Ministère de l'Enseignement Supérieur,
de la Recherche Scientifique et de la Technologie
Direction Générale des Etudes Technologiques
Institut Supérieur des Etudes Technologiques de Djerba

Support de Cours

BASES DE DONNEES

Élaboré par :

Mejdi BLAGHGI & Anis ASSÈS

Public cible :

Informatique (Niveau 2)

Options : Informatique Réseaux & Informatique Industrielle

Version 2007

Avant propos

Pré requis

- Notion de fichier
- Notion de variables

Objectifs généraux

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

- Découvrir les concepts de base des bases de données.
- Analyser une étude de cas donné afin de dégager le modèle entités/associations et le modèle relationnel associé.
- Apprendre à utiliser un langage normalisé d'accès aux données (SQL).

Niveau cible

Etudiants du deuxième niveau des Etudes Supérieures Technologiques.

Options : Informatique Réseaux et Informatique Industrielle.

Volume horaire

- 1h 30 de cours intégré.

Soit en total : 22,5h

- 1h 30 de Travaux Pratiques pour chaque groupe

Soit en total : 22,5h

Moyens pédagogiques

Tableau

Micro-ordinateurs

Vidéo projecteur

Polycopiés de Travaux Dirigés

Evaluation

- Coefficient : 2
- Devoirs surveillé : 30%
- Examen TP, Moyenne des TP (Comptes rendus) : 30%
- Examen semestriel : 40%

Tables de matières

Liste des figures

Chapitre : 1

Introduction aux bases de données

Objectifs spécifiques

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

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

Plan du chapitre

I. Introduction

II. Les systèmes à fichiers

III. Les bases de données

IV. Les SGBD

V. Les niveaux d'abstraction

Volume horaire

4 heures et demi

I. Introduction

Toute entreprise (établissements d'enseignement, ministères, banques, agences de voyages, transport, santé, …) dispose d'un ensemble de données qu'elle propose de stocker, organiser, manipuler et exploiter.

- Stockage et organisation : saisir, insérer les informations et les enregistrer dans les emplacements appropriés dans le système.
- Manipulation : chercher, sélectionner, mettre à jour et supprimer les données archivées dans le système.
- Exploitation : récupérer les données et les informations nécessaires afin de prendre une décision.

Au fil des années, les quantités de données deviennent de plus en plus grandes. Certains experts (dans le domaine des statistiques) estiment même que le volume de données collectées par une entreprise double tous les vingt mois.

à Explosion vertigineuse du volume de données au sein des entreprises.
à Recourir à un moyen efficace de stockage, de manipulation et d'exploitation de données.

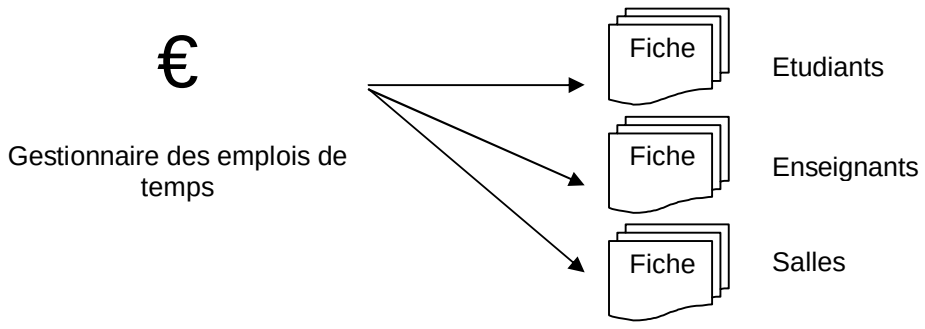
II. Les systèmes à fichiers

II.1. Définition

Fichier : est un récipient de données identifié par un nom constitué par un ensemble de fiches contenant des informations système ou utilisateur.
Les informations relatives à un même sujet sont décrites sur une fiche.

Gestionnaire de fichiers : structuré autour d'un noyau appelé analyseur, qui assure les fonctions de base, à savoir la création/destruction des fichiers, lecture/écriture d'une suite d'octets à un certain emplacement dans un fichier, l'allocation de la mémoire, la localisation et la recherche des fichiers sur les volumes mémoires.

Exemples :



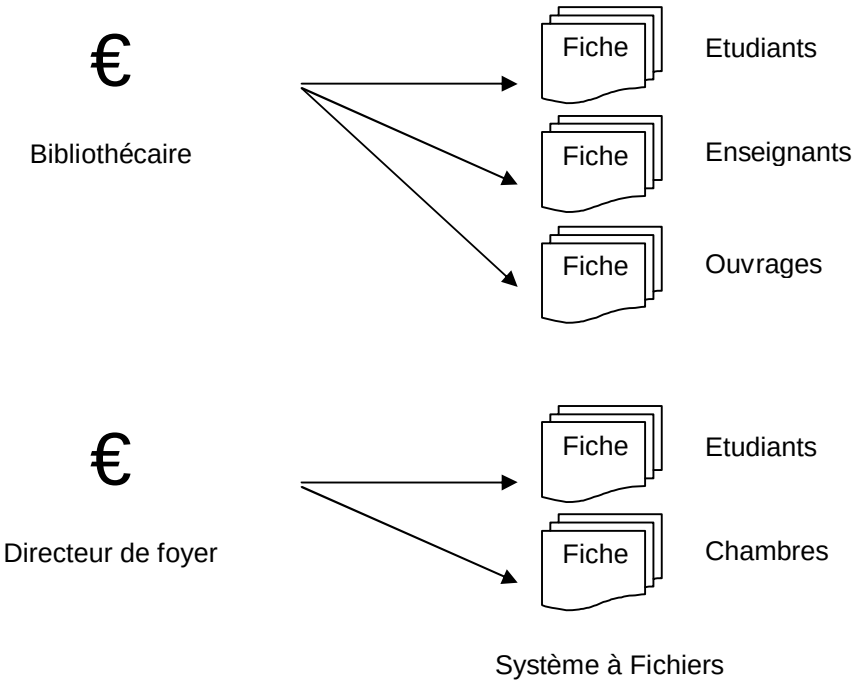


Figure 1. Organisation de données en systèmes à fichiers

II.2. Limites

Les systèmes à fichiers soulèvent certaines limites à savoir :

- Dispersion des informations.
- Contrôle différé de données.
- Risque d'incohérence de données.
- Redondance de données.
- Difficulté d'accès, d'exploitation, d'organisation et de maintenance.

III. Les bases de données

III.1. Définition

Une base de données est un ensemble de données d'entreprise enregistrées sur des supports accessibles par l'ordinateur pour satisfaire simultanément plusieurs utilisateurs de façon sélective et en temps opportun.[1]

Donc, une base de données nécessite :

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

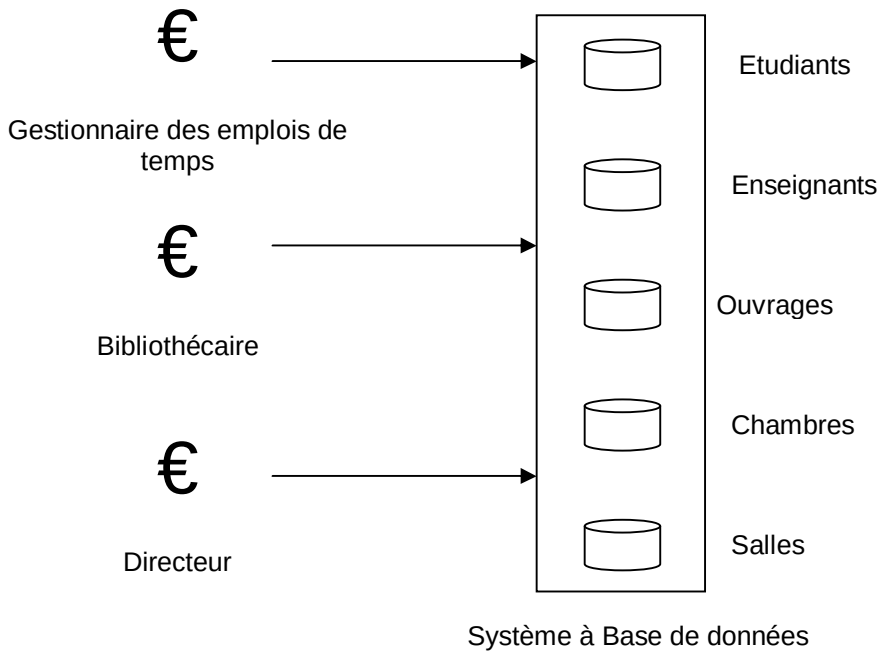


Figure 2. Organisation de données en systèmes à base de données

III.2. Types d'utilisateurs de bases de données

On distingue essentiellement trois types différents d'utilisateurs bases de données à savoir : [2]

- Administrateurs de bases de données : Ils gèrent la base (les accès, les droits des utilisateurs, les sauvegardes, les restaurations, …)
- Concepteurs de bases de données et développeurs d'application : Ils représentent ceux qui définissent, décrivent et créent la base de données.
- Utilisateurs finaux : manipulent la base de données. Il est possible de distinguer des familles d'utilisateurs avec des droits différents vis-à-vis de l'accès à la base. On suppose qu'ils n'ont aucune connaissance sur les bases de données.

IV. Les SGBD

IV.1. 1. Définition

Un SGBD (Système de Gestion de Bases de Données) est un logiciel qui permet à des utilisateurs de définir, créer, mettre à jour une base de données et d'en contrôler l'accès [3]. C'est un outil qui agit comme interface entre la base de données et l'utilisateur. Il fournit les moyens pour définir, contrôler, mémoriser, manipuler et traiter les données tout en assurant la sécurité, l'intégrité et la confidentialité indispensables dans un environnement multi-utilisateurs.

