Développement d'applications avec les bases de données

Michel Meynard

Table des Matières

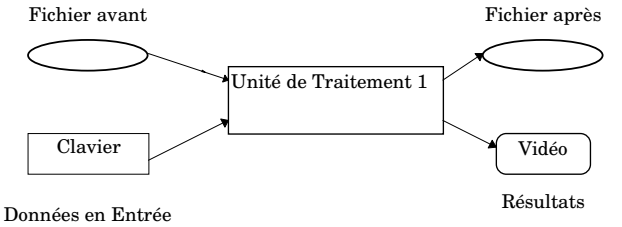
1. Introduction

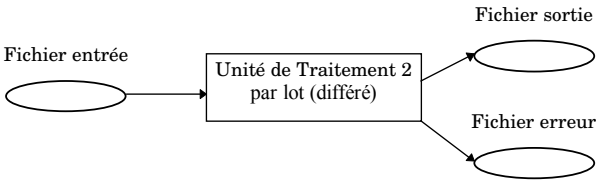
1.1 Généralités, définitions

Pourquoi a-t-on besoin des Systèmes de Gestion de Bases de Données (SGBD) ? Au début de l'informatique, les ordinateurs disposaient de peu de capacité de stockage : les entrées se faisaient par l'intermédiaire de cartes perforées et l'ordinateur réalisait le calcul sur les entrées.

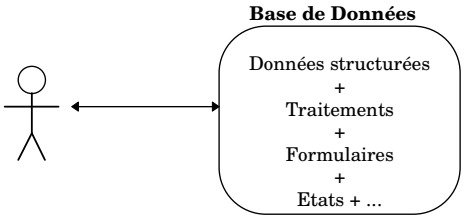
Progressivement les entrées vont se faire par l'intermédiaire d'écran-clavier d'où le problème du stockage de l'information entrée (fichiers).

Les premiers problèmes abordés étaient ceux du domaine de la gestion.





Progressivement on passe à l'idée suivante :



Une **base de données** (BD) peut être définie comme une **collection d'informations structurées** (fichiers, records, tables) modélisant une « entreprise » de l'univers, et mémorisée sur un **support permanent**.

Une telle base de données est manipulée par des logiciels spécifiques ou généraux qui permettent la **consultation** (query) et la **mise à jour** (update) de la base.

Un système de gestion de base de données (**SGBD**) est un logiciel général qui permet à l'utilisateur de manipuler une ou plusieurs BD dans des termes abstraits, sans tenir compte de la façon dont l'ordinateur les maintient.

1.2 Historique

L'évolution historique des bases de données, suit les avancées réalisées dans des domaines divers comme : Systèmes d'exploitation, Langages de programmation, Logique.

1. Avant les années 60, le développement des SGF partagés comportant :
   - des fonctions de base (création, destruction, allocation de mémoire, localisation) ;
   - des fonctions de contrôle (partage, résistance aux pannes, sécurité et confidentialité des données.
2. Durant les années 60, naissance de la première génération de SGBD :
   - séparation de la description des données et des programmes d'application écrits dans un langage hôte ;
   - avènement des langages navigationnels : **modèles navigationnels.**
3. Durant les années 70, naissance de la deuxième génération de SGBD :
   - modèle Entité-Relation ;
   - modèle Relationnel.
4. Années 80-2000 :
   - troisième génération : Modèle Orienté Objet ;
   - bases de Données Déductives.

IDS Integrated Data Store
IMS Information Management System
CODASYL Conférence on Data System Language

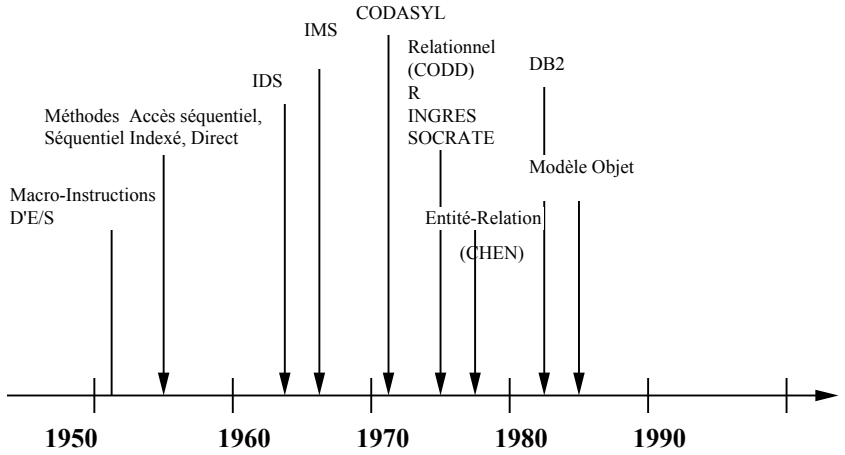


Figure 1 : Evolution de la gestion des données

Actuellement, les SGBD les plus courants sont :

Oracle	Le leader du marché, plates-formes Unix et WinX
DB2 d'IBM	Grands comptes sur Centraux IBM
SQL Server de Microsoft	Très utilisé sur les serveurs Windows
Access de Microsoft	BD bureautique sur WinX
PostgreSQL	Logiciel libre très performant sous Unix
MySQL	Logiciel libre rapide et limité sous WinX, Unix (rapide donc Web)
MapInfo, ArcInfo	SIG : SGBD spésialisés
Lotus Notes	Système d'Information documentaire (Workflow)

1.3 Constituants principaux d'un SGBD

Les S.G.B.D. sont axés sur les données et leur qualité, donc sur leur mise à jour, leur cohérence, leur protection.

Une base de données au sein d'un SGBD est sous la surveillance d'un **DBA** (Administrateur de la Base) et peut être utilisées de différentes manières (structuration, applications programmées, requêtes, ...) par différentes catégories d'utilisateur.

La figure, Structure d'un SGBD, indique les utilisations possibles des BDs gérées par un SGBD.

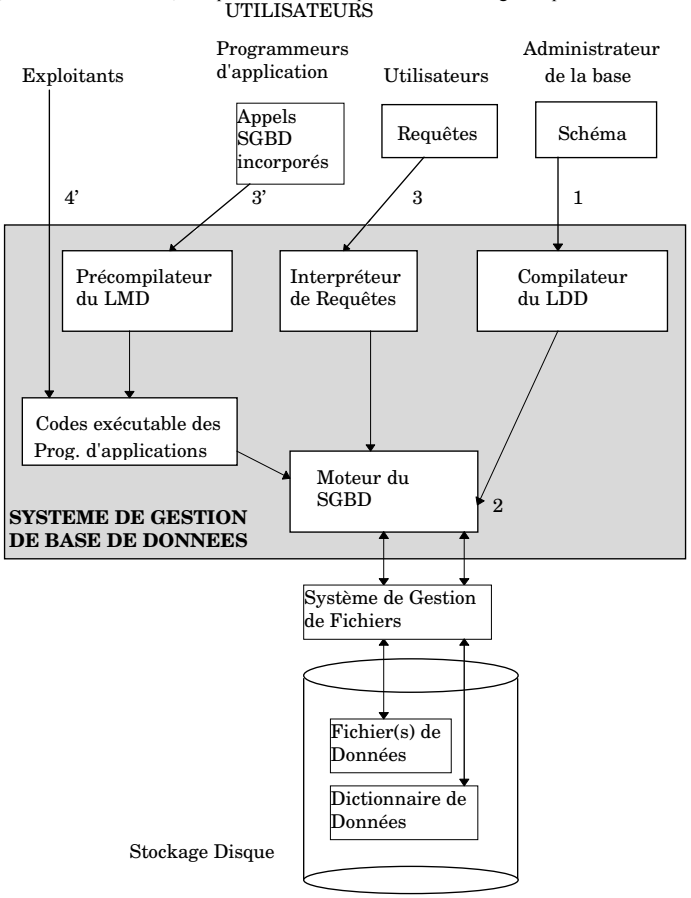


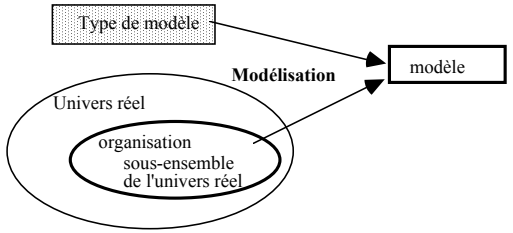
Figure 2 : Structure d'un SGBD

Les fonctions essentielles d'un SGBD sont des fonctions d'**organisation**, d'**interrogation** et de **contrôle** des données :

- Organisation : le **Langage de Définition des Données** (LDD) permet la structuration des données. Les utilisateurs de la base ne doivent pas se préoccuper de la structuration physique des données, celles-ci sont donc vues et décrites avec un certain degré d'abstraction dans un langage spécifique, le LDD.
- Le **Langage de Manipulation de Données**(LMD) associe souvent un langage de programmation et un langage d'interrogation. L'interrogation de la base doit être possible par des personnes de qualifications différentes : (programmeurs, utilisateurs finaux).
- Le Gestionnaire de la Base de Données, appelé également **moteur**, peut être vu comme le coeur du SGBD car à partir des résultats de la compilation du LDD, et des requêtes utilisateurs, il traduit en une suite d'opérations sur le ou les fichiers qui contiennent réellement les données. Il doit posséder les qualités suivantes : **Sécurité, Intégrité, Performance**. Pour cela, le SGBD doit utiliser le SGF du système d'exploitation sous-jacent. Pour améliorer la portabilité du SGBD, l'interface avec le SGF est généralement minimale : On utilise un fichier non fragmenté du SGF pour stocker la BD (Access), voire l'ensemble des BD (Oracle). Dans d'autres cas, IMS, DB2, une partition entière est dédié au SGBD qui inclut un SGF spécifique. Pour améliorer l'efficacité de certains accès, des structures de données particulières (hachage, index linéaire, b-arbre) sont utilisées.

1.4 Modélisation

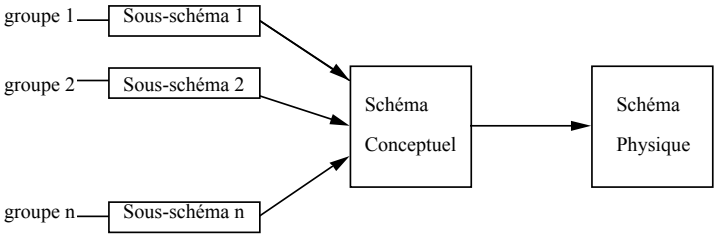
De manière très générale, une représentation de la réalité d'une organisation constitue un modèle. La **modélisation** d'une Base de Données suppose une **méthode** de modélisation (spécifications), des outils d'aide à l'analyse ... La méthode permet notamment d'assurer certaines **contraintes d'intégrité** régissant la cohérence des données.



Remarque : la modélisation dans le cadre des bases de données ne s'intéresse qu'aux données elles-mêmes. En revanche, dans les Systèmes d'Information (SI) la modélisation porte aussi sur les traitements.

Entre le Réel et la Machine, on admet généralement trois niveaux d'abstraction (norme ANSI/SPARC American National Standard for Information Systems/Standard Planning And Requirement Committee) :

- **Niveau interne** : c'est la Base de Données vue par le SGBD. Ce niveau est subdivisé en :
  1. un niveau logique ou d'accès ;
  2. un niveau physique.
- **Niveau conceptuel** : le **modèle conceptuel** de la Base de Données est une abstraction du monde réel également nommé **schéma conceptuel**. Il existe plusieurs méthodes de modélisation que nous verrons ultérieurement.
- **Niveau externe** : le schéma conceptuel est souvent une abstraction trop large et complexe pour un utilisateur donné. Aussi on introduit la notion de **vue**, de **sous-schéma**, qui est abstraction partielle de la base. L'utilisateur voit la base par « une fenêtre » appelée sous-schéma ou vue.



Niveaux d'abstraction

Diverses modèlisations « conceptuelles » de données sont utilisées : dans un premier temps, nous rappellerons brièvement les modèles conceptuels navigationnels.

1.5 Les Modèles Navigationnels

Nous indiquons ci-après les caractéristiques essentielles des modèles hiérarchiques et réseau.

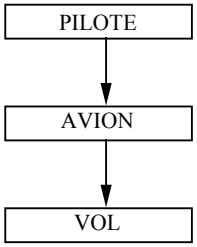
1.5.1 Le Modèle hiérarchique

Le modèle hiérarchique est employé par des systèmes très répandus comme IMS [1](IBM). Dans le modèle hiérarchique, la structure des données est décrite par deux concepts :

- l'**enregistrement logique** ou **article** (record) qui regroupe l'ensemble des propriétés ou **champs** constituant les caractéristiques d'une entité ;
- le **lien orienté** associant les enregistrements logiques selon une arborescence : un article père est relié à 0 ou N articles fils, tandis qu'un fils n'a qu'**1** père.

