Bases de Données

Rosine CICCHETI, Lotfi LAKHAL, Sebastien NEDJAR

Rosine CICCHETI
Lotfi LAKHAL
Sebastien NEDJAR

Bases de données
1re année

Première partie

Introduction

1 Approche par application

On informatise les entreprises, les organisations (l'univers réel) en étant guidé par les programmes.

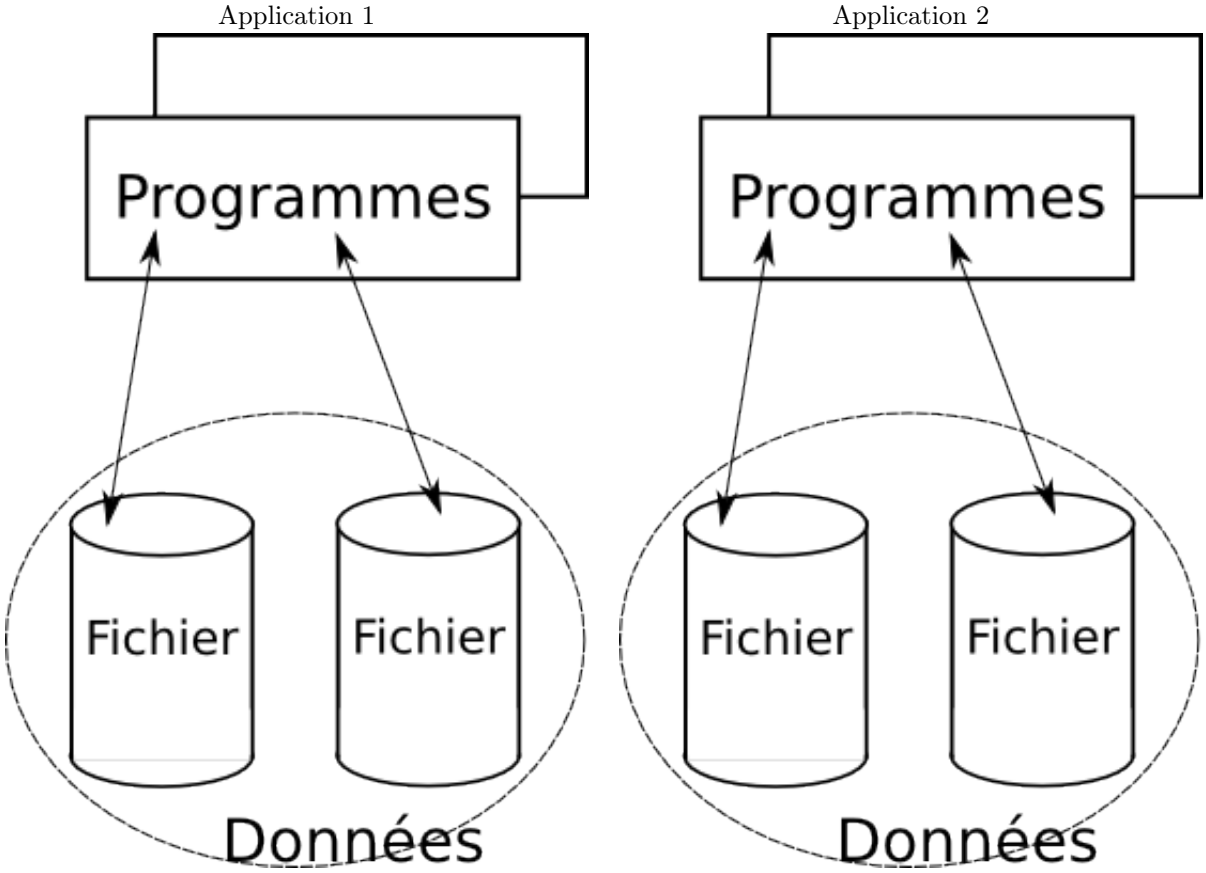

Figure 1 – Approche par application

Ces approches ont conduit à la « crise du logiciel ».

Inconvénients

- Redondance de données : les données sont dupliquées autant de fois qu'on en a besoin dans les différentes applications
  - perte de place : déperdition de stockage
  - danger d'incohérence des données lors des mises à jour
- Dépendance entre données et programmes d'où maintenance accrue
- Dépendance entre niveaux logique et physique : vision de l'utilisateur des données et ce qui est effectivement stocké sur son disque
- Difficulté de développer de nouvelles applications non prévues (informatique décisionnelle)

Avantage : démarche relativement facile à mettre en œuvre et progressive

Rosine CICCHETI
Lotfi LAKHAL
Sebastien NEDJAR

2 Approche base de données

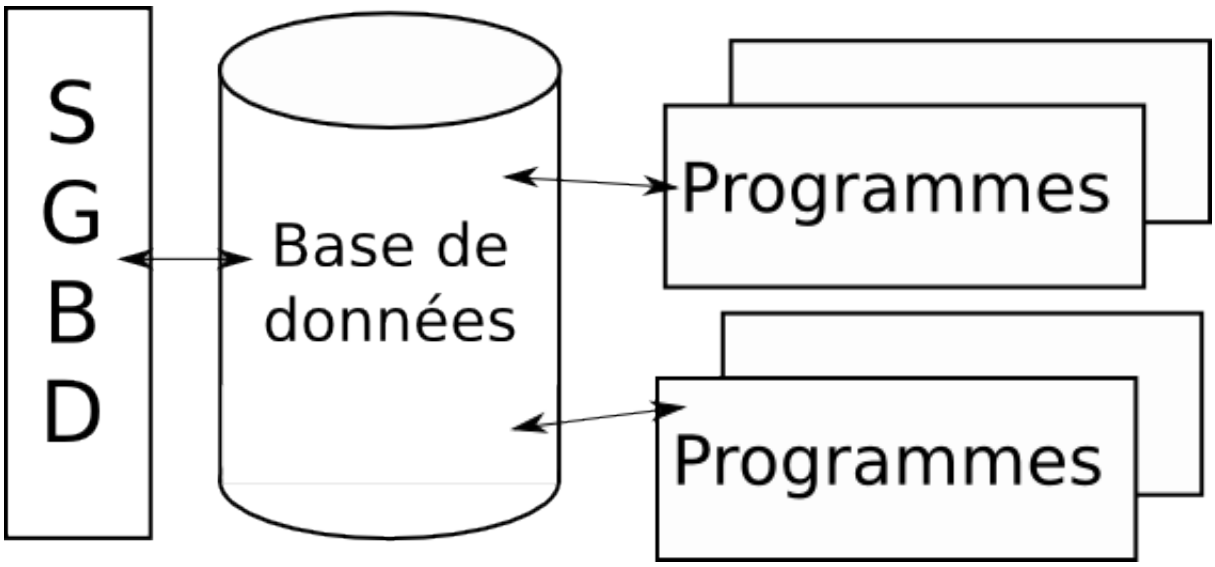

FIGURE 2 – Approche base de données

SGBD : Système de Gestion de Bases de Données

Avantages
- Élimination de la redondance de données
- Indépendance données/programmes
- Indépendance entre niveaux logique et physique
- Facilité de développer de nouvelles applications non prévues

Inconvénients démarche longue, coûteuse, difficile à mettre en œuvre

Modèles de bases de données
- Modèle hiérarchique (1960)
- Modèle Codasyl (1964)
- Modèle relationnel (1970)
- Modèle orienté objet (1990)

Rosine CICCHETI
Lotfi LAKHAL
Sebastien NEDJAR

Deuxième partie

Le modèle relationnel

3 Introduction

3.1 3 composantes

Les concepts relationnels

Domaine, relations, tuples, attributs, clef primaire, clef étrangère, relation statique, relation dynamique, schéma relationnel.