IV.2. 2. Objectifs

On distingue essentiellement trois fonctions des SGBD à savoir :

- Description des données : Langage de Définition de Données (LDD).

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

En d'autres termes, les objectifs d'un SGBD peuvent être résumés dans les points suivants :

- Intégrité de données.
- Indépendance données/programmes : Ajout d'un nouveau champ ne provoque aucun problème vis à vis des autres champs.
- Manipulation facile de données : un utilisateur non informaticien peut manipuler simplement les données.
- Sécurité et administration de données.
- Efficacité d'accès aux données.
- Redondance contrôlée de données.
- Partage de données.

IV.3. 3. Historique

Jusqu'aux années 60 : Organisation classique en fichiers : systèmes à fichiers.

Fin des années 60 : Première génération : apparition des premiers SGBD.

- Séparation de la description des données de leur manipulation par les programmes d'application.
- Basés sur des modèles navigationnels :
    - Modèles hiérarchiques : modèles père fils, structures d'arbres, … (Exemples : Adabas, DL/ 1 ou IMS (Information Management System))
    - Modèles réseaux : modèle père fils, structure de graphes. (Exemples : SOCRATE, IIDS, IDMS, IDS2, IMS2).

À partir de 1970 : Deuxième génération de SGBD à partir du modèle relationnel.

- Enrichissement et simplification des SGBD afin de faciliter l'accès aux données pour les utilisateurs.
- Modèle mathématique de base : l'algèbre relaionnelle
- Exemples : ORACLE, SQL Server, DB2, Access, PostgreSQL, MySQL.

Début des années 80 : Troisième génération de SGBD basées sur des modèles plus complexes [4] :

- SGBD déductifs : modèle logique, algèbre booléenne, souvent avec un autre SGBD (généralement relationnel) qui gère le stockage, présence d'un moteur d'inférence.
- SGBD à objets (SGBDOO) : modèles inspirés des langages de programmation orientée objet tels que Java, C++ … utilisation de l'encapsulation, l'héritage, le polymorphisme, …
- Exemples : ONTOS, ObjectStore, VERSANT, ORION, O2.

V. Les niveaux d'abstraction

La conception d'une base de données passe essentiellement, comme le montre la figure 3, par trois niveaux d'abstraction à savoir :

V.1. Niveau externe

Ce niveau présente une vue de l'organisation (ou d'une partie) par des utilisateurs ou des applications. En effet, il prend en charge le problème du dialogue avec les utilisateurs, c'est-à-dire l'analyse des demandes de l'utilisateur, le contrôle des droits d'accès de l'utilisateur, la présentation des résultats.

V.2. Niveau conceptuel

Il s'agit d'une représentation abstraite globale de l'organisation et de ses mécanismes de gestion. Ce niveau assure les fonctions de contrôle global (optimisation globale des requêtes, gestion des conflits d'accès simultanés, contrôle général de la cohérence de l'ensemble…).

V.3. Niveau interne

Ce niveau s'occupe du stockage des données dans les supports physiques et de la gestion des structures de mémorisation et d'accès (gestion des index, des clés, ...)

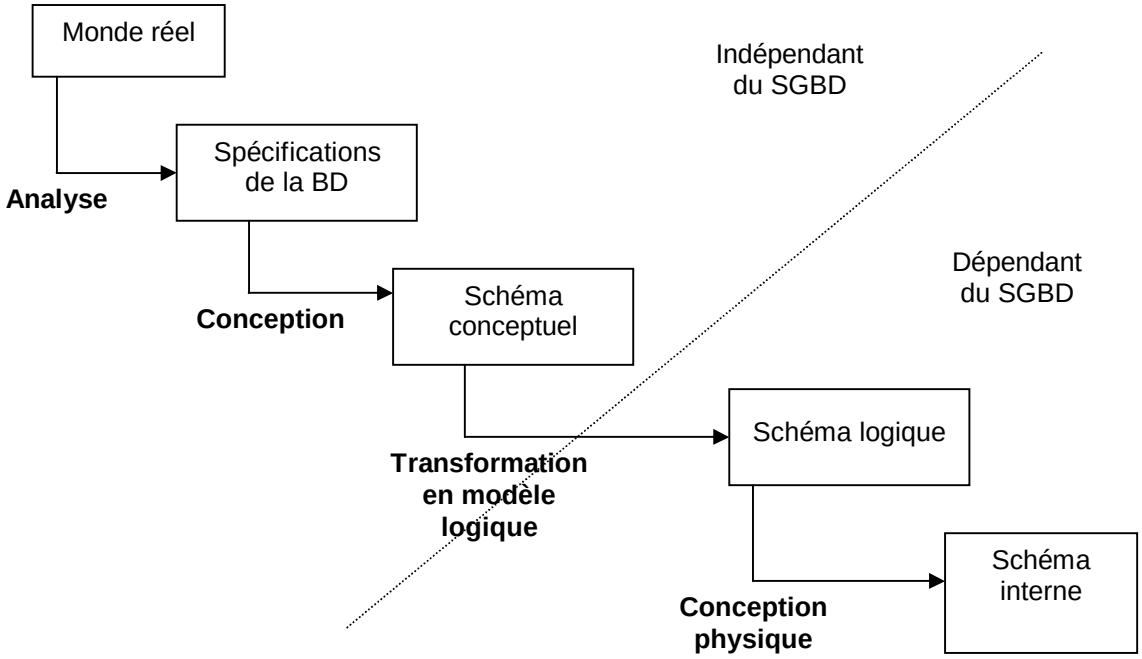


Figure 3. Niveaux d'abstraction

Chapitre : 2

Le modèle Entités/Associations

Objectifs spécifiques

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

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

Plan du chapitre

I. Généralités

II. Concepts de base

III. Associations et cardinalités

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

Volume horaire

4 heures et demi

I. Généralités

- Le modèle Entités/Associations est généralement connu par le modèle E/A.
- C'est un modèle conceptuel conçu dans les années 1970 qui résulte des travaux de BACHMAN, CHEN, TARDIEU.
- Il est essentiellement utilisé pour la phase de conception initiale.
- Il utilise une représentation graphique.
- Mise en œuvre de la base de données : transformation du schéma E/A en un schéma logique de SGBD.

II. Concepts de base

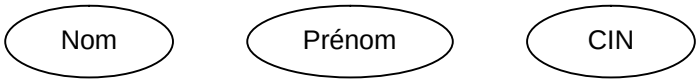
II.1. Attribut

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

Exemples : Nom, prénom, date_naissance, immatricule_voiture, raison sociale, …

Notation : On présente les attributs par des ellipses contenant leurs noms.

Exemples :



Propriétés :

- *Attribut simple* : attribut non décomposable en d'autres attributs.
  Exemples : Nom, Prénom, NCI, Email, Téléphone.

- *Attribut composé* : la valeur de l'attribut est une concaténation des valeurs de plusieurs attributs simples.
  Exemple : Adresse (Rue, Code postal, Ville).

- *Attribut dérivé* : la valeur de l'attribut est calculée ou déduite à partir des valeurs des autres attributs.
  Exemples : Age, Moyenne, Durée

- *Valeur nulle (NULL)* : pour un attribut, c'est une valeur non définie.
  Exemple : Pour un client dont on ne connaît pas sa date de naissance, l'attribut date_naissance prend la valeur NULL.

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

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

Exemple : Si le prix des produits est compris entre 1DT et 5DT, alors le domaine de l'attribut prix est [1..5].

II.2. Entité

Définitions :

- Une entité est un objet de l'univers du discours = (sujet, thème)
- Un type d'entité permet de définir de façon conceptuelle une entité dont tous les membres partagent les mêmes caractéristiques. [5]
- Une occurrence d'entité est constituée par l'ensemble des valeurs de chacune des propriétés d'un type d'entité.

  Exemple :

  Type d'entité : Personne

  Occurrences :

Mohamed	Ben Salah	05652124
Ali	Ben Abdallah	07412148
Salah	Ben Ali	06457986

Identifiant d'une entité : caractérise de façon unique les occurrences d'un type d'entité.

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

Notation : Chaque entité est représentée par un rectangle et doit avoir un identifiant qui doit être souligné.

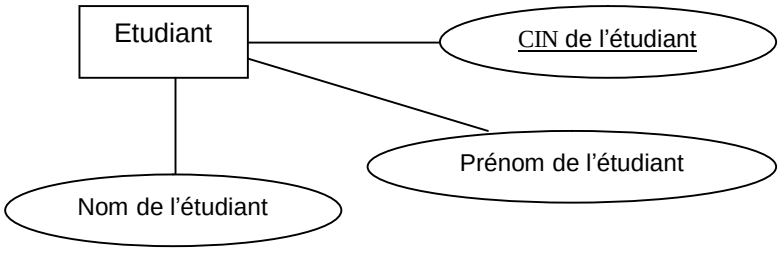


Figure 4. Exemple d'entité avec ses attributs

III. Associations et Cardinalités

III.1. Associations

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

Exemple : Les clients commandent des produits.

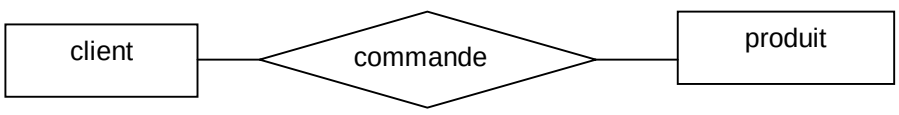


Figure 5. Exemple d'association entre deux entités

Les entités client, produit sont dites participantes à la relation commande.

