Bases de Données - HMIN112M

Mastère Informatique Pour les Sciences - IPS

2020-2021

I. Mougenot & A-M. Chifolleau

Table des matières

Chapitre 1

Introduction

Contents

1.1 Généralités, définitions

L'objectif du cours est de donner quelques éléments de réponse à la question suivante : **Pourquoi a-t-on besoin des Bases de Données (BD) et 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 qui est saisie. Les fichiers viennent alors apporter des solutions à ce problème (Figure 1.1). Les premiers problèmes abordés étaient ceux du domaine de la gestion ce qui explique l'orientation précoce des bases de données dans ce domaine.

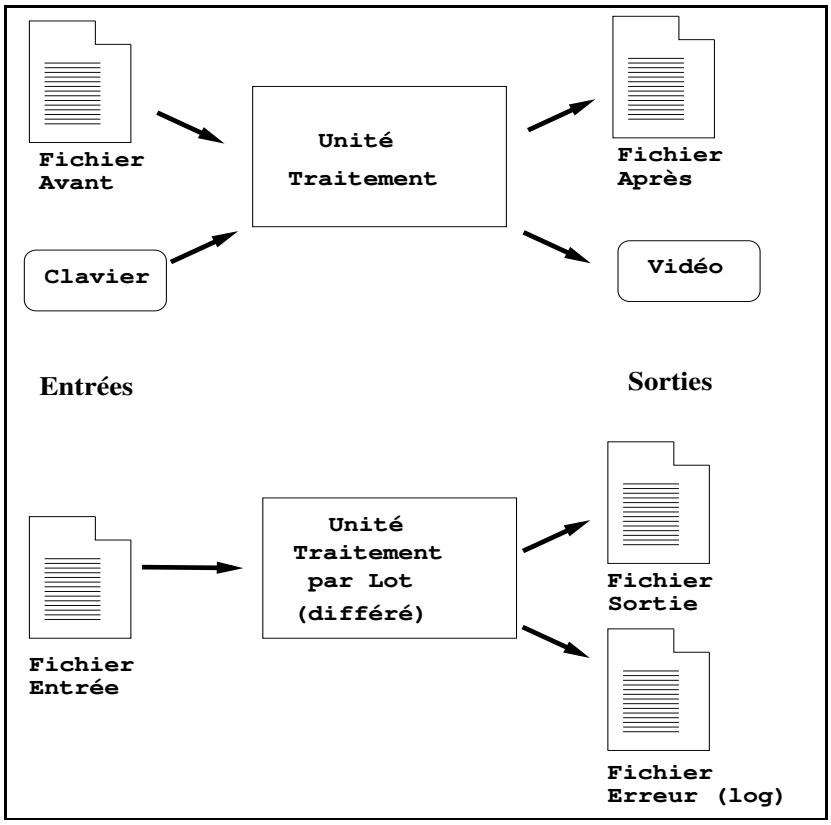


FIGURE 1.1 – Stockage via des fichiers

Progressivement, on en arrive à l'idée d'une véritable interaction entre l'utilisateur et le système.

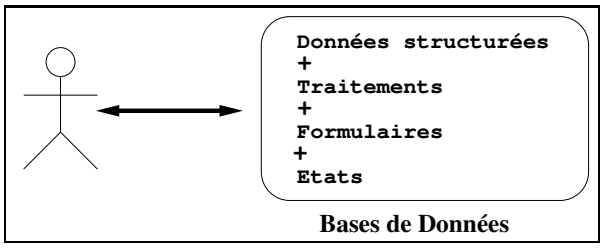


FIGURE 1.2 – Vision base de données

Définitions importantes

- Une **Base de Données (BD)** peut être définie comme une *collection d'informations structurées* (enregistrements, tables, ldots) 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.

L'évolution historique des bases de données, suit les avancées réalisées dans des domaines divers comme les systèmes d'exploitation, les langages de programmation ou encore la logique.

Quelques faits marquants

- Avant les années 60, développement des SGF (Systèmes de Gestion de Fichiers) 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.
- 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* (à savoir hiérarchique et réseau).
- Durant les années 70, naissance de la deuxième génération de SGBD
  - modèle entité-relation (ou entité-association),
  - modèle relationnel.
- Années 80- 90
  - troisième génération : modèle orienté objet,
  - bases de données déductives,
  - objet-relationnel,
  - ...
- Années 2000 - de nos jours
  - systèmes distribués, passage à l'échelle : mouvance NOSQL (Not Only SQL)
    - à base d'agrégats : clé-valeur, document, colonne
    - graphe

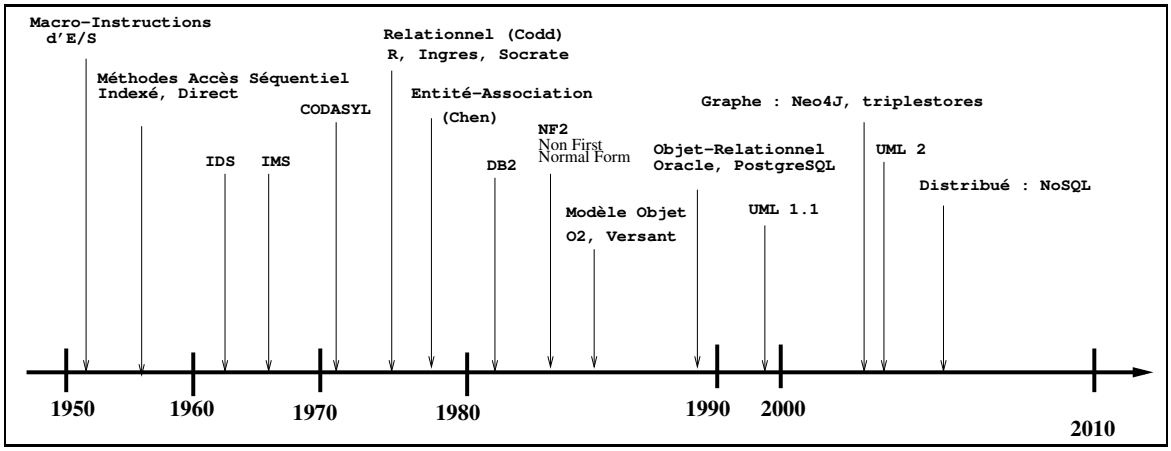


FIGURE 1.3 – Evolution de la gestion des données

Eléments de légende de la figure

IDS Integrated Data Store (modèle réseau General Electric)
IMS Information Management System (IBM modèle hiérarchique puis modèle réseau)
CODASYL Conference on Data System Language

Les SGBD les plus courants (pour plus d'exhaustivité voir `https://db-engines.com/en/ranking`)

- SGBD relationnels (ou objet-relationnel) commerciaux
  - Oracle, Oracle http ://www.oracle.com/index.html, (Windows, Unix),
  - Sybase, Sybase http ://www.sybase.com/home,
  - DB2, IBM http ://www-306.ibm.com/software/data/(Centraux IBM),
  - Informix IBM http ://www-306.ibm.com/software/data/informix/,
  - SQL Server Microsoft http ://www.microsoft.com/sql/default.asp,

  - ACCESS Suite Office Microsoft http ://office.microsoft.com/en-us/default.aspx.
- SGBD relationnels open source
  - PostgreSQL, www.postgresql.org, (Unix, Linux, MacOS X),
  - MySQL http ://www.mysql.com, (Windows, Unix, Linux, MacOS X).
- SGBD objets (plus ou moins disponibles)
  - Matisse, http ://www.matisse.com/ (Unix, Windows)
  - O2 (retiré du marché en 1999),
  - ObjectStore, http ://www.objectstore.net/index.ssp
  - GemStone, http ://www.gemstone.com/
  - Poet, http ://www.poet.com/en/indexjs.html
- SGBD NoSQL
  - Document : CouchDB, couchdb.apache.org
  - Colonne : HBase, hbase.apache.org
  - Graphe : Neo4J, neo4j.com
  - Triplestore : Jena TDB, jena.apache.org/documentation/tdb

1.2 Constituants principaux d'un SGBD

Les SGBDs sont axés sur les données et leur qualité, donc sur leur mise à jour, leur cohérence, ou encore 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 (applications programmées, requêtes, ...) par différentes catégories d'utilisateur.
La figure 1.2, "Structure d'un SGBD", indique les utilisations possibles des BDs gérées par 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.
- **Interrogation** : 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).
- **Contrôle** : Le Gestionnaire de la Base de Données, appelé également *moteur*, peut être vu comme le coeur du SGBD.
  Il traduit les résultats de la compilation du langage de définition des données et des requêtes utilisateurs en une suite d'opérations sur le ou les fichiers qui contiennent réellement les données.
  Le gestionnaire 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 le cas des systèmes IMS ou 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.

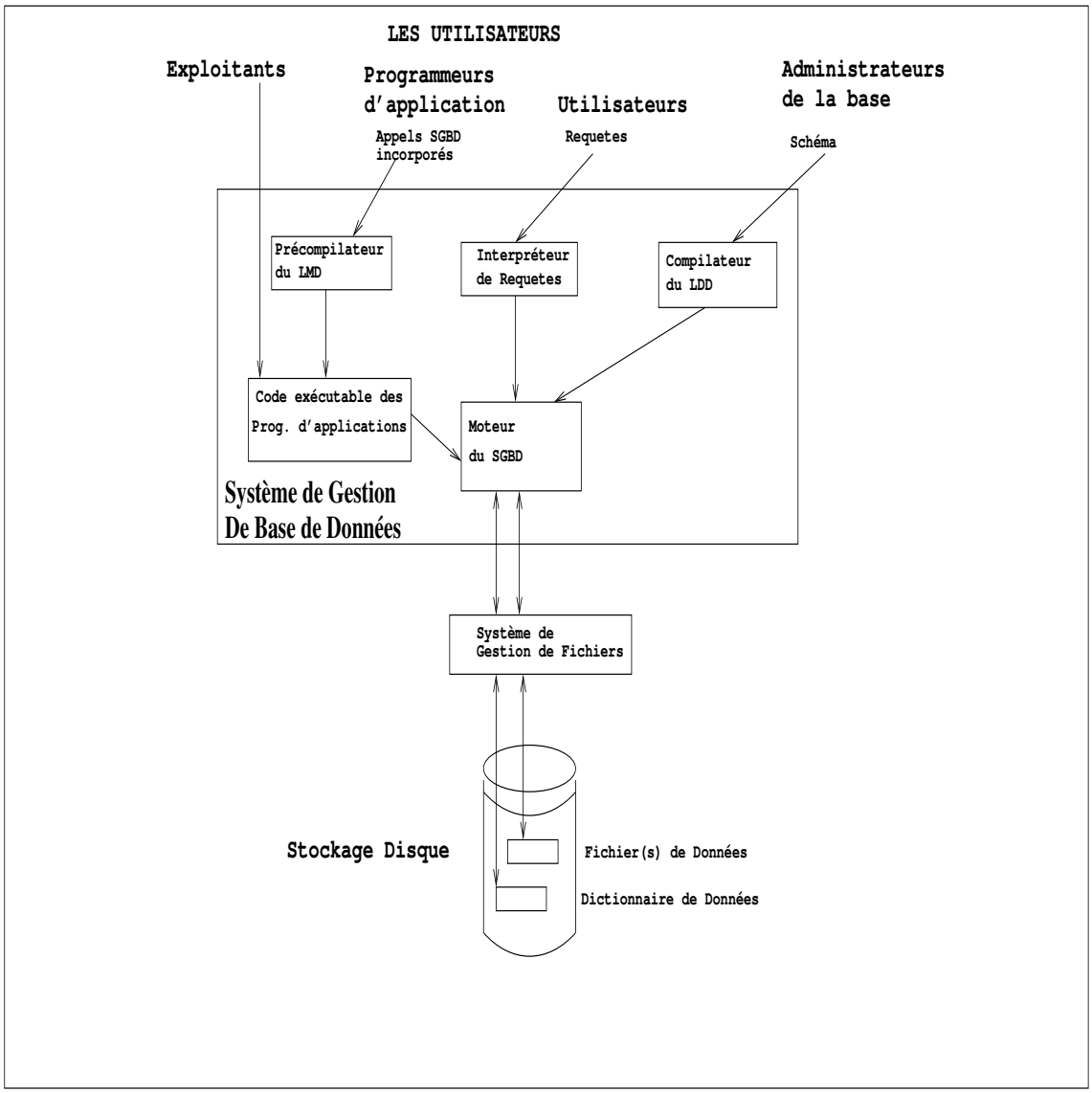


FIGURE 1.4 – Structure d'un SGBD

1.3 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.

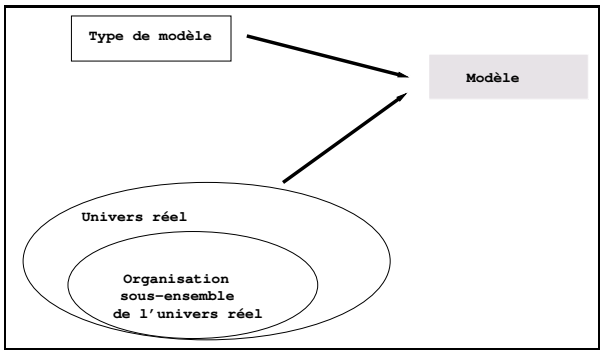


FIGURE 1.5 – Vision générale de la modélisation

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).

Niveaux ANSI/SPARC

- **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.
- **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.