Algèbre relationnel

Projection, sélection, jointure, division, union, intersection, différence.

Contraintes d'intégrité relationnelle

Intégrité de domaine, inéligibilité de relation, intégrité de références.

3.2 Rappels mathématiques

Produit cartésien

Soient A et B deux ensembles tels que :

$$A \times B = \{(a, b)/a \in A, b \in B\}$$

Exemple :

$$A = \{1, 2, 3\}\ B = \{4, 5\}$$
$$A \times B = \{(1, 4), (1, 5), (2, 4), (2, 5), (3, 4), (3, 5)\}$$

Relation binaire

Soient A et B deux ensembles et $R \subseteq A \times B$ une relation entre les éléments de A et de B. R contient les couples de $A \times B$.

Exemple :

$$R_1 = \{a_1, b_1\}\ R_2 = \{(a_1, b_1), (a_2, b_2)\}$$

4 Concepts relationnels

4.1 Domaine sémantique

Un domaine sémantique est un ensemble de valeurs atomiques, ou simples. Chaque domaine est caractérisé par un type de base (numérique : entier, réel ; alphanumérique : string, date,...).

Exemple

— D_NOMAV = Domaine de nom des avions
— D_NOMPIL = Domaine de nom des pilotes

— D_NOMAV = {'A310','A330','A340','B727','B747'}
— D_NOMPIL = {'DUPONT','ROGER'}

Le domaine sémantique a comme objectif de vérifier la validité des comparaisons (évite de comparer D_NOMPIL avec D_NOMAV).

4.2 Relations/tuples

Soient $D_1, D_2, ..., D_n$, n domaines non nécessairement distincts. Une relation R est le sous-ensemble du produit cartésien de ces n domaines :

$$R \subseteq D_1 \times D_2 \times \cdots \times D_n$$

La relation en base de données est une relation n-aire avec des n-uplets comme éléments.

Exemple
- AVION
  - D_NUMAV
  - D_NOMAV
  - D_CAP
  - D_VILLE
- AVION⊆D_NUMAV×D_NOMAV×D_CAP×D_VILLE

Tuple représente un avion : (100,'A310',250,'MARSEILLE')

4.2.1 Représentation tabulaire d'une relation

AVION={(100,'A310',250,'MARSEILLE'),(200','B747',450,'PARIS')}

AVION	NUMAV	NOMAV	CAP	VILLE
	100	A310	250	MARSEILLE
	200	B747	450	PARIS

Table 1 – Représentation tabulaire de la relation AVION

Ici,
- AVION est l'en-tête
- NUMAV, NOMAV, CAP, VILLE sont les attributs
- Le corps est constitué des différents tuples
- Les tuples sont les différentes lignes du tableau
- Le corps est l'extension du schéma relationnel
- L'en-tête est le schéma de la relation/intention

4.3 Attributs

L'attribut est un rôle joué par un domaine dans une relation pour éliminer les ambiguïtés (D_VILLE×D_VILLE).
VOL⊆D_NUMVOL×D_VILLE×D_VILLE×D_HEURE×D_HEURE
Attribut NUMVOL, VILLE_DEP, VILLE_ARR, H_DEP, H_ARR
Un attribut joue un rôle dans un domaine, un domaine a un type défini.
Ainsi, (IT100, 'PARIS', 'LONDRES', 18, 20) avec les attributs
- D_NUMVOL ⇒ NUMVOL
- D_VILLE ⇒ VILLE_DEP
- D_VILLE ⇒ VILLE_ARR
- D_HEURE ⇒ HEURE_DEP
- D_HEURE ⇒ HEURE_ARR

Chaque attribut est défini sur un domaine.

4.4 Clef primaire

Une clef primaire est un attribut, ou groupe d'attributs, permettant d'identifier de manière unique les tuples de la relation.

Exemple NUMAV ne contient aucun doublon, et est ainsi le domaine de clef primaire de la relation AVION.

4.5 Domaine primaire

C'est le domaine dans lequel est défini un attribut, une clef primaire.

Exemple
- D_NUMAV : domaine primaire
- NUMAV : clef primaire

4.6 Clef étrangère

C'est un attribut, ou groupe d'attribut, défini sur un ou plusieurs domaines primaires, qui est clef primaire dans une autre relation.

Exemple
- AVION(NUMAV, NOMAV, CAP, LOC)
- VOL(NUMVOL, NUMAV#, NUMPIL#, VILLE_DEP, VILLE_ARR, HEURE_DEP, HEURE_ARR)
- PILOTE(NUMPIL, NOMPIL, ADR, SALAIRE)

4.7 Schéma relationnel

C'est le schéma de chaque relation, la liste des attributs. La clef étrangère sert à exprimer des liens sémantiques entre les relations du schéma relationnel. C'est l'ensemble du schéma de relations liées à travers des clefs primaires et étrangères. Chaque schéma de relation est décrit par le nom de la relation et la liste de ses attributs : AVION(NUMAV, NOMAV, CAP, LOC).

4.8 Autres concepts

Degré et cardinalité d'une relation

Le degré d'une relation est le nombre d'attributs d'une relation.
La cardinalité d'une relation est le nombre de ses tuples.

Relations dynamiques et statiques

Une relation dynamique est une relation qui contient une clef étrangère.
Une relation statique est une relation indépendante, qui ne contient pas de clef étrangère.

Attributs comparables ou compatibles

Deux attributs sont comparables ou compatibles s'ils sont définis sur le même domaine.

5 Contraintes d'intégrité relationnelles

5.1 Contraintes d'intégrité statique/structurelle

Ce sont les règles de cohérence de la base induite par les concepts relationnels. Elles sont invariantes et ne dépendant pas de l'application.

Contrainte de domaine

Toute valeur affectée à un attribut doit être dans le domaine de l'attribut.

Contrainte de relation

Toute valeur affectée à la clef primaire doit être unique et obligatoire (doit exister).

Contrainte de référence

Toute valeur affectée à la clef étrangère doit exister dans la clef primaire associée.

5.2 Contraintes dynamiques/applicatives

Les règles de cohérence de la base de données dépendent de l'application.

Exemple Pour un vol, on a VILLE_DEP<>VILLE_ARR et HEURE_DEP<>HEURE_ARR.

La contrainte statique est vérifiée automatiquement par le SGBD. Les contraintes dynamiques sont spécifiées et programmées par l'administrateur de la base de données.

6 Algèbre relationnel

6.1 Généralités

Opérateurs relationnels

Relation unaire Projection, séléction.

Relation binaire Jointure, division.

Opérateurs ensemblistes

- Union, qui s'apparente à un OR
- Intersection, qui s'apparente à un AND
- Différence, qui s'apparente à un NOT

6.2 Opérateurs relationnels

6.2.1 Projection

Définition Soit $R(U)$ une relation d'attributs U telle que
- $U = \{A_1, A_2, \ldots, A_n\}, X \subseteq U$
- $RS = \text{PROJECTION}(R/X)$
- $RS = \{t(X)/r \subset R\}$

PILOTE	NUMPIL	NOMPIL	ADR	SALAIRE
	100	Dupont	Marseille	5000
	200	Durand	Marseille	4000
	300	Martin	Nice	3000

TABLE 2 – Représentation tabulaire de la relation PILOTE

Exemple : quels sont les noms et adresses des pilotes ?

$RS = \text{PROJECTION}(PILOTE/NOMPIL, ADR)$

RS	NOMPIL	ADR
	Dupont	Marseille
	Durand	Marseille
	Martin	Nice

TABLE 3 – Projection de PILOTE sur NOMPIL et ADR

$RS = \text{PROJECTION}(PILOTE/ADR)$

RS	ADR
	Marseille
	Nice

TABLE 4 – Projection de PILOTE sur ADR

La projection élimine les doublons.