La base de données est alors constituée d'une **collection d'arbres** (forêt) dont les racines constituent les points d'accès à la BD. Les requêtes de consultation ou de mise à jour consistent donc en une navigation à l'intérieur de cette forêt.

Exemple :



Arbre TRANSPORT_AERIEN

Article PILOTE Racine
PNOM caractère(20)
ADRESSE caractère (30)
.....

Article AVION Parent = PILOTE
ANOM caractère(10)
CAPACITE entier

Article VOL Parent = AVION
VNOM caractère(5)
ORIGINE caractère(20)
DESTINATION caractère(20)

Schéma hiérarchique de la base de données « aviation »

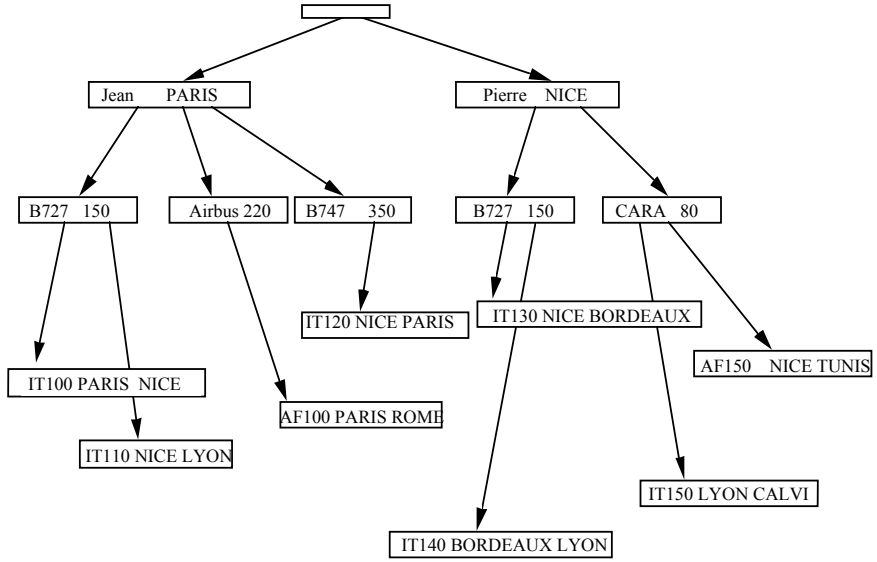
Le modèle présente l'avantage d'être simple mais présente quelques faiblesses dues à la structure arborescente :

- seule une association [1-N] sera naturellement représentée ;
- il y a duplication d'enregistrements communs à deux instances du même lien (exemple du B127 150) ou même pire : duplications d'ensembles d'articles.(exemple des conducteurs).

Certaines redondances peuvent être évitées par la définition d'articles virtuels pointeurs vers des articles réels. Les langages de définition et de manipulation des Données sont intégrés dans un langage de programmation hôte (par exemple PL1 ou COBOL).

LE LMD permet de parcourir les arbres d'articles, de mémoriser des positions dans ces arbres, d'insérer, de supprimer, ou de modifier des articles dans ces arbres.

[1] Information Management System dont l'étude commença en 1966 en vue de l'opération spatiale Apollo pour ensuite devenir un des SGBD les plus répandus dans les années 1970.



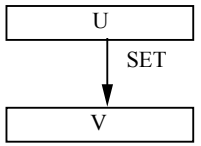
Une instance du schéma « aviation »

Un autre exemple de schéma hiérarchique concernant une société d'assurances et sa BD ASSURE :
ASSURE→VOITURE, ASSURE→DEUXROUES,
VOITURE→CONDUCTEURV, DEUXROUES→CONDUCTEURD
Dans ce schéma un même conducteur de deux-roues et de voiture sera enregistré dans deux ensembles d'enregistrements distincts : CONDUCTEURV et CONDUCTEURD. Une solution consiste à créer alors une autre BD CONDUCTEUR(NUMC, NOM, PRENOM, ...) et à n'enregistrer dans la BD ASSURE que les NUMéros de Conducteurs. Dans ce cas, le volume des données est optimisé mais le temps de traitement sera plus long (ouverture de deux BD).

1.5.2 Le modèle Réseau

Proposé par le groupe DBTG (Data Base Task Group) et utilisé par IDS (CII HB), IDMS, le modèle réseau résout certains problèmes du modèle hiérarchique. Les concepts utilisés sont les mêmes que ceux du modèle hiérarchique : enregistrements logiques (**record**) et liens (**set**). Cependant le rôle des liens se diversifie : liens CODASYL, anneaux SOCRATE, ...

Un lien entre deux types d'articles appelés **SET CODASYL** matérialise une association entre ces deux types d'articles distincts.



Contraintes :

- Tout v de V est relié à un seul u de U.
- A chaque u de U on peut associer un ensemble S(u) d'articles v.
- U est dit **propriétaire** du SET, V est dit **membre** du SET.
- u et S(u) constitue une occurrence du SET.
- Chaque SET est **nommé.**

Le schéma conceptuel est présenté sous la forme d'un graphe, d'un réseau connectant des **types d'article** comportant un ou plusieurs **champs** par des **types de set** nommés.

Exemple :

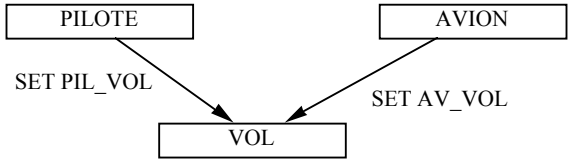
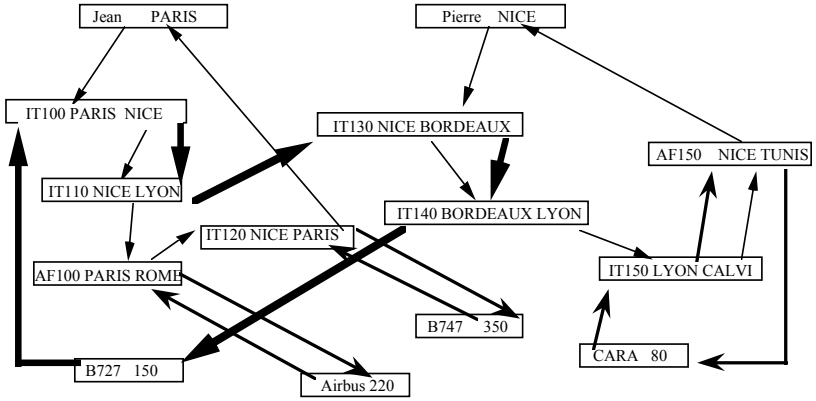


Schéma de la base de données réseau « aviation »



Une occurrence du schéma

Le langage de manipulation des données permet :

- d'entrer dans la base par des points d'accès positionnés sur certains types d'articles ;
  de suivre les liens (naviguer) : opérations S (premier fils), S^F (frère suivant), S^{-1} (père) ;
- l'insertion, la suppression et la modification d'articles et de liens.

Les opérations disponibles doivent être, comme pour le modèle hiérarchique, immergées dans un langage hôte (Cobol, Pascal ...).

Le principal reproche formulé à l'encontre de ces modèles est leur **« indépendance physique »** faible. Dans les modèles navigationnels, l'imbrication implémentation physique et description conceptuelle est trop étroite au niveau de la description du schéma pour apporter la réelle indépendance souhaitée.

2. Le Modèle Relationnel

2.1 Introduction

Proposé à partir de 1970 (travaux de CODD), le modèle relationnel de données connaît un grand succès pour les raisons suivantes :

- représentation simple des données à l'aide du concept unique de relation ;
- existence d'une démarche rigoureuse (normalisation) permettant la construction du schéma ;
- SGBD relationnels et langages d'interrogation renommés (SQL).

Dans ce chapitre nous nous intéresserons aux concepts de ce modèle, et à la description d'un langage « théorique » attaché au modèle relationnel : le langage algébrique. Attention à la distinction **théorie/pratique**. Le modèle relationnel est un modèle mathématique (théorique) qui est implémenté dans de nombreux SGBD (pratique). L'implémentation du modèle est généralement moins contraignante que le modèle (valeurs nulles, doublons).

2.2 Définitions

Le modèle relationnel n'utilise que le concept mathématique de **relation**. Une relation dans la théorie des ensembles est un sous-ensemble du **produit cartésien** de **domaines**. Un **domaine** est un ensemble de valeurs.

Exemples de domaines:

- un ensemble d'entier ;
- un ensemble de chaînes de caractères ;
- {rouge,vert,bleu}.

Le **produit cartésien** des domaines D1,D2,...,Dn est dénoté par : D1 x D2 x....x Dn. C'est l'ensemble des **n-uplets** ou n-uplets (v1,v2,...,vn) tels que vi $\in$ Di pour i=1,2,...,n.

Une **relation** sur les domaines D1,D2,...,Dn est un sous-ensemble du produit cartésien de ces domaines.

On appelle n l'**arité** de la relation, et **n-uplets** (ou n-uplets) les éléments de la relation.

Le nombre de n-uplets d'une relation est appelé son **cardinal** (ou sa cardinalité).

Il est courant de représenter une relation sous forme d'une **table**, où les lignes correspondent aux n-uplets et les colonnes aux **attributs**.

Exemple :
la table **ACHAT** représente les noms de personnes ayant acheté une voiture une certaine année, avec un kilométrage donné.

NOM	NIMM	ANNEE	KILOMETRAGE
Dupont	1234UV34	1990	36000
Martin	3456YT75	1993	78100
Durand	4321IO13	1999	12000
Dupont	7865UU34	2000	0

Remarques :

- on donne un nom différent à chaque colonne pour les distinguer (notion de **rôle**) et on appelle **attribut** d'une relation un couple (nom, domaine) ;
- l'ordre des attributs n'a pas d'importance si les valeurs des n-uplets « suivent » ;
- de même, l'ordre des n-uplets n'a pas d'importance ;
- d'un point de vue théorique, une relation ne peut pas posséder deux n-uplets identiques car c'est un ensemble. D'un point de vue pratique, les **doublons** existent dans les SGBD relationnels ;
- d'un point de vue théorique, un attribut d'un n-uplet possède **toujours une valeur** qui est un élément du domaine associé à cet attribut. D'un point de vue pratique, la valeur **NULL** est un élément spécial associé aux attributs qui n'ont pas été saisis. On peut voir cette non-valeur comme un élément appartenant à tous les domaines ;
- Synonymes Théorie/Pratique : relation/table, n-uplet/ligne , attribut/colonne, domaine/type, cardinal/taille.

On appelle **Schéma de la relation** le nom de la relation suivi de la liste de ses attributs entre parenthèses : nomRelation(nomAtt1, nomAtt2, ...).

Pour reprendre l'exemple ci-dessus :
ACHAT(NOM CHAR(20), NIMM CHAR(8), ANNEE NUMBER(4), KILOMETRAGE NUMBER(7))

La représentation tabulaire de la relation à un instant donné est souvent désigné par **instance** de la relation. Le schéma d'une relation est une organisation logique (on peut parler d'intention de la relation) qui donne l'interprétation de la relation. Une instance du schéma ou **relation** correspond au contenu à un instant donné (à l'extension).

2.3 Schéma de base de données relationnelle

Un **schéma de base de données relationnel** est constitué d'un ensemble de **schémas de relation** définis sur des ensembles d'attributs et soumis à un certain nombre de **contraintes** dites contraintes d'intégrité qui expriment la sémantique de la représentation en imposant certaines règles.

Du point de vue théorique, chaque attribut d'un **schéma** possède un **nom unique** : deux attributs de deux tables ayant la même sémantique doivent donc être nommés différemment. Dans la pratique, on peut nommer de la même façon deux attributs appartenant à deux tables. En cas d'ambiguïté, pour désigner un tel attribut, on le préfixera par le nom de sa table d'origine : NOM_TABLE(NOM_ATTRIBUT).

Exemple de schéma (pratique) :
ARTICLE(REF CHAR(8), DESIGNATION CHAR(20), PRIX_UNIT MONETAIRE)
COMMANDE(N_COM NUMBER(8), CLIENT CHAR(20), DATE_COM DATE)
LIGNE_COM(N_COM NUMBER(8), REF CHAR(8), QUANTITE NUMBER)

Dans l'exemple précédent, on distinguera bien COMMANDE(N_COM) et LIGNE_COM(N_COM) qui sont deux colonnes tout à fait distinctes !