III.2. Cardinalités

Définition : Les associations sont caractérisées par des cardinalités. La cardinalité M-N attachée à une entité indique les nombres minimal et maximal d'instance d'associations pour une instance de cette entité.

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

III.3. Types de cardinalités

Notations :

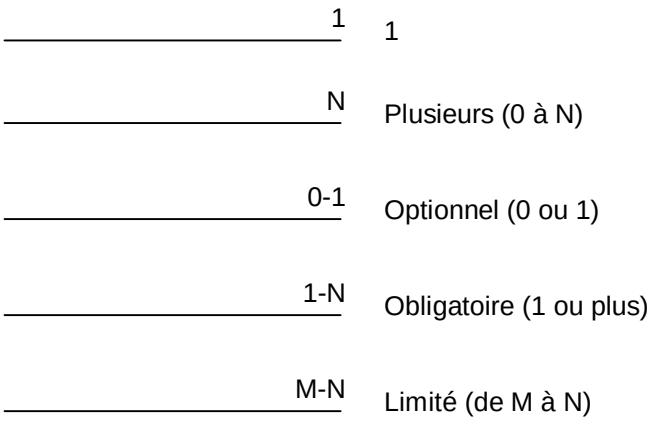


Figure 6. Notation des cardinalités d'associations

Exemples :

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

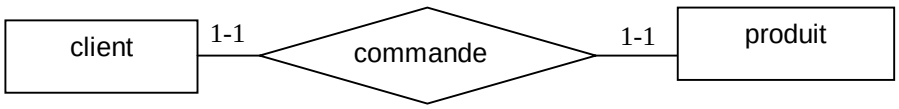


Figure 7. Exemple d'association de type 1-1

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

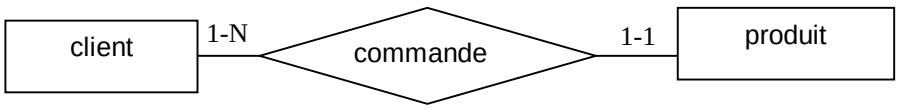


Figure 8. Exemple d'association de type 1-N

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

- *Association M-N :* Un client donné commande plusieurs produits. Un produit donné est commandé par un ou plusieurs clients.

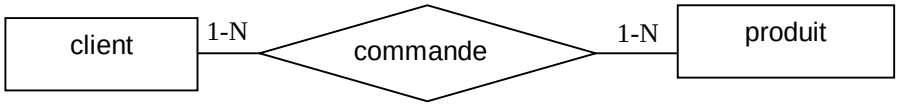


Figure 9. Exemple d'association de type M-N

III.4. Attributs d'association

Dans une association M-N, il est possible de caractériser l'association par des attributs.

Exemple : Une commande est passée à une date donnée et concerne une quantité de produit fixe.

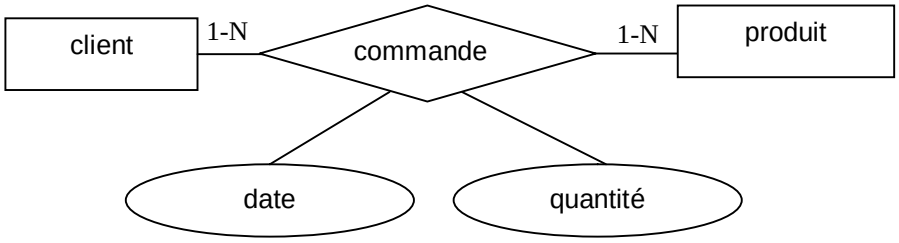


Figure 10. Exemple d'association de type M-N

Remarques :

- On peut avoir une association réflexive au niveau de la même entité.
- Une association peut l'être entre plus que deux entités.

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

IV.1. Démarche

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

1. Recueil des besoins et identification des différents attributs.
2. Regrouper les attributs par entités.
3. Identifier les associations entre les entités ainsi que les attributs y associés.
4. Evaluer les cardinalités des associations.

IV.2. Exemple illustratif : Gestion simplifié de stock

Spécifications :

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

Les produits sont caractérisés par un numéro de produit, une désignation et un prix unitaire.

Chaque produit est fourni par un fournisseur unique (mais un fournisseur peut fournir plusieurs produits).

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

Solution :

Les différents attributs associés à ces spécifications peuvent être résumés comme suit :

Nom de l'attribut	Désignation de l'attribut
numCl	numéro du client.
nomCl	nom du client.
prenomCl	prénom du client.
datenaisCl	date de naissance du client.
adrCl	adresse du client.
qtePc	quantité de produits commandés.
dateCd	date de la commande
numPd	numéro du produit.
designPd	désignation du produit.
puPd	prix unitaire du produit.
numFr	numéro du fournisseur.
rsFr	raison sociale du fournisseur.
emailFr	adresse email du fournisseur.
adrFr	adresse du fournisseur.

Les entités avec leurs attributs sont :

Nom de l'entité	Attributs de l'entité
Client	numCl, nomCl, prenomCl, datenaisCl, adrCl
Produit	numPd, designPd, puPd
Fournisseur	numFr, rasFr, emailFr, adrFr

Les associations entre les entités :

Nom de l'association	Entités participantes	Attributs associés
commande	client et produit	qtePc, dateCd
fourniture	fournisseur et produit	-

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

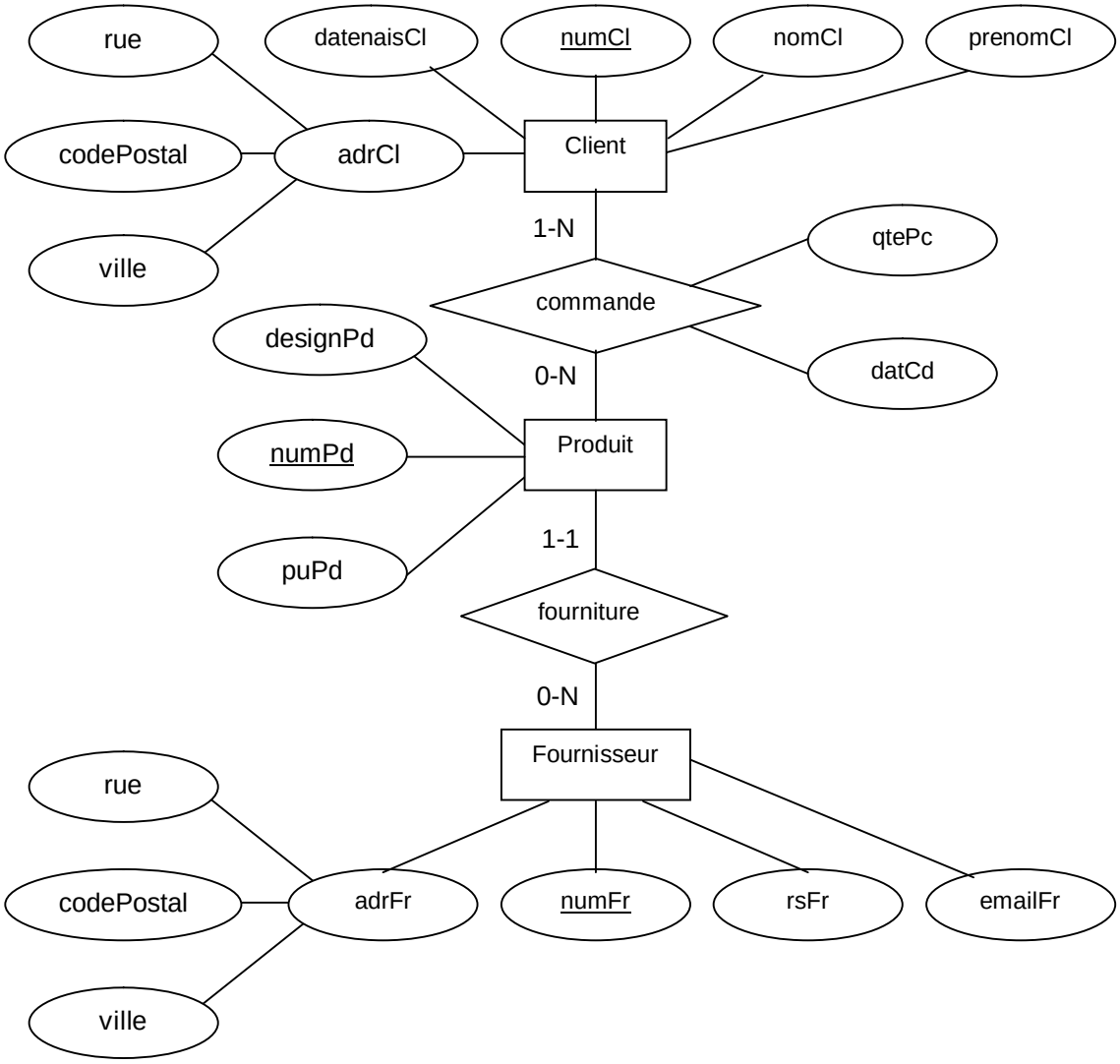


Figure 11. Modèle E/A de l'exemple illustratif

Chapitre : 3

Le modèle relationnel

Objectifs spécifiques

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

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

Plan du chapitre

Volume horaire

3 heures

I. Généralités

Définition : Le modèle relationnel est un modèle logique associé aux SGBD relationnels.

Exemples : Oracle, DB2, SQLServer, Access, Dbase, ….