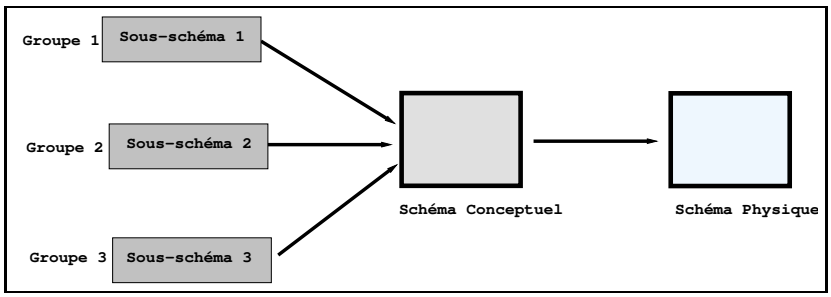


FIGURE 1.6 – Niveaux d'abstraction

Chapitre 2

Notation UML et conception de BD

Contents

Le formalisme UML (Unified Modelling Language) permet de "modéliser" la partie de la réalité concernée par le projet d'informatisation. Le concepteur dispose de plusieurs vues sur le modèle global du système : la vue cas d'utilisation (vision plutôt fonctionnelle exprimée en terme de services attendus), la vue structurelle (vision "statique" sur les objets du système), la vue dynamique (vision relative aux scénarios d'exécution du système, à la collaboration entre objets du système, à l'évolution des objets du système au cours du temps) ainsi que des vues liées au déploiement et à l'organisation des composants du système.

Dans le cadre du cours, nous nous attacherons à montrer comment vue structurelle, vue cas d'utilisation et vue dynamique permettent la conception de bases de données.

2.1 Modèle fonctionnel

La vision "externe" et fonctionnelle du système consiste à préciser quels sont les acteurs utilisateurs du système et quels types de service ils sont en mesure d'attendre du système.

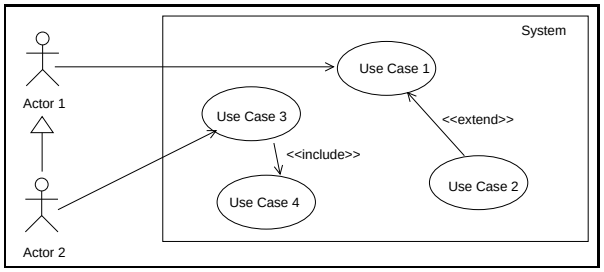


FIGURE 2.1 – diagramme de cas d'utilisation

Les cas d'utilisation peuvent être liés entre eux, par exemple par le lien "extend" indiquant que le cas d'utilisation peut être étendu par un autre cas d'utilisation (traitement d'exceptions par exemple).

2.2 Modèle structurel

La partie "statique" d'un modèle peut être décrite à l'aide de diagrammes de classes et de diagrammes d'objets. Un diagramme de classes montre l'organisation structurelle des "choses" du système en terme de classes et d'associations entre ces classes. Un diagramme d'objets montre quelques instances (ou individus) qui satisfont le diagramme de classes.

2.2.1 Diagramme de classes

Un **diagramme de classes** distingue plusieurs éléments (dénotés par *classifier* dans la notation UML) :

— **Classe** une classe est le descripteur d'un ensemble d'objets de structure, comportement et relations similaires. Le modèle associé décrit l'intension [1] de la classe et à l'exécution la classe sera associée à une extension [2] (ensemble d'instances).

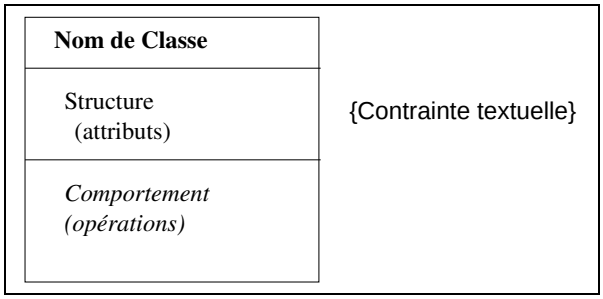


FIGURE 2.2 – Classe

— **Association** Dans le cas le plus simple une *association binaire* relie exactement deux classes (incluant la possibilité d'une association binaire établie sur la même classe).

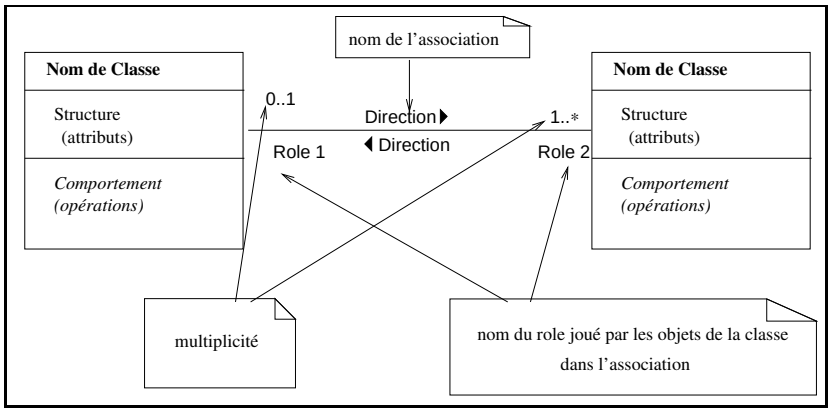


FIGURE 2.3 – Association

Les multiplicités possibles sont 0..1, 1..1, 0..*,1..* et m..n si l'on connait les valeurs exactes de n et m. Les associations peuvent être renseignées par des contraintes textuelles comme par exemple {ordonné}. Une association binaire peut aussi être reliée à une "classe association" (qui lui permet d'acquérir ainsi de nouvelles propriétés).

1. L'intension ou compréhension de la classe désigne toutes les propriétés qui concourent à la description de la classe

2. L'extension désigne l'ensemble des objets ou individus qui se conforment à la description de la classe

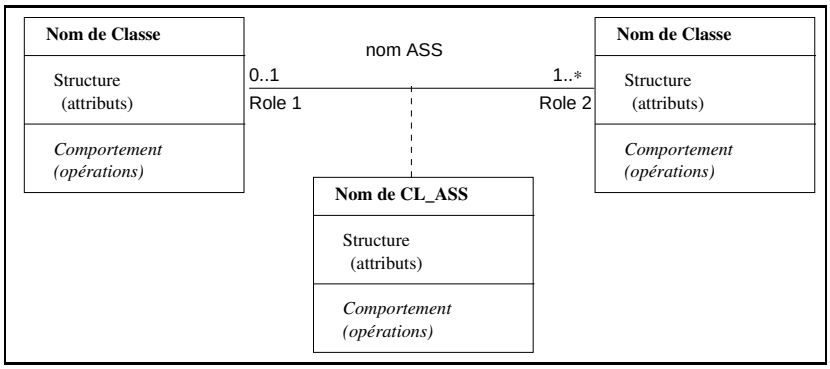


FIGURE 2.4 – Classe Association

Remarque : le nom de la classe_association est le même que celui de l'association. Les associations peuvent être plus complexes et relier plus de 2 classes.

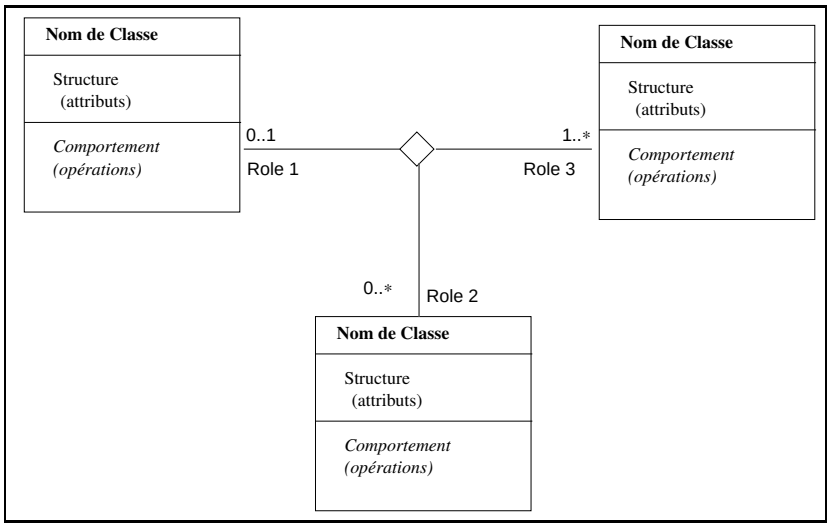


FIGURE 2.5 – Association ternaire

Des associations "spécifiques", telles que l'**agrégation** et la **composition** peuvent être utilisées, elles introduisent des contraintes complémentaires entre objet composite (le "tout") et objet composite ("élément sous-partie du tout"). Les contraintes sont plus fortes dans le cas de la composition (l'objet composant a la même durée de vie que son composite et il ne peut être partagé entre plusieurs composites). Il est alors dit que le composant dépend existentiellement de son composite. Le losange (blanc quand il s'agit d'une agrégation, et noir quand il s'agit d'une composition) est placé du côté de la classe composite.

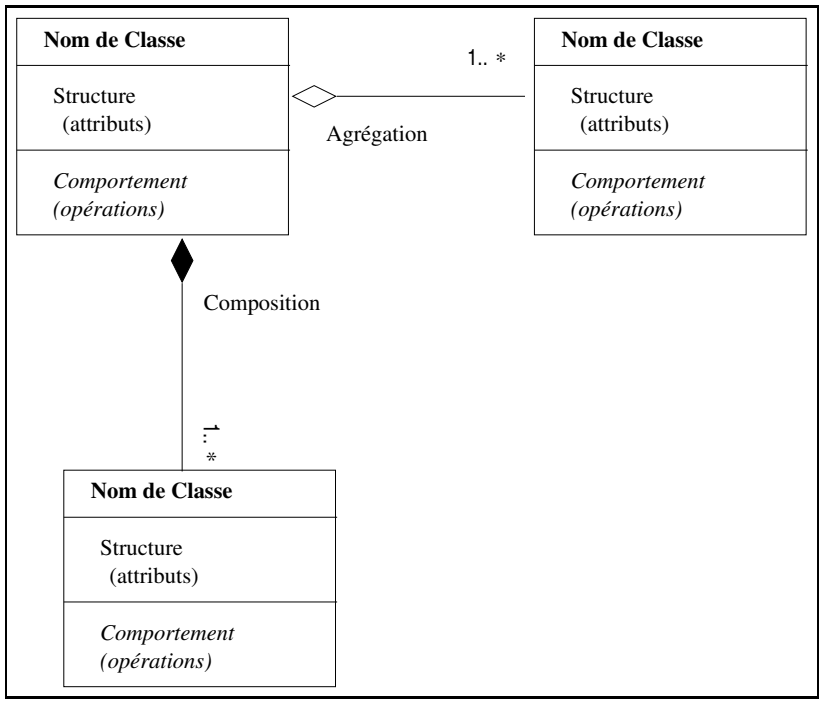


FIGURE 2.6 – Mécanismes d'agrégation et de composition

— **Spécialisation/Généralisation**
Pour exprimer le fait que des "classifications" peuvent être réalisées soit à partir de factorisations, soit à partir de spécialisations, on représente une association particulière entre classes.

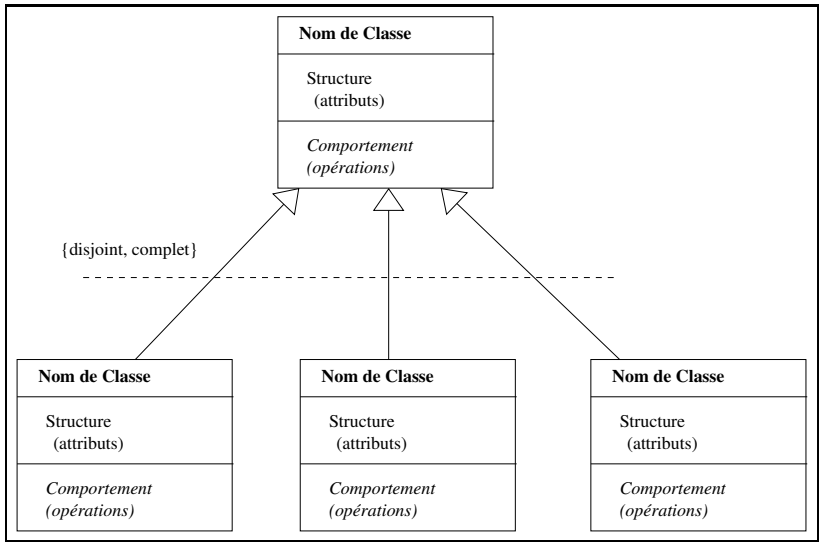


FIGURE 2.7 – Mécanisme de spécialisation/généralisation

2.2.2 Diagramme d'objets

Un diagramme d'objets présente des instances conformes au diagramme de classes correspondant. Il permet de vérifier une partie des contraintes exprimées sur le diagramme de classes.