2.4 Les contraintes

Les **contraintes d'intégrité** exprimables au niveau schéma peuvent être variées :

- contraintes de domaine ;
- dépendances fonctionnelles, identifiants, clés ;
- contraintes référentielles ou contraintes d'inclusion ;

Un type important de contraintes intervenant au niveau d'un schéma de relation est la dépendance fonctionnelle.

2.4.1 Contrainte de domaines

Définition *extensive* ou *intensive* du domaine d'un attribut :

- l'attribut Nom du schéma de relation Fournisseurs est contraint à être une chaîne de caractères de longueur 20 ;
- l'attribut Couleur du schéma de relation Voitures a ses valeurs dans l'ensemble {rouge,vert,bleu,noir,blanc} ;
- l'attribut mois est compris entre 1 et 12 ;
- le prix unitaire doit être strictement positif.

Présence obligatoire ou non d'une valeur pour un attribut : NULL ou NOT NULL.

L'existence de la valeur d'un attribut peut être lié à la valeur d'un autre attribut : par exemple, l'attribut Nommarital d'un schéma de relation Personne ne prend de valeur que si la valeur de l'attribut Sexe est « féminin ».

Règles de calcul indiquant comment la valeur d'un attribut est déduite de la valeur d'un ou plusieurs attributs. Par exemple l'attribut date de naissance est contraint a avoir la même valeur pour l'année que les caractères 2 et 3 de l'attribut N_SS contraint à être une chaîne de caractères de longueur 13.

Autre contrôles (CHECK) de la valeur d'un attribut.

La représentation des contraintes (de domaines) dépend du LDD du SGBD : SQL, autre,...

Exemple :
LIGNE_COM(N_COM NUMBER(8), REF CHAR(8), QUANTITE NUMBER)
N_COM, REF, QUANTITE non NULLS
QUANTITE>0

2.4.2 Dépendance fonctionnelle

D'autres contraintes entre attributs existent que nous allons détailler (DF, DI). Les DJ seront étudiées lors du processus de normalisation.

Définition : Soit R(A1,A2,....An) un schéma de relation et G et D deux sous-ensembles de {A1,A2,...,An}, on dit que **G détermine D** ou que **D dépend fonctionnellement de G** et on note **G→D** ssi pour toute instance r de R, tous les n-uplets u,v de r qui ont même valeur dans G ont aussi même valeur dans D.

c'est-à-dire : ($\forall$ r instance de R) ($\forall$ u,v $\in$ r tel que u[G] = v[G] alors u[D] = v[D])

Remarques :
si G$\cup$D est l'ensemble des attributs de R, on ne peut avoir deux n-uplets u,v $\in$ r instance de R coïncidant sur G car sinon ils seraient égaux. Or les relations sont des ensembles, donc un n-uplet ne figure qu'une seule fois. En pratique, les valeurs nulles sont interdites pour tout attribut de la partie gauche d'une DF !

Exemple :
Soit la relation VOITURE(N_IMM, COULEUR, TYPE, PUIS) et les règles suivantes :
- une voiture est identifiée par un numéro d'immatriculation N_imm ;
- à un type de voiture correspond une puissance.

On peut associer les DF suivantes : {N_IMM} → {N_IMM, COULEUR, TYPE, PUIS} ; {TYPE} → {PUIS}

2.4.3 Identifiant et clé

La notion de dépendance fonctionnelle permet de définir la notion d'**identifiant** ou surclé : K est **identifiant** de R sur U, l'ensemble des attributs de R, ssi K$\subseteq$U et K→U.

Deux n-uplets distincts ne peuvent avoir le même identifiant (sinon ils coïncident entièrement et R contient 2 n-uplets égaux). Dans la théorie relationnelle, U est toujours un identifiant de R (trivial).

Exemple :
ETUDIANT(N_ETUD, N_SS, NOM, ...) possède plusieurs identifiants : {N_ETUD}, {N_SS}, mais aussi {N_ETUD, NOM}, ...
LIGNE_COM(N_COM NUMBER(8), REF CHAR(8), QUANTITE NUMBER) possède l'identifiant composé {N_COM, REF} en supposant qu'une commande ne possède pas deux lignes de commande du même article.

Définition d'une clé de schéma de relation
Soit R(A1,A2,....An), un schéma de relation et K un sous-ensemble de {A1,A2,...,An}, on dit que K est une **clé** pour R ssi :
- *propriété 1* : K →A1,A2.....An ;
- *propriété 2* :pour tout K' inclus dans K si K' → A1,A2,....An, alors K=K'.

La deuxième condition introduit la **minimalité au sens de l'inclusion**, de l'ensemble des attributs qui constitue une clé. Autrement dit, K est clé de R ssi non $\exists$A$\in$K tel que : K-{A}→U.

Exemples :
- ETUDIANT(N_ETUD, N_SS, NOM, ...) possède 2 clés : {N_ETUD} et {N_SS}.

- Dans LIGNE_COM(N_COM, REF, QUANTITE), {N_COM, REF, QUANTITE} est identifiant mais n'est pas clé. La clé est {N_COM, REF}.

Remarques :
- si plusieurs clés sont possibles, on les appelle **clés candidates**. On en choisit généralement une qui sera la **clé primaire** ;
- le choix de la bonne clé primaire d'une table est primordial et comme toute contrainte, ce choix devra s'appuyer sur la sémantique de la table. Dans ETUDIANT, par exemple, la clé primaire devrait être N_ETUD car on s'intéresse à des étudiants et non à des assurés sociaux ;
- les clés primaires composées d'un unique attribut numérique (N_CLIENT, N_FOURN, N_COM, ...) sont très courantes pour des raisons d'efficacité de recherche, cependant elles ne sont pas très « parlantes » pour les utilisateurs qui préfèrent des critères de recherche plus « humains » (nom du client et date de la commande pour une commande par exemple) ;
- par conséquent, dans une application BD, il faudra souvent penser à des recherches selon différents critères : d'une part numérique sur la clé, d'autre part par raffinements successifs. Un étudiant qui a perdu sa carte d'étudiant et qui en désire un duplicata ne se souviendra pas de son N_ETUD, mais de son nom, prénom, date de naissance, ...
- la représentation de la clé primaire dans un schéma relationnel consiste à souligner les attributs membres de la clé primaire. Par exemple, ARTICLE(<u>REF</u>, DESIGNATION, PRIX_UNIT), COMMANDE(<u>N_COM</u>, CLIENT, DATE_COM), LIGNE_COM(<u>N_COM, REF</u>, QUANTITE).

Remarques sur la pratique :
Dans la pratique, les doublons sont permis dans la plupart des SGBD. Une table sans clé primaire déclarée pourra donc posséder des doublons. Par contre, si une **clé primaire** est déclarée pour cette table, **aucun doublon** n'est permis. De plus, aucun des attributs constituant la clé primaire ne peut avoir une valeur **NULL**.
Pour modéliser d'autres clés candidates, les SGBD admettent généralement une contrainte d'intégrité **UNIQUE**. A la différence de la clef primaire, les attributs constituant la contrainte UNIQUE, peuvent avoir une valeur **NULL**. Par conséquent, une table contenant une CI UNIQUE et pas de clé primaire, peut contenir des doublons. Remarquons que l'on peut associer la CI NOT NULL à la CI UNIQUE pour éviter la valeur nulle.

2.4.4 Dépendance d'inclusion et Contrainte d'Intégrité Référentielle

Soit un schéma relationnel S contenant deux tables A(A1, A2, ..., An) et B(B1, B2, ..., Bk). Soient une séquence d'attributs de A (Aa, Ab, ..., Ai) et une séquence de même longueur d'attributs de B (Ba, Bb, ..., Bi). Une dépendance d'inclusion entre ces deux suites d'attributs notée **(Aa, Ab, ..., Ai)$\subseteq$(Ba, Bb, ..., Bi)** existe ssi toute instance du schéma vérifie que pour toute ligne de A, les valeurs de (Aa, Ab, ..., Ai) existent dans au moins une ligne de B sur les attributs (Ba, Bb, ..., Bi).

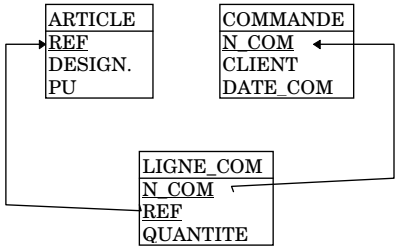
Remarquons qu'il n'existe aucune contrainte sur les deux tables A, B et sur les attributs membres de la DI. On peut avoir par exemple :
EMPLOYE(<u>MATRICULE</u> NUMBER(6), NOM CHAR(20), NOM_SUP CHAR(20), ...) avec la DI : NOM_SUP $\subseteq$ NOM. Cette contrainte exprime que tous les noms de supérieurs hiérarchiques sont des noms d'employés.

En règle générale, les DI sont utilisées pour créér des « liens » entre les tables afin d'éviter certaines incohérences. On les appelle alors des contraintes de référence, ou Contraintes d'Intégrité Référentielle (**CIR**) et elles sont des DI dont le membre de droite doit être un **identifiant** de la relation cible.
L'exemple précédent devient alors :
EMPLOYE(<u>MATRICULE</u> NUMBER(6), NOM CHAR(20), MAT_SUP NUMBER(6), ...) avec MAT_SUP $\subseteq$ MAT.

Exemple : soit le schéma : ARTICLE(<u>REF</u>, DESIGNATION, PRIX_UNIT), COMMANDE(<u>N_COM</u>, CLIENT, DATE_COM), LIGNE_COM(<u>N_COM, REF</u>, QUANTITE),
- **LIGNE_COM(N_COM) $\subseteq$ COMMANDE(N_COM)** interdit d'ajouter une ligne de commande qui ne correspondrait à aucune commande en cours.
- **LIGNE_COM(REF) $\subseteq$ ARTICLE(REF)** interdit d'ajouter une ligne de commande d'un article n'existant pas.

L'implémentation des CIR déclaratives (FOREIGN KEY N_COM REFERENCES COMMANDE(N_COM)) en SQL dans les SGBD est très pratique pour éviter notamment certaines incohérences dues à des fautes de frappe : la « clé étrangère » doit exister comme identifiant dans la table référencée !

On représente souvent les Contraintes d'Intégrité Référentielles dans un schéma appelé parfois multigraphe des références où des arcs représentent les CIR :



Multigraphe de références

Remarques sur la pratique :

Dans la pratique, la partie droite d'une CIR doit être une clé primaire ou être UNIQUE (NULL ou NOT NULL). La partie gauche, nommée clé étrangère, peut être NULL.

2.5 Les langages d'interrogation

L'algèbre relationnelle est un langage d'interrogation « **théorique** » basé sur l'utilisation d'opérateurs appliqués à des relations. C'est donc un langage **impératif** permettant de décrire précisément la façon d'obtenir un résultat. Un autre langage théorique, dit langage prédicatif, permet de **déclarer** la forme du résultat souhaité sans indiquer le moyen d'y parvenir. La plupart des langages **pratiques**, tel que SQL, s'inspirent de ces deux approches.

2.6 L'algèbre relationnelle

Les opérations de base sont de deux sortes :

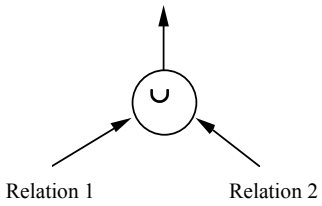
- les opérations ensemblistes ;
- les opérations spécifiques relationnelles.

2.6.1 Opérations ensemblistes

2.6.1.1 UNION

Opération portant sur **deux relations de même schéma** Relation 1 et Relation 2 consistant à construire la relation 3 de même schéma ayant pour n-uplets ceux appartenant à R1 ou à R2 ou aux deux relations.

On peut la noter :
UNION (Relation 1, Relation 2)
ou Relation 1 ∪ Relation 2



UNION

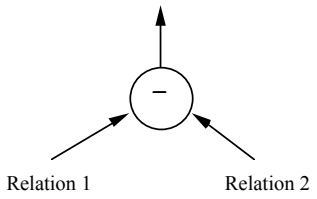
Exemple : fusion de deux relations ARTICLE_CUISINE(REF, DESIGN, PU) et ARTICLE_SALON(REF, DESIGN, PU)

2.6.1.2 DIFFERENCE

Opération portant sur les deux relations 1 et 2 de **même schéma**, le résultat est la relation 3 de même schéma et ayant pour n-uplets ceux appartenant à R1 et n'appartenant pas à R2.

On peut la noter :

DIFFERENCE (Relation 1, Relation 2)
ou Relation 1 - Relation 2



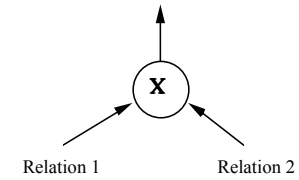
DIFFERENCE

Exemple : différence de deux relations ARTICLE_99(REF, DESIGN, PU) et ARTICLE_2000(REF, DESIGN, PU).