Propriétés
— Degré$(RS) \leq$ Degré(R)
— Cardinalité$(RS) \leq$ Cardinalité(R)

6.2.2 Sélection

Soient $R(U)$ une relation d'attributs $U = \{A_1, A_2, \ldots, A_n\}$ constante et Ai une valeur dans le domaine de A.

$$RS = \text{SELECTION}(R/Ai\theta\text{constante})$$

θ est l'un des opérateurs suivants : $=$, $<$, $>$, $<>$ [1], $\leq$, $\geq$.

Exemple : chercher les pilotes habitant à Marseille
$RS = \text{SELECTION}(PILOTE/ADR =' \text{MARSEILLE}')$

RS	NUMPIL	NOMPIL	ADR	SALAIRE
	100	Dupond	Marseille	5000
	200	Durand	Marseille	4000

TABLE 5 – Pilotes habitant à Marseille

1. opérateur de différence, équivalent à ! =

Propriétés
— Degré(RS) = Degré(R)
— Cardinalité(RS) $\leq$ Cardinalité(R)
On a ainsi les opérateurs suivants :
— La projection : sert à découper verticalement une relation
— La sélection : sert à découper horizontalement une relation
Toujours commencer par la sélection

Exemple : quels sont les noms des pilotes habitant à Marseille ?
$R1 = \text{SELECTION}(PILOTE/ADR =' Marseille')$
$RS = \text{PROJECTION}(R1/NOMPIL)$

RS	NOMPIL
	Dupont
	Durand

Table 6 – Sélection et projection

6.2.3 Jointure

La jointure est un opérateur binaire : elle nécessite deux relations.
Soient $R(U_1), S(U_2)$ deux relations d'attributs $U_1 = \{A_1, A_2, \ldots, A_n\}$ et $U_2 = \{B_1, B_2, \ldots, B_n\}$ et $A_i \in U_1, B_j \in U_2$ deux attributs compatibles.

$$RS = \text{JOINTURE}(R, S/A_i \theta B_j)$$
$$RS = \{(t, s)/t \in R, s \in S, t(A_i) \theta S(B_j)\}$$

avec θ un opérateur de comparaison.
Le schéma de RS contient tous les attributs de A et tous les attributs de S.

R	A	B	C
	a_1	b_1	c_1
	a_2	b_1	c_1
	a_3	b_2	c_1

S	A	D
	a_1	d_1
	a_1	d_2
	a_2	d_3
	a_2	d_4

RS	A	B	C	A	D
	a_1	b_1	c_1	a_1	d_1
	a_1	b_1	c_1	a_1	d_2
	a_2	b_1	c_1	a_2	d_3
	a_2	b_1	c_1	a_2	d_4

Table 7 – Jointure entre R et S sur A

Exemple : quels sont les noms des pilotes en service ?

PILOTE	NUMPIL	NOMPIL	ADR	SAL
	10	Dupont	Nice	5000
	20	Durand	Marseille	6000
	30	Dupont	Marseille	4000

VOL	NUMVOL	NUMPIL	NUMAV	VILLE_DEP	VILLE_ARR
	IT100	10	100	Marseille	Paris
	IT200	10	100	Paris	Marseille
	IT300	30	200	Toulous	Paris

Table 8 – Relations PILOTE et VOL

$R1 = \text{JOINTURE}(PILOTE, VOL/NUMPIL = NUMPIL)$
$RS = \text{PROJECTION}(R1/NOMPIL)$

R1	NUMPIL	NOMPIL	ADR	SAL	NUMVOL	NUMPIL	NUMAV	VILLE_DEP	VILLE_ARR
	10	Dupont	Nice	5000	IT100	10	100	Marseille	Paris
	10	Dupont	Nice	5000	IT200	10	100	Paris	Marseille
	30	Dupont	Marseille	4000	IT300	30	200	Toulouse	Paris

RS	NOMPIL
	Dupont
	Dupont

TABLE 9 – Jointure entre PILOTE et VOL sur NUMPIL

Différents types de jointures

Équijointure

$$RS = \text{JOINTURE}(PILOTE, VOL/NUMPIL = NUMPIL)$$

Jointure naturelle

Enlève la deuxième occurreence d'un attribut identique (ici, enlève NUMPIL).

RS	NUMPIL	NOMPIL	ADR	SAL	NUMVOL	NUMAV	VILLE_DEP	VILLE_ARR
	10	Dupont	Nice	5000	IT100	100	Marseille	Paris
	10	Dupont	Nice	5000	IT200	100	Paris	Marseille
	30	Dupont	Marseille	4000	IT300	200	Toulouse	Paris

TABLE 10 – Jointure naturelle entre PILOTE et VOL sur NUMPIL

Jointure par <>

$$RS = \text{JOINTURE}(PILOTE, VOL/NUMPIL <> NUMPIL)$$

RS	NUMPIL	NOMPIL	ADR	SAL	NUMVOL	NUMPIL	NUMAV	VILLE_DEP	VILLE_ARR
	10	Dupont	Nice	5000	IT300	30	200	Toulouse	Paris
	20	Durand	Marseille	6000	IT100	10	100	Marseille	Paris
	20	Durand	Marseille	6000	IT200	10	100	Paris	Marseille
	20	Durand	Marseille	6000	IT300	30	200	Toulouse	Paris
	30	Dupont	Marseille	4000	IT100	10	100	Marseille	Paris
	30	Dupont	Marseille	4000	IT200	10	100	Paris	Marseille

TABLE 11 – Jointure entre PILOTE et VOL avec NUMPIL<>NUMPIL

Autojointure

$$RS = \text{JOINTURE}(PILOTE, PILOTE/NUMPIL = NUMPIL)$$

RS	NUMPIL	NOMPIL	ADR	SAL	NUMPIL	NOMPIL	ADR	SAL
	10	Dupont	Nice	5000	10	Dupont	Nice	5000
	20	Durand	Marseille	6000	20	Durand	Marseille	6000
	30	Dupont	Marseille	4000	30	Dupont	Marseille	4000

TABLE 12 – Autojointure de PILOTE sur NUMPIL

6.2.4 Division

Soient $R(B, A)$ et $S(A)$ deux relations respectivement binaire et unaire, avec A_r et A_s définis sur le même domaine.

$$RS = \text{DIVISION}(R, S/A_r, A_s)$$

A_r et A_s n'ont pas forcément le même nom.

$$RS = \{t(B)/t \in R, \forall s \in S, (t(B), S(A)) \in R\}$$

Contraintes R a deux attributs, S un seul. Les attributs de la division doivent être compatibles.

$$RS = \text{DIVISION}(R, S/A, A)$$

R	B	A
	b_1	a_1
	b_1	a_2
	b_2	a_1

S	A
	a_1
	a_2

RS	B
	b_1

TABLE 13 – Division de R par S

b_1 est associé dans R à toutes les valeurs de A.

$$B = \{b_1, b_2\} \Rightarrow \{(b_1, a_1), (b_1, a_2)\}$$

Or b_2 n'est pas associé dans R à a_2 donc il n'est pas conservé.

Remarques

— *La division exprime « tous les », ou bien « au moins tous les »* $\Rightarrow \forall$

— *La jointure exprime « un », « il existe »* $\Rightarrow \exists$

Exemple : numéros des pilotes qui conduisent tous les avions de sa compagnie
NUMAV est l'attribut de la division.

$R1 = \text{PROJECTION}(AVION/NUMAV)$

$R2 = \text{PROJECTION}(VOL/NUMAV, NUMPIL)$

$RS = \text{DIVISION}(R1, R2/NUMAV, NUMAV)$

Les pilotes apparaissant dans le résultant sont associés à tous les avions de la compagnie.