Objectifs du modèle relationnel :

- *Indépendance physique* : indépendance entre programmes d'application et représentation interne de données.
- *Traitement des problèmes de cohérence et de redondance de données* : problème non traité au niveau des modèles hiérarchiques et réseaux.
- *Développement des LMD non procéduraux* : modélisation et manipulation simples de données, langages faciles à utiliser.
- *Devenir un standard*.

II. Concepts de base

II.1. Relation

Définition : Une relation R est un ensemble d'attributs {A1, A2,…, An}.

Notation : R(A1, A2, …, An)

Exemple : Soit la relation Produit(numPd, designPd, puPd)

La relation Produit est l'ensemble des attributs {numPd, designPd, puPd}

Remarque : Chaque attribut Ai prend ses valeurs dans un domaine dom(Ai).

Exemple : Le prix unitaire puPd est compris entre 0 et 10000. D'où dom(puPd) = [0, 10000].

II.2. Tuple

Définition : Un tuple est un ensemble de valeurs t=<V1 V2 … Vn> où Vi appartient à dom(Ai).
Il à noter que Vi peut aussi prendre la valeur nulle.

Exemple : <2, 'Prod1', 300>

II.3. Contraintes d'intégrité

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

Types :

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

  Notation : la clé primaire doit être soulignée.

Exemple :

Soit la relation Produit(numPd, designPd, puPd)

L'attribut numPd présente la clé primaire de la relation Produit.

- *Clé étrangère :* attribut qui est clé primaire d'une autre relation.

  Notation : La clé étrangère doit être précédée par #.

  Exemple :

  Produit(numPd, designPd, puPd, # numFr)

  Le tuple Produit(1, 'Produit1', 100, 1) signifie que le produit de numéro 1 a comme fournisseur numéro 1.

- *Contraintes de domaine :* les attributs doivent respecter une condition logique.

III. Traduction E/A - relationnel

III.1. Règles

Pour traduire un modèle Entités / Associations en modèle relationnel, on applique les règles suivantes :

- Chaque entité devient une relation. Les attributs de l'entité deviennent attributs de la relation.
- L'identifiant de l'entité devient clé primaire de la relation.
- Chaque association 1-1 est prise en compte en incluant la clé primaire d'une des relations comme clé étrangère dans l'autre relation.
- Chaque association 1-N est prise en compte en incluant la clé primaire de la relation dont la cardinalité maximale est N comme clé étrangère dans l'autre relation.
- Chaque association M-N est prise en compte en créant une nouvelle relation dont la clé primaire et la concaténation des clés primaires des relations participantes. Les attributs de l'association sont insérés dans cette nouvelle relation.

III.2. Application

L'exemple du modèle Entités/Associations élaboré dans le chapitre précédent (Chapitre 2, IV.2. Exemple illustratif) se traduit en modèle relationnel comme suit :

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

PRODUIT (numPd, designPd, puPd, #numFr)

FOURNISSEUR (numFr, rsFr, emailFr, adrFr)

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

IV. Les dépendances fonctionnelles (DF)

IV.1. Définition

Soit R (X, Y, Z) une relation où X, Y, et Z sont des ensembles d'attributs. Z peut être vide.

On dit que Y dépend fonctionnellement de X ou X détermine Y noté (X →Y) si étant donné une valeur de X, il lui correspond une valeur unique de Y.

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

NumProd →Dési,

Dési →PrixUni

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

IV.2. Propriétés des dépendances fonctionnelles

Les dépendances fonctionnelles obéissent à certaines propriétés connues sous le nom d'axiomes d'Armstrong.[6]

- Réflexivité : $Y \subset X \Rightarrow X \rightarrow Y$
- Augmentation : $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$
- Transitivité : $X \rightarrow Y$ et $Y \rightarrow Z \Rightarrow X \rightarrow Z$

D'autres propriétés se déduisent de ces axiomes :

- Union : $X \rightarrow Y$ et $X \rightarrow Z \Rightarrow X \rightarrow YZ$
- Pseudo-transitivité : $X \rightarrow Y$ et $YW \rightarrow Z \Rightarrow XW \rightarrow Z$
- Décomposition : $X \rightarrow Y$ et $Z \subset Y \Rightarrow X \rightarrow Z$

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

V. Normalisation

V.1. Objectifs de la normalisation

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

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

Cette relation présente différentes anomalies.

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

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

Objectifs de la normalisation :

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

V.2. Première forme normale (1FN)

Définition : Une relation est en 1FN si tout attribut est atomique (n'est pas décomposable).

Exemple : Les relations PERSONNE (Nom, Prénoms, Age) et DEPARTEMENT (Nom, Adresse, Tel) ne sont pas en 1FN si les attributs Prénoms et Adresse peuvent être du type [Med, Ali] ou [73, Rue 20 Mars, Tunis] respectivement.

V.3. Deuxième forme normale (2FN)

Définition : Une relation est en 2FN si :

- elle est en 1FN ;
- tout attribut non clé primaire est dépendant de la clé primaire entière.

Exemple :

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

La relation COMMANDE_PRODUIT(NumProd, Quantite, NumFour, Ville) n'est pas en 2FN car on a NumProd, NumFour → Quantité et NumFour → Ville.

La décomposition suivante donne deux relations en 2FN :

COMMANDE (NumProd, NumFour, Quantité) ;

FOURNISSEUR (NumFour, Ville).

V.4. Troisième forme normale (3FN)

Définition : Une relation est en 3FN si :

- elle est en 2FN ;
- il n'existe aucune DF entre deux attributs non clé primaire.

Exemple :

La relation COMPAGNIE (Vol, Avion, Pilote) avec les DF :

Vol → Avion, Avion → Pilote et Vol → Pilote

est en 2FN, mais pas en 3FN.

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

La décomposition suivante donne deux relations en 3FN qui permettent de retrouver (par transitivité) toutes les DF : R1 (Vol, Avion) ; R2 (Avion, Pilote).

Chapitre : 4

L'algèbre relationnelle

Objectifs spécifiques

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

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

Plan du chapitre

I. Définition

II. Opérateurs ensemblistes

III. Opérateurs spécifiques

IV. Exercice d'application

Volume horaire

1 heures et demi

I. Définition

L'algèbre relationnelle est définie comme étant l'ensemble d'opérateurs qui s'appliquent aux relations.

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

L'algèbre relationnelle permet d'effectuer des recherches dans les relations.

II. Opérateurs ensemblistes

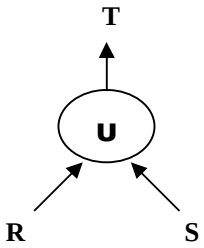
II.1. Union

Définition :

T= R U S ou T=UNION(R, S) contient tous les tuples appartenant aux deux relations R et S.

R et S doivent avoir même schéma.

Notation :



Exemple : R et S sont les relations PRODUIT de deux sociétés qui fusionnent et veulent unifier leur catalogue.

R		
NumProd	**DesigProd**	**PrixProd**
1	Produit1	100
2	Produit2	200

S		
NumProd	**DesigProd**	**PrixProd**
3	Produit3	500
2	Produit2	200

Alors :

T = R U S		
NumProd	**DesigProd**	**PrixProd**
1	Produit1	100
2	Produit2	200
3	Produit3	500

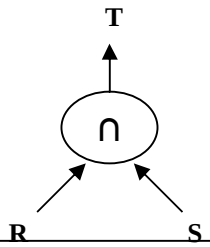
II.2. Intersection

Définition :

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

R et S doivent avoir même schéma.

Notation :



Exemple : T regroupe les produits communs aux catalogues de deux sociétés.

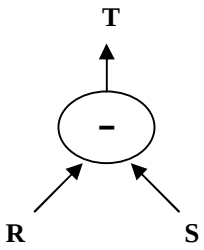
T = R ∩ S		
NumProd	**DesigProd**	**PrixProd**
2	Produit2	200

II.3. Différence

Définition :

T=R – S ou T=MINUS(R, S) permet de retirer les tuples de la relation S existant dans la relation R.

Notation :



Exemple : Retirer les produits de la relation S existant dans la relation R.

R		
NumProd	**DesigProd**	**PrixProd**
1	Produit1	100
2	Produit2	200

S		
NumProd	**DesigProd**	**PrixProd**
2	Produit2	200

Alors :

T = R - S		
NumProd	**DesigProd**	**PrixProd**
1	Produit1	100

II.4. Division

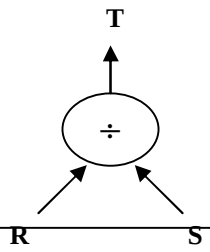
Définition :

T = R ÷ S ou T = DIVISION(R,S)

Où R(A1, A2, …, An) et S(Ap+1, …, An)

T(A1, A2, …, Ap) contient tous les tuples tels que la concaténation à chacun des tuples de S donne toujours un tuple de R.

Notation :



Exemple : La division de R par S fournit la relation T.

R		
NumProd	**DesigProd**	**PrixProd**
1	Produit1	100
2	Produit2	200

S
DesigProd
Produit2

Alors :

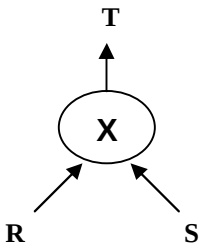
T = R ÷ S	
NumProd	**PrixProd**
1	100

II.5. Produit cartésien

Définition :

T=R X S ou T=PRODUCT(R, S) permet d'associer chaque tuple de R à chaque tuple de S.