2.6.1.3 PRODUIT CARTESIEN

Opération portant sur **deux relations** 1 et 2, consistant à construire une relation 3 ayant pour schéma la juxtaposition de ceux des relations opérandes et pour n-uplets toutes les combinaisons des n-uplets des relations 1 et 2.

On peut la noter :
Relation 1 × Relation 2



PRODUIT CARTESIEN

Exemple : produit cartésien de deux relations COULEUR(COUL) et FORME(GEOMETRIE).

2.6.1.4 INTERSECTION

R1∩R2=R1-(R1-R2) (voir ci-après).

Exemple : intersection de deux relations DEBITEUR_CC(CLIENT) et COMPTE_EPARGNE(CLIENT).

Exercice : quelles sont les arités et cardinalités de R1*R2, avec *∈ {∪,∩,×,-}.

2.6.2 Opérations spécifiques :

2.6.2.1 SELECTION

Opération sur **une relation** produisant une relation de même schéma mais ne comportant que les seuls n-uplets vérifiant la condition précisée en opérande.

On peut la noter :

SELECTION (Relation 1, condition)
condition du type :
Attribut opérateur Valeur
ou Relation 1 **:** condition
ou $\sigma_{condition}$ (Relation 1)

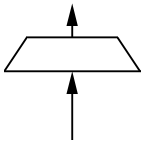
SELECTION

Exemple : sélection des articles de moins de 10 F : $\sigma_{PU<10}$ (ARTICLE).

2.6.2.2 PROJECTION

Opération sur **une relation** consistant à composer une relation 2 en enlevant à la relation initiale tous les attributs non mentionnés en opérande et en éliminant les n-uplets qui constituent des "doublons".

On peut la noter :
PROJECTION (Relation 1, att i,....) ou
Relation 1 **[**atti,....**]** ou
$\Pi_{atti,..}$(Relation 1)



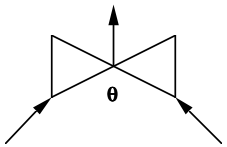
Relation 1

PROJECTION

Exemple : projection des références d'articles : $\prod_{REF}$ (ARTICLE).

2.6.2.3 JOINTURE

Opération consistant à rapprocher selon une condition, les n-uplets de **deux relations** 1 et 2 afin de former une relation 3 qui contient l'ensemble de tous les n-uplets obtenus en concaténant un n-uplet de 1 et un n-uplet de 2 vérifiant la condition de rapprochement. (celle-ci est du type Attribut1 opérateur Attribut 2).



JOINTURE

On peut la noter :
- JOIN(Relation 1, Relation 2, Condition) ou :
  R $\bowtie_{\theta}$ **S**
-

Remarques :
- **jointure naturelle** : égalité des attributs de même nom en ne conservant qu'une seule fois ces attributs dans la relation résultat ; par exemple, COMMANDE $\bowtie$ LIGNECOM donne (N_COM, CLIENT, DATE_COM, REF, QUANTITE). Pour toutes les jointures non naturelles, les attributs de même nom sont **renommés** dans le résultat d'une jointure.
- **équijointure** : égalité entre attributs de noms différents ; par exemple, ETUDIANT(N_ET,NOM_ET)$\bowtie_{NOM\_ET=NOM\_PROF}$ PROF(N_PROF, NOM_PROF) donne (N_ET, NOM_ET, N_PROF, NOM_PROF)
- **auto-jointure** : jointure non naturelle d'une table avec elle-même ; par exemple, EMPLOYE(MAT, NOM, MATDIR) $\bowtie_{G.MAT=D.MATDIR}$ EMPLOYE donne (MAT, NOM, MATDIR, MAT', NOM', MATDIR'). G signifie Gauche, D signifie Droite. Qu'est-ce qu'une auto-jointure naturelle ?
- **thèta-jointure** : opérateur θ quelconque {<, >, <=, >=,.<>} ; par exemple, NOTE(N_ET, VALEUR) $\bowtie_{G.VALEUR>D.VALEUR}$NOTE donne (N_ET,VALEUR,N_ET',VALEUR').
- Les jointures sont les opérations indispensables pour relier les informations de différentes tables. On utilisera souvent les CIR pour les jointures habituelles.

2.6.3 Exercices

Soit le schéma PERSONNEL suivant : EMPLOYE(MAT, NOM, MATSUP, SERV) avec MATSUP⊆MAT et SERV⊆SERVICE(NSERV) ; SERVICE(NSERV, LIBELLE, DIR) avec DIR⊆EMPLOYE(MAT). Ecrire les requêtes algébriques répondant aux questions suivantes :
1. Ensemble de tous les noms d'employés ?
2. Quel est le nom de l'employé de matricule 123 ?
3. Nom du directeur du service 43 ?
4. Services dans lesquels le directeur n'est pas employé dans celui-ci ?
5. Matricules des employés dont le supérieur hiérarchique est le directeur de leur service ?

2.6.4 Opérations complémentaires

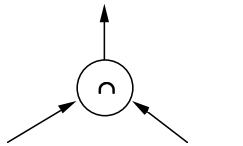
2.6.4.1 INTERSECTION

Elle peut être obtenue à partir de la différence ensembliste :R1 ∩ R2 = R1-(R1-R2) ou R2-(R2-R1).

Opération portant sur **deux relations** 1et 2 de **même schéma** consistant à réaliser une relation 3 de même schéma ayant pour n-uplets ceux appartenant à la fois à 1 et 2.

On peut la noter :

INTERSECTION(Relation 1,Relation 2) ou Relation 1 ∩ Relation 2

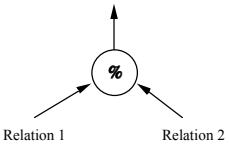


INTERSECTION

Exemple :
soit la relation PARENTE(PARENT,ENFANT) ; Π_{PARENT}(PARENTE)∩Π_{ENFANT}(PARENTE) donne les personnes ayant au moins un enfant et un parent dans l'instance.

2.6.4.2 QUOTIENT

La relation quotient d'une relation Relation 1 de schéma R1(A1,A2,...An) par une Relation2 de schéma R2(Ap+1,Ap+2,...An) est la relation R1 **%** R2 de schéma Q (A1,A2,...Ap) constituée des p-uplets t tels que pour tout (n-p)-uplet u de R2, le n-uplet tu est dans R1



QUOTIENT

Remarque :

R1(X,Y) **%** R2(Y) = R1[X] - ((R1[X] × R2 (Y)) - R1(X,Y)) [X]= $\cap_{v\in R2}$(R1 (X, Y) **:** (Y=v)) [X])

Exemple : soit la relation MODELE(NUM, COUL) et la relation COLORI(COULEUR). MODELE%COLORI donne les numéros de modèle qui existent dans tous les coloris.

2.6.4.3 ANTIPROJECTION

Opération portant sur une relation R(A1,A2) permettant d'obtenir les n-uplets dont la projection sur l'attribut A2 est associée à toutes les valeurs distinctes possibles des projections de la relation sur l'attribut A1. Elle est notée R(A1, A2)]A2[et définie par :

R(A1, A2)]A2[= R(A1, A2) % R [A2] = $\cap_{v \in R[A2]}$(R (A1,A2) : (A2=v)) [A1])

Exemple : MODELE(NUM, COUL)]COUL[donne les numéros de modèle qui existent dans toutes les couleurs existantes de modèle.

2.6.4.4 SEMI-JOINTURE

Opération portant sur deux relations R1, R2 notée R1 ⋉ R2 qui consiste à projeter sur les attributs de R1 le résultat de la jointure naturelle de R1 avec R2.

$$R1 \ltimes R2 = \Pi_{attributs(R1)} R1 \bowtie R2$$

Exemple : soit la relation MODELE(NUM, COULEUR) et la relation COLORI(COULEUR, METAL). MODELE⋉COLORI fournit les modèles ayant une couleur référencée dans COLORI.

2.6.4.5 SOMME et COMPLEMENT

La somme de deux relations R(X) + S(Y) et le complément d'une relation R(X) notée ¬R(X) sont deux opérateurs qui nécessitent la connaissance des domaines associés aux attributs.

Soit R(A1, A2, ..., An) avec Ai∈ Di alors ¬R=D1×D2×...×Dn - R

Soit S(B1, B2,..., Bp) avec Bi∈ Fi et A1=B1, ..., Ak=Bk alors :

$$R+S=R\times(F1\times F2\times...\times Fp)\cup\Pi_{1,2,...k,\ p+1,...,(p+n),k+1,...,p}(S\times(D1\times D2\times...\times Dn))$$

Exemple :
Soient les relations R(PIECE, FOURNISSEUR) et S(PIECE, PROJET). Les domaines associés aux divers attributs sont :
D1 pour PIECE = {écrou, boulon, vis}

D2 pour FOURNISSEUR = {pierre, paul, alice}
D3 pour PROJET ={a, b, c}

R

PIECE	FOURNISSEUR
écrou	pierre
écrou	paul
boulon	alice

¬ R

PIECE	FOURNISSEUR
écrou	alice
boulon	pierre
boulon	paul
vis	pierre
vis	paul
vis	alice

S

PIECE	PROJET
écrou	a
écrou	b
boulon	a

R +S

PIECE	FOURNISSEUR	PROJET
écrou	pierre	a
écrou	pierre	b
écrou	pierre	c
écrou	paul	a
écrou	paul	b
écrou	paul	c
écrou	alice	a
écrou	alice	b
boulon	alice	a
boulon	alice	b
boulon	pierre	a
boulon	paul	a
boulon	alice	c

2.6.5 Résultat

L'ensemble des opérateurs {union, différence, produit cartésien, sélection, projection} permet d'engendrer tous les autres opérateurs, mis à part la somme et le complément. Un langage possédant ces opérateurs ou pouvant les engendrer est dit **complet.**

2.7 Les langages tabulaires et graphiques

Les langages d'interrogation commercialisés s'inspirent des langages algébriques ou des langages prédicatifs mais introduisent des fonctionnalités supplémentaires (tri, regroupement, fonctions chaînes de caractère, de date, fonction d'agrégation). De plus, ils offrent souvent une interface utilisateur agréable. Enfin, ils intègrent généralement LMD et LDD dans un même langage.

2.7.1 Access

Access est un SGBD de Microsoft offrant une interface graphique intégrée. La plupart des commandes du LDD et du LMD s'effectuent par la saisie d'information dans des tableaux. L'exemple de la Figure 3 permet d'obtenir la liste suivante :

Nom	Prénom	TypeLieu.Nom	TypeTel	Numéro de téléphone
Meynard	Michel	travail	tél	04.67.41.85.40
Meynard	Michel	travail	fax	04.67.41.85.00

Tableau 1: résultat de la requête Access

Dans la partie haute, on place les tables ainsi que les liens de **jointure** entre tables. Dans la partie basse on indique les colonnes à afficher (**projection**) ainsi que celles sur lesquelles portent des critères de **sélection**. Les critères sur une même ligne sont connectés par des **et** logiques, tandis que des **ou** connectent les différentes lignes. Ainsi la requête « exemple » permet de connaître tous les numéros de téléphone du lieu de travail ainsi que tous les numéros de fax du domicile.

D'autres types de requêtes, ajout, suppression, mise-à-jour, sont concevables selon le même esprit.

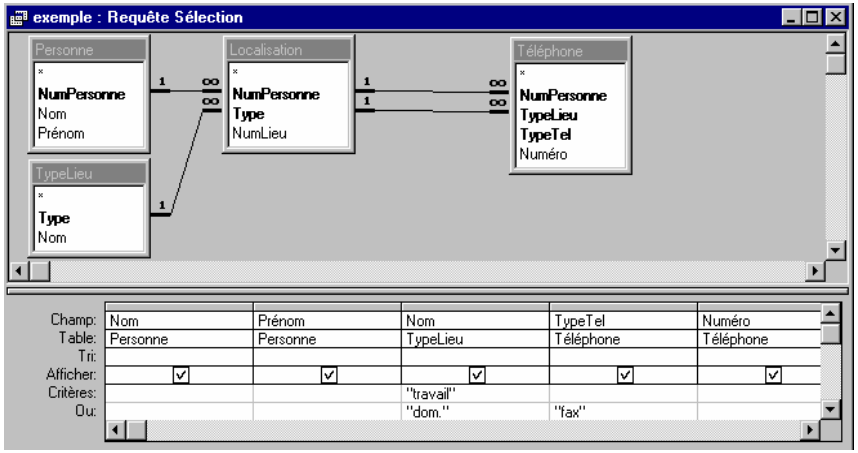
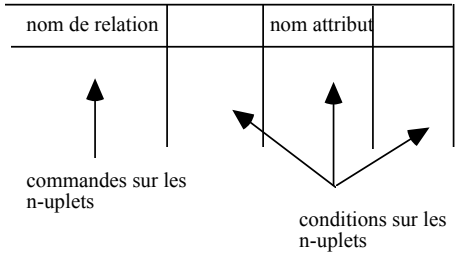


Figure 3: exemple de requête Access

2.7.2 QBE

Query By Example est un langage (LDD et LMD) tabulaire de type prédicatif à variables domaines développé par IBM. Il est associé à divers SGBD dont DB2. L'utilisateur dispose de "squelette" ou de "cadre" pour chaque relation concernée.