6.3 Opérateurs ensemblistes

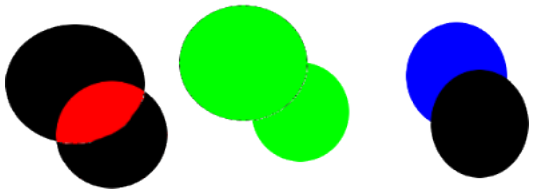

FIGURE 3 – Opérateurs ensemblistes

— ROUGE : intersection ($R \cap S$; $RS = \text{INTERSECTION}(R, S) = \{t/t \in R \wedge t \in S\}$)
— VERT : union ($R \cup S$; $RS = \text{UNION}(R, S) = \{t/t \in R \vee t \in S\}$)
— BLEU : différence ($R - S$; $RS = \text{DIFFERENCE}(R, S) = \{t/t \in R \wedge t \notin S\}$)

R et S sont uni-compatibles si :
— Degré(R) = Degré(S)
— Les attributs de R et S sont définis sur le même domaine

Exemple
Soient $R(A, B, C)$ et $S(D, E, F)$ des relations. R et S sont uni-compatibles si
— A est sur le même domaine que D
— B est sur le même domaine que E
— C est sur le même domaine que F

Exemple : noms des pilotes habitant Nice et gagnant 5000€

$R1 = \text{SELECTION}(PILOTE/ADR =' Nice')$
$R2 = \text{SELECTION}(PILOTE/SAL = 5000)$
$R3 = \text{INTERSECTION}(R1, R2)$
$RS = \text{PROJECTION}(R3/NOMPIL)$

$R1 = \text{SELECTION}(PILOTE/ADR =' Nice')$
$R2 = \text{SELECTION}(R1/SALAIRE = 5000)$
$RS = \text{PROJECTION}(R2/NOMPIL)$

Exemple : noms des pilotes qui n'ont jamais assuré de vol

$R1 = \text{PROJECTION}(VOL/NUMPIL)$
$R2 = \text{PROJECTION}(PILOTE/NUMPIL)$
$R3 = \text{DIFFERENCE}(R2, R1)$
$R4 = \text{JOINTURE}(PILOTE, R3/NUMPIL = NUMPIL)$
$RS = \text{PROJECTION}(R4/NOMPIL)$

Rosine CICCHETI
Lotfi LAKHAL
Sebastien NEDJAR

Troisième partie

Dépendances fonctionnelles

7 Définition

Soient $r(R)$ une relation d'attributs R et $x, y \in R$. y dépend fonctionnellement de x, ou x détermine y, ou x implique y, $X \to Y$, si et seulement si $\forall t, t' \in r$. Si $t(x) = t'(x)$, alors $t(y) = t'(y)$, tous les tuples ayant même valeur sur x ont même valeur sur y, à chaque valeur de x correspond au plus une valeur de y,

$$\underbrace{|r(x,y)|}_{\text{Cardinalité de } r(x,y)} = \underbrace{|r(x)|}_{\text{Cardinalité de } r(x)}.$$

8 Axiomes d'Armstrong

8.1 Réflexivité

Si $y \subseteq X$ alors $x \to y$ ($x \to x$).

8.2 Augmentation

Si $x \to y$ et $w \subseteq z$ alors $x, z \to y, w$ ($x, z \to y$ si $w = \emptyset$).

8.3 Transitivité

Si $x \to y$ et $y \to z$ alors $x \to z$.

8.4 Pseudo-transitivité

Si $x \to y$ et $y, z \to t$ alors $x, y \to t$.

8.5 Décomposition

Si $x \to y, z$ alors $x \to y$ et $x \to z$.

8.6 Union

Si $x \to y$ et $x \to z$ alors $x \to y, z$.

9 Dépendance fonctionnelle minimale

$X \to A$ est une dépendance fonctionnelle minimale si et seulement si
— A est un attribut unique (les cibles de la DF [2] minimale comportent un seul attribut)
— $\forall x \in X, X - x \nrightarrow A$ (tous les attributs de la source de la DF sont nécessaires)

Exemple
— $D \to A$ est une DF minimale (car $\emptyset \nrightarrow A$)
— $BD \to A$ n'est pas une DF minimale (car $D \to A$, B n'est pas nécessaire)

10 Clef candidate

$X \subseteq \mathbb{R}$ est une clef candidate si et seulement si $X \to \mathbb{R}$.

2. dépendance fonctionnelle

11 Clef candidate minimale

$X \subseteq \mathbb{R}$ est une clef candidate minimale si et seulement si
— X est une clef candidate minimale
— $\forall x \in X, X - x \nrightarrow \mathbb{R}$ (tous les attributs de X sont nécessaires pour déterminer R)

12 Clef primaire

$X \subseteq \mathbb{R}$ est une clef primaire si et seulement si X est une clef candidate minimale.

Critères de choix d'une clef primaire parmi les clefs candidates minimales

1. nombre d'attributs (choisir le plus petit)
2. type (les types numériques sont prioritaires)

13 Attributs non-clef

$A \in \mathbb{R}$ est un attribut non-clef si et seulement si A n'est pas une clef candidate minimale.

Rosine CICCHETI
Lotfi LAKHAL
Sebastien NEDJAR

Quatrième partie

Théorie de modélisation

14 Objectif

Un bon schéma relationnel est un schéma relationnel qui présente le moins de redondance de données et le moins d'anomalies de stockage dans un environnement de mise à jour.

14.1 Mauvais schéma relationnel

ENSEIGNANT	NOM	FONCTION	SALAIRE
	CICCHETI	PROF	5000
	MIRANDA	PROF	5000
	LAKHAL	PROF	5000
	DUPONT	ASSISTANT	2000
	CASALI	MC	4000
	LAPORTE	MC	4000

Table 14 – Mauvais schéma relationnel

Ce schéma relationnel est mauvais :
- **Redondance de données** : (Prof,5000) se répète plusieurs fois
- **Anomalie de stockage** lors des opérations de mises à jour
  - Ajout d'une nouvelle fonction et d'un nouveau salaire (exemple : (-,MA,3000)) impossible car la valeur de la clef primaire n'est pas définie
  - Baisser le salaire des professeurs nécessite plusieurs opérations
  - Supprimer l'enseignant DUPONT fait perdre de l'information (les assistants gagnent 2000€).

14.2 Bon schéma relationnel

ENSEIGNANT	NOM	FONCTION
	CICCHETI	PROF
	MIRANDA	PROF
	DUPONT	MA
	CASALi	MC

FONCTION	FONCTION	SALAIRE
	PROF	5000
	MA	3000
	MC	4000

Table 15 – Bon schéma relationnel

15 Formes normales

Il y a trois formes normales : 1NF, 2NF et 3NF. Plus le degré de normalité d'une relation est important, moins on a de redondances de données et d'anomalie de stockage dans un contexte de mise à jour.

15.1 1NF (1st Normal Form)

Une relation $r(\mathbb{R})$ est en 1NF si et seulement si tous les attributs de $r(\mathbb{R})$ sont mono-valués (pour tout tuple de r, chaque attribut prend une seule valeur).

Exemple de relation N1NF

LIVRE	CODE	TITRE	AUTEUR
	C_1	T_1	A_1
	C_2	T_2	A_1, A_2
	C_3	T_4	A_2, A_3, A_4

TABLE 16 – Relation N1NF

AUTEUR est un attribut multivalué, LIVRE n'est donc pas en 1NF.