Notation :



Exemple :

R		
NumProd	**DesigProd**	**PrixProd**
1	Produit1	100
2	Produit2	200

S
NumFour
1

Alors :

T = R ÷ S			
NumProd	**DesigProd**	**PrixProd**	**NumFour**
1	Produit1	100	1
2	Produit2	200	1

III. Opérateurs spécifiques

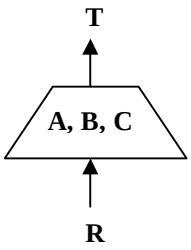
III.1. Projection

Définition :

T=∏<A, B, C>(R) ou T= PROJECT(R / A, B, C)

T ne contient que les attributs A, B et C de R.

Notation :



Exemple : Nom et prénom des clients.

R		
numCli	**prenomCli**	**nomCli**
1	Salah	Ben Salah
2	Ali	Ben Ali

Alors :

T = PROJECT(R / prenomCli, nomCli)	
prenomCli	**nomCli**
Salah	Ben Salah
Ali	Ben Ali

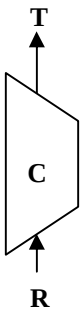
III.2. Restriction

Définition :

T=б <C > (R) ou T=RESTRICT(R / C)

T ne contient que les attributs de R qui satisfont la condition C.

Notation :



Exemple : La liste des clients ayant pour prénom « Ali ».

R		
numCli	**prenomCli**	**nomCli**
1	Salah	Ben Salah
2	Ali	Ben Ali

Alors :

T = RESTRICT(Clients / prenom='Ali')		
numCli	**prenomCli**	**nomCli**
2	Ali	Ben Ali

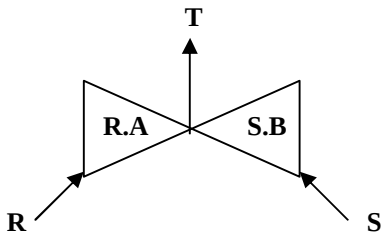
III.3. Jointure

Définition :

T=R >< S ou T=JOIN(R, S)

Produit cartésien R X S et Restriction A=B sur les attributs A de R et B de S.

Notation :



Exemple : La liste des étudiants ainsi que leurs classes.

Etudiant			
NumEtud	**NomEtud**	**PrenomEtud**	**NumClass**
1	Ben Salah	Salah	1
2	Ben Ali	Ali	2

Classe	
NumClass	**NomClass**
1	Info11
2	Info12

Alors :

Etudiant				
NumEtud	**NomEtud**	**PrenomEtud**	**NumClass**	**NomClass**
1	Ben Salah	Salah	1	Info11
2	Ben Ali	Ali	2	Info12

IV. Exercice d'application

Soit le modèle relationnel suivant :

CLIENT (NOC, NOM, ADRESSE)
SERVICE (NOS, INTITULE, LOCALISATION)
PIECE (NOP, DESIGNATION, COULEUR, POIDS)
ORDRE (#NOP, #NOS, #NOC, QUANTITE)

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

1. Détermine les NOS des services qui ont commandé pour le client C1.
2. Déterminer les NOS des services qui ont commandé une pièce P1 pour le client C1.
3. Donner les NOS des services qui ont commandé une pièce de couleur rouge pour le client C1.
4. Donner les NOC des clients qui ont en commande au moins toutes les pièces commandées par le service S1.
5. Donner les NOC des clients qui ont en commande des pièces figurant uniquement dans les commandes du service S1.
6. Donner les NOP des pièces qui sont commandées au niveau local (pour ces pièces, l'adresse du client et la localisation du service sont identiques).

Chapitre : 5

Le langage SQL

Objectifs spécifiques

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

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

Plan du chapitre

I. Présentation de SQL
II. Définition de données
III. Manipulation de données
IV. Interrogation de données
V. Contrôle de données

Volume horaire

6 heures

I. Présentation de SQL

SQL signifie Structured Query Language est le langage des bases de données relationnelles répandant à la fois aux problématiques de création des objets de base de données (modèle), de manipulation des données (algèbre relationnelle), de gestion de la sécurité (droits d'accès), de traitements locaux de données (procédures).[7]

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

Il est utilisé par les principaux SGBDR du marché : Oracle, SQL Server, MySQL, Access, DB2, …

Remarque : Il existe plusieurs implémentations de SQL chez les principaux éditeurs (Voir Annexes).

Dans le reste de ce chapitre, l'implémentation utilisée est celle du SGBD Oracle.

II. Définition de données

II.1. Création des tables

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

```
CREATE TABLE nom_table (
        Attribut1 type1,
        Attribut2 type2,
        ….,
        Contrainte1,
        Contrainte2,
            …
) ;
```

Type des données : [8]

- NUMBER(N) : Entier à N chiffres
- NUMBER(N , M) : Réel à N chiffres au total, M après la virgule.
- DATE : Date complète (date et/ou heure)
- VARCHAR(N), VARCHAR2(N) : chaîne de N caractères (entre ' ') dont les espaces enfin de la chaîne seront éliminés (longueur variable).
- CHAR(N) : Chaîne de N caractères (longueur fixe).

Contraintes d'intégrité

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

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

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

Création :

- CONSTRAINT nom_contrainte UNIQUE (colonne1, colonne2, …) : interdit qu'une colonne, ou la concaténation de plusieurs colonnes, contiennent deux valeurs identiques.

- CONSTRAINT nom_contrainte PRIMARY KEY (attribut1, attribut2, …) : l'ensemble des attributs attribut1, attribut2, … forment la clé primaire de la relation.

- CONSTRAINT nom_contrainte FOREIGN KEY (attribut_clé_étrangère) REFERENCES nom_table (attribut_référence) : l'attribut de la relation en cours représente la clé étrangère qui fait référence à la clé primaire de la table indiquée.

- CONSTRAINT nom_contrainte CHECK (condition) : contrainte là où on doit obligatoirement satisfaire la condition telle qu'elle est énoncée.

- *Exemple* :

```
CREATE TABLE CLASSE (
        NUMCLASSE NUMBER(5),
        NOMCLASSE VARCHAR2(10)
) ;

CREATE TABLE ETUDIANT (
        NCE NUMBER(5),
        NOM VARCHAR2(10),
        PRENOM VARCHAR2(10),
        NUMCLASSE VARCHAR2(10),
        CONSTRAINT PK_ETUDIANT PRIMARY KEY(NCE),
        CONSTRAINT U_NOM UNIQUE(NOM),
        CONSTRAINT CK_NOM CHECK(NOM = UPPER(NOM)),
        CONSTRAINT FK_ETUDIANT_CLASSE FOREIGN KEY(NUMCLASSE)
        REFERENCES CLASSE(NUMCLASSE)
) ;
```

II.2. Renommage des tables

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

```
RENAME Ancien_Nom
TO Nouveau_Nom ;
```

Exemple : Etant donné l'entité ETUDIANT, si on souhaite la renommer par STUDENT, on écrit :

```
RENAME ETUDIANT
TO STUDENT ;
```

II.3. Destruction des tables

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

```
DROP TABLE nom_table ;
```

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

Exemple : Pour supprimer la table ETUDIANT ainsi que son contenu, on fait recours à l'instruction :

```
DROP TABLE ETUDIANT ;
```

II.4. Modification des tables

Il existe plusieurs modifications que l'on peut effectuer sur une table donnée.

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

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

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

```
ALTER TABLE COMMANDE
ADD (MONTANT NUMBER(10,3)) ;
```

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

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

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

```
ALTER TABLE COMMANDE
MODIFY (MONTANT NUMBER(12,3)) ;
```

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

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

Exemple : Ajouter une contrainte à la table Commande qui permet d'obliger des insertions de montants positifs

```
ALTER TABLE COMMANDE
ADD CONSTRAINT CK_MONTANT CHECK(MONTANT >= 0) ;
```

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

```
ALTER TABLE nom_table
DROP CONSTRAINT nom_contrainte ;
```

Exemple : Supprimer la contrainte ck_montant de la table Commande

```
ALTER TABLE COMMANDE
DROP CONSTRAINT CK_MONTANT ;
```

III. Manipulation de données

III.1. Ajout de données

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

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

Exemple : Etant donné la table Etudiant(NCE, nom, prenom, ville). Si on souhaite insérer les informations d'un nouvel étudiant disposant des informations suivantes (1234, Ben Salah, Salah, Djerba), on écrit :

```
INSERT INTO ETUDIANT
VALUES (1234, 'BEN SALAH', 'SALAH', 'DJERBA') ;
```

III.2. Modification de données

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

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

Exemple : Etant donné la table Etudiant (NCE, nom, prenom, ville). Si jamais l'étudiant Salah Ben Salah habite maintenant à Tunis, on écrit dans ce cas :

```
UPDATE ETUDIANT
SET VILLE='TUNIS'
WHERE NCE=1234 ;
```

III.3. Suppression de données

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

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

Exemple : Si on souhaite supprimer l'étudiant de NCE 1234 de la table Etudiant, on écrit :

```
DELETE FROM ETUDIANT
WHERE NCE=1234 ;
```

IV. Interrogation de données

IV.1. Généralités

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

Syntaxe :

L'ordre SELECT possède six clauses différentes, dont seules les deux premières sont obligatoires.[9]

Elles sont données ci-dessous dans l'ordre dans lequel elles doivent apparaître quand elles sont utilisées :

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

IV.2. Projection

Tous les attributs d'une table :

```
SELECT *
FROM nom_table ;
```

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

Exemple : Liste de tous les étudiants

```
SELECT *
FROM ETUDIANT ;
```

Quelques attributs :

```
SELECT attribut1, attribut2, …
FROM nom_table ;
```

Exemple : Liste des noms des étudiants sans duplication

```
SELECT DISTINCT NOM
FROM ETUDIANT ;
```