2.7.2.1 Langage d'interrogation



A partir de ces squelettes l'utilisateur décrit ce qu'il souhaite obtenir. Les requêtes sont posées en utilisant des variables « domaines » (_A dans l'exemple suivant), des constantes, des opérateurs booléens. Les colonnes contenant la commande P. (Print) seront imprimées (projection).

Exemple :

PILOTE	Numpl	Nompl	Adresse	Salaire
	_A	P.	Paris	>15000

VOL	Numvol	Numpl	Avion	..etc
		_A	P.	

correspond à la requête :

{Nompl Avion / ∃Salaire ∃Numvol ∃Numpl ∃Adresse (PILOTE (Numpl, Nompl, "Paris", Salaire) ∧ VOL(Numvol, Numpl, Avion, etc) ∧ (Salaire >15000))}

Dans les requêtes, l'utilisateur peut utiliser :

- des conditions complexes introduites dans une "boîte" condition ;
- les opérateurs agrégats SUM, AVG, MAX, MIN, CNT.

2.7.2.2 Langage de définition et de manipulation de données

Les requêtes relatives à l'insertion, la mise à jour et la suppression de n-uplets se construisent de manière analogue aux requêtes d'interrogation, mais on dispose dans la zone « commande sur les n-uplets » d'ordre correspondant I., U., D.

Les requêtes de définition de tables et de vues sont bâties sur le même principe.

Exemples :

- Création de la table PILOTE :

I. PILOTE I.	Numpl	Nompl	Adresse	Salaire
KEY I.	Y	N	N	N
TYPE I.	Char	Char	Char	Float
DOMAIN I.	Numeros	Noms	Adresses	Salaires

- Création de la vue PIL_PAR :

I. VIEW PIL_PAR I.	Nompl	Salaire
	_A	_B

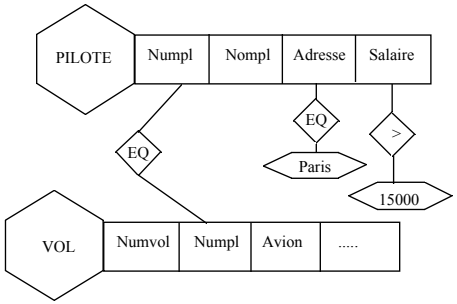
PILOTE	Numpl	Nompl	Adresse	Salaire
		_A	Paris	_B

2.7.3 CUPID

Casual User Pictorial Interface Design est un langage graphique associé au SGBD Ingres.

L'usager dispose de figures géométriques répertoriées dans un menu : un hexagone symbolise une relation, un rectangle un attribut, un losange un opérateur..

Exemple :



Le développement des « Interfaces Homme-Machine » donne naissance à divers langages d'interrogation graphiques ou textuel qui tendent à simplifier la conception des requêtes pour les utilisateurs finaux.

3. SQL

3.1 Introduction

Le langage SQL (Structured Query Language) est un dérivé de SEQUEL, lui-même dérivé de SQUARE, langage d'interrogation associé au prototype System R de chez IBM.

SQL est utilisé dans de nombreux SGBD commercialisés :

- DB2 (IBM) (gros ordinateurs sous systèmes d'exploitation MVS, VM/CMS) ;
- PostgreSQL (Linux), Informix ;
- ORACLE (gros ordinateurs et micro-ordinateurs) ;
- msql, mysql ;
- Access, SQL Server, SQL anywhere (micro-ordinateurs).

SQL a un caractère **prédicatif** ou assertionnel lié à la clause WHERE. Cependant il a également une tendance impérative (algébrique) parce qu'il autorise l'imbrication des SELECT. Cependant, la plupart des optimiseurs de requêtes savent désimbriquer les requêtes, notamment dans Oracle.

En SQL :

- TABLE désigne une collection de lignes (ROW) ou n-uplets. Attention, généralisation de la notion de relation car plusieurs lignes peuvent être identiques. On parle alors de doublons (voir options ALL/DISTINCT) ;
- COLUMN désigne un attribut ;
- ROW désigne un n-uplet ;
- VIEW désigne une vue, c'est-à-dire le résultat d'une requête. Ce résultat est évalué **dynamiquement** au moment où on pose une requête sur cette vue ;
- SCHEMA désigne un ensemble de tables et de vues ;
- les minuscules et les majuscules ne sont **pas différenciées** pour les mots-clés, ni pour les identificateurs ;
- le caractère « ; » est le terminateur de commande ;

3.2 Langages formels

Un langage formel, SQL, Java, C, Pascal, est décrit à l'aide d'une **grammaire**. Une grammaire permet d'indiquer toutes les constructions **syntaxiques** valides du langage. Une **règle** de grammaire est composé d'une partie gauche décrivant un symbole **non terminal**, d'un séparateur **::=**, et d'une partie droite constituée d'une séquence de symboles. Par exemple :

type ::= { CHAR(longueur) | VARCHAR(longueur) | DATE }

On définit ici le symbole non terminal type comme pouvant prendre la valeur CHAR(35) ou DATE ou VARCHAR(10). Les conventions d'écriture suivantes sont adoptées :

- les mots écrits en **minuscules** sont des symboles non terminaux (type, longueur) ;
- les mots écrits en **majuscules** sont des symboles terminaux (CHAR, (, DATE) ;
- { mot1 | mot2 | mot3 } indique un choix exclusif entre l'un des 3 mots ;
- { mot1 mot2 } ... : le parenthésage peut être utilisé pour forcer les priorités
- [mot] indique le caractère optionnel de mot ;
- mot ... indique la répétition possible 0 à n fois de mot.

Un exemple de règle :

exp ::= { exp + exp | exp * exp | exp - exp | exp / exp | chiffre chiffre ... }

Exercice :

Montrer que l'expression 34+65*2 est correcte.

3.3 Langage d'interrogation : la commande SELECT

Le Langage de Manipulation de Données ou LMD peut être décomposé en deux parties : le langage d'interrogation (ou de consultation) et le langage de modification. En SQL, la commande SELECT appelée consultation ou question ou interrogation (query) est fondamentale. Elle n'a pas de rapport sémantique direct avec l'opération algébrique de sélection (σ).

3.3.1 Syntaxe

```
selectcommand ::=
SELECT [DISTINCT | ALL]
  { *
  | { [schema.]{table | view | snapshot}.* | expr [c_alias]
      [, { [schema.]{table | view | snapshot}.* | expr [c_alias] } ] ...
    }
  }
FROM [schema.]{table | view | snapshot} [t_alias]
     [, [schema.]{table | view | snapshot} [t_alias] ] ...
[WHERE condition ]
[ [START WITH condition] CONNECT BY condition]
[GROUP BY expr [, expr] ... [HAVING condition] ]
[{UNION | UNION ALL | INTERSECT | MINUS} selectcommand ]
[ORDER BY {expr|position} [ASC | DESC] [, {expr | position} [ASC | DESC]] ...]
[FOR UPDATE [OF [[schema.]{table | view}.]column
                [, [[schema.]{table | view}.]column] ...] [NOWAIT] ]
;
```

3.3.2 Sémantique

La commande SELECT comporte un certain nombre de **clauses**, c'est-à-dire des parties de commandes débutant par un mot-clé.

3.3.2.1 clause SELECT

- DISTINCT : suppression des doublons ;
- ALL : non suppression des doublons (valeur par défaut) ;
- * : toutes les colonnes de toutes les tables, vues, clichés ;
- schema : nom du schéma contenant les tables, vues ou clichés (par défaut, celui de l'utilisateur qui exécute la requête) ;
- table.*, view.*, snapshot.* : toutes les colonnes de la table, de la vue ou du cliché précisé ;
- exp : expression sur les colonnes utilisant des opérateurs et des fonctions ;
- c_alias : alias de colonnes sous la forme : **AS** `nouveauNom` ;

3.3.2.2 clause FROM

- table, view, snapshot : nom des tables, vues et/ou clichés contenant les colonnes ;
- t_alias : alias de table, vue, cliché ;

Exemples :

- Visualisation de toutes les lignes et colonnes : **SELECT * FROM article ;**
- Visualisation de toutes les lignes avec quasi-projection : **SELECT ref, prix FROM article ;** Ceci n'est pas une projection dans la mesure où on peut obtenir des n-uplets identiques ! Projection : **SELECT DISTINCT ref, prix*1.196 AS prix_ttc FROM article ;**
- Produit cartésien : **SELECT DISTINCT A.ref, C.n_com FROM article A, commande C ;**
- Expression complexe : **SELECT TO_CHAR(date_embauche, 'DD/MM/YY HH24:MI:SS') AS Debut FROM emp;**

3.3.2.3 clause WHERE

Cette clause n'est pas obligatoire mais est souvent la plus utile !

3.3.2.3.1 Opérateurs de comparaison

Les conditions de sélection et de jointure peuvent utiliser les opérateurs de comparaison : <, <=, =, >, >=, <> ou !=.

Exemples :

- **SELECT * FROM article WHERE prix<150 ;**
- **SELECT etudiant.nom, note.valeur FROM etudiant E,note N WHERE N.n_et=E.n_et;**
- **SELECT A.pere, B.fils FROM parente A, parente B WHERE A.fils=B.pere ;** A et B sont des variables lignes ou **alias** nécessaires en cas d'auto-jointure, et souvent utiles pour abréger la requête ;

3.3.2.3.2 Opérateurs logiques

Les conditions de sélection et de jointure peuvent utiliser les opérateurs logiques **NOT**, **AND**, **OR**.

Exemples :
- **SELECT ref FROM article WHERE prix>=200 AND design='chemise' ;**
- **SELECT * FROM COMMANDE, LIGNE_COM WHERE CLIENT='Dupont' AND COMMANDE.N_COM=LIGNE_COM.N_COM ;** lignes de commande du client Dupont (jointure naturelle) : le nom de la colonne pour être identifié est préfixé par le nom de sa table suivi d'un point. Il est possible d'effectuer des « auto-jointures » :

3.3.2.3.3 Autres opérateurs

On peut également utiliser :

1. des prédicats :
   - BETWEEN teste l'appartenance d'une valeur à un intervalle donné (intervalle fermé). Par exemple, **salaire BETWEEN 9000 AND 15000**
   - IN teste l'appartenance d'une valeur à une liste donnée : **A [NOT] IN (S)**. Par exemple : **lieu IN ('Versailles','Paris','Montpellier','Vierzon')**
   - LIKE recherche toute valeur d'une colonne de type chaîne de caractères contenant une chaîne de caractères donnée : **A LIKE modèle**. Par exemple : **lieu LIKE '__ri_'**. Le '_' représente un et un seul caractère tandis que '%' représente toute chaîne de caractères, la chaîne vide comprise : **lieu LIKE '%er%'**
   - EXISTS teste la non vacuité d'un ensemble ou liste : **EXISTS S**. Utile quand S est une sous-requête.
2. des comparaisons quantifiees : les opérateurs (**=, <, >, <=, >=, <>** représentés par Θ) peuvent être quantifiés. Soit S une expression dénotant un ensemble :
   - **A Θ ANY (S)** signifie (∃x S(x)Λ AΘx)
   - **A Θ ALL (S)** signifie(∀x S(x)Λ AΘx)

   Utile quand S est une sous-requête.

3.3.2.3.4 Sous-requête ou requête imbriquée

la condition exprimée dans la clause WHERE peut porter sur une sous-requête ou requête imbriquée; celle-ci doit :
- être **enfermée dans des parenthèses** et
- n'avoir **qu'une seule** colonne ou expression dans sa clause SELECT (excepté avec le prédicat EXISTS) et
- retourner une **et une seule ligne** excepté quand elle est précédée de IN, ALL, ou ANY.