15.1.1 Normalisation en 1NF

Cas 1 : ajout d'attributs

LIVRE	CODE	TITRE	AUTEUR 1	AUTEUR 2	AUTEUR 3
	C_1	T_1	A_1		
	C_2	T_2	A_1	A_2	
	C_3	T_4	A_2	A_3	A_4

TABLE 17 – Ajout d'attributs

Cas 2 : ajout d'une relation toute clef

LIVRE	CODE	TITRE
	C_1	T_1
	C_2	T_2
	C_3	T_4

AUTEURS	CODE	AUTEUR
	C_1	A_1
	C_2	A_1
	C_2	A_2
	C_3	A_2
	C_3	A_3
	C_3	A_3

TABLE 18 – Relation toute clef

15.2 2NF (2nd Normal Form)

Une relation $r(\mathbb{R})$ est en 2NF si et seulement si :
— elle est en 1NF
— il n'y a pas de DF entre une partie d'une clef candidate minimale et un attribut qui n'appartient pas à une clef candidate minimale

DF problématique $\underbrace{X}_{\text{une partie d'une CCM}} \rightarrow \underbrace{A}_{\text{un attribut qui n'appartient pas à une CCM}}$

Exemple COMMANDE(NO_PROD, NO_FOURN, NOM_FOURN, QTE_FOURN) avec

$$\mathbb{F}_{\text{commande}} = \begin{cases} \text{NO\_PROD, NO\_FOURN} \rightarrow \text{NOM\_FOURN, QTE\_FOURN} \\ \underbrace{\text{NO\_PROD}}_{\text{une partie d'une CCM}} \rightarrow \underbrace{\text{NOM\_FOURN}}_{\notin \text{ CCM}} \end{cases}$$

Une seule clef candidate minimale : (NO_PROD,NO_FOURN).
COMMANDE n'est pas en 2NF.

15.2.1 Normalisation en 2NF

Théorème de décomposition de Casey-Delabel :
« Soit $r(\mathbb{R})$ une relation, avec $x, y \subseteq \mathbb{R}$ et $x \rightarrow y$. On a r = JointureNaturelle$(r_1, r_2/x = x)$ avec r_1 = Projection$(r/x, y)$ et r_2 = Projection$(r/y, x)$. »

Ce théorème nous garantit que la décomposition de $r(x, y, z)$ en $r_1(x, y)$ et $r_2(x, z)$ est réversible (sans perte d'information).

Exemple de normalisation en 2NF
COMMANDE(NO_PROD, NO_FOURN, NOM_FOURN, QTE_FOURN) devient :
COMMANDE(NO_PROD,NO_FOURN,QTE_FOURN)
$\underbrace{r_1}_{\text{FOURNISSEUR}}$ (NO_FOURN, NOM_FOURN)

Remarque lorsque les clefs candidates minimales ne comportent qu'un seul attribut, la relation est automatiquement en 2NF.

15.3 3NF (3rd Normal Form)

Une relation $r(\mathbb{R})$ est en 3NF si et seulement si :
— $r(\mathbb{R})$ est en 2NF
— il n'y a pas de DF entre un attribut ou groupe d'attributs qui n'est pas une clef candidate minimale, et un attribut qui n'appartient pas à une clef candidate minimale

DF problématique $\underbrace{X}_{\text{n'est pas une CCM}} \rightarrow \underbrace{A}_{\notin \text{ CCM}}$

15.3.1 Normalisation en 3NF

PRODUIT(NO_PROD, LIBELLE, CODE_TVA, TAUX_TVA) avec

$$\mathbb{F}_{\text{PRODUIT}} = \begin{cases} \text{NO\_PROD} \rightarrow \text{LIBELLE, CODE\_TVA, TAUX\_TVA} \\ \text{CODE\_TVA} \rightarrow \text{TAUX\_TVA} \end{cases}$$

devient :
TVA(CODE_TVA, TAUX_TVA)
PRODUIT(NO_PROD, LIBELLE, CODE_TVA)

Rosine CICCHETI
Lotfi LAKHAL
Sebastien NEDJAR

Cinquième partie

Algorithmes et méthodes de normalisation

Objectif : aider à la conception d'un schéma relationnel en 3NF

16 Concepts de base

16.1 Dérivabilité des DF

Soit $\mathbb{F}$ un ensemble de DF sur $r(\mathbb{R})$ et $x, y \subseteq \mathbb{R}$. On dit que $x \to y$ est dérivable à partir de $\mathbb{F}(\mathbb{F} \vdash x \to y)$ si $x \to y$ peut être obtenue à partir de $\mathbb{F}$ en appliquant les axiomes d'Armstrong.

Exemple

$$\mathbb{F} = \begin{cases} A \to B \\ B \to C \end{cases}$$

16.2 Couverture d'un ensemble de DF

Soit $\mathbb{F}$ un ensemble de DF sur $r(\mathbb{R})$. $\mathbb{G}$ est une couverture de $\mathbb{F}$ si et seulement si :
— $\mathbb{F} \vdash \mathbb{G}$ $(\forall x \to A \in \mathbb{G}, \mathbb{F} \vdash x \to A)$
— $\mathbb{G} \vdash \mathbb{F}$ $(\forall x \to A \in \mathbb{F}, \mathbb{G} \vdash x \to A)$
$\mathbb{G}$ est une couverture de $\mathbb{F}$ et $\mathbb{F}$ est une couverture de $\mathbb{G}$.

16.3 Couverture minimale d'un ensemble de DF

Soit $\mathbb{F}$ un ensemble de DF. $\mathbb{M}$ est une couverture minimale de $\mathbb{F}$ si et seulement si :
— Toutes les DF de $\mathbb{M}$ sont de la forme $X \Rightarrow A$ (un attribut cible)
— $\forall x \to A \in \mathbb{M}, \forall x \in X, \mathbb{M} \forall X - x \to A$ (chaque DF de $\mathbb{M}$ est minimale)
— $\forall x \to A \in \mathbb{M}, \mathbb{M} - \{x \to A\} \vdash X \to A$ (chaque DF de $\mathbb{M}$ est nécessaire)
Si ces trois conditions sont vraies alors $\mathbb{M}$ est une couverture minimale de $\mathbb{F}$.

16.4 Fermeture d'Armstrong

Soit $\mathbb{F}$ un ensemble de DF sur $r(\mathbb{R})$ et $Z_1, Z_2, \ldots, Z_n$ une suite cumulative $(Z_1 \subseteq Z_2 \subseteq Z_3 \ldots \subseteq Z_n)$. La fermeture d'Armstrong d'un ensemble d'attributs selon un ensemble de DF est notée $(X^+_{\mathbb{F}})$.

16.4.1 Application +

$S(\mathbb{R}) \to P(\mathbb{R})$
$X \to X^+_{\mathbb{F}} = Z_n$ avec $Z_{n+1} = Z_n, Z_1 = X, Z_n = Z_{n-1} \cup \{A \in \mathbb{R} / x \to A \in \mathbb{F}, X \subseteq Z_{n-1}\}$

16.4.2 Exemple : $BD^+_{\mathbb{F}}$, la fermeture de BD selon $\mathbb{F}$

$$BD^+_{\mathbb{F}} = Z_n$$
$$Z_1 = BD$$
$$Z_2 = BD \cup \{E, A\} = ABDE$$
$$Z_3 = ABDE \cup \{C\} = ABCDE$$
$$Z_4 = ABCDE \cup \{\mathbb{F}\} = Z_3$$

$X^+_{\mathbb{F}} = X \cup$ tous les attributs déterminés par X (directement ou indirectement)

Rosine CICCHETI
Lotfi LAKHAL
Sebastien NEDJAR

17 Algorithmes de normalisation

17.1 Algorithme de calcul d'une clef candidate minimale

— Entrée : $\mathbb{F}$ un ensemble de DF sur $r(\mathbb{R})$
— Sortie : $X \subseteq \mathbb{R}$ une clef candidate minimale pour $r(\mathbb{R})$

Algorithme

$X := \mathbb{R}$
Pour tout $A \in \mathbb{R}$
    si $(X - A)^+_{\mathbb{F}} = \mathbb{R}$ alors $X := X - A$
Renvoyer X