Un exemple :

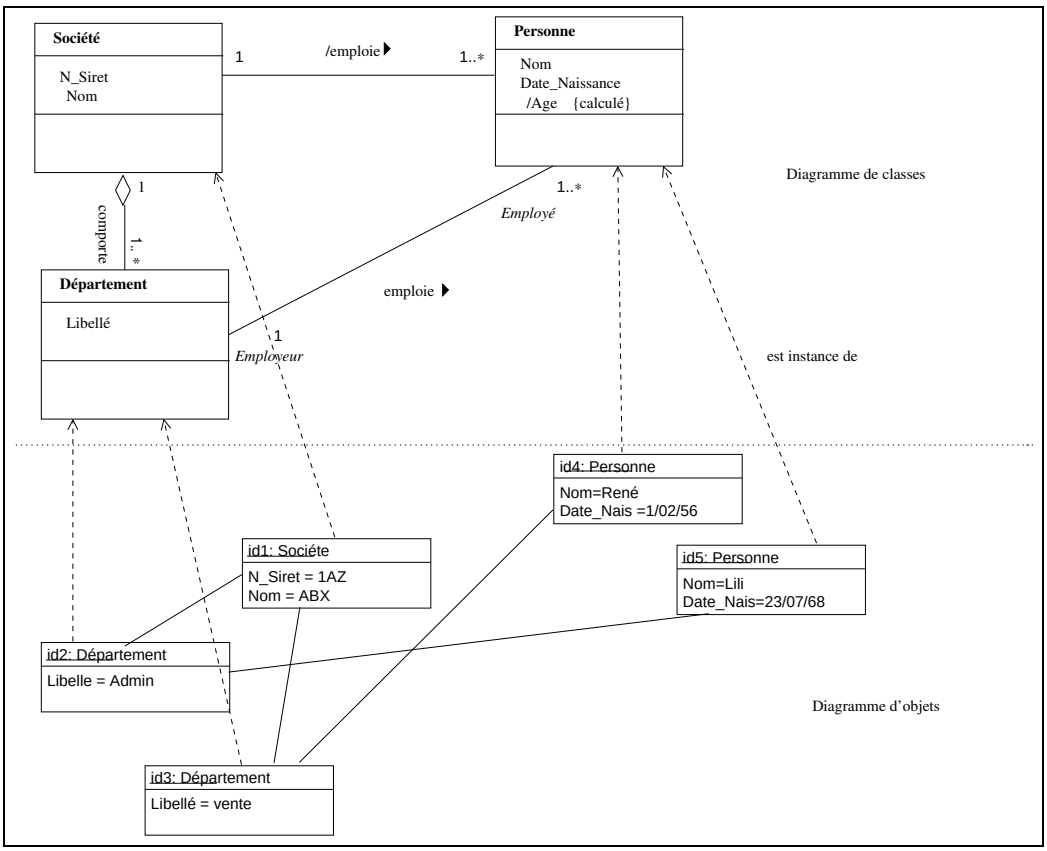


FIGURE 2.8 – Diagrammes de classes et d'objets

Remarque : L'association /emploie entre Société et Personne est une association dite dérivée (dénotée par le signe /), en effet elle est "calculable" à partir de l'association emploie liant les personnes et leur département et de l'association d'agrégation entre la Société et ses départements.

2.3 Modèle dynamique

Le modèle dynamique peut être représenté par divers diagrammes : diagrammes de collaboration, de séquences et d'états. Nous nous restreindrons aux diagrammes de séquences qui expriment le déroulement d'un scénario d'exécution possible au sein du système. Ces diagrammes traduisent les cas d'utilisation en s'appuyant sur les objets du diagramme structurel.

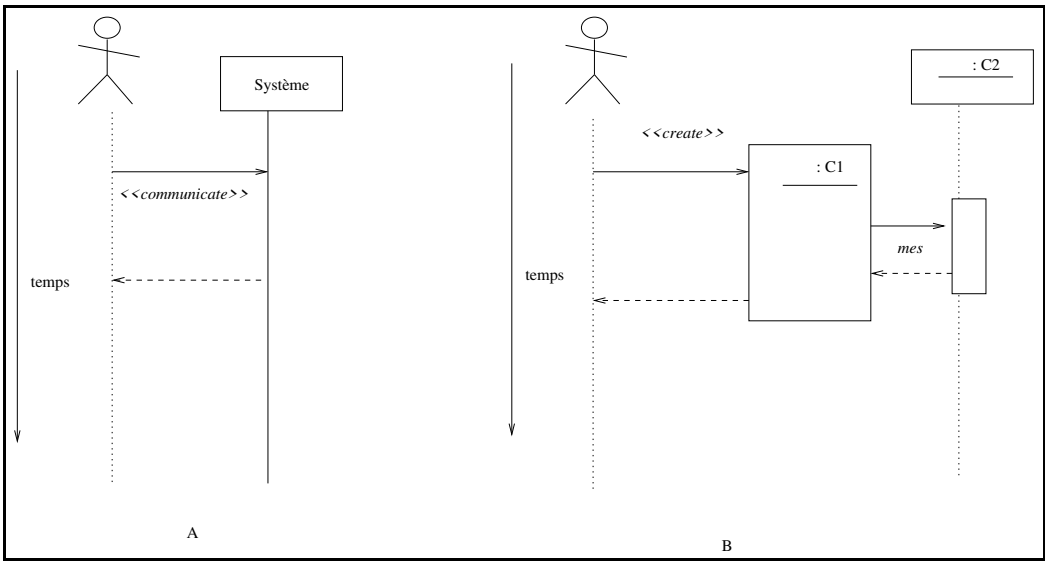


FIGURE 2.9 – Diagramme de séquences

La figure ci-dessus montre sur la partie gauche (diagramme A) un acteur qui communique avec le système selon un cas d'utilisation. La partie droite (diagramme B) donne le détail d'une communication qui en fait consiste à créer une nouvelle instance de la classe C1, lors de la création un message (mes) est envoyé vers une instance de la classe C2 qui préexiste dans le système.

2.4 Vers la conception de bases de données

Nous allons nous concentrer sur les règles de passage du modèle structurel (diagramme de classes) au modèle relationnel.

Règles de passage standard *Toute classe et toute association* vont donner naissance à un schéma relationnel dont la description va être décrite dans le dictionnaire des données.

NOM DE LA TABLE	NOM ATTRIBUT	DOMAINE DE L'ATTRIBUT	VALEURS NULLES AUTORISÉES	OBLIGATOIRE

FIGURE 2.10 – Description de la table dans le dictionnaire de données

Attention : en "objet" la description de la classe ne comprend pas forcément un ou plusieurs attributs dont les valeurs seraient uniques pour chaque objet de la classe, il faut donc systématiquement ajouter un attribut ID_Table.

Exemple (traduction du schéma précédent)
Les **classes** donnent :

Société	Nom Attribut	Domaine de l'attribut	Valeurs nulles autorisées	Obligatoire
	ID_Société	Identifiant	non	oui
	N_Siret	Chaine 20 {unique}	non	oui
	Nom	Chaine 40	oui	non

Figure 2.11 – Table Société

Personne	Nom Attribut	Domaine de l'attribut	Valeurs nulles autorisées	Obligatoire
	ID_Personne	Identifiant	non	oui
	Nom	Chaine 50	non	oui
	Date_Naissance	date	non	oui
	Age	calculé	non	calculé

Figure 2.12 – Table Personne

Département	Nom Attribut	Domaine de l'attribut	Valeurs nulles autorisées	Obligatoire
	ID_Département	Identifiant	non	oui
	Libellé	Chaine 20{unique}	non	oui

Figure 2.13 – Table Département

Il faut alors pour passer au codage en SQL choisir la clé primaire parmi les clés "candidates" c'est à dire ID_table ou un des attributs à valeur unique. Ici pour la table Société N_Siret Clé primaire (Id_société ne sera donc pas utilisé). Même remarque pour Département Libellé peut servir de clé primaire.

Code SQL

```
Create table Société
N_Siret varchar(20) not null primary key,
Nom varchar(50));

Create table Personne
Id_Personne number not null primary key,
Nom varchar(50)) not null,
date_naissance date not null,
age number not null);

Create table Département
Libellé varchar(20)) not null primary key);
   Remarque : L'attribut age est calculé, ce qui signifie que lors des insertions dans la table
Personne la valeur de l'âge ne sera pas donnée par l'utilisateur mais obtenue à partir d'un
calcul effectué à partir de la date de naissance.
```

Pour les **associations,** on obtient

Soc_comporte_Dept	Nom Attribut	Domaine de l'attribut	Valeurs nulles autorisées	Obligatoire
	N_Siret	Identifiant	non	oui
	Libellé	Identifiant	non	oui

Figure 2.14 – Table Soc_comporte_Dept

Dept_emploie_Pers	Nom Attribut	Domaine Attribut	Valeurs nulles autorisées	Obligatoire
	Libellé	Identifiant	non	oui
	ID_Personne	Identifiant	non	oui

Figure 2.15 – Table Dept_emploie_Pers

La clé primaire est la conjonction des identifiants récupérés dans chaque table associée.

Code SQL

```
Create table Soc_comporte_Dept
N_Siret varchar(20) not null ,
Libellé varchar(20) not null,
primary key (N_Siret,Libellé));

Create table Dept_emploie_Pers
Id_Personne number not null,
Libellé varchar(20)) not null,
primary key (ID_Personne,Libellé));
```

Remarque : l'association dérivée Société-Emploie-Personne peut donner lieu à une vue.

Code SQL

```
Create View Soc_emploie_Pers as
Select Id_Personne,N_Siret
From Dept_emploie_Pers A, Soc_comporte_Dept B
Where A.Libelle=B.Libellé ;
```

Règles relatives au multiplicité maximale 1 Si dans une association la multiplicité maximale liée à un rôle est 1, la table correspondant à l'association n'est plus nécessaire, il faut alors :

- si la multiplicité minimale est 1 (c'est à dire si on a 1..1) faire migrer l'identifiant du rôle concerné dans la table correspondant au deuxième rôle et établir la contrainte de "référence" (inclusion stricte).
- si la multiplicité minimale est 0 (c'est à dire si on a 0..1) faire migrer l'identifiant du rôle concerné dans la table correspondant au deuxième rôle mais la contrainte de "référence" ne sera pas vérifiée systématiquement (il peut y avoir des éléments non liés).

Ici les tables Soc_comporte_dept et Dept_emploie_Pers ne sont donc pas nécessaires, mais il faut modifier les tables Personne et Département.

Personne	Nom Attribut	Domaine de l'attribut	Valeurs nulles autorisées	Obligatoire
	ID_Personne	Identifiant	non	oui
	Nom	Chaine 50	non	oui
	Date_Naissance	date	non	oui
	Age	calculé	non	calculé
	Libellé	Chaine 20 référence	non	oui

Figure 2.16 – Table Personne

Département	Nom Attribut	Domaine de l'attribut	Valeurs nulles autorisées	Obligatoire
	Libellé	Identifiant	non	oui
	N_Siret	Chaine 20 référence	non	oui

Figure 2.17 – Table Département

Code SQL

```
Create table Personne
(Id_Personne number not null primary key,
Nom varchar(50)) not null,
Date_Naissance date not null,
Age number not null,
Libellé varchar(20) foreign key references Departement(libellé));

Create table Département
(Libellé varchar(20)) not null primary key, N_Siret varchar(20) foreign key references Société(N_Siret));
```

Règles relatives aux classes_associations Les classes associations permettent en fait d'ajouter des propriétés à l'association concernée. Donc si l'association donne lieu à une table il faut rajouter dans la description de cette table ces propriétés et elles feront partie de la clé primaire. Si l'association ne donne pas lieu à une table, il faut faire migrer ces propriétés avec l'identifiant du rôle.

Règles relatives à la généralisation/spécialisation Plusieurs stratégies possibles :

- générer la table correspondant à la classe de "plus haut niveau" en récupérant l'ensemble des informations des sous-classes.
  Principal danger : la table sera "creuse" c'est à dire avec de nombreuses valeurs nulles pour des attributs qui caractérisaient les sous classes.
- générer les tables correspondant aux classes de niveau feuilles en récupérant l'ensemble des informations "héritées".
  Principal danger : les identifications dans les tables obtenues ne doivent pas se recouper (si la spécialisation est simple)
- générer l'ensemble des tables relatives à chaque classes de la hiérarchie
  Ici on a toujours le problème des identifications mais qui peut être résolu en utilisant la contrainte d'intégrité référentielle. Il reste cependant que pour reconstituer la totalité de l'information il faut avoir recours à des jointures.

Chapitre 3

Modèle Relationnel

Contents

3.1 Introduction

Proposé à partir de 1970 (travaux de E.F. 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 (SQL) ayant fait leur preuve.

Dans ce chapitre, nous nous intéresserons aux concepts de ce modèle, et à la description des langages "théoriques" attachés au modèle relationnel à savoir les langage algébriques et prédicatifs.

3.2 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'entiers ;
— un ensemble de chaînes de caractères ;
— {rouge,vert,bleu}.

Le **produit cartésien** des domaines D1,D2,...,Dn est dénoté par :

$$D1 \times D2 \times .... \times Dn$$

et est vu comme l'ensemble des n-uplets ou tuples (v1,v2,...,vn) tels que vi $\in$ Di pour i=1,2,...,n.
Une **relation** sur les domaines D1,D2,...,Dn est alors un sous-ensemble du produit cartésien de ces domaines.
On appelle le nombre de domaines n l'**arité** de la relation et **n-uplets** (ou encore tuples) 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.

Ordinateur	Nom	Constructeur	Prix	Année
	Emac G4	Apple	836	2004
	Latitude	Dell	1000	2003
	Armada	HP Compaq	500	2001
	Vaio	Sony	1300	2004

Figure 3.1 – Une extension possible de la relation Ordinateur

Remarques :

- un nom différent est attribué à chaque colonne de manière à les distinguer (notion de rôle),
- le couple (nom,domaine) est appelé **attribut** de la relation. L'ordre des attributs tout comme l'ordre des n-uplets demeure sans importance,
- on appelle **schéma de la relation** le nom de la relation suivi de la liste de ses attributs. Pour reprendre l'exemple de la relation Ordinateur, le schéma de la relation est ainsi : Ordinateur (Nom : caractère(10), Constructeur : caractère(10), Prix : numérique, Année entier, clé **Nom,Constructeur**)

La notion de **clé** (rencontrée dans le modèle E-A au travers de la notion d'identifiant) s'applique à la structure relationnelle. Ainsi les attributs Nom et Constructeur forment une clé pour la relation Ordinateur. En d'autres termes, deux tuples de la relation ne peuvent avoir la même valeur à la fois pour Nom et Constructeur.