Exemples :
- **SELECT DISTINCT ncom FROM lignecom WHERE ref IN (SELECT ref FROM article WHERE design='chemise');**

Cette requête imbriquée est équivalente à une autre requête non imbriquée. Laquelle ? L'exécution d'une requête imbriquée est plus longue que celle d'une requête non imbriquée équivalente. Cependant certains SGBD (Oracle) possèdent un optimiseur de requêtes supprimant les imbrications inutiles.
- **SELECT DISTINCT ncom FROM lignecom L WHERE NOT EXISTS (SELECT * FROM article A WHERE A.ref=L.ref);**

Quelle est la sémantique de la requête précédente ?

3.3.2.4 clause GROUP BY

3.3.2.4.1 Fonctions d'agrégat

Ces fonctions permettent d'agréger les valeurs des colonnes de toutes les lignes sélectionnées :
- MIN([DISTINCT | ALL] expression), MAX([DISTINCT | ALL] expression) : minimum, maximum ;
- COUNT(* | [DISTINCT | ALL] expression) : nombre de lignes dont l'expression n'est pas NULL ;
- SUM([DISTINCT | ALL] expression) : somme ;
- AVG([DISTINCT | ALL] expression), STDDEV([DISTINCT | ALL] expression), VARIANCE([DISTINCT | ALL] expression) : moyenne, écart-type, variance ;

Exemples :
- SELECT COUNT(DISTINCT *) FROM article ;
- SELECT AVG(prix) AS prix_moyen FROM article ;
- SELECT MAX(note) AS meilleure_note FROM examen ;

3.3.2.4.2 Groupement

Il est souvent utile d'agréger des résultats non pas pour toute la table mais par **groupes** de lignes : rechercher la note maximale de chaque étudiant ! En cas de regroupement, toutes les expressions de la clause SELECT doivent être soit un agrégat, soit un groupe.

Exemples :
- SELECT N_ET, MAX(note) AS meilleure_note FROM examen GROUP BY N_ET ;
- SELECT N_ET, MATIERE, MIN(note) AS min, AVG(note) AS moy, MAX(note) AS max FROM examen GROUP BY N_ET, MATIERE ;

3.3.2.4.3 Clause HAVING

On peut également regrouper conditionnellement grâce à la clause HAVING. Les expressions de cette clause doivent absolument être des expressions de regroupement.

Exemples :
- SELECT design, MAX(prix) AS prix_max FROM article WHERE design LIKE '%ch%' GROUP BY design HAVING prix_max<200 ; catégories d'articles contenant « ch » n'ayant aucun exemplaire de prix supérieur ou égal à 200.
- SELECT N_ET, AVG(note) FROM examen WHERE matiere='BD' GROUP BY N_ET HAVING AVG(note)<10 ; les recalés de BD !

Comme la clause WHERE, une clause HAVING peut comporter une sous-requête.

3.3.2.5 clause UNION, INTERSECT ou MINUS

Opérations ensemblistes entre plusieurs commandes SELECT. Par exemple : SELECT col1 FROM tab1 **UNION** SELECT col2 FROM tab2 ; Les informations résultantes de chaque requête doivent être du même type.

3.3.2.6 clause ORDER BY

Tri ascendant ou descendant selon certaines colonnes. Les colonnes utilisées dans la clause ORDER BY doivent faire partie de la clause SELECT.

3.3.2.7 clause START WITH

Utilisée dans les tables représentant une hiérarchie (père, fils). Non standard.

3.3.2.8 clause FOR UPDATE

Verrouille les lignes sélectionnées afin d'éviter leur consultation/modification avant leur mise à jour. Non standard.

3.3.3 Expressions

Les expressions SQL sont assez peu standardisées !

3.3.3.1 Généralités

Une expression SQL peut contenir :
- des noms de **colonnes** (PRIX, SALAIRE, ...),
- des littéraux **constants** (123, 345.78, 'Dupont', #12/31/1999#, ...),
- des **fonctions** portant sur une liste d'expressions,
- combinés au moyen d'**opérateurs** et de **parenthèses**.

3.3.3.2 Opérateurs arithmétiques

Opérateurs : +, -, *, /, - unaire. Les opérateurs de multiplication et de division sont prioritaires par rapport aux opérateurs d'addition et de soustraction. Des parenthèses peuvent être utilisées pour forcer l'évaluation de l'expression dans un ordre différent de celui découlant de la priorité des opérateurs.

3.3.3.3 Fonctions arithmétiques

- ABS(nb) : valeur absolue ;
- ROUND(n[,m]), TRUNC(n[,m]), CEIL(n), FLOOR(n) : arrondi à m chiffres après la virgule, troncature, partie entière supérieure, inférieure ;

- SIN(n), COS(n), TAN(n)
- EXP(n), POWER(m,n), LN(n), LOG(m,n) : exponentielle, puissance, logarithme, log à base m
- MOD(m,n) : modulo
- SIGN(nb) : renvoie -1 si nb est négatif, 0 si nb est nul, 1 si nb est positif.
- SQRT(n) : racine carrée

3.3.3.4 Opérateur et fonctions de chaînes

- Opérateur de **concaténation** : ||
- chaîne CONCAT(chaîne1,chaîne2) : concaténation ;
- chaîne SUBSTR(chaîne,m[,n]) : partie commençant au caractère m et ayant une longueur de n ;
- chaîne INITCAP(chaîne) : première lettre de chaque mot en majuscule et toutes les autres en minuscule ;
- chaîne LOWER(chaîne), UPPER(chaîne) : toutes lettres en minuscules, majuscules ;
- chaîne LPAD(chaîne,long,[char]), RPAD(chaîne,n,[char]) : complète ou tronque chaîne pour qu'elle ait comme longueur « long » en ajoutant éventuellement à gauche (droite) le caractère (ou la chaîne de caractères) char (par défaut, un espace) ;
- chaîne LTRIM(chaîne[,ens]), RTRIM(chaîne[,ens]) : supprime de chaîne tous les caractères qui sont dans ens à gauche (à droite). Arrêt : premier caractère qui n'est pas dans ens (par défaut, ens = espace) ;
- chaîne REPLACE(chaine, avant, apres) : remplacements ;
- chaîne TRANSLATE(chaîne, avant, apres) : remplace chaque caractère de chaîne présent dans avant par le caractère situé à la même position dans apres. Si avant contient plus de caractères que apres, suppression des caractères correspondants ;
- int INSTR(chaîne, sous-chaîne, debut, occ) : position du premier caractère de chaîne correspondant à l'occurence occ de sous-chaîne en commençant la recherche à la position début ;
- int LENGTH(chaîne) : longueur de chaîne ;

3.3.3.5 Opérateurs et fonctions sur les dates

Opérateurs de date :

- date +/- nombre : est une date obtenue en ajoutant le nombre de jours nombre à la date date ;
- date2 - date1 : est le nombre de jours entre les deux dates ;

Fonctions de dates :

- SYSDATE : date et l'heure courantes du système d'exploitation hôte ;
- DATE NEXT_DAY(date, nom_du_jour) : date du prochain jour de la semaine dont le nom est nom_de_jour ;
- DATE LAST_DAY(d) : date du dernier jour du mois de d ;
- DATE ADD_MONTHS(d, n) : ajoute n mois à date. Si le mois obtenu a moins de jours que le jour de d, le jour obtenu est le dernier du mois.
- number MONTHS_BETWEEN(date2, date1) : nombre de mois entre date2 et date1, si date2 est après date1 le résultat est positif, sinon le résultat est négatif. Si les jours date2 et date1 sont les mêmes, ou si ce sont les derniers jours du mois, le résultat est un entier. La partie fractionnaire est calculée en considérant chaque jour comme 1/31ème de mois
- DATE ROUND(date[,précision]) : d arrondie à l'unité spécifiée dans précision. L'unité de précision est indiquée en utilisant un des masques de mise en forme de la date. On peut ainsi arrondir une date à l'année, au mois, à la minute,... Par défaut la précision est le jour ;
- DATE TRUNC(d[,précision]) : d tronquée à l'unité spécifiée dans précision. Les paramètres sont analogues à ceux de la fonction ROUND.

3.3.3.6 Fonctions de conversion

- ASCII(chaine) : nombre correspondant au code ascii du premier caractère de chaine ;
- CHR(nombre) : caractère dont nombre est le code ascii ;
- TO_NUMBER(chaîne) : convertit chaîne en sa valeur numérique ;
- TO_CHAR(nombre,format) : conversion de nombre chaîne en fonction de la chaîne format :
  - 9 représente un chiffre (non représenté si non significatif) ;
  - 0 représente un chiffre (représenté même si non significatif) ;
  - . : point décimal apparent ;
  - V : position du point décimal non apparent ;
  - , : une virgule apparaîtra à cet endroit ;
  - $: un $ précédera le premier chiffre significatif ;
  - B : le nombre sera représenté par des blancs s'il vaut 0 ;

  - EEEE : le nombre sera représenté avec un exposant (le spécifier avant MI ou PR) ;
  - MI : le signe négatif sera à droite ;
  - PR : un nombre négatif sera entre <> ;
- TO_CHAR(date,format) : conversion d'une date en chaîne de caractères en fonction de la chaîne format. Exemple : TO_CHAR(#12/31/2001#,'yy') donne '01'
  - scc : siècle avec signe ;
  - cc :siècle ;
  - sy,yyy : année (avec signe et virgule) ;
  - y,yyy : année(avec virgule) ;
  - yyyy :année ;
  - yyy : 3 derniers chiffres de l'année
  - yy : 2 derniers chiffres de l'année
  - y : dernier chiffre de l'année
  - q : numéro du trimestre dans l'année
  - ww : numéro de la semaine dans l'année
  - w : numéro de la semaine dans le mois
  - 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
  - ss : secondes
  - sssss : secondes après minuit
  - j : jour du calendrier julien

  Dates en lettres (en anglais) :

  - syear ou year : année en toutes lettres
  - month : nom du mois
  - mon :nom du mois abrégé sur 3 lettres
  - day : nom du jour
  - dy : nom du jour abrégé sur 3 lettres
  - am ou pm : indication am ou pm
  - bc ou ad : indication avant ou après Jésus Christ

  Tout caractère spécial inséré dans le format sera reproduit tel quel dans la chaîne de caractères résultat.
- TO_DATE(chaîne, format) : convertit une chaîne en date. Le format est identique à celui de la fonction TO_CHAR ; TO_DATE('31-12-01','dd-mm-yy') donne la date #12/31/2001#

3.3.3.7 Autres fonctions

- GREATEST(expr1, expr2,...), LEAST(expr1, expr2,...) : valeur maximale, minimale ;
- NVL(expr1, expr2) : expr1 si non nulle, sinon expr2
- DECODE(crit, val1, res1 [, val2, res2 ...], def) : choisit une valeur parmi une liste d'expressions, en fonction de la valeur de crit (case crit of val1 :res1 ; val2 : res2 ; ...)
- bool IS [NOT] NULL(expr) est vrai si l'expression expr est NULL, faux sinon

3.4 Remarques sur le SELECT

Certains SGBD ne fournissent pas toutes les possibilités de la commande SELECT décrites ci-dessus. Par exemple, l'opération MINUS de différence ensembliste n'existe pas en ACCESS, pas plus que l'intersection ! De plus, les opérations de regroupement peuvent fournir un moyen efficace de réaliser certaines opérations telles que le quotient. On peut également utiliser des opérations particulières telles que les jointures à gauche.

3.4.1 MINUS

L'opération de jointure externe, ou jointure gauche, correspond à une jointure dans laquelle toutes les lignes de la table de gauche sont sélectionnées dans le résultat même si elles n'ont pas de correspondance dans la table de droite. Elle est très utile pour les participations optionnelles. Sa syntaxe est (parfois) :

```
SELECT ... FROM A LEFT JOIN B ON condition
```

Toutes les lignes de la table A seront alors présentes. Par exemple :

```
SELECT E.nom, N.valeur FROM etud E LEFT JOIN note N ON E.numet=N.numet ;
```

Les absents à l'examen seront tout de même représentés avec une note NULL.