IV.3. Restriction

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

```
SELECT attribut1, attribut2, …
FROM nom_table
WHERE predicat ;
```

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

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

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

WHERE exp1 = exp2

Exemple : Liste des étudiants qui s'appellent Ali

```
SELECT *
FROM ETUDIANT
WHERE PRENOM = 'ALI' ;
```

WHERE exp1 != exp2

Exemple : Liste des étudiants qui ne s'appellent pas Ali

```
SELECT *
FROM ETUDIANT
WHERE PRENOM != 'ALI' ;
```

WHERE exp1 < exp2

WHERE exp1 <= exp2

Exemple : Liste des étudiants ayant les NCE inférieurs à 100

```
SELECT *
FROM ETUDIANT
WHERE NCE <= 100 ;
```

WHERE exp1 > exp2_

WHERE exp1 >= exp2

Exemple : Liste des étudiants ayant les NCE supérieurs à 100

```
SELECT *
FROM ETUDIANT
WHERE NCE >= 100 ;
```

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

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

Exemple : Liste des étudiants ayant les NCE compris entre 100 et 200

```
SELECT *
FROM ETUDIANT
WHERE NCE BETWEEN 100 AND 200 ;
```

WHERE exp1 **LIKE** exp2 :

LIKE teste l'égalité de deux chaînes en tenant compte des caractères jokers dans la 2ème chaîne:

« _ » remplace un caractère exactement,

« % » remplace une chaîne de caractères de longueur quelconque (y compris de longueur nulle)

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

```
SELECT *
FROM ETUDIANT
WHERE NOM LIKE 'A_%' ;
```

WHERE exp1 **IN** (exp2, exp3, …)

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

Exemple : Liste des étudiants ayant les prénoms appartenant à la liste (Ali, Salah, Bechir)

```
SELECT *
FROM ETUDIANT
WHERE PRENOM IN ('ALI', 'SALAH', 'BECHIR') ;
```

WHERE exp **IS NULL**

Exemple : Liste des étudiants dont les prénoms sont non définis

```
SELECT *
FROM ETUDIANT
WHERE PRENOM IS NULL ;
```

WHERE EXISTS (sous_interrogation)

La clause EXISTS est suivie d'une sous interrogation entre parenthèses, et prend la valeur vraie s'il existe au moins une ligne satisfaisant les conditions de la sous interrogation.

Exemple : Liste des noms des classes qui ont au moins un étudiant ayant le nom 'Ali'.

```
SELECT NOMCLASSE FROM CLASSE
WHERE EXISTS   (SELECT * FROM ETUDIANT
                WHERE ETUDIANT.NUMCLASSE = CLASSE.NUMCLASSE
                AND NOMETUDIANT='ALI');
```

Remarque :

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

IV.4. Tri

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

Syntaxe :

ORDER BY exp1 [DESC], exp2 [DESC], …

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

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

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

Exemple : Liste des étudiants ordonnés par ordre croissant des NCE et décroissant des noms

```
SELECT *
FROM ETUDIANT
ORDER BY NCE, NOM Desc;
```

IV.5. Regroupement

IV.5.1. La clause GROUP BY

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

Syntaxe :

GROUP BY exp1, exp2, …

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

Remarques :

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

Exemples :

Liste des départements ainsi que le nombre de leurs employés

```
SELECT DEPT, COUNT(*)
FROM EMP
GROUP BY DEPT;
```

Liste des départements ainsi que le nombre de leurs secrétaires

```
SELECT DEPT, COUNT(*)
FROM EMP
WHERE POSTE = 'SECRETAIRE'
GROUP BY DEPT;
```

IV.5.2. La clause HAVING

HAVING sert à préciser quels groupes doivent être sélectionnés.

Elle se place après la clause GROUP BY.

Syntaxe :

HAVING predicat

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

Exemple :

```
SELECT DEPT, COUNT(*)
FROM EMP
WHERE POSTE = 'SECRETAIRE'
GROUP BY DEPT
HAVING COUNT(*) > 1 ;
```

IV.6. Opérateurs ensemblistes

IV.6.1. Union

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

Exemple : Liste des ingénieurs des deux filiales

```
SELECT * FROM EMP1 WHERE POSTE = 'INGENIEUR'
UNION
SELECT * FROM EMP2 WHERE POSTE = 'INGENIEUR' ;
```

IV.6.2. Différence

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

Exemple : Liste des départements qui ont des employés dans la première filiale mais pas dans la deuxième

```
SELECT DEPT FROM EMP1
MINUS
SELECT DEPT FROM EMP2;
```

IV.6.3. Intersection

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

Exemple : Liste des départements qui ont des employés dans les deux filiales

```
SELECT DEPT FROM EMP1
INTERSECT
SELECT DEPT FROM EMP2;
```

IV.7. Jointure

IV.7.1. Définition

Quand on précise plusieurs tables dans la clause FROM, on obtient le produit cartésien des tables.
Le produit cartésien de deux tables offre en général peu d'intérêt.
Ce qui est normalement souhaité, c'est de joindre les informations de diverses tables, en précisant quelles relations les relient entre elles. C'est la clause WHERE qui permet d'obtenir ce résultat. Elle vient limiter cette sélection en ne conservant que le sous-ensemble du produit cartésien qui satisfait le prédicat.

Exemple : Liste des noms des étudiants avec les noms de leurs classes

```
SELECT NOMETUDIANT, NOMCLASSE
FROM ETUDIANT, CLASSE
WHERE ETUDIANT.NUMCLASSE = CLASSE.NUMCLASSE ;
```

IV.7.2. Jointure d'une table à elle même

Il peut être utile de rassembler des informations venant d'une ligne d'une table avec des informations venant d'une autre ligne de la même table.
Dans ce cas, il faut renommer au moins l'une des deux tables en lui donnant un synonyme, afin de pouvoir préfixer sans ambiguïté chaque nom de colonne.

Exemple : Lister les employés qui ont un supérieur en indiquant pour chacun le nom de son supérieur

```
SELECT EMP.NOME EMPLOYE, SUPE.NOME SUPERIEUR
FROM EMP, EMP SUPE
WHERE EMP.SUP = SUPE.MATR ;
```

V. Contrôle de données

V.1. Gestion des utilisateurs

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

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

V.1.1. Création d'un utilisateur

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

```
CREATE USER utilisateur IDENTIFIED BY mot_de_passe ;
```

Exemple :

```
CREATE USER ALI IDENTIFIED BY Ae3OPd ;
```

V.1.2. Modification d'un compte utilisateur

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

```
ALTER USER utilisateur IDENTIFIED BY nouveau_mot_de_passe ;
```

Exemple :

```
CREATE USER ALI IDENTIFIED BY A23ePs ;
```

V.1.3. Suppression d'un utilisateur

Syntaxe : Pour supprimer un compte utilisateur, on écrit :

```
DROP USER utilisateur [CASCADE] ;
```

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

Exemple :

```
DROP USER Ali CASCADE ;
```

V.2. Gestion des privilèges

V.2.1. Attribution de privilèges

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

Syntaxe :

```
GRANT privilège
[ON table]
TO utilisateur [WITH GRANT OPTION] ;
```

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

Principaux Privilèges :

SELECT : lecture
INSERT : insertion
UPDATE : mise à jour
DELETE : suppression
DBA, ALL : tous les privilèges

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

Exemples :

```
GRANT SELECT
```

ON ETUDIANT
TO PUBLIC ;

GRANT UPADATE, DELETE
ON ETUDIANT
TO Ali **WITH GRANT OPTION** ;

V.2.2. Suppression des privilèges

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

Syntaxe :

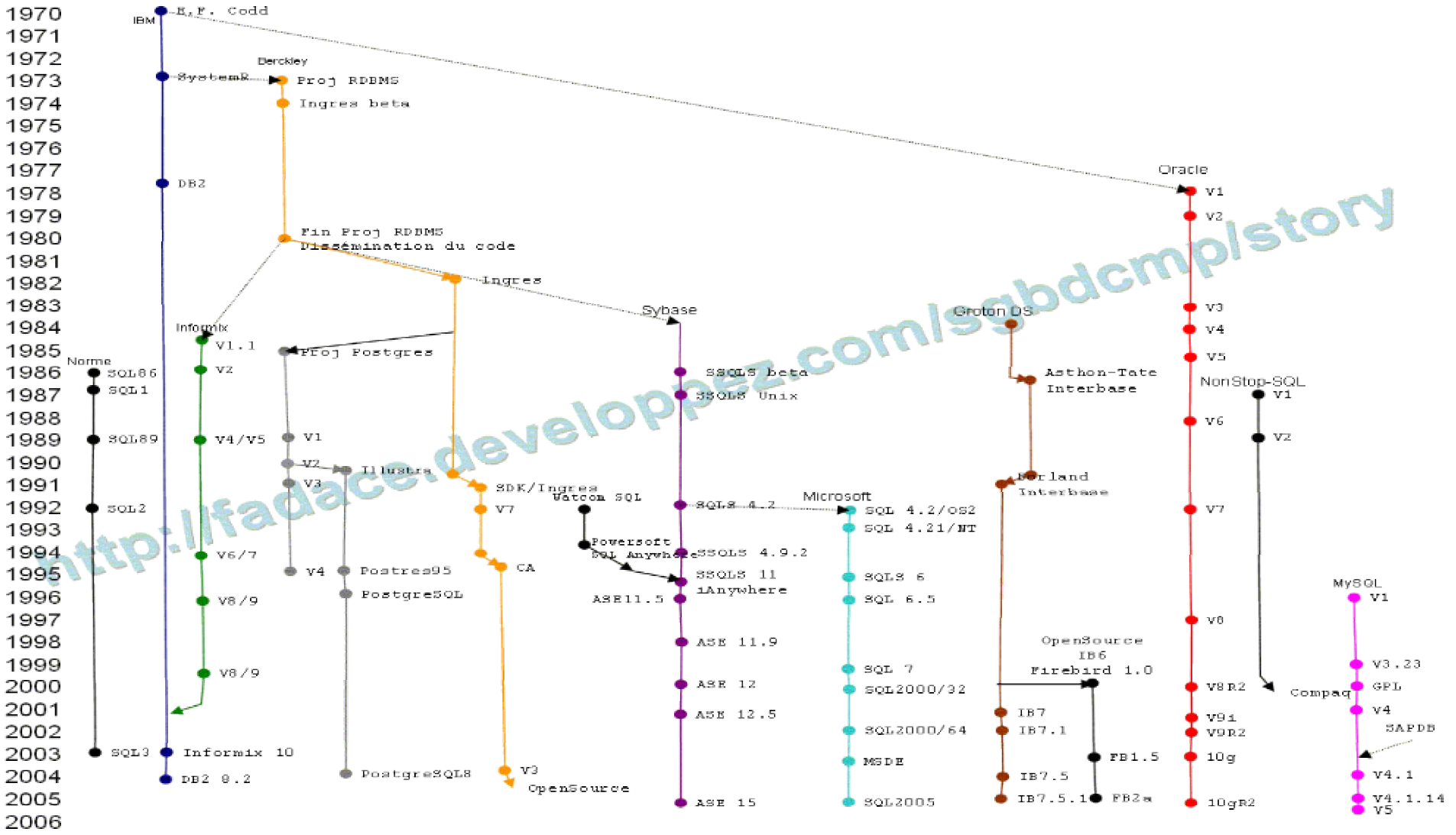
REVOKE privilège
[**ON** table]
FROM utilisateur ;