Généralisation :

Soit $\mathcal{R}$ un schéma de relation défini sur un ensemble U d'attributs, et K un sous-ensemble de U. K est appelé clé si il n'existe pas de couples de n-uplets dans les relations R (réalisations de $\mathcal{R}$), ayant même valeurs d'attributs pour les attributs de K.

3.3 Schéma de base de données relationnel

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.
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 instanciation du schéma ou relation ou encore **instance de la relation** correspond au contenu à un instant donné (relation en extension).

Un schéma de relation est donc constitué :

- d'un nom,
- de la liste des attributs,
- de la liste des contraintes imposées.

Exemple de schéma relationnel

Societe(numSiret entier, nom char(15), ville char(20), pays char(15))
Salarie (numEmploye entier, nom char(20), prenom char(20), fonction char(20), numSociete entier)
Ordinateur (nomOrdi char(10), marqueOrdi char(15), type char(15), processeur char(15), numSociete entier)
AffecteA (numEmploye entier, nomOrdi char(15), marqueOrdi char(15))

3.4 Les contraintes

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

— contraintes de domaine,
— dépendances fonctionnelles,
— contraintes référentielles ou contraintes d'inclusion.

3.4.1 Contraintes de domaine

Elles peuvent revêtir plusieurs formes :
— définition extensive ou en intention du domaine d'un attribut avec pour exemples :
  — l'attribut nomOrdi du schéma de relation Ordinateur est contraint à être une chaîne de caractères de longueur 10,
  — l'attribut marqueOrdi du schéma de relation Ordinateur a ses valeurs dans l'ensemble {Dell, HP Compaq, Apple, Sony, IBM}.
  — présence obligatoire ou non d'une valeur pour un attribut à l'aide de la contrainte NULL ou NOT NULL.
— l'existence de la valeur d'un attribut peut être liée 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 (numéro de sécurité sociale). L"attribut N_SS est contraint à son tour à être une chaîne de caractères de longueur 13.
— contraintes de dépendances entre attributs que nous allons détailler (DF ou dépendances fonctionnelles, DJ ou dépendances de jointure, DI ou dépendances d'inclusion). Les DJ et DI seront étudiées lors du processus de normalisation.

3.4.2 Dépendances fonctionnelles

Définition :

Soit $\mathcal{R}$(A1,A2,....An) un schéma de relation et G et D deux sous-ensembles de l'ensemble A1,A2,...,An. On dit que G détermine D ou que D dépend fonctionnellement de G et on note G $\rightarrow$ D si pour toute relation R de $\mathcal{R}$ acceptable, tous les n-uplets u,v de R qui ont même valeur dans G ont aussi même valeur dans D.

$$\boxed{\text{ssi } (\forall \text{ R instance de } \mathcal{R})\ (\forall \text{ u,v} \in \text{R tel que u[G]} = \text{v[G]} \Rightarrow \text{u[D]} = \text{v[D]})}$$

En d'autres termes, si deux n-uplets de R coïncident sur G alors ils coïncident sur D.

Remarque :
si DG est l'ensemble des attributs de $\mathcal{R}$, on ne peut avoir deux n-uplets u,v $\in$ R instance de $\mathcal{R}$ coïncidant sur G car sinon ils seraient égaux. Or les relations sont des ensembles, un n-uplet ne figure qu'une seule fois (doublons interdits).

Exemple : A l'ensemble de phrases suivantes :
— une voiture est identifiée par un numéro d'immatriculation N_immat,
— une voiture a une couleur ;
— à une voiture correspond un type ;
— un type de voiture correspond une puissance.

On peut associer l'ensemble de DF suivant :
{N_imm $\rightarrow$ Type, N_imm $\rightarrow$ Couleur, Type $\rightarrow$ Puissance}
La notion de dépendance fonctionnelle permet de préciser la notion de surclé et de clé : K est surclé de $\mathcal{R}$ sur U ssi K $\rightarrow$ U
Deux n-uplets distincts ne peuvent avoir la même surclé (sinon ils coïncident entièrement et R contient 2 n-uplets égaux). U est toujours une surclé (triviale)

Définition d'une clé de schéma de relation

Soit $\mathcal{R}$(A1,A2,....An), un schéma de relation et K un sous-ensemble de A1,A2,...,An, on dit que K est une clé pour $\mathcal{R}$ si :

— 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 de l'ensemble d'attributs qui constitue une clé.

$$\boxed{\text{K est clé de } \mathcal{R} \text{ ssi } \neg\ \exists\ \text{A} \in \text{K tel que K-}\{\text{A}\} \rightarrow \text{U}}$$

Remarque : Si plusieurs clés sont "possibles", on en choisit une dite clé primaire, les autres sont appelées clés candidates.

3.5 Les langages d'interrogation "théoriques"

Nous ne traiterons ici que de l'algèbre relationnelle. Les opérations de base sont de deux sortes :

1. les opérations ensemblistes,
2. les opérations spécifiques relationnelles.

Opérations ensemblistes

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.

Notations possibles :

UNION (Relation 1, Relation 2)
Relation 1 $\bigcup$ Relation 2

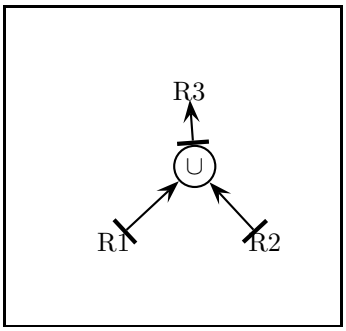


FIGURE 3.2 – Notation graphique de l'union

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

Notations possibles :

DIFFERENCE (Relation 1, Relation 2)
Relation 1 − Relation 2

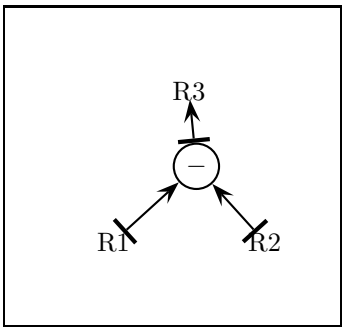


FIGURE 3.3 – Notation graphique de la différence

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.

Notations possibles :

Relation 1 × Relation 2

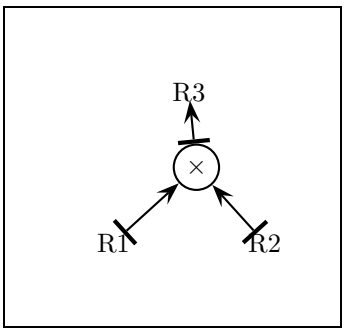


FIGURE 3.4 – Notation graphique du produit cartésien

Opérations spécifiques

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.

Notations possibles :

SELECTION (Relation 1, condition) avec condition du type : Attribut opérateur Valeur (opérateurs possibles : <, >,=,<>, ...)
Relation 1 : condition
$\sigma_{condition}$(Relation 1)

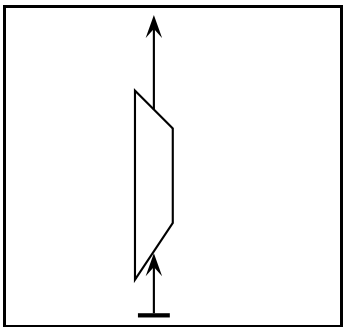


FIGURE 3.5 – Notation graphique de la sélection

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".

Notations possibles :

PROJECTION (Relation 1, att i,....)
Relation 1 [atti,....]
Πatti,..(Relation 1)

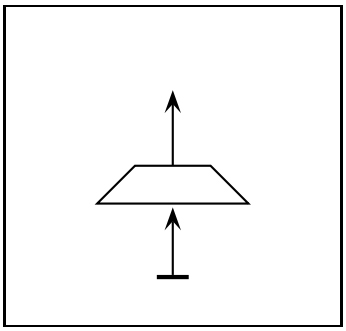


FIGURE 3.6 – Notation graphique de la projection

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).

Notations possibles :

JOIN(Relation 1, Relation 2,Condition)
R1 ⋈ R2

Remarques :

Si l'opérateur noté de manière générique Θ est l'opérateur d'égalité (=), l'opération est appelée équijointure.
La jointure naturelle (notée ⋈), en d'autres termes la jointure pour laquelle la condition est l'égalité des attributs de même nom en ne conservant qu'une seule fois ces attributs dans la relation 3, sera le plus souvent utilisée.
une Θ-jointure désignera une jointure mettant en jeu un opérateur autre que l'égalité (<,>,<>,>=, ...)

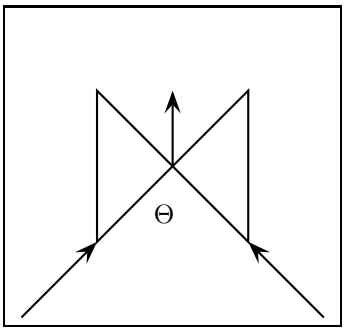


FIGURE 3.7 – Notation graphique de la jointure

Opérations complémentaires

INTERSECTION L'intersection peut être obtenue à partir de la différence
R1 ⋂ R2 = R1-(R1-R2) ou R2-(R2-R1)
Opération portant sur deux relations 1 et 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.

Notations possibles :

INTERSECTION(Relation 1,Relation 2)
Relation 1 ⋂ Relation 2

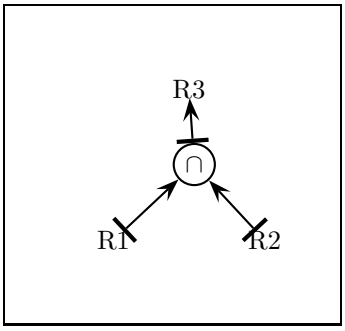


FIGURE 3.8 – Notation graphique de l'intersection

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
Remarque :
R1(X,Y) ÷ R2(Y) = R1[X] - ((R1[X] × R2 (Y)) - R1(X,Y)) [X]

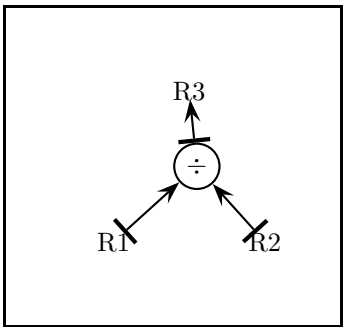


FIGURE 3.9 – Notation graphique du quotient

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 [A1]

SEMI-JOINTURE Opération portant sur deux relations notée Relation 1 $\ltimes$ Relation 2 qui consiste à projeter sur les attributs de R le résultat de la jointure naturelle Relation 1 $\bowtie$ Relation 2
Relation 1 $\ltimes$ Relation 2 = $\Pi_{attdeRelation1}$(Relation 1 $\bowtie$ Relation 2)

COMPLEMENT Le complément d'une relation R(X) notée ¬R(X) est un opérateur qui nécessite la connaissance des domaines associés aux attributs.
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}

R	PIÈCE	FOURNISSEUR
	écrou	pierre
	écrou	paul
	boulon	alice

FIGURE 3.10 – Relation R en extension

¬R	PIÈCE	FOURNISSEUR
	écrou	alice
	boulon	pierre
	boulon	paul
	vis	alice
	vis	pierre
	vis	paul

FIGURE 3.11 – le complément de R

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

3.6 Les Langages d'interrogation réels : SQL

Les langages d'interrogation commercialisés s'inspirent des langages algébriques ou des langages prédicatifs mais introduisent des fonctionnalités supplémentaires (tris, groupage-partitionnement, fonctions chaînes de caractère, fonction d'agrégation).
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 commerciaux (relationnels) :

- DB2 (IBM), SQL/DS (gros ordinateurs sous systèmes d'exploitation MVS, VM/CMS)
- Postgres (Linux), Informix ;
- ORACLE, Sybase, MySQL (gros ordinateurs et micro-ordinateurs) ;
- Access, SQL Server (micro-ordinateurs) ;
- DB4, Paradox (micro-ordinateurs).

En SQL, la **table** désigne une collection de lignes (**ROW**) ou n-uplets. Une table peut être vue comme une généralisation de la notion de relation ; plusieurs lignes peuvent en effet y être identiques. On parle alors de doublons (voir options ALL/DISTINCT). De plus la colonne (**COLUMN**) désigne un attribut.

SQL en tant que langage de manipulation de données (LMD)

Nous allons illustrer les commandes SQL au travers de la base de données exemple présentée précédemment.

La commande SELECT En SQL, la commande SELECT appelée consultation ou question ou interrogation (query) est fondamentale. Elle n'a aucun rapport sémantique avec l'opération algébrique de sélection (σ).