Soient 2 tables A(x,y) et B(x,y), la différence ensembliste A-B peut être obtenue de la manière suivante :

```
SELECT DISTINCT A.x, A.y FROM A LEFT JOIN B ON A.x=B.x AND A.Y=B.y WHERE B.x IS
NULL ;
```

3.4.2 INTERSECT

De la même manière, si l'intersection n'existe pas dans un dialecte SQL, on pourra simuler A∩B par :

```
SELECT DISTINCT A.x, A.y FROM A, B WHERE A.x=B.x AND A.y=B.y ;
```

3.4.3 Quotient

L'opération algébrique de quotient peut être obtenue par une suite d'opérations algébriques : produit cartésien, projection, différence. En SQL, on peut la réaliser efficacement de la manière suivante. Soient 2 tables A(x,y,z,t) et B(z,t), A%B est obtenu par :

```
SELECT x,y FROM A,B WHERE A.z=B.z AND A.t=B.t GROUP BY x,y HAVING
COUNT(*)=(SELECT COUNT(*) FROM B) ;
```

3.5 Langage de modification de données

3.5.1 Ajout de lignes

```
insertcmd ::=
INSERT INTO [schema.]table | view
     [ (column [, column] ...) ]
    VALUES (expr [, expr] ...) | subquery
```

- table, view : table ou vue dans laquelle les lignes seront insérées. Si c'est un nom de vue qui est précisé, les données seront insérées dans la table basée sur la vue ;
- column : les autres colonnes non mentionnées seront alors remplies de NULL ;
- VALUES : valeurs en extension de chaque colonne ;
- subquery : commande SELECT ;

Exemples :

- INSERT INTO article VALUES (123, 'chemise', 134) ;
- INSERT INTO client_2000 SELECT DISTINCT * FROM client_99 WHERE solde>=0 ;

3.5.2 Suppression de lignes

```
deletecmd ::=
DELETE [FROM] [schema.] {table | view} [alias]
     [WHERE condition]
```

- table, view : table ou vue dans laquelle les lignes seront supprimées. Si c'est un nom de vue qui est précisé, les données seront supprimées dans la table basée sur la vue ;
- alias : alias de table ;
- where : condition de destruction de la ligne ; sans clause where, suppression de toutes les lignes !

Exemple :

- DELETE FROM article WHERE ref=123 ;

3.5.3 Mise à jour de lignes

```
updatecmd ::=
UPDATE [schema.] {table | view} [alias]
     SET  (column [, column] ...) = (subquery)
        |  column =  expr | (subquery)
     [,     (column [, column] ...) = (subquery)
        |  column =  expr | (subquery)   ] ...
    [WHERE condition]
```


- column : nom de la colonne qui sera modifiée ;
- expr : nouvelle valeur de la colonne ;
- subquery : SELECT qui renvoie les nouvelles valeurs affectées aux colonnes correspondantes ;
- WHERE : restreint les lignes modifiées à celles pour lesquelles la condition est vraie. Si on omet cette clause toutes les lignes sont modifiées ;

Exemple :

- UPDATE article SET PRIX=PRIX*1.1 WHERE ref>123 ;
- UPDATE examen SET note=10 WHERE note>=9.5 ;

3.6 Langage de définition de données

Le langage de définition de données permet de déclarer tous les objets décrivant la structure (c-à-d le schéma) de la base : tables, attributs, index...

Il existe différentes variantes syntaxiques au niveau de la définition des données, nous utiliserons celle du SQL ORACLE.

3.6.1 Définition de table

Définir une table c'est déclarer son nom et sa structure. La définition de chaque attribut consiste à donner son nom, son type, sa taille et éventuellement des contraintes d'intégrité (par exemple, valeur NULL).

```
createtablecmd ::=
CREATE TABLE [schema.]table
     (  column datatype [DEFAULT expr] [column_constraint] ...
      | table_constraint
    [,  column datatype [DEFAULT expr] [column_constraint] ...
      | table_constraint ]...)

    [ [PCTFREE  integer] [PCTUSED  integer]
      [INITRANS integer] [MAXTRANS integer]
      [TABLESPACE tablespace]
      [STORAGE storage_clause]
    | [CLUSTER cluster (column [, column]...)] ]
    [ ENABLE   enable_clause
    | DISABLE disable_clause ] ...
    [AS subquery]
```

- column : nom d'une colonne de la table. Le nombre de colonnes possibles dans une table est compris entre 1 et 254 ;

- datatype : type de la colonne ;
- DEFAULT : valeur qui sera affectée à cette colonne si, lors d'un INSERT, on ne lui en précise pas ;
- column_constraint : contrainte d'intégrité qui fait partie de la définition de la colonne ;
- table_constraint : contrainte d'intégrité qui fait partie de la définition de la table ;
- TABLESPACE : la tablespace dans laquelle la table sera crée.
- CLUSTER : pour les tables faisant partie d'un cluster ;
- ENABLE : active une contrainte d'intégrité (défaut) ;
- DISABLE : désactive une contrainte d'intégrité ;
- AS subquery : insère les lignes renvoyées par la sous requête dans la table, à sa création.

3.6.1.1 Types de données

Les types de données admis par ORACLE sont :

- NUMBER[(longueur,[précision]) : données numériques flottantes dont la valeur est comprise entre 10^-130 et 10^125 avec une précision de 38 chiffres ;
  - longueur :précise le nombre maximum de chiffres significatifs stockés (par défaut 38),
  - précision : donne le nombre maximum de chiffres après la virgule (par défaut 38), sa valeur peut être comprise entre -84 et 127. Une valeur négative signifie que le nombre est arrondi à gauche de la virgule.
- CHAR(longueur) : chaînes de caractères de longueur fixe. Longueur doit être inférieur à 255, sa valeur par défaut est 1 ;
- VARCHAR(longueur) : chaînes de caractères de longueur variable. Longueur doit être inférieur à 2000, il n'y a pas de valeur par défaut ;

- DATE : date et heure ;
- RAW(longueur) : caractères non imprimables (octets qcq) ;
- LONG : chaînes de caractères de longueur variable et inférieure à 2^31 -1. Les colonnes de ce type sont soumises à certaines restrictions ;
  - une table ne peut pas contenir plus d'une colonne de ce type ;
  - les colonnes de ce type ne peuvent pas apparaître dans des contraintes d'intégrité ;
  - les colonnes de ce type ne peuvent pas être indexées ;
  - les colonnes de ce type ne peuvent pas apparaître dans des clauses : WHERE, GROUP BY, ORDER BY ou CONNECT BY ainsi que dans un DISTINCT.

Exemple :

```
CREATE TABLE PILOTE (
    numpl NUMBER (6) NOT NULL,
    nompl CHAR (20),
    salaire NUMBER (8),
    adresse CHAR (80)
    ) ;
```

Remarque :

Si aucune zone (SPACE) n'est spécifiée, la création a lieu dans la zone privée de l'utilisateur. La table créée est vide. Il est possible de créer une table non vide à partir d'autres tables existant dans la base.

La requête définie dans la clause AS détermine les n-uplets et éventuellement les attributs de la table créée. Le type et la taille des attributs seront hérités des attributs cités dans la requête. Le nombre d'attributs spécifiés explicitement doit être égal au nombre d'attributs cités dans la requête.

Exemple :

```
CREATE TABLE pilote-province AS
    SELECT numpl, nompl
    FROM PILOTE
    WHERE NOT adresse = 'Paris' ;
```

3.6.1.2 Contraintes d'intégrité

```
column_constraint ::=
[CONSTRAINT constraint]
{ [NOT] NULL
| {UNIQUE | PRIMARY KEY}
| REFERENCES [schema.]table [(column)]
        [ON DELETE CASCADE]
| CHECK (condition)
}
{ [USING INDEX [PCTFREE integer]
                [INITRANS integer] [MAXTRANS integer]
                [TABLESPACE tablespace]
                [STORAGE storage_clause] ]
  [EXCEPTIONS INTO [schema.]table]
  [{ENABLE | DISABLE}]
}
```

```
table_constraint ::=
{
[CONSTRAINT constraint]
{ {UNIQUE | PRIMARY KEY} (column [,column] ...)
| FOREIGN KEY (column [,column] ...)
         REFERENCES [schema.]table [(column [,column] ...)]
         [ON DELETE CASCADE]
| CHECK (condition)
}
{ [USING INDEX [PCTFREE integer]
                [INITRANS integer] [MAXTRANS integer]
                [TABLESPACE tablespace]
                [STORAGE storage_clause] ]
  [EXCEPTIONS INTO [schema.]table]
  [{ENABLE | DISABLE}]
}
```


- constraint : nom de la contrainte ; si absent, c'est le SGBD qui la nomme ! Mieux vaut nommer avec des règles de nommage : table_pk, table1_table2_fk, ...
- NULL, NOT NULL : la colonne peut contenir ou pas des valeurs NULL ;
- UNIQUE : identifiant ou NULL ;
- PRIMARY KEY : identifiant ;
- FOREIGN KEY (column ...) : liste des colonnes locales référençant des colonnes extérieures ; les colonnes locales sont appelées « clé étrangère » bien qu'elles ne soient absolument pas identifiantes la plupart du temps !
- REFERENCES table(column ...) : liste des colonnes extérieures devant être soit **PRIMARY KEY**, soit **UNIQUE** ;
- ON DELETE CASCADE : en cas de suppression de la valeur extérieure, suppression des références à cette valeur !
- CHECK(condition) : contrainte d'intégrité de **contrôle** de la valeur : par exemple, CHECK(mois>0 AND mois<13), ou CHECK(salaire > prime) ;
- EXCEPTIONS INTO table : en cas d'ajout de lignes ne respectant pas le contrôle, cette ligne sera ajoutée à table ;
- DISABLE : désactive une contrainte (par défaut activée ENABLE) ;

3.6.1.2.1 Suppression de contrainte

```
DROP CONSTRAINT nom_contrainte ;
```

3.6.2 Suppression de table

```
DROP TABLE nom_table ;
```

A ne pas confondre avec DELETE !

3.6.3 Vue

Une vue est définie par une requête SELECT. Cette requête est effectuée à chaque fois que la vue est utilisée dans une autre requête SELECT. Le contenu de la vue est donc **dynamiquement** constitué, contrairement à une création de table. Une vue permet également d'agir sur les tables sur lesquelles elle repose. Pour cela, il faut bien entendu que la clé primaire de la table à modifier fasse partie de la vue.

```
createviewcmd ::=
CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW [schema.]view
    [(alias [,alias]...)]
    AS subquery
```

- OR REPLACE : remplace la vue si elle existe déjà ;
- FORCE : sans s'inquiéter de l'existence de la table et des privilèges sur celle-ci ;
- NOFORCE : uniquement si la table existe et si les privilèges adéquats existent (défaut) ;
- view : nom de la vue créée ;
- alias : nom des colonnes de la vue ;
- AS subquery : requête ;

Exemple :

CREATE VIEW nom_note AS SELECT nom, valeur FROM etudiant, note WHERE etudiant.n_et=note.n_et ;

Une vue est supprimée par :

```
DROP VIEW nom_vue ;
```

Remarques sur les vues

Lorsque les vues n'existent pas dans le SGBD, on peut les simuler en créant des tables temporaires (create table tempo as select ...) et en les détruisant après utilisation. Risque lié au contrôle de concurrence !

3.6.4 Index

Les index ou accélérateurs d'accès aux données sont définis par :

```
createindexcmd ::=
CREATE [UNIQUE] INDEX nom_index
ON nom_table (nom_col1 , nom_col2, ...)
[PCTFREE nombre]
[COMPRESS | NOCOMPRESS]
```

```
[ROWS = nombre_lignes] ;
```

- UNIQUE : on interdit que deux lignes aient la même valeur dans la colonne indexée (identifiant) ;
- PCTFREE : pourcentage de place laissée libre dans les blocs d'index à la création de l'index. Cette place libre évitera une réorganisation de l'index des les premières insertions de nouvelles clés. La valeur par défaut est 20% ;
- NOCOMPRESS indique que l'on ne veut pas comprimer les clés ;
- nombre_lignes : estimation du nombre de lignes, permettant d'optimiser l'algorithme de classement..

Un index peut être créé dynamiquement sur une table contenant déjà des lignes. Il sera ensuite tenu à jour automatiquement lors des modifications de la table. Un index peut porter sur plusieurs colonnes, la clé d'accès sera alors la concaténation des différentes colonnes. On peut créer plusieurs index indépendants sur une même table.

Les requêtes SQL sont transparentes au fait qu'il existe un index ou non. C'est l'optimiseur du système de gestion de bases de données qui, au moment de l'exécution de chaque requête, recherche s'il peut s'aider ou non d'un index. S'il n'existe pas d'index, le SGBD peut générer un index temporaire afin accélérer l'exécution de la requête.

Exemples :

- CREATE UNIQUE INDEX article_xpk ON article (ref) ;
- CREATE INDEX etudiant_xnom ON etudiant (nom, prenom) ;

Un index est supprimé par :

```
DROP INDEX nom_index;
```

3.6.5 Définition de synonymes

```
CREATE [PUBLIC] SYNONYM synonyme FOR [schema.] {table|vue} ;
```

Nommer autrement une table ou une vue est possible. Ceci est important pour le partage d'une BD par plusieurs utilisateurs :

```
CREATE PUBLIC SYNONYM operation FOR comptabilite.operation ;
```

Ainsi tous les utilisateurs pourront accéder à la table des opérations comptables comme si celle-ci étaient dans leur propre schéma (de nom égal à leur nom d'utilisateur).