17.2 Algorithme de calcul d'une couverture minimale

Étapes de calcul

1. Décomposer les DF (un seul attribut cible par DF)
2. Suppression des attributs de la source de chaque DF
3. Suppression des DF redondantes

17.3 Algorithme de calcul de la fermeture d'Armstrong

— Entrée : $\mathbb{F}$ un ensemble de DF sur $r(\mathbb{R})$ et $X \leq \mathbb{R}$
— Sortie : $X^+_{\mathbb{F}}$ la fermeture de X selon $\mathbb{F}$

Algorithme

Fermeture:= X
Répéter
    DF:= X
    Pour tout $X \rightarrow A \in \mathbb{F}$ faire
        Si $x \leq$Fermeture et $A \notin$Fermeture alors Fermeture:=Fermeture$\cup A$
Jusqu'à DF=Fermeture
Renvvoyer Fermeture

Rosine CICCHETI
Lotfi LAKHAL
Sebastien NEDJAR

Sixième partie

Modélisation de bases de données

Jusqu'à présent, nous avons présenté des approches de conception faisant abstraction de l'univers réel. À partir d'une liste d'attributs et d'un ensemble de dépendances fonctionnelles, on arrivait à trouver le schéma normalisé relationnel directement. Cette approche a des limites :

— l'identification des DF est ardue voire impossible du premier coup
— les algorithmes ont un coût qui augmente exponentiellement avec le nombre d'attributs
— un schéma relationnel est difficilement compréhensible pour l'utilisateur/client

Pour résoudre ces problèmes, on va ajouter une phase à la conception de la base de données :

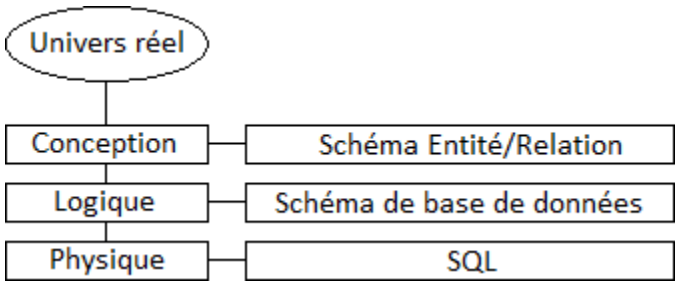

Figure 4 – Conception de base de données

18 Schéma entité/relation

Définition Le schéma entité/relation est une représentation statique. Il a pour but de décrire simplement les données ainsi que les liens pouvant exister.

18.1 Les types d'entités

Une entité est la représentation d'un objet (matériel ou immatériel) identifiable de l'univers réel. Par exemple, une voiture dont le numéro de série est X123. Comme en programmation orientée objet, où les objets similaires sont regroupés en classe, les entités sont regroupées en types d'entités.

Un type d'entité est la description d'entités qui possèdent les mêmes caractéristiques. Par abus de langage, il peut arriver d'utiliser le mot « entité » à la place de « type d'entité ». Pour parler d'entités, préférer « occurrence ». Pour parler de type d'entité, préférer « TE ».

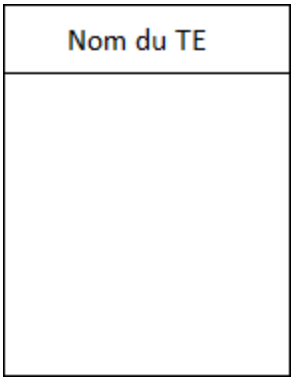

Figure 5 – Représentation graphique d'un type d'entité

18.2 Les propriétés

Une propriété (ou attribut) est une information élémentaire (c'est à dire non déductible d'autres informations) qui présent un intérêt pour le domaine étudié. L'ensemble de valeurs permises pour une propriété est son domaine.

Il faut être vigilant en choisissant le nom d'une propriété car il ne doit pas être ambigu.

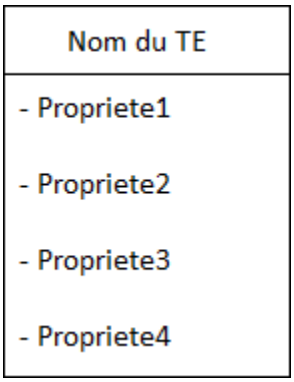

FIGURE 6 – Représentation graphique des proprietés

18.3 Les identifiants (ou clefs)

L'identifiant d'un TE est un ensemble de propriétés dont la valeur permet d'identifier de manière unique une entité (occurrence).

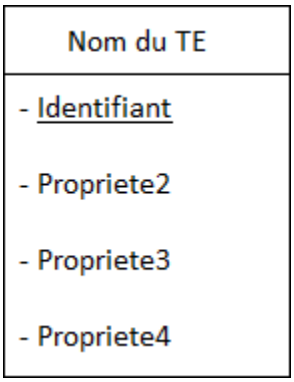

FIGURE 7 – Représentation graphique des identifiants

18.4 Associations et types d'associations

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

Un type d'association est un ensemble d'associations qui partagent les mêmes caractéristiques.

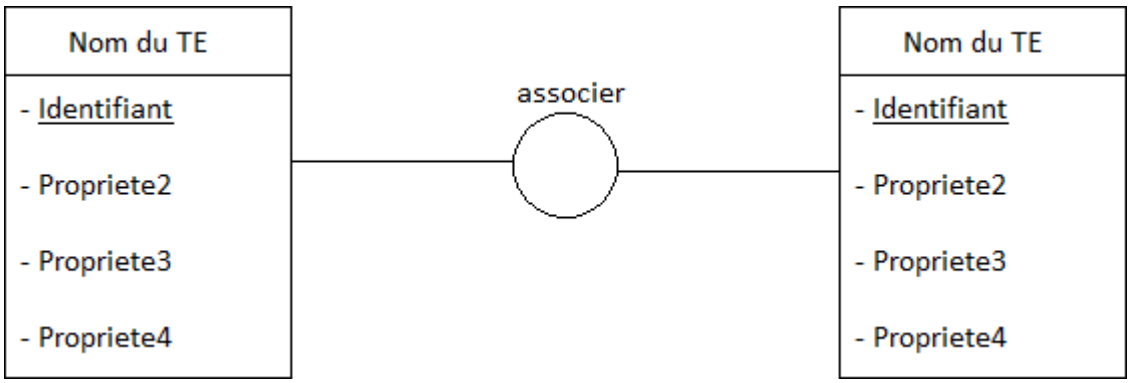

FIGURE 8 – Représentation graphique d'une association

18.4.1 Associations

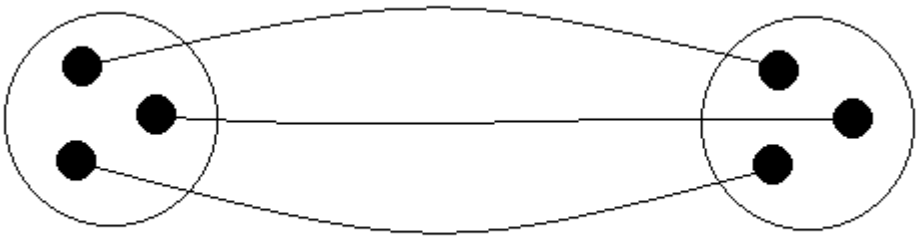

FIGURE 9 – Association « un-à-un »

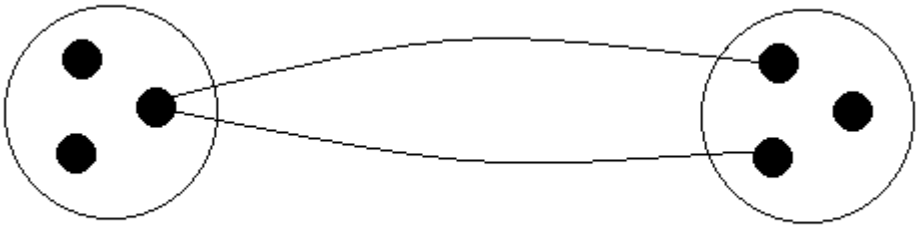

FIGURE 10 – Association « un-à-plusieurs »