Forme générale (partielle) :

```
SELECT [ALL | DISTINCT] { liste <colonne/expression> | * }
FROM liste [<proprietaire>.] { <table> | <vue> } [<alias >]
[WHERE <condition>]
[GROUP BY {liste expression } [HAVING condition]] ;
```

Sémantique : la commande visualise une table par sélection, projection et jointure d'autres tables, aux doublons près.
Exemples :

Visualisation de toutes les lignes et colonnes d'une table (notée Tab)
SELECT * FROM Tab ;
Illustration SELECT * FROM Ordinateur ;
Visualisation de toutes les lignes et d'une partie des colonnes d'une table
SELECT col1,col2 FROM Tab ;
Illustration SELECT nomOrdi, marqueOrdi from Ordinateur ;
Ceci n'est pas une projection dans la mesure où nous pouvons obtenir des n-uplets identiques !
Si d'aventure, nous souhaitons éliminer les n-uplets identiques, le mot SELECT doit être suivi de DISTINCT :
SELECT DISTINCT col1,col2 FROM Tab ;
Illustration $\Pi_{nomOrdi,marqueOrdi}$(Ordinateur)
SELECT DISTINCT nomOrdi, marqueOrdi from Ordinateur ;

Visualisation d'une partie des lignes d'une table
SELECT * FROM Tab WHERE condition ;
Cette opération est une sélection si la table de départ est une relation.
Illustration $\sigma_{type='portable'}$(Ordinateur)
SELECT DISTINCT * FROM Ordinateur WHERE type='portable' ;
Pour exprimer des conditions nous pouvons avoir recours aux opérateurs de comparaison, aux connecteurs logiques ou encore à certains prédicats. Nous détaillons ces opérateurs, connecteurs et prédicats dans un paragraphe ci-dessous.
Visualisation de l'ensemble (ou d'une partie) des données de plusieurs tables ou vues
Nous définissons au préalable deux tables notées tab1 et tab2 et ayant pour schémas tab1(col1,col2,col3) et tab2(col1,col4).
SELECT tab1.col1,col2,col4 FROM tab1,tab2 WHERE tab1.col1=tab2.col1 ;
Cette requête effectue une jointure entre tab1 et tab2, la colonne col1 est commune aux deux tables, le nom de la colonne pour être identifié est préfixé par le nom de sa table suivi d'un point.
Illustration $\sigma_{marqueOrdi='Dell'}$(AffecteA) $\bowtie$ (Salarie)
SELECT * FROM AffecteA, Salarie WHERE AffecteA.numEmploye = Salarie.numEmploye AND marqueOrdi='Dell' ;
Il est également possible d'effectuer des "auto-jointures" (jointure d'une table avec elle même ou encore utilisation différentielle de la même table) :
SELECT A.col1, B.col2 FROM tab1 A, tab1 B WHERE NOT (A.col1=B.col1) AND A.col2=B.col3 ; A et B sont des variables lignes ou alias.
Illustration Un exemple d'utilisation différentielle est donnée ci-dessous :
SELECT x.idarticle, x.qtestock, y.idarticle, y.qtestock from article x, article y
WHERE x.qtestock = 2*y.qtestock ;
Sous-requêtes imbriquées. 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 ne doit avoir qu'une seule colonne ou expression dans sa clause SELECT, retourner une et une seule ligne excepté quand elle est précédée de IN,ALL,ANY.
Le prédicat EXISTS est le seul qui permet l'emploi de SELECT * dans une sous-requête.
Illustration
SELECT nom, salaire FROM employe
WHERE salaire > ALL (SELECT salaire FROM employe WHERE departement = 'production') ;

les opérations ensemblistes
SQL supporte les opérations d'union, d'intersection et de différence au travers des opérateurs UNION, INTERSECT et MINUS. SELECT col1 FROM tab1
{UNION|INTERSECT|MINUS}
SELECT col2 FROM tab2 ;
Les informations résultantes de chaque requête (ici col1 et col2) doivent être du même type.
Illustration
SELECT nom, prenom from Etudiant
UNION
SELECT nom, prenom from Enseignant ;

Opérateurs de comparaison, prédicats et connecteurs logiques

les opérateurs de comparaison =, <, >, <=,>=, <>. Ces opérateurs (Θ) peuvent être quantifiés. Soit S une expression dénotant un ensemble :
A Θ ANY(S) signifie $(\exists\ x)(S(x) \wedge A\ \Theta\ x)$
A Θ ALL(S) signifie $(\forall\ x)(S(x) \wedge A\ \Theta\ x)$
les 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'. SQL offre deux caractères génériques de substitution (le pourcentage pour la substitution de zéro à plusieurs caractères et le souligné (ou underscore) pour la substitution d'un et un seul caractère). Par exemple : lieu LIKE '_ri_' ou encore lieu LIKE '%er%'
- EXISTS teste la non vacuité d'un ensemble : EXISTS S
- NULL est un valeur non renseignée qui est par conséquente différente de zéro ou du blanc. SQL offre une comparaison sur les valeurs nulles à l'aide du prédicat IS [NOT] NULL

les opérateurs logiques : NOT, AND, OR

Les conditions élémentaires de sélection et de jointure vont pouvoir utiliser les opérateurs logiques de conjonction (AND), de disjonction (OR) et de négation (NOT)

Des extensions

les fonctions d'agrégat

Ces fonctions d'évaluation de tables (au travers d'expressions) vont permettre d'effectuer des calculs en regroupant toutes les lignes ensembles.
- COUNT([DISTINCT|ALL] expression)
- MIN([DISTINCT|ALL] expression)
- MAX([DISTINCT|ALL] expression)
- AVG([DISTINCT|ALL] expression)
- SUM([DISTINCT|ALL] expression)

Illustration

comptage du nombre de lignes :

```
SELECT COUNT(*)
FROM Ordinateur
WHERE marqueOrdi='Sanyo';
```

Autre Illustration

Moyenne et somme pour les colonnes de type numérique :
Minimum, Maximum pour les colonnes de type numérique ou chaîne de caractères

```
SELECT AVG(prix) AS Prix_Moyen)
FROM Ordinateur
```

le groupement au travers de la clause GROUP BY

Il peut s'avérer utile d'agréger les résultats non pas pour toute la table mais par groupes de lignes. L'opération de regroupement conditionnel s'effectue en outre par la clause : GROUP BY <liste colonne> HAVING <condition>. La clause HAVING effectue une sélection à l'image du WHERE mais de groupes de la partition et non plus des lignes de la table. Attention, chaque colonne ne participant pas au regroupement doit être calculée via une fonction d'évaluation (count, min, max, avg, sum).

Illustration

```
SELECT fonction, count(numEmploye)
FROM Salarie
GROUP BY fonction;
```

Autre Illustration

```
SELECT fonction
FROM Salarie
GROUP BY fonction HAVING COUNT(numEmploye)>=10;
```

les opérations de tri

SQL autorise le tri croissant (par défaut ASC) ou décroissant (DESC) sur une ou plusieurs colonnes. Les colonnes utilisées dans la clause ORDER BY doivent cependant faire partie de la clause SELECT

Illustration

```
SELECT nom, prenom, fonction, numSociete
FROM Salarie
```

ORDER BY nom, prenom ;

les opérateurs arithmétiques
Les opérateurs de multiplication et de division (∗ et /) sont prioritaires par rapport aux opérateurs d'addition et de soustraction (+ et -). Les parenthèses vont permettre de forcer les ordres de priorité si nécessaire.
les fonctions numériques
SQL offre un ensemble de fonctions traitant les données de type numérique qui peuvent être exprimées dans des expressions dans la clause SELECT ou WHERE :
— ABS(n) retourne la valeur absolue de n
— CEIL(n) retourne le plus petit entier supérieur ou égal à n
— COS(n) retourne le cosinus de n (n exprimé en radian)
— COSH(n) retourne le cosinus hyperbolique de n (n exprimé en radian)
— EXP(n) retourne la valeur exponentielle de n
— FLOOR(n) retourne la valeur entière de n
— LN(n) retourne le logarithme népérien de n (n¿0)
— LOG(m,n) retourne le logarithme à base de m, de n
— MOD(m,n) retourne le reste de la division entière de m par n
— POWER(m,n) retourne la valeur de m à la puissance n
— ROUND(n,|m] retourne la valeur de n arrondi à m postions à droite du point décimal
— SIGN(n) retourne -1 si n<0, 0 si n=0 et 1 si n>0
— SIN(n) retourne le sinus de n (n exprimé en radian)
— SINH(n) retourne le sinus hyperbolique de n (n exprimé en radian)
— SQRT(n) retourne la racine carrée de n
— TAN(n) retourne la tangente de n (n exprimé en radian)
— TANH(n) retourne la tangente hyperbolique de n (n exprimé en radian)
— TRUNC(n,m) retourne la valeur n tronquée à m positions décimales.
quelques fonctions de chaînes de caractères
L'opération de concaténation peut être réalisée sur les chaînes à l'aide de l'opérateur || (C1 || C2).
— INITCAP(char) met en majuscule la première lettre de chaque mot de la chaîne
— LOWER(char) met la chaîne de caractères char en minuscules
— UPPER(char) retourne la chaîne char avec toutes les lettres en majuscule
— LENGTH(char) retourne la longueur de la chaîne char en caractères
quelques fonctions de de date
Les opérateurs + et - peuvent être utilisés pour ajouter ou soustraire un nombre de jours à une date.
— SYSDATE retourne la date et l'heure courante du système
— ADD_MONTHS(d,n) ajoute n mois à la date d. Le résultat est une date
— MONTHS_BETWEEN(d1,d2) retourne le nombre de mois entre les dates d1 et d2.
— ROUND(D [, format]) retourne la longueur de la chaîne char en caractères

Forme complète de la commande SELECT

```
SELECT [ALL|DISTINCT] liste_de_selection
FROM liste_de_tables
[WHERE condition]
[[START WITH condition] CONNECT BY condition ]
[GROUP BY liste_d'expression [HAVING condition]]
[{UNION|UNION ALL|INTERSECT|MINUS} commande SELECT]
[ORDER BY {expr|position} [ASC|DESC] [,{expr|position} [ASC|DESC]]...
```

Quelques illustrations supplémentaires

Afficher des commerciaux qui se voient affecter un ou des ordinateurs de la marque Sanyo (expression au travers d'une jointure)
SELECT DISTINCT Salarie.numEmploye, nom, prenom FROM Salarie, AffecteA WHERE

Salarie.numEmploye=AffecteA.numEmploye AND marqueOrdi='Sanyo' AND fonction='commercial' ;
Afficher des employés qui se voient affecter un ou des ordinateurs de la marque Sanyo (expression au travers d'un select imbriqué)
SELECT DISTINCT numEmploye, nom, prenom FROM Salarie WHERE numEmploye IN (SELECT numEmploye from AffecteA WHERE marqueOrdi='Sanyo') AND fonction='commercial' ;
Afficher des employés qui se voient affecter un ou des ordinateurs de la marque Sanyo (formulation à l'aide de l'opérateur d'existence EXISTS)
SELECT DISTINCT numEmploye, nom, prenom FROM Salarie WHERE EXISTS(SELECT * from AffecteA WHERE marqueOrdi='Sanyo' and Salarie.numEmploye=AffecteA.numEmploye) AND fonction='commercial' ;

SQL a un caractère prédicatif ou assertionnel lié à la clause SELECT. Cependant il a également une tendance impérative (algébrique) parce qu'il autorise l'imbrication des SELECT.

SQL en tant que Langage de Définition de données (LDD)

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.

Définition de table Définir une table revient à déclarer son nom et sa structure. La définition de chaque attribut consiste à en donner un nom, un type, une taille et éventuellement à lui apposer des contraintes d'intégrité (par exemple, dans l'exemple illustratif numpl porte une contrainte de non nullité **NOT NULL**).

```
CREATE TABLE <table>
(définition d'une colonne | définition d'une contrainte, ...)
[[SPACE <définition_espace_stockage>] [PCTFREE n]/ [CLUSTER <cluster> (liste <colonne>)]
/ AS <commande SELECT données provenant d'une requête>]

Colonne ::= nom type [DEFAULT expression] [contrainte de colonne] ...
```

Les types de données admis par ORACLE sont :
CHAR(n) chaîne de caractères de longueur fixe (n<=240)
VARCHAR(n) ou VARCHAR2(n) : chaîne de caractères de longueur variable n (n<=2000)
LONG chaîne de caractères de longueur pouvant aller jusqu'à 65535 caractères ;
NUMBER(longueur,[précision]) nombre positif ou négatif sur n digits avec d digits à droite du point décimal
INTEGER entier
DATE données temporelles
ROWID chaîne hexadécimale représentant l'adresse unique d'une ligne de la table. Sa valeur est retournée par la pseudo-colonne de même nom.

Exemple de création de table

CREATE TABLE PILOTE (numpl NUMBER (6) NOT NULL, nompl CHAR (20), salaire NUMBER (8) DEFAULT 800, adresse CHAR (80)) ;

Remarque

Si aucune zone de stockage (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 au travers de la <commande SELECT> (requête portant sur des tables existantes). Il n'est pas nécessaire de spécifier les attributs. 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 WHERE adresse = 'Paris' ;

L'effet de cette commande est de créer une table pilote-province ayant deux attributs et les n-uplets vérifiant la condition adresse 'Paris'.

Notion de valeurs nulles Une valeur nulle (NULL) indique un 'non renseignement' pour un enregistrement donné. Une chaîne vide ou un 0 ne donnant pas forcément l'intention voulue, NULL spécifie que la valeur de ce champ est indéfinie indépendamment du type.