3.6.6 Modification de la structure d'une table

Modifier une table revient à :

- ajouter un ou plusieurs attributs ;
- ajouter une ou plusieurs contraintes ;
- modifier le type d'un attribut ou une contrainte ;
- activer/désactiver une contrainte ;
- modifier les clauses de stockage de la table.

La suppression d'un attribut n'est pas exprimable directement en SQL; elle peut-être réalisée d'une façon indirecte par la création d'une nouvelle table en omettant de recopier l'attribut.

```
altertablecmd ::=
ALTER TABLE [schema.]table
{  ADD      column datatype [DEFAULT expr] [column_constraint] ...
              | table_constraint
         |  (  column datatype [DEFAULT expr] [column_constraint] ...
              | table_constraint
           [,  column datatype [DEFAULT expr] [column_constraint] ...
              | table_constraint ] ...
            )
|  [MODIFY    column [datatype] [DEFAULT expr]
         |  ( column [datatype] [DEFAULT expr]
            [, column datatype [DEFAULT expr]] ...
            )
|  PCTFREE  integer | PCTUSED  integer
|  INITRANS integer] | MAXTRANS integer
|  STORAGE storage_clause
|  DROP drop_clause ...
|  ALLOCATE EXTENT [( [SIZE integer [K|M] ]
                     [DATAFILE 'filename']
```



```
                     [INSTANCE integer]
                    )]
|  ENABLE  enable_clause | DISABLE disable_clause
```

- ADD :ajoute une colonne ou une contrainte d'intégrité ;
- MODIFY : modifie une colonne ou une contrainte
- ENABLE, DISABLE : active, désactive une contrainte d'intégrité ou un trigger associé à la table.

Exemples :

- Par exemple, ajout de l'attribut Licence dans la table pilote se fait par :

```
ALTER TABLE pilote ADD licence CHAR(40) NOT NULL;
```

- L'activation d'une contrainte :

```
ALTER TABLE pilote ENABLE CONSTRAINT pilote_pk ;
```

La modification du type d'un attribut peut concerner :

- la réduction de la taille : seulement si l'attribut ne contient que des valeurs NULL ;
- la modification du type : même condition que précédemment ;
- l'augmentation de la taille est toujours possible ;
- la suppression de l'interdiction de présence de valeurs nulles (passage de NOT NULL à NULL) : toujours possible.

3.6.6.1 Activation des contraintes d'intégrité (Oracle)

```
{ ENABLE | DISABLE } CONSTRAINT contrainte ;
```

En cas d'activation d'une contrainte, il faut pouvoir gérer les lignes qui violent celle-ci ! La clause EXCEPTIONS INTO permet d'indiquer une table des lignes incohérentes qu'il faudra mettre à jour. En attendant, la contrainte est active pour les insertions et mises à jour nouvelles. Les « vieilles » lignes ne respectant pas la contrainte sont tout de même conservées. Il vaut mieux les mettre à jour ou les supprimer.

Exemple :

```
CREATE TABLE rejets (nligne ROWID, nomSchema VARCHAR2(30), nomTable
VARCHAR2(30), nomContr VARCHAR2(30)) ;
ENABLE CONSTRAINT pilote_pk EXCEPTIONS INTO rejets ;
... traitement ou éventuellement
DELETE FROM pilote WHERE ROWID IN (SELECT nligne FROM rejets WHERE
nomContr='pilote_pk') ;
```

3.6.7 Renommer une table

On peut changer le nom d'une table en la renommant. Mais il faut prendre soin de répercuter cette modification au niveau des applications et des vues définies sur cette table.

```
RENAME old TO new ;
```

La possibilité de définir des synonymes permet de pallier aux problèmes posés par le renommage d'une table. En effet, une table peut-être référencée soit par son nom initial soit par un synonyme.

3.7 Les droits

L'accès d'un utilisateur est contrôlé au moment de la connexion. (login et password). Les droits d'accès à un objet (table, vue, ..) dépendent :

- de son autorité administrative ;
- du fait d'être ou non propriétaire des objets ;
- d'autorisations explicites accordées.

Donner des droits :

```
GRANT { droit [,droit]... | ALL }
  ON ressource
  TO { usager [,usager]... | PUBLIC }[WITH GRANT OPTION] ;
```

donne à un ou plusieurs usagers (ou à tous, PUBLIC) les **droits** d'accès spécifiés :
ALTER, DELETE, INDEX, INSERT, SELECT, UPDATE, ALL

Supprimer des droits :

```
REVOKE { droit [,droit]... | ALL }
  ON ressource
  FROM { usager [,usager]... | PUBLIC } ;
```

Créer des utilisateurs

Les utilisateurs possédant l'autorité d'un DBA (administrateur de Base) peuvent créer de nouveaux utilisateurs par la commande :

```
GRANT [CONNECT,][RESOURCE,][DBA]
  TO user [,user]...
    [IDENTIFIED BY passwd [,passwd]...] ;
```

4. Langages de programmation

4.1 Introduction

Quelle que soit la puissance d'un langage de requêtes, il est souvent nécessaire de faire appel à un langage de programmation. Plusieurs approches sont possibles :

- approche par intégration : le langage est intégré au SGBD, PL/SQL pour Oracle, Visual Basic pour Access. Les instructions du langage de manipulation de données existant sont « plongées » au sein d'un langage de programmation. Pour cela, on introduit des constructions spécifiques (curseurs, recordset) ;
- approche par extension : les requêtes sont incorporées dans du code C(++) ou Java ou Perl ou PHP ou ... Le langage de programmation hôte est enrichi d'un certain nombre d'instructions ou de macro-instructions (C) permettant la manipulation des données. Ensuite, la compilation avec une bibliothèque fournie avec le SGBD permet au programme d'application de faire des appels au moteur du SGBD. Les interfaces de programmation ODBC ou JDBC procède de cette approche en permettant la connexion d'une application au SGBD et la soumission de requêtes.

5. Installation PostgreSQL

1. extraire les fichiers de postgresql-8.2.3-1.zip dans un répertoire temporaire
2. lancer postgresql-8.2.msi et choisissez le Français
3. sélectionner le support des langues
4. configuration du service : noter le nom du compte de service « postgres » et adjoignez lui un mot de passe **à ne pas oublier** (utilisateur qui lance le service) **!**
5. notez le numéro de port 5432 ; acceptez les connexions externes (pas seulement localhost ; il faudra éditer le fichier pg_hba.conf) ; choisissez une locale France, french ; codage LATIN1 ; superutilisateur : c'est le DBA : il peut être le même que celui qui lance le service…
6. installer lo (large objects), autoinc.
7. terminer l'installation

5.1 Démarrage

- Pour lancer le service, menu Démarrer, PostgreSQL 8.2, lancer le service.
- Lancer pgAdminIII, repérez la bd postgres, créer une nouvelle base de données « test », dans le schéma public, ajouter une table etudiant contenant les colonnes suivantes, …

```
CREATE TABLE etudiant
(
 numet serial NOT NULL,
 nom text,
 prenom text
)
ALTER TABLE etudiant
 ADD CONSTRAINT etudiant_pkey PRIMARY KEY(numet);
```

- Lancer psql sur postgres :
  - select user; -> postgres
  - select now();
  - select current_database(); -> postgres
  - \c test
  - insert into etudiant(nom,prenom) values ('dupont','pierre'),('martin','jacques');
  -
- Afficher la table etudiant à l'aide pgAdminIII, notez les nouvelles lignes
- Remarquez les problèmes d'affichage dans la fenêtre dos psql !
- \! cmd.exe /c chcp 1252 -> change le code de page de la fenêtre dos
- Changer la propriété de police de la fenêtre psql afin d'utiliser Lucida
- Lancer \? et vérifier le bon affichage des accents !

5.2 Divers

- Pour ajouter des données en mode graphique, dans pgAdminIII, afficher les données d'une table, et insérer de nouvelles lignes en bas de la table.
- Pour ajouter de nombreuses lignes à une table depuis un fichier ou depuis l'entrée standard (stdin), utiliser COPY etudiant FROM fic.txt ;
- Pour sauvegarder la bd test dans pgAdmin, cliquer droit sur la bd, sauvegarder, donner un nom de fichier « testbackup.sql », PLAIN, commandes insert ;

5.3 Connexion psql à un serveur distant

Sur le serveur :

1. Tout d'abord, il faut lancer l'application pare-feu afin d'ouvrir le port TCP 5432 ;
2. Il faut ensuite configurer le serveur PostgreSQL à l'aide de 2 fichiers de configuration situés dans « c:/Program Files/PostgreSQL/8.2/data » :
3. vérifier dans postgresql.conf qu'il existe une ligne : listen_addresses = '*'
4. ajouter la ligne suivante dans pg_hba.conf :

```
# TYPE      DATABASE  USER  CIDR-ADDRESS     METHOD
host  all               all   192.168.0.0/24   password
```


5. panneau de config, outils d'administration, services, redémarrer le service postgreSQL pour prendre en compte les changements de configuration ;

Sur le client :

1. psql –h machineServeur –U utilisateur bd
   soit psql –h 192.168.0.2 –U postgres test
   192.168.0.2 peut être remplacé par le nom de la machine serveur

5.4 Installation et utilisation de PostgreSQL ODBC

1. extraire les fichiers de psqlodbc-08_02_0400.zip dans un répertoire temporaire puis lancer l'installation du driver psqlODBC et de la documentation sur le client.
2. une fois le driver odbc installé sur la machine cliente, il reste à créer un « Data Source Name » associé à une bd particulière : panneau de config, outils d'administration, source de données (ODBC), onglet source de données système pour permettre l'accès à d'autres utilisateurs depuis d'autres machines, ajouter, choisir le driver PostgreSQL Unicode, (le client et le serveur peuvent être la même machine)
   a. Data Source : choisissez un nom parlant : « connexion test »
   b. Database : test
   c. Server : localhost ou le nom du serveur distant 192.168.0.2
   d. Username : postgres ou un autre utilisateur
   e. Password : donnez le password associé à l'utilsateur
3. Toujours tester avant de sauvegarder.
4. Une fois la source de données système installée, on peut interagir sur la bd test depuis une autre application : programme C, autre sgbd comme access, …
5. ouvrir Access, créer une bd, une table, y mettre quelques lignes, puis dans l'onglet table d'Access :
   a. cliquer droit sur la table, exporter, choisir le format ODBC, choisir un nom de table distante, sélectionner la source de données « connexion test » et lancer la création ;
   b. vérifier sur le serveur que la table a bien été créée.

Remarquons que depuis la machine cliente, on peut utiliser une source de données système créée localement et associée à un serveur distant, ou bien au moment de sélectionner la source, on peut aller chercher une source de données fichier située à distance sur le réseau !

6. Conclusion

Pour aller plus loin que ce cours introductif aux Systèmes de Gestion de Bases de Données, on peut prendre différentes directions :

- sur le plan théorique et pratique, le développement des bases de données déductives utilisant la logique ;
- sur le plan théorique et pratique, les SGBD à Objets qui reprennent les principes de la programmation par objets ;
- les méthodes de conception de schéma de BD relationnelles, Merise, normalisation ;
- sur le plan pratique, le développement d'une BD en utilisant un langage de programmation style PL/SQL, des triggers, du C ;
- les transactions pour le contrôle de concurrence ;
- sur le plan pratique, le modèle 3 tiers des architectures client-serveur et l'interface d'accès aux données CGI Perl, PHP3, ASP, JSP, ...
- sur le plan pratique, les interfaces ODBC, JDBC de connexion aux bases de données ;
- les outils de conception et de développement d'applications BD : Power AMC, Rational Rose, Oracle Designer, RAD (WinDev, Power Builder, ...) ;
- l'administration des SGBD : sécurité, tolérance aux pannes ;
- ...

On le voit avec cette liste non exhaustive, les SGBD sont utilisés dans toute l'informatique de gestion. Une bonne compréhension des concepts théoriques sous-jacents aux SGBD commerciaux permet un passage facilité d'un SGBD à un autre, et surtout une comparaison possible des avantages et inconvénients de chacun. Le développement d'internet et donc des clients légers web intensifie la complexité architecturale des applications BD. Les temps de réponse de certains SGBD « relationnels » ne sont pas les seuls éléments à prendre en compte lors d'un développement de site web. En effet, la maintenance, le développement du site et l'évolution de la BD associée peut parfois prendre beaucoup plus de temps avec des SGBD ne rendant pas des services déclaratifs essentiels : CIR, trigger, transaction ... La migration d'un SGBD à un autre est toujours une opération coûteuse et dangereuse, le standard SQL n'étant pas respecté, aussi il est toujours préférable de choisir un SGBD pouvant supporter une charge plus importante que celle prévue au démarrage.