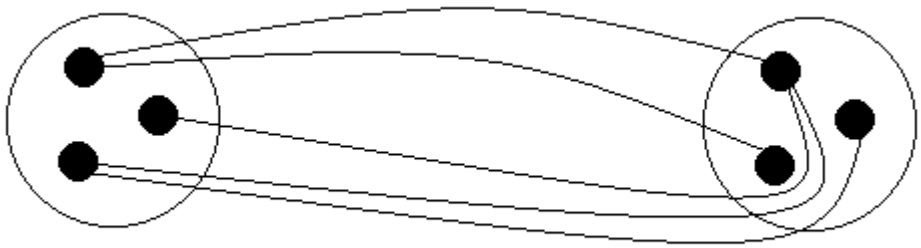

FIGURE 11 – Association « plusieurs-à-plusieurs »

18.4.2 Cardinalités

Les cardinalités permettent de caractériser la multiplicité du lien qui existe entre les occurrences et l'association à laquelle elle est reliée. La cardinalité d'un type d'association (TA) est le nombre de fois minimal et maximal qu'une entité peut intervenir dans une association de ce type.

Une TA dont l'une des cardinalités est (n,1) est dite hiérarchique.

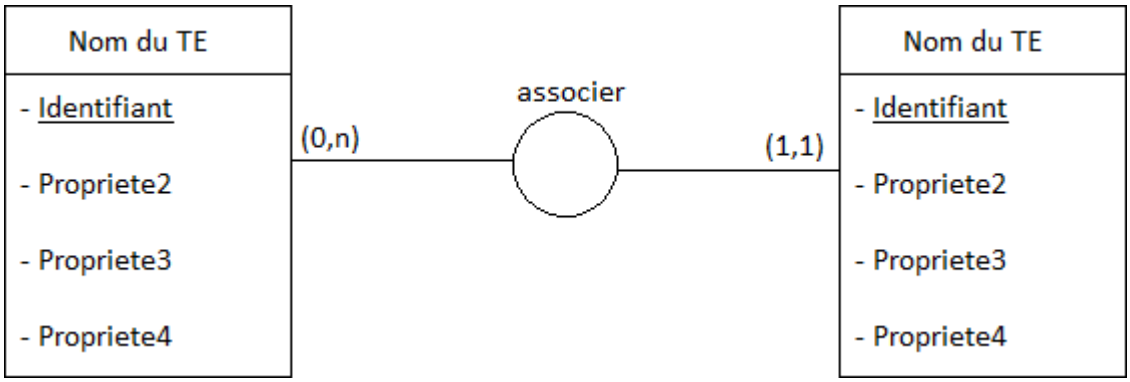

FIGURE 12 – Représentation des cardinalités

18.4.3 Dimension d'une TA

La dimension d'un TA est le nombre de TE qui participent à l'association.

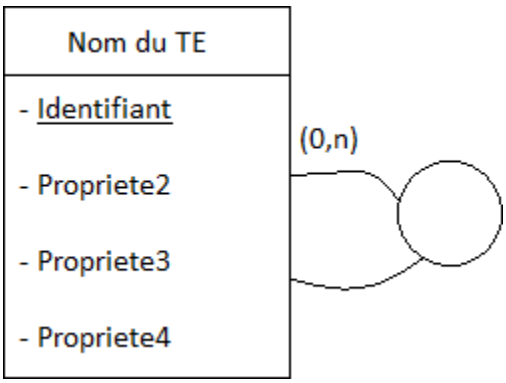

FIGURE 13 – TA unaire (reflective)

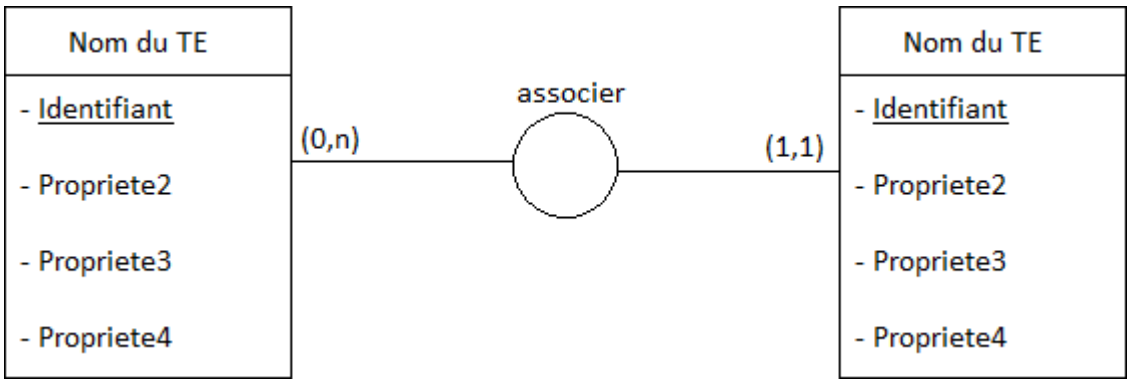

FIGURE 14 – TA binaire

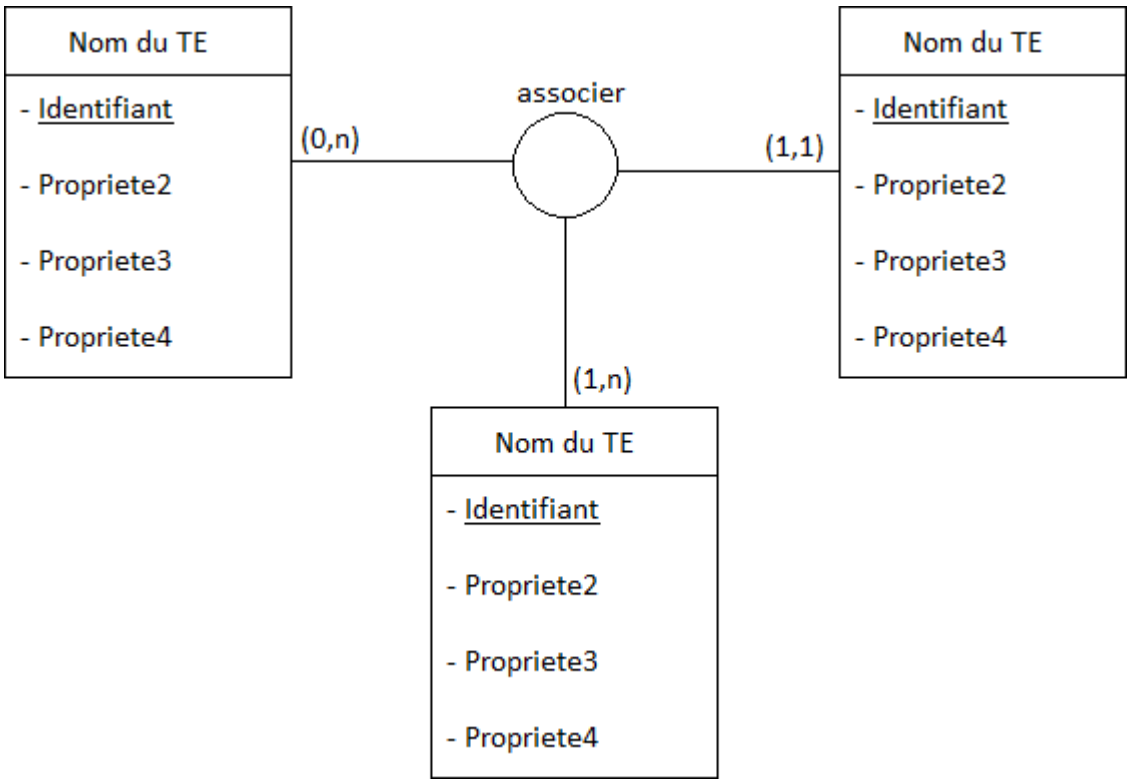

FIGURE 15 – TA ternaire

18.4.4 Identification d'une association

Une association n'a pas d'existence en dehors des entités liées. On utilise donc pour l'identifier la combinaison des clefs des TE liés. La valeur de l'identifiant doit être unique.