Contraintes d'intégrité Une contrainte d'intégrité permet d'exercer un contrôle sur la valeur d'une colonne et vaêtre définie lors de la création d'une colonne de la manière suivante

```
contrainte d'intégrité de colonne ::=
[CONSTRAINT nom_de_contrainte]
{ [NOT] NULL |
{ UNIQUE | PRIMARY KEY}
| REFERENCES [schema.]table[[colonne)]
[ON DELETE CASCADE]
{EXCEPTIONS INTO [schema.]table | DISABLE}
```

Il est permis de nommer les contraintes au travers du mot-clé CONSTRAINT. Cette contrainte sera alors plus facile de manipulation (notamment au travers de la table USER_CONSTRAINTS du dictionnaire des donnnées)

NULL spécifie que la colonne peut contenir des valeurs nulles. A contrario, NOT NULL interdit l'insertion de valeurs nulles pour la colonne considérée.
UNIQUE interdit deux lignes ayant la même valeur pour la colonne (ORACLE pose un index sur la colonne)
PRIMARY KEY indique une colonne porteuse de contrainte de clé primaire. Les valeurs de la colonne doivent alors être uniques et non nulles. (pose d'un index sur la colonne de manière automatique par ORACLE)
CHECK indique une contrainte de domaine, la condition indiquée doit être vérifiée pour chacune des valeurs insérées.
REFERENCES définit une contrainte d'intégrité référentielle par rapport à une clé unique ou primaire. ON DELETE CASCADE est optionnelle et permet de maintenir la contrainte d'intégrité référentielle lors de la suppression de tuples.
EXCEPTIONS INTO désigne une table d'exception existante dans laquelle ORACLE place des informations sur les lignes qui violent une contrainte d'intégrité activée
DISABLE est un mot-clé permettant de désactiver la contrainte. Par défaut, ORACLE active cette contrainte.

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

```
CREATE [UNIQUE] INDEX <index>
ON [<propriétaire>.]<table> (liste <colonne> [ASC/DESC]);
```

Remarque : l'option UNIQUE spécifie que l'ensemble des attributs de l'index est un identifiant de la table.

Définition de vue

```
CREATE VIEW [<propriétaire>.]<vue> [liste (<colonne>) ] AS <commande SELECT> ;
```

Définition de synonymes

Nommer autrement une table ou une vue est possible par :

```
CREATE [PUBLIC] SYNONYM <synonyme> FOR [<propriétaire>.]<table>/<vue> ;
```

Modification de de la structure d'une table Modifier une table revient soit à ajouter un ou plusieurs attributs, soit à modifier le type d'un attribut, soit encore à supprimer un attribut.
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.

Ajout de nouveaux attributs dans une table : pour ajouter un attribut, il suffit de le nommer et de donner son type et éventuellement sa taille et des contraintes (NOT NULL) ;

```
ALTER TABLE <table> ADD (liste <colonne> <type> [NULL/NOT NULL]...) ;
```

Par exemple, ajout de l'attribut Type_Licence dans la table pilote se fait par : ALTER TABLE pilote ADD (Type_Licence CHAR (40)) ;
Modification du type d'un attribut : cette modification peut concerner

- la réduction de la taille : seulement si l'attribut ne contient que des valeurs 'nulles'
- 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.

La modification est réalisée par la commande :

```
ALTER TABLE <table> MODIFY (liste <colonne> [<type>] [NULL/NOT NULL] ) ;
```

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 <ancienne_table> TO <nouvelle_table> ;
```

La possibilité de définir des synomymes 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.

Mise à jour des données d'une table

Ajout d'un ou plusieurs n-uplet (lignes) littéraux dans une table :

```
INSERT INTO <table> [liste <colonne>] VALUES (liste >valeur>) ;
```

Ajout d'une ou plusieurs lignes sélectionnées à partir d'autres tables :

```
INSERT INTO <table> <commande SELECT>;
```

Mise à jour de données à l'intérieur d'une table :

```
UPDATE <table>
        SET liste <colonne>=<expression>
        [WHERE condition ];
```

Remarque : si la clause WHERE n'existe pas, la mise à jour concerne toutes les lignes de la table.

Suppression d'une table C'est une opération rare et qui est à manipuler avec prudence surtout lorsqu'il existe des vues définies sur cette table. Cette commande est utile pour supprimer des tables de travail.

```
DROP TABLE <table> [CASCADE CONSTRAINTS];
```

Elle a pour effet de supprimer la table spécifiée, et tous les index, synonymes correspondants. Il est possible de supprimer le synonyme d'une table :

```
DROP [PUBLIC] SYNONYM <synonyme>  ;
```

Suppression d'un index :

```
DROP INDEX <index> [ON <table>] ;
```

Suppression d'une vue :

```
DROP VIEW <vue> ;
```

Suppression de données à l'intérieur d'une table :

```
DELETE FROM <table> [WHERE <condition>] ;
```

Remarque : si la clause WHERE n'existe pas, la suppression concerne toutes les lignes de la table.

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 liste <droit>/ALL ON <ressource> TO liste <usager>/PUBLIC [WITH GRANT OPTION] ;
```

GRANT donne à un ou plusieurs usagers (ou à tous, PUBLIC) les droits d'accès spécifiés sur les ordres SQL suivants : ALTER, DELETE, INDEX, INSERT, SELECT, UPDATE, ALL.

Supprimer des droits

```
REVOKE liste <droit >/ALL ON <ressource> FROM liste <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 liste <user> [IDENTIFIED BY liste <passwd> ] ;
```

Chapitre 4

Concept de transaction

Contents

4.1 Définitions

Un SGBD se doit de mettre à la disposition de tous les utilisateurs un ensemble cohérent de données. Il est alors essentiel de garantir la cohérence des données lors de manipulations simultanées par différents utilisateurs. Les concepts de transaction et d'accès concurrents vont oeuvrer dans ce sens.

Définition du concept de transaction

L'usager manipule la base de données soit au travers d'un langage de requête, soit au travers des programmes qui font appel au SGBD. L'exécution d'une requête ou d'un programme fait naître au niveau du SGBD une occurrence de transaction.

Une transaction est une unité de traitement séquentiel (séquence d'actions cohérente), exécutée pour le compte d'un usager, qui appliqée à une base de données cohérente, restitue une base de données cohérente. Les opérations de la transaction doivent être soit exécutées entièrement, soit pas du tout, nous amenant à définir le début et la fin d'un transaction.

Début de la transaction

Il est défini de manière implicite par la première commande SQL exécutée ou par la fin d'une transaction précédente (par annulation ou validation de cette dernière).

Fin de la transaction

Elle est soit implicite, soit explicite :

- fin explicite d'une transaction à l'aide des commandes COMMIT (validation des opérations élémentaires) et ROLLBACK (annulation des opérations élémentaires))
- fin implicite d'une transaction
  - exécution d'une commande de définition de données (CREATE, ALTER, RENAME et DROP) : toutes les opérations exécutées depuis le début de la transaction sont validées
  - fin normale d'une session ou d'un programme avec déconnexion d'Oracle : la transaction est validée
  - fin anormale d'un programme ou d'une session (sortie sans déconnexion d'Oracle) : la transaction est annulée

Découpage d'une transaction

Dans Oracle, il est possible de découper une transaction en insérant des points de repère (save-points). Ces points de repère permettent ensuite d'annuler une suite d'opérations.

Par exemple :

```
insert into EMP (num,nom) values (1,'martin');
insert into EMP (num,nom) values (2,'miller');
insert into EMP (num,nom) values (3,'regnier');
SAVEPOINT  Les_trois_premiers ;
insert into EMP (num,nom) values (4,'morand');
insert into EMP (num,nom) values (5,'kessel');
insert into EMP (num,nom) values (6,'royer');
ROLLBACK TO Les_trois_premiers ;
```

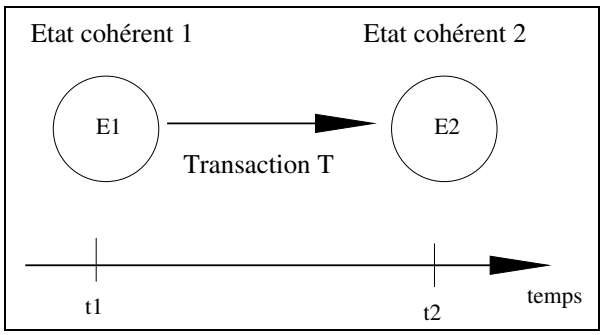


FIGURE 4.1 – Modèle de la transaction

4.2 Mécanisme de gestion des transactions (transaction ACID)

Le mécanisme de gestion des transactions doit notamment assurer que chacune des transactions possède les propriétés suivantes :

- **Atomicité** : lors d'une exécution d'une transaction, toutes ses actions sont exécutées ou bien aucune ne l'est.
- **Cohérence** : les modifications apportées à la b.d lors d'une transaction doivent être valides c'est à dire respecter les contraintes d'intégrité.
- **Isolation** : chaque transaction est isolée, de manière à éviter des incohérences lors d'exécutions concurrentes
- **Durabilité ou Permanence** : les effets d'une transaction qui s'est exécutée correctement doivent survivre à une panne

4.2.1 Isolation d'une transaction et concurrence

Même si les transactions sont cohérentes, leur entrelacement peut conduire à des incohérences globales. Trois exemples vous illustrent ces incohérences potentielles.

Illustration

TEMPS	T1	ETAT DE LA BASE	T2
t1	lire(X)	(X=100)	—
t2	—		lire(X)
t3	X := X+100		—
t4	—		X := X+200
t5	écrire(X)	(X=200)	—
t6	—	(X=300)	écrire(X)

FIGURE 4.2 – Exemple 1 d'incohérence liée à un entrelacement de transactions

Les transactions T1 et T2 exécutent une suite d'actions (lecture, affectation, écriture) concurrentes sur la variable X. La variable X a la valeur 300 à la fin des deux transactions alors que sa valeur aurait été de 400 si les deux transactions T1 et T2 s'étaient succédées.

Temps	T1	Etat de la base	T2
t1	X :=10	(X=5, Y=10)	—
t2	écrire(X)	(X=10, Y=10)	—
t3	—		X := 30
t4	—	(X=30, Y=10)	écrire(X)
t5	—		Y :=60
t6	Y :=20	(X=30, Y=60)	écrire(Y)
t6	écrire(Y)	(X=30, Y=20)	—

Figure 4.3 – Exemple 2 d'incohérence liée à un entrelacement de transactions

La base de données est soumise à la contrainte d'intégrité Y=2X (X=5 et Y=10 initialement). Les deux transactions T1 et T2 respectent la contrainte de manière individuelle. Leur entrelacement aboutit à un conflit sur la contrainte d'intégrité. La base de données se trouve, de fait, dans un état incohérent (X=30, Y=20 au final).

Temps	T1	Etat de la base	T2
t1	lire(X)	(X=5, Y=10)	—
t2	—	(X=5, Y=10)	—
t3	—		X := 30
t4	—	(X=30, Y=10)	écrire(X)
t5	—		Y :=60
t6	—	(X=30, Y=60)	écrire(Y)
t6	lire(Y)		—

Figure 4.4 – Exemple 3 d'incohérence liée à un entrelacement de transactions

La contrainte d'intégrité est toujours Y=2X (X=5 et Y=10 initialement). La base est cohérente après les deux transactions mais T1 a lu des valeurs incohérentes (X=5 et Y=60)

Isolation

La base de données est découpée en granules (niveau de granularité : tuple, page, table fixé par l'administrateur). L'isolation d'un élément dans une transaction est obtenu par le mécanisme de verrou (lock). Ces verrous sont définis par deux opérations :

- verrouiller(A) : cette opération oblige toute transaction à attendre le déverrouillage de l'élément A si elle a besoin de cet élément
- déverrouiller(A) : la transaction effectuant cette opération libère le verrou qu'elle avait obtenu sur A et permet à une autre transaction candidate en attente d'obtenir le sien.

L'utilisation de verrou entraîne éventuellement la mise en attente de transaction. Ceci a pour conséquence deux types de problèmes :

- la famine : lorqu'un verrou est relâché sur A, le système choisit parmi les transactions candidates en attente. Si les transactions sont liées au niveau de priorité de l'utilisateur, certaines transactions attendront longtemps. La solution est alors d'instaurer une file d'attente et de gérer les demandes de verrou en respectant l'ordre d'entrée dans la file.
- l'interblocage (deadlock) : Cette situation se présente lorsqu'un ensemble de transactions attendent mutuellement le déverrouillage d'éléments actuellement verrouillés par des transactions de cet ensemble. Par exemple, si T1 obtient un verrou sur A, puis T2 obtient un

verrou sur B et qu'enfin T1 demandant un verrou sur B soit mise en attente, la demande ultérieure par T2 d'un verrou sur A entraîne l'interblocage des deux transactions. Les solutions consistent la plupart du temps à faire intervenir le système qui "tue" les transactions.

Transactions sérialisables

Définition Une exécution d'un ensemble de transactions est sérialisable si et seulement si elle est équivalente à une exécution séquentielle (ou série) de transactions. Qand les transactions sont arbitraires, la sériabilité est en effet la seule à pouvoir assurer un bon "entrelacement".

Réalisation La sériabilité peut être obtenue en imposant aux transactions que tous les verrouillages précèdent tous les déverrouilages. Les transactions sont dites à deux phases : une phase d'acquisition des verrous puis une phase de libération.

4.2.2 Atomicité d'une transaction

La gestion de transaction nécessite la propriété d'atomicité d'une transaction c'est à dire d'assurer que les opérations mêmes les plus complexes englobées au sein d'une transaction ne soient perçues que comme une opération unique. De plus il faut que toutes les modifications liées à cette opération soient effectuées en bloc ou annulées en bloc, de telle sorte que l'utilisateur sache, quelles que soient les circonstances, l'état des données.
C'est le système qui est à même d'assurer l'atomicité des transactions. Le modèle général de la transaction sera le suivant :

Tdébut
—
Actions isolation, atomicité (panne=>défaire)
—
Tfin
Validation : calcul de la validité de la transaction - certification
Point de validation (commit)
Permanence (panne =>refaire éventuellement)
Vrai fin de transaction

Ce que l'on appelle le point de validation (en anglais commit) est fixé pour toute transaction. Des "calculs" vont s'exécuter au cours de ce point de validation de mnaière à accepter ou à rejeter la transaction finissante (certification). Les valeurs modifiées sont ensuite recopiées dans la base.
Comme une transaction peut ne pas se terminer normalement soit à cause de pannes diverses, soit par intervention du système lors d'un interblocage, nous pouvons raffiner les définitions :

- avant un point de validation, une panne entraîne la perte (totale à cause de l'atomicité) de la transaction,
- après le point de validation, la transaction sera visible (au bout d'un certain temps) par les autres.