Exemples :

REVOKE SELECT
ON ETUDIANT
FROM Ali ;

Annexes

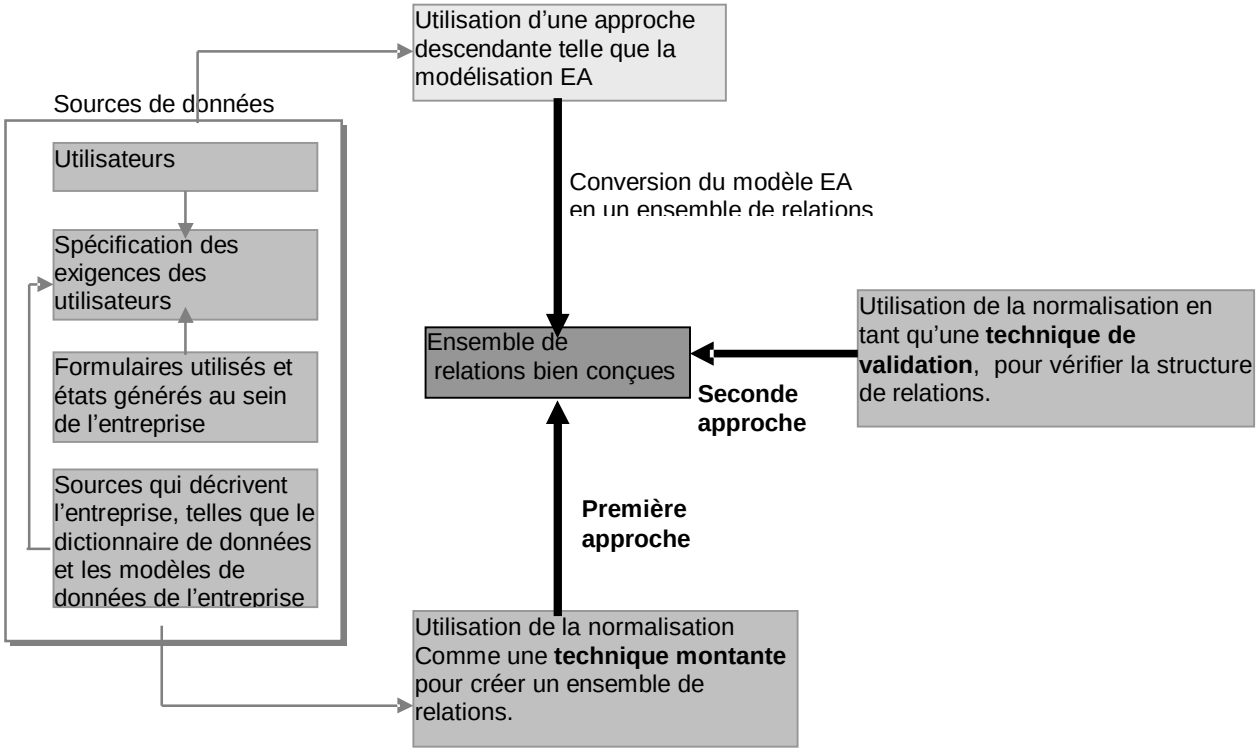
Annexe I : Historique des SGBDR



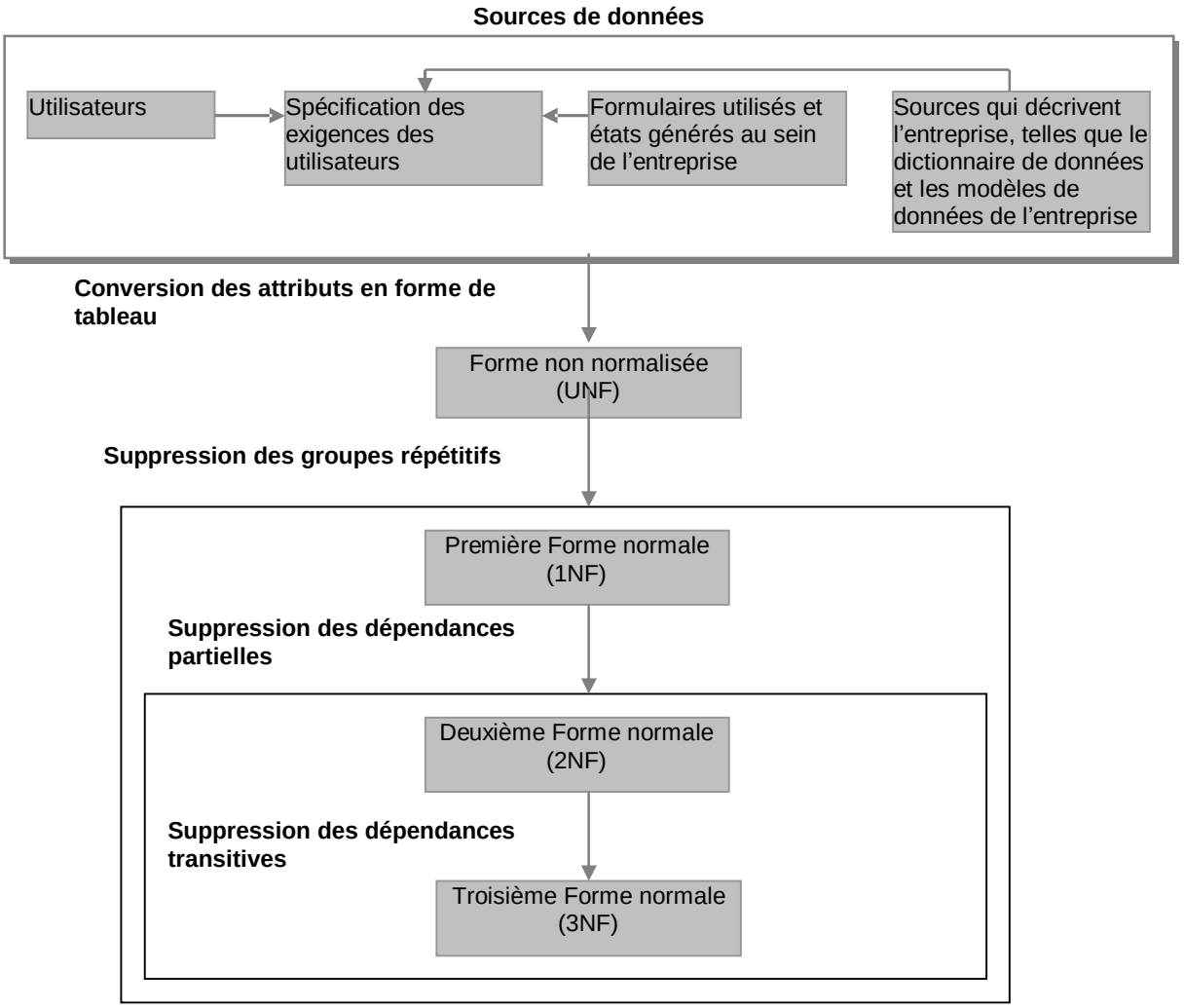
http://fadace.developpez.com/sgbdcmp/story

