles séquentiels.

5.4. L'intégration aux langages de programmation « embedded SQL

Un programme SQL intégré est en fait un programme ordinaire (C, C++, java) dans lequel nous insérons des commandes SQL incluses dans des sections spécialement marquées. Ainsi les instructions Embedded SQL commencent par les mots *EXEC SQL* et se terminent par un point-virgule (« ; »).

Ces sections se présentent toujours de la manière suivante :

```
EXEC SQL instructions_SQL ;
```

Exemple en C :

```
   EXEC SQL BEGIN DECLARE SECTION ;
     char requete[MAX_QUERY_LENGTH];                /* declaration de variable hote */
   EXEC SQL END DECLARE SECTION ;
While (1){                                          /* programme C */
          printf("\n Nouvelle requête: ");
          scanf(''%s '',requete) ;
```

```
   EXEC SQL PREPARE q FROM : requete ;
   EXEC SQL EXECUTE q;
   printf("\n Nouvelle requête: ");
    scanf(''%s '',requete) ;
  EXEC SQL EXECUTE IMMEDIATE: requete;
}
```