19 Dérivation relationnelle

La dérivation relationnelle est la transformation d'un schéma Entités/Associations en un schéma relationnel normalisé (3NF). Pour ce faire, on applique trois règles simples :

1. Dérivation des TE : 1 TE = 1 relation
2. Dérivation des TA non-hiérarchiques : 1 TA non-hiérarchique = 1 relation. La clef primaire de la relation est l'identifiant du TA.
3. Dérivation des TA hiérarchiques : 1 TA hiérarchique = 1 clef étrangère du côté de la cardinalité (n,1)

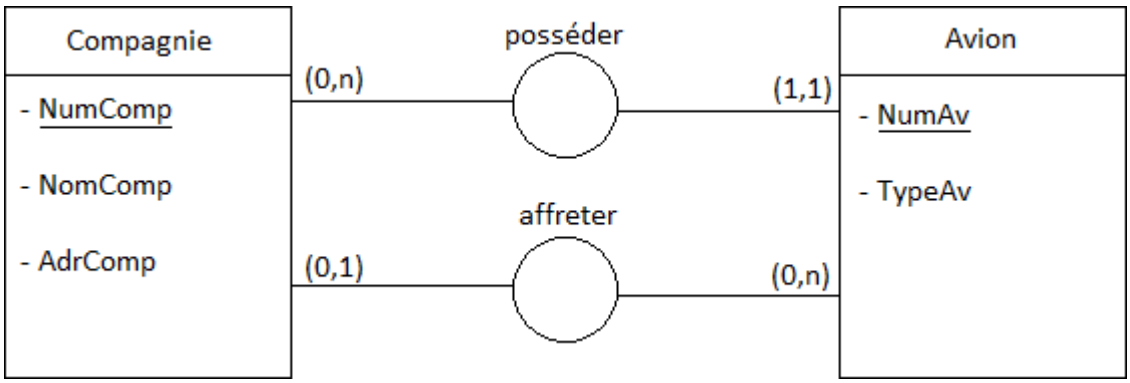

FIGURE 16 – Dérivation relationnelle

— COMPAGNIE(NumComp, NomComp, AdrComp)
— AVION(NumAv, TypeAv, NumComp#)
— AFFRETER(NumComp#, NumAv#)

20 Règles de conception

1. tout TE doit être identité (avoir une clef)
2. tout TA non(hiérarchique doit être identifié par la combinaison des clefs des TE liés
3. tout TE doit être normalisé (3NF)
4. un TA hiérarchique ne doit pas avoir d'attribut
5. un TA de degré supérieur à 1 ne peut pas être hiérarchique

Rosine CICCHETI
Lotfi LAKHAL
Sebastien NEDJAR

Septième partie

Le langage SQL

SQL (Structured Query Language) prend trois aspects :
— langage de manipulation de données (LMD) : requêtes et mises à jours des données (*INSERT,DELETE, UPDATE*)
— langage de définition de données (LDD) : manipulation de la structure, création de relations, ajout d'attributs,...
— langage de contrôle des données (LCD) : intégrité, confidentialité
SQL est une norme ANSI, mais suivant les SGBD, on a des variantes (dialectes) de SQL comme SQL*Plus d'Oracle.

21 SQL comme langage de manipulation des données (LMD)

Toute requêtes SQL s'exprime sous forme de blocs.

21.1 Expression des projections

Les attributs projetés sont mentionnés dans la clause *SELECT*, séparés par des virgules.

Exemple : numéros et noms des pilotes

```
SELECT  NumPil, NomPil
FROM    PILOTE
```

Quand on doit conserver tous les attributs, on utilise *.
En SQL, il n'y a pas d'élimination automatique des doublons. C'est à la charge de l'utilisateur grâce au mot-clef *DISTINCT*.

Exemple : numéros et noms des pilotes

```
SELECT  DISTINCT NumPil, NomPil
FROM    PILOTE
```

21.2 Expression des sélections

Les sélections s'expriment dans la clause *WHERE* comme en langage algébrique.

Exemple : numéros des pilotes habitant à Marseille

```
SELECT  NumPil
FROM    PILOTE
WHERE   ADR='Marseille'
```

On peut combiner des conditions avec *AND* et *OR*.

Exemple : numéros des pilotes nommés Dupont habitant à Marseille

```
SELECT  NumPil
FROM    PILOTE
WHERE    ADR='Marseille'
         AND NOMPIL='Dupont'
```

21.2.1 Prédicats de sélection

On peut utiliser les prédicats suivants dans la clause *WHERE* :
— <attribut> *BETWEEN* <valeur1> *AND* <valeur2>
— <attribut> *IN*(<valeur1>, <valeur2>,...)
— <attribut> *LIKE* 'valeur_générique%' (_ = n'importe quel caractère, % = n'importe quel nombre de caractères)
— <attribut> *IS NULL*

21.3 Calculs horizontaux

Ces calculs peuvent figurer soit dans la clause *SELECT* (le résultat du calcul fait partie de la requête), soit dans la clause *WHERE* (la condition porte sur le résultat du calcul). Il est possible d'utiliser les opérateurs +, -, *, / sur les nombres, + et - sur les dates, et || (concaténation) sur les chaines ; il est aussi possible d'utiliser les fonctions *UPPER* (passer une chaîne en majuscules), *LOWER* (passer une chaîne en minuscule) et *LENGTH* (renvoie la taille de la chaîne).

21.4 Calculs verticaux

Les cinq fonctions classiques sont :
— *SUM* (somme)
— *AVG* (moyenne)
— *COUNT* (comptage)
— *MIN* (valeur minimum)
— *MAX* (valeur maximum)

Un calcul vertical s'effectue une seule fois pour tous les tuples d'une relation. Il est possible de combiner les calculs horizontaux et verticaux.

Exemple : donner le total des salaires des pilotes

```
SELECT  SUM(SAL)
FROM    PILOTE
```

Les valeurs nulles sont ignorées. Le seul cas où *SUM*, *AVG*, *MIN* et *MAX* renvoient *NULL* est quand toutes les valeurs sont nulles. Pour donner une valeur par défaut au lieu de *NULL*, on utilise *NVL*.

Exemple : somme des salaires et primes des pilotes

```
SELECT  NVL(SUM(SAL),0) + NVL(SUM(Prime),0)
FROM    PILOTE
```

21.5 Jointures prédicatives

Elles s'expriment dans la clause *WHERE* de la même façon qu'en langage algébrique : <attribut1> θ <*attribut2*>, où θ est un opérateur de comparaison.

Exemple : numéro des avions localisés dans la ville de départ du vol IT100

```
SELECT  NUMAV
FROM    AVION, VOL
WHERE   NUMVOL='IT100' AND LOC=VILLE_DEP
```

Exemple : numéro et nom des pilotes faisant au moins un vol au départ de Nice

```
SELECT  DISTINCT NOMPIL, PILOTE.NUMPIL
FROM    PILOTE, VOL
WHERE   VILLE_DEP='Nice' AND PILOTE.NUMPIL=VOL.NUMPIL
```

Exemple : auto-jointure, nom des pilotes habitant