Annexe II : La normalisation

I. Intérêt de la normalisation dans la conception de base de données



II. Illustration graphique du processus de normalisation



Annexe III : Implémentation de SQL chez les principaux éditeurs

I. Différents types SQL disponibles

Type SQL	Oracle	IBM DB2	SQL Server	PostGreSQL	MySQL
CHAR	Oui	Oui	Oui	Oui	Oui
VARCHAR	Oui	Oui	Oui	Oui	Oui
NCHAR	Oui	GRAPHIC[(n)]	Oui	Non	Oui
NVARCHAR	Oui	VARGRAPHIC[(n)]	Oui	Non	Oui
CLOB	Oui	Oui	TEXT	TEXT	LONGTEXT
NCLOB	Oui	Non	NTEXT	Non	Non
DATE	Oui	Oui	Non	Oui	Oui
TIME	Non	Oui	Non	Oui	Oui
TIMESTAMP	Oui	Oui	DATETIME	Oui	DATETIME(2)
INTERVAL	Oui	Non	Non	(1)	Non
TIME WITH TIME ZONE	Non	Non	Non	Oui	Non
TIME STAMP WITH TIME ZONE	Oui	Non	Non	Oui	Non
BIGINT	Oui	Oui	Oui	Oui	Oui
INTEGER	NUMBER	Oui	Oui	Oui	Oui
SMALLINT	Oui	Oui	Oui	Oui	Oui
FLOAT	Oui	Oui	Oui	Non	Oui
REAL	Oui	Oui	Oui	Oui	Oui
DOUBLE PRECISION	Oui	Oui	Non	Oui	Oui
NUMERIC	Oui	NUMERIC	Oui	Oui	Oui
DECIMAL	Oui	Non	Non	Oui	Oui
BIT	Non	Non	BINARY	Oui	Non
BIT VARYING	RAW	Non	VAR BINARY	Oui	Non
BOOLEAN	Non	Non	BIT	Oui	Non
BLOB	Oui	Oui	IMAGE	Non	LONGBLOB
LOCATOR	Non	Oui	Non	Non	Non
DATALINK	Non (3)	Oui	Non	Non	Non
MULTISET	Non (4)	Non	Non	Non	Non
XML	Oui (5)	Oui	Oui	Non	Non

(1) PostGresSQL possède un type INTERVAL spécifique.

(2) MySQL limite le date time à une précision de la seconde.

(3) Oracle n'implément pas le DATALINK, mais une référence de fichier via le type BFILE.

(4) Pas de MULTISET pour Oracle, mais un VARRAY proche avec utilisation de la fonction MULTISET.

(5) Oracle encapsule son XML dans un BLOB ce qui ne permet pas l'indexation interne du document stocké.

II. Longueur maximale des noms des objets :

Objet	Oracle	IBM DB2	MS SQL Server	PostGreSQL	MySQL
Base	8	8	128	63	64 (1)
Table, vue	30	128	128	63	64 (1)
Colonne	30	30	128	63	64
Contrainte	30	18	128	63	64
Index	30	128	128	63	64

(1) peut être moins long en fonction de l'OS.

Annexe IV : Présentation de quelques fonctions SQL

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

I. Fonctions arithmétiques

ABS(n) : valeur absolue de n

MOD(n1, n2) : n1 modulo n2

POWER(n, e) : n à la puissance e

ROUND(n[,p]) : arrondit n à la précision p (0 par défaut)

SIGN(n) : -1 si n< 0, 0 si n=0 et 1i n>0

SQRT(n) : racine carré de n

TO_CHAR(n, format) : convertit n en chaîne de caractères

TRUNC(n[,p]) : tronque n à la précision p (0 par défaut)

GREATEST(n1, n2, …) : maximum de n1, n2, …

LEAST(n1, n2, …) : minimum de n1, n2, …

TO_NUMBER(chaîne) : convertit la chaîne de caractères en numérique

II. Fonctions chaînes de caractères

DECODE(crit, val1, res1[,val2, res2, …], defaut)

permet de choisir une valeur parmi une liste d'expressions, en fonction de la valeur prise par une expression servant de critère de sélection : elle prend la valeur res1 si l'expression crit a la valeur val1, prend la valeur res2 si crit a la valeur val2, … ; si l'expression crit n'est égale à aucune des expressions val1, val2, …, DECODE prend la valeur defaut par défaut.

Les expressions résultat (res1, res2, …, defaut) peuvent être de types différents : caractère et numérique, ou caractère et date (le résultat est du type de la première expression rencontrée dans le DECODE).

Les expressions val et res peuvent être soit des constantes, soit des colonnes ou même des expressions résultats de fonctions.

LENGTH(chaîne)

prend comme valeur la longueur de la chaîne.

SUBSTR(chaîne, position[, longueur])

extrait de la chaîne "chaîne" une sous-chaîne de longueur "longueur" commençant en position "position" de la chaîne.

Le paramètre longueur est facultatif (par défaut, la sous-chaîne va jusqu'à l'extrémité de la chaîne).

INSTR(chaîne, sous_chaîne[, pos[, n]])

prend comme valeur la position de la sous-chaîne dans la chaîne (les positions sont numérotées à partir de 1). 0 signifie que la sous-chaîne n'a pas été trouvée dans la chaîne.
La recherche commence à la position "pos" de la chaîne (paramètre facultatif qui vaut 1 par défaut). Une valeur négative de "pos" signifie une position par rapport à la fin de la chaîne.
Le dernier paramètre "n" permet de rechercher la nième occurrence de la sous-chaîne dans la chaîne. Ce paramètre facultatif vaut 1 par défaut.

UPPER(chaîne)
convertit les minuscules en majuscules

LOWER(chaîne)
convertit les majuscules en minuscules

LPAD(chaîne, long[, car])
complète ou tronque "chaîne" la longueur "long". La chaîne est complétée à gauche par le caractère (ou la chaîne de caractères) "car".
Le paramètre "car" est optionnel. Par défaut, chaîne est complétée par des espaces.

RPAD(chaîne, long[, car])
fonction analogue à LPAD mais, "chaîne" étant complétée à droite.

LTRIM(chaîne, car)
supprime les caractères à l'extrémité gauche de la chaîne "chaîne" tant qu'ils appartiennent à l'ensemble de caractères "car".

RTRIM(chaîne, car)
fonction analogue à LTRIM mais, les caractères étant supprimés à l'extrémité droite de la chaîne.

REPLACE(chaîne, ch1, ch2)
remplace "ch1" par "ch2" dans "chaîne"

TO_CHAR
La fonction TO_CHAR permet de convertir un nombre ou une date en chaîne de caractère en fonction d'un format :
Pour les nombres : TO_CHAR(nombre, format)
nombre est une expression de type numérique
format est une chaîne de caractère pouvant contenir les caractères suivants :

- 9 représente un chiffre (non représenté si non significatif)
- 0 représente un chiffre (représenté même si non significatif)
- . point décimal apparent
- , une virgule apparaîtra à cet endroit
- $ un $ précèdera le premier chiffre significatif

- B le nombre sera représente par des blancs s'il vaut zéro
- MI le signe négatif sera à droite
- PR un nombre négatif sera entre < >

Pour les dates : TO_CHAR(date, format)

format indique le format sous lequel sera affichée date. C'est une combinaison de codes ; en voici quelques uns :

- YYYY : année en 4 chiffres
- YY : deux derniers chiffres de l'année
- WW : numéro de la semaine dans l'année
- MM : numéro du mois
- DDD : numéro du jour dans l'année
- DD : numéro du jour dans le mois
- D : numéro du jour dans la semaine
- HH ou HH12 : heure (sur 12 heures)
- HH24 : heure (sur 24 heures)
- MI : minutes

Tout caractère spécial insère dans le format sera reproduit dans la chaîne de caractères résultat. On peut également insérer dans le format une chaîne de caractères quelconque, à condition de la placer entre guillemets.

TO_NUMBER(chaîne)

convertit une chaîne de caractères en nombre (quand la chaîne de caractères est composée de caractères numériques)

ASCII(chaîne)

donne le code ASCII du premier caractère de chaîne.

CHR(n)

donne le caractère de code ASCII(n)

TO_DATE(chaîne, format)

permet de convertir une chaîne de caractères en donnée de type date. Le format est identique à celui de la fonction TO_CHAR.

Fonctions de travail avec les dates

ROUND(date, précision)

arrondit la date à la précision spécifiée. La précision est indiquée en utilisant un des masques de mise en forme de la date.

TRUNC(date, précision)

tronque la date à la précision spécifiée (similaire à ROUND).

SYSDATE

a pour valeur la date et l'heure courante du système d'exploitation hôte.

III. Fonctions de groupe

Les fonctions de groupes peuvent apparaître dans le SELECT ou le HAVING. Ce sont les fonctions suivantes :

AVG : moyenne

SUM : somme

MIN : plus petite des valeurs

MAX : plus grande des valeurs

VARIANCE : variance

STDDEV : écart type

COUNT(*) : nombre de lignes

COUNT(col) : nombre de valeurs non nulles de la colonne

COUNT(**DISTINCT** col) : nombre de valeurs non nulles différentes

IV. Autres fonctions

USER

a pour valeur le nom sous lequel l'utilisateur est entré dans Oracle.

NVL(exp, val)

permet de remplacer une valeur NULL de « exp » du par une valeur par défaut « val »

Bibliographie

[1] Georges GARDARIN « Bases de données » Eyrolles 2003.

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

[3] Nicolas LARROUSSE « Création de bases de données » PEARSON Education 2006.

[4] Pierre CRESCENZO « Support de cours magistraux de Bases de données » disponible sur le site : « http://www.crescenzo.nom.fr/CMBasesDeDonnees » visité le 12/04/2007.

[5] Pierre CARRIER, Rémy HUDON, Suzanne JEAN « Bases de données dans le développement de systèmes » Gaêtan Morin 1991.

[6] Jean-Pierre CHEINEY, Philippe PICOUET, Jean-Marc SAGLIO « Systèmes de Gestion de Bases de Données » disponible sur le site : « http://www.bd.enst.fr/polyv7 » visité le 24/03/2007.

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

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

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