Cette méthode s'appelle la **validation à deux phases**. Elle suppose souvent l'existence d'une mémoire stable, dans laquelle au point de validation, les nouvelles valeurs devront être enregistrées. Une transaction ayant atteint son point de transaction sera dite **validée** (elle ne peut plus être remise en cause). Une transaction n'ayant pas encore atteint son point de validation sera dite **vivante**.

4.2.3 Permanence

Le problème de la permanence est de faire en sorte que lorsqu'une transaction a atteint son point de validation, les effets de la transaction soient conservées sur la base quelles que soient les circonstances. La solution est obtenue par l'intermédiaire de fichiers journaux qui conservent la trace des transactions successives qui ont été effectuées sur la base.
Lors d'une transaction qui effectue une mise à jour sur la base, la base passe par un ancien état à un nouvel état et on conserve dans le journal l'identification des éléments modifiés, leur ancienne valeur et leur nouvelle valeur.

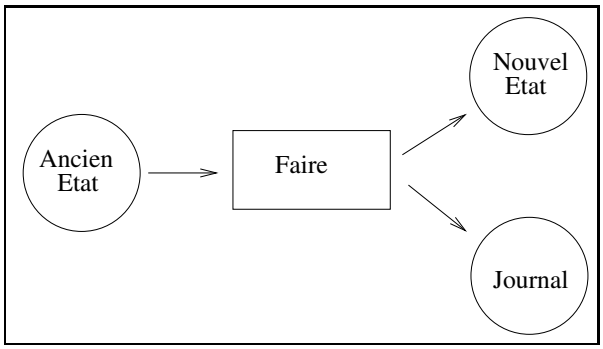


FIGURE 4.5 – Mécanisme de journalisation

Lors d'une panne, si la transaction doit être défaite, on se sert du nouvel état et de l'ancienne valeur se trouvant dans le journal pour reconstituer l'ancien état.

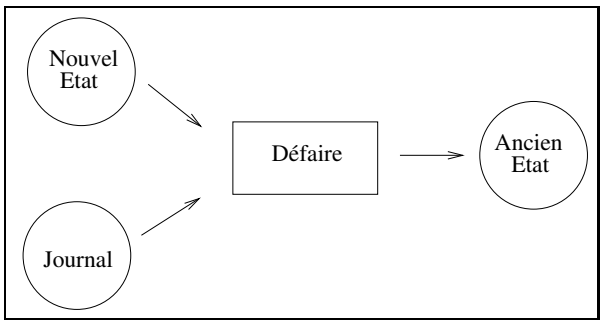


FIGURE 4.6 – Mécanisme de journalisation en cas de panne

Si la transaction doit être refaite, il faut partir de l'ancien état et du journal pour reconstruire le nouvel état.

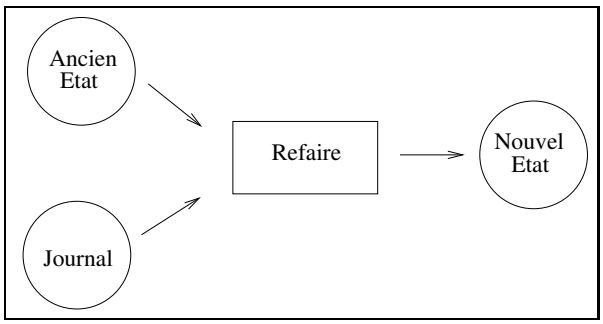


FIGURE 4.7 – Mécanisme de journalisation en cas de panne

Chapitre 5

Théorie de la conception des BD relationnelles

Contents

5.1 Introduction

Nous avons appris auparavant à manipuler les bases de données relationnelles, nous allons nous intéresser maintenant à la manière d'élaborer une "base relationnelle". Nous formulons l'hypothèse de départ que la phase d'analyse préalable nous a fourni un schéma relationnel "universel" : $\mathcal{R}$ défini sur un ensemble d'attributs U, plus un ensemble de prédicats sur la sémantique des attributs, plus un ensemble de contraintes d'intégrité.

5.2 Difficultés liées au caractère relationnel de la modélisation

Avant de se pencher sur les "formes normales", reprenons les "pièges" auxquels expose la structure relationnelle. Si l'on modélise mal, on peut tomber sur des répétitions, des inaptitudes, des pertes d'informations. Nous donnons l'exemple d'une modélisation relative au domaine bancaire : $\mathcal{R}$(nomAgence,numPrêt,client,montant,chiffreAffaires) correspondant au prédicat : "Une agence de nom nomAgence possède un chiffre d'affaires et consent des prêts de numéros donnés à des clients pour un montant déterminé".
Les contraintes suivantes (dépendances fonctionnelles) sont établies :

- nomAgence → chiffreAffaires
- client numPrêt → montant

Soit une relation du schéma notée R

nomAgence	numPrêt	client	montant	chiffreAffaires
Montpellier	17	Dupont	1000	90M
Montpellier	23	Durand	2000	90M
Béziers	15	Nestor	1500	45M
Nîmes	20	Adam	500	85M
Béziers	16	Achille	2000	45M
Nîmes	18	Désiré	1500	85M
...	...	...	...	...

FIGURE 5.1 – Relation R

Divers problèmes

- On veut insérer un nouveau client "Amanda" qui contracte le prêt 30 de montant 1500 auprès de l'agence de Béziers, il faut rajouter aussi l'information redondante relative au chiffre d'affaires 45M.
- Si une agence change de chiffre d'affaires, tous les n_uplets relatifs à cette agence doivent subir la modification.
- Si une nouvelle agence se crée sans encore aucun client, comment insérer l'information sans valeur nulle ? Si on tolère les valeurs nulles, les contraintes ne sont plus respectées.

Idée de décomposition Le schéma initial est décomposé en sous-schémas, mais plusieurs décompositions sont alors envisageables.

Première décomposition envisagée :
$\mathcal{R}1$(nomAgence, chiffreAffaires) et $\mathcal{R}2$(nomAgence, numPrêt, client, montant)
$\mathcal{R}1$ = ΠnomAgence, chiffreAffaires($\mathcal{R}$)
$\mathcal{R}2$ = ΠnomAgence,numPrêt,client,montant($\mathcal{R}$)
Avec $\mathcal{R} = \mathcal{R}1 \bowtie \mathcal{R}2$ et les contraintes sont toujours valables.

Seconde décomposition envisagée :
On peut aussi envisager la décomposition de $\mathcal{R}$ en
$\mathcal{R}1$(nomAgence, numPrêt, montant) et $\mathcal{R}2$(client, montant) et $\mathcal{R}3$(nomAgence, chiffreAffaires) dont une occurrence est donnée ci-dessous :

nomAgence	numPrêt	montant
Montpellier	17	1000
Montpellier	23	2000
Béziers	15	1500
Nîmes	20	500
Béziers	16	2000
Nîmes	18	1500
...	...	...

FIGURE 5.2 – R1

CLIENT	MONTANT
Dupont	1000
Durand	2000
Nestor	1500
Adam	500
Achille	2000
Désiré	1500
...	...

FIGURE 5.3 – R2

NOMAGENCE	CHIFFREAFFAIRES
Montpellier	90M
Béziers	45M
Nîmes	85M
...	...

FIGURE 5.4 – R3

Requêtage Supposons que l'on souhaite poser la requête suivante : "quelles sont les agences auprès desquelles Achille a emprunté de l'argent ?" à la base relationnelle provenant de la deuxième décomposition.
En algèbre relationnelle, cela s'exprime sous la forme $\Pi_{nomAgence}(\sigma_{client='Achille'}(\text{R2}) \bowtie \text{R1})$
La jointure s'opère sur l'attribut commun montant et la requête donne comme résultat Montpellier et Béziers. De manière informelle, nous voyons que cette deuxième décomposition n'est pas satisfaisante. Dans la première décomposition, l'intersection des ensembles d'attributs des schémas relationnels $\mathcal{R}1$ et $\mathcal{R}2$ donne nomAgence clé de $\mathcal{R}1$. Dans la deuxième décomposition, l'intersection des ensembles d'attributs des schémas relationnels $\mathcal{R}1$ et $\mathcal{R}2$ donne montant. Or un client peut emprunter dans une même agence le même montant pour des prêts différents, ou bien emprunter le même montant dans des agences différentes, ...
Nous nous attacherons donc à décrire une méthodologie amenant à une **"bonne décomposition"**.

5.3 Normalisation sous dépendances fonctionnelles

Les dépendances fonctionnelles constituent un type important de contrainte d'intégrité intervenant dans une structure relationnelle et vous ont déjà été présentés dans le cadre de ce cours. Nous vous en rappelons cependant la définition. Soit $\mathcal{R}$(A1,A2,....An) un schéma de relation et G et D deux sous-ensembles de l'ensemble A1,A2,...,An. On dit que G détermine fonctionnellement ou identifie D, que D dépend fonctionnellement de G ou est identifié par G et on note G $\rightarrow$ D si pour toute relation R de $\mathcal{R}$ acceptable, tous les n-uplets u,v de R qui ont même valeur dans G ont aussi même valeur dans D.

ssi ($\forall$ R instance de $\mathcal{R}$) ($\forall$ u,v $\in$ R tel que u[G] = v[G] $\Rightarrow$ u[D] = v[D])

En d'autres termes, si deux n-uplets de R coïncident sur G alors ils coïncident sur D. u[G] note la projection de u sur G.
Remarque :
si DG est l'ensemble des attributs de $\mathcal{R}$, on ne peut avoir deux n-uplets u,v $\in$ R instance de $\mathcal{R}$ coïncidant sur G car sinon ils seraient égaux. Or les relations sont des ensembles, un n-uplet ne figure qu'une seule fois (doublons interdits).

Exemple : Soit le schéma relationnel EmploiDuTemps(professeur,jour,heure,salle,classe,matière) associé au prédicat intentionnel "tout enseignement d'une matière est effectué par un professeur pour une classe donnée dans une salle donnée à un jour et une heure donnés".
L'ensemble des dépendances fonctionnelles suivant peut lui être associé :
{
professeur → matière : un professeur n'enseigne qu'une seule matière
professeur jour heure → classe : un professeur à un jour et une heure donnés enseigne à une seule classe
jour heure salle → professeur : à un jour et une heure donnés et dans une salle donnée ne peut se trouver qu'un seul professeur
jour heure classe → salle : à un jour et une heure donnés et pour une classe donnée ne peut être associée qu'une seule salle
}
Notons que, selon l'observateur qui modélise, l'ensemble de dépendances fonctionnelles peut être différent : {professeur → matière ; jour heure classe → professeur ; jour heure salle → classe}
Le problème sera de savoir si les ensembles de DF aboutissent au même schéma normalisé.

5.3.1 Equivalence entre ensembles de DF

Fermeture transitive d'un ensemble de DF

Définitions

- *Conséquence logique*
  Etant donné un schéma relationnel $\mathcal{R}$(U,F), une dépendance fonctionnelle f sera une conséquence logique de F (au sens des formules logiques) si f est vérifiée pour toute réalisation R de $\mathcal{R}$.
- *Fermeture transitive d'un ensemble de DF*
  Etant donné un ensemble F de DF, d'autres DF peuvent en être la conséquence logique. Comme les DF sont en nombre fini (si U est fini), l'ensemble des conséquences existe et s'appelle la **fermeture transitive** de F, notée $\mathbf{F}^+$.
- *Exemple*
  Si (1) X → Y et (2) Y → Z sont deux DF sur $\mathcal{R}$ alors X → Z est une DF sur $\mathcal{R}$.

- *Démonstration*
  En effet, soit $\mathcal{R}$ contenant les attributs XYZ,
  Si t et t' sont tels que t[X]=t'[X] alors (1) t[Y]=t'[Y] et (2) t[Z]=t'[Z].
  Il existe plusieurs moyens de dériver toutes les conséquences d'un ensemble de DF, nous en citerons un, celui utilisant des règles de dérivation appelées *axiomes d'ARMSTRONG.*

Règles principales

- DF1 **rélexivité** si Y ⊆ X ⊆ U alors X → Y
- DF2 **augmentation** si X → Y et Z est un ensemble d'attributs tel que Z ⊆ U, alors XZ → YZ
- DF3 **transitivité** si X → Y et Y → Z alors X → Z

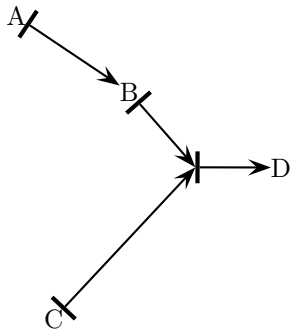
Nous admettons sans le démontrer que l'ensemble de ces règles de dérivation constituent un ensemble valide (sain et consistant) et complet.

Définition

Un système de règles de dérivation est valide (sain ou consistant) si lorsque f est dérivée de F à l'aide du système de règles, f est une conséquence logique de F (le sytème de règles ne permet donc d'engendrer que des DF valables).
Un système de règles de dérivation est complet si lorsque f est une conséquence logique de F alors f est dérivable de F en utilisant ce système de règles.
On utilise aussi des conséquences de ces règles qui sont utiles en pratique lors des manipulations de DF.

Fermeture transitive d'un ensemble d'attributs par rapport à un ensemble de DF

Définition L'ensemble des attributs directement atteignables à partir de X par rapport à F est appelé fermeture transitive des attributs de X par rapport à F et notée X^+.
Exemple : Soit F = {A → B, BC → D} et l'hypergraphe correspondant



$A^+/F = AB$
$B^+/F = B$
$C^+/F = C$
$D^+/F = D$
$BC^+/F = BCD$
$AC^+/F = ABCD$

Il existe un algorithme général simple pour calculer X^+.

Algorithme Fermeture_Trans(X,F,X^+)
{ calcule dans X^+ la fermeture de X par rapport à F, ensemble de DF}
$X^+ := X$; ENU := F ; {ENU ensemble des DF non utilisées}
tant que <∃ une DF G → D dans ENU tel que G ∈ X^+>
faire choisir (et enlever) un arc G → D de ENU tel que G ∈ X^+
$X^+ := X^+ \bigcup \{D\}$;

Remarque : d'autres algorithmes plus sophistiqués ont été développés.
Théorème :
Soit F un ensemble de DF, la dépendance G → D est dérivée de F (G → D ∈ F^+) ssi D ∈ G^+/F.
Applications :
Par définition, deux ensembles de dépendances fonctionnelles F et F' sont **équivalents**
ssi $F^+ = F'^+$
ssi $F^+ \in F'^+$ et $F'^+ \in F^+$
ssi F ∈ F'^+ et F' ∈ F^+
ssi toute DF de F ∈ F'^+ et toute DF de F' ∈ F^+
ssi pour toute DF de F, G → D alors G^+ par rapport à F' contient D et
pour toute DF de F', G → D alors G^+ par rapport à F contient D
Grâce au théorème précédent, on ramène le calcul de la fermeture transitive de l'ensemble des dépendances fonctionnelles F à celui des fermetures des parties gauches des dépendances fonctionnelles de F.

Eliminer la redondance : L'objectif est d'arriver à une **Couverture Irredondante Minimale**

a. **redondance d'une DF (G → D) par rapport à un ensemble F de DF**
Définition : G → D est redondant par rapport à F
ssi D ∈ G^+ par rapport à (F - {G → D})
ssi $(F - \{G \to D\})^+ = F^+$
Pour chasser la redondance des dépendances fonctionnelles, rappelons qu'il faut décomposer

G → D1 D2 D3 ... Dn en n dépendances fonctionnelles **élémentaires** (un seul attribut en partie droite) : G → D1, G → D2, G → D3 ...

b. **minimisation des parties gauches de DF**
Définition : A ∈ G est redondant dans G → D par rapport à F
ssi **D ∈ {G-{A}}$^+$ par rapport à F**
Une dépendance fonctionnelle est dite **minimale** si elle ne contient aucun attribut redondant en partie gauche

c. **couverture irredondante minimale (CIM) d'un ensemble F de DF**
C'est un ensemble F' de DF équivalent à F qui ne contient aucune DF redondante et dont les parties gauches sont minimales. Pour en obtenir une (il n'y a pas forcément unicité), il faut appliquer les tests a et b dans cet ordre.

Clés de schéma relationnel

a **Surclé K d'un schéma relationnel**
Définition : K est surclé (ou identifiant) de $\mathcal{R}$ sur U ssi K → U est une DF sur $\mathcal{R}$.
Deux n-uplets distincts ne peuvent avoir la même surclé (sinon ils coïncident entièrement et une réalisation R de $\mathcal{R}$ contient 2 tuples égaux). U est toujours une surclé (triviale).

b **Clé d'un schéma relationnel**
Définition : K est une clé ssi K est une surclé de $\mathcal{R}$ et K est minimale (ne contient pas strictement une autre surclé K').
En d'autres termes K est clé ssi ¬ ∃ A ∈ K tel que K-{A} → U

c **Calcul d'une clé**
On esaie de minimiser la partie gauche de U → U.
Remarque : Une clé sera représentée par un soulignement continu au niveau du schéma relationnel associé.
Exemple : EMPLOI_DU_TEMPS(Professeur,Jour,Heure,Salle,Classe,Matière)

Algorithme Clé(U,F,K)
{calcule une clé K d'un schéma relationnel $\mathcal{R}$ sur U avec un ensemble de DF F}
K := U ; {surclé}
tant que ∃ un attribut A ∈ K tel que K-{A} → U
faire choisir un tel attribut A ;
K := K - {A} ;

Il existe d'autres algorithmes capables de calculer l'ensemble des clés possibles pour un schéma $\mathcal{R}$.

5.3.2 Propriétés d'une décomposition

La normalisation va permettre de décomposer un schéma relationnel $\mathcal{R}$(U,F) en un ensemble de schémas relationnels ayant une bonne "forme" incorporant l'ensemble des dépendances fonctionnelles F et réduisant la redondance des données et par suite réduisant les diverses difficultés de mise à jour citées initialement.
On appelle décomposition d'un schéma $\mathcal{R}$ défini sur U en sous-schémas $\mathcal{R}_i$ définis sur U_i sous ensembles de U ssi **U = ⋃ des U**$_i$ (recouvrement de l'ensemble des attributs).
Les décompositions effectuées doivent présenter un certain nombre de propriétés que nous allons détailler.

Décomposition sans perte

Définitions : On appelle **décomposition sans perte** d'un schéma $\mathcal{R}$ soumis à un ensemble de dépendances fonctionnelles F en sous-schémas $\mathcal{R}_i$ définis sur {U_i} ssi pour toutes les instances R de $\mathcal{R}$ vérifiant F, on a R = Π_{U1}(R) ⋈ Π_{U2}(R) ⋈ ... ⋈ Π_{Un}(R))

Test d'une décomposition

Algorithme Test_Sans_Perte($\mathcal{R}$,{U$_i$},F) : booléen
{retourne vrai si {U_i} est une décomposition sans perte}
construire un tableau T de n colonnes (n étant le cardinal de U) et de k lignes (k est le nombre de sous-schémas de la décomposition) : la ligne i correspond au schéma $\mathcal{R}_i$ et la colonne j correspond à l'attribut j de U.

Pour tous les attributs Aj si Aj $\in \mathcal{R}_i$ mettre T(i,j)=aj sinon T(i,j)=bij
Pour toute DF G D de F **faire**
Pour toutes les lignes qui ont même partie gauche (sur tous les attributs Aj de G) **faire** coïncider les attributs de D de la manière suivante :
— si l'un d'entre eux est de la forme aj, mettre les autres à aj
— sinon les mettre égaux à l'un d'entre eux (bij par exemple)
Résultat := la table T possède au moins une ligne remplie de ai.

Les théorèmes suivants peuvent aussi permettre de vérifier qu'une décomposition est sans perte.

Théorème 1
Si $\mathcal{R}$ soumis à l'ensemble de DF F se décompose en $\{\mathcal{R}_1, \mathcal{R}_2\}$ alors cette décomposition est sans perte par rapport à F ssi
(U1 $\bigcap$ U2) $\rightarrow$ (U1 - U2) ou (U1 $\bigcap$ U2) $\rightarrow$ (U2 - U1)
ssi les attributs communs sont clés de l'une ou de l'autre des schémas relationnels décomposés

Théorème 2
Si $\mathcal{R}$ soumis à l'ensemble de DF F se décompose en {U1, U2, ..., Un} sans perte
et si U1 soumis à Π_{U1}(F) se décompose en {V1,V2} sans perte,
alors {V1, V2, U2, ..., Un} est une décomposition sans perte de $\mathcal{R}$ soumis à F.

Décomposition avec préservation des DF

Définitions : La projection d'un ensemble de DF sur un ensemble d'attributs est l'ensemble des DF de F^+ qui ont tous leurs attributs dans Z.
Π_Z(F) ={G $\rightarrow$ D | G $\rightarrow$ D $\in F^+$ et GD $\in$ Z}
Une décomposition préserve un ensemble F de DF ssi $\{\bigcup \Pi_{\mathcal{R}_i}(F)\}^+ = F^+$ ssi "l'union des projections est équivalent à l'ensemble initial".
Exemples : $\mathcal{R}$ sur ABC soumis à F={A $\rightarrow$ B, B $\rightarrow$ C}

a) la décomposition sur AC, BC préserve-t'elle les DF ?
   F^+ sans les dépendances triviales = {A $\rightarrow$ B, B $\rightarrow$ C, A $\rightarrow$ C}
   Π_{AC}(F) ={A $\rightarrow$ C}, Π_{BC}(F) ={B $\rightarrow$ C}
   Π_{AC}(F) $\bigcup$ Π_{BC}(F) ={A $\rightarrow$ C B $\rightarrow$ C}
   et A $\rightarrow$ B n'est pas retrouvée donc la décomposition ne préserve pas les DF
b) au contraire la décomposition AB, BC les préserve.

Il existe des méthodes pour vérifier la préservation des DF dans une décomposition mais nous ne les examinerons pas au niveau de ce cours.

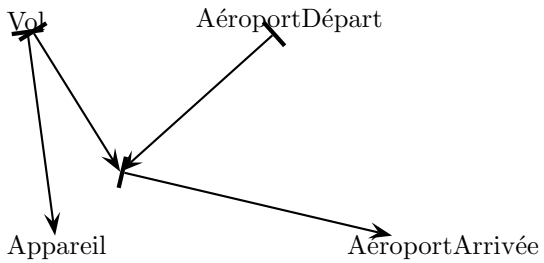
5.4 Formes normales

5.4.1 1ère FN

Un schéma relationnel $\mathcal{R}$ est en première forme normale si tous les attributs de U sont atomiques (ou monovalués).

5.4.2 2ème FN

Un schéma relationnel $\mathcal{R}$ défini sur U et sur un ensemble de dépendances fonctionnelles F est en 2FN ssi tout attribut n'appartenant pas à une clé dépend directement d'une clé.
Exemple : Soit $\mathcal{R}$(Appareil, Vol, AéroportDépart, AéroportArrivée)
avec F={Vol $\rightarrow$ Appareil, Vol AéroportDépart $\rightarrow$ AéroportArrivée}.



La clé de $\mathcal{R}$ est Vol AéroportDépart.
AéroportArrivée dépend directement de la clé mais Appareil dépend d'une partie de la clé et non de la clé entière donc le schéma n'est pas en 2FN.

5.4.3 3ème FN

Un schéma relationnel $\mathcal{R}$ défini sur U et sur un ensemble de dépendances fonctionnelles F est 3FN

- ssi les parties gauches des dépendances fonctionnelles sont clés
- ssi pour les DF dont la partie gauche n'est pas clé, la partie droite appartient à une clé

En d'autres termes, un schéma de relation $\mathcal{R}$ soumis à F (CIM) est en 3ième forme normale ssi pour toute DF G $\rightarrow$ D de F,

- **soit G est une clé**
- soit D appartient à une clé

5.4.4 3ème BCNF (forme normale de Boyce-Codd)

un schéma de relation $\mathcal{R}$ soumis à F (CIM) est en BCNF ssi pour toute DF G $\rightarrow$ D de F, **G est une clé**.

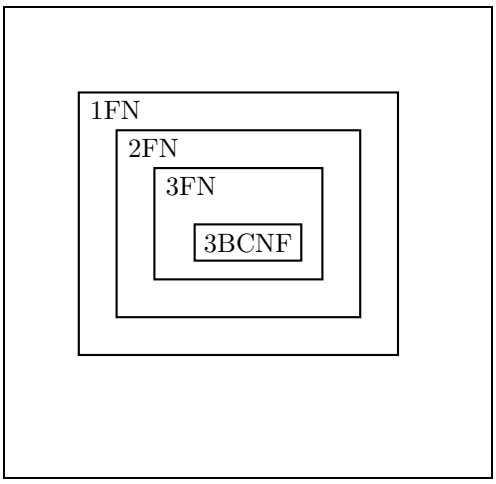


FIGURE 5.5 – Imbrication des diverses formes normales

5.4.5 Décomposition en 3FN

Théorème :
Tout schéma relationnel peut se décomposer en schémas sous $3^{\text{ième}}$ forme normale sans perte et avec préservation des dépendances fonctionnelles.
Il existe divers algorithmes de décomposition (nous n'en étudierons qu'un "classique" dû à Bernstein).

Algorithme de décomposition en $3^{\text{ième}}$ forme normale
l'algorithme prend en entrée un schéma $\mathcal{R}$ soumis à F et donne en résultat $\{\mathcal{R}_i(\mathrm{U}_i,\mathrm{F}_i)\}$ tel que tout $\mathcal{R}_i$ est en 3FN. Le schéma relationnel $\mathcal{R}$ est lui même le résultat d'une phase de modélisation.
calculer une CIM (F1) à partir de l'ensemble de toutes les dépendances fonctionnelles élémentaires de F en chassant la redondance des attributs de partie gauche, puis les dépendances fonctionnelles redondantes.
trouver les clés de F1
regrouper les DF de F1 ayant même partie gauche. Fabriquer les schémas de relation maximaux par l'inclusion ensembliste
$\mathcal{R}_i = \{\mathrm{U}_i = \mathrm{Max}(\mathrm{G} \bigcup \mathrm{D})\,;\, \{\ \mathrm{G}_j \rightarrow \mathrm{D}_j\,;\, \mathrm{G}_j \bigcup \mathrm{D}_j \in \mathrm{U}_i\}\}$
si aucune clé ne figure dans une des relations produites à l'étape précédente, ajouter un schéma de relation $\mathcal{R}_i = \{\mathrm{K}\,;\, \mathrm{K} \rightarrow \mathrm{K}\}$
Cet algorithme peut conduire à une décomposition en 3BCNF (meilleur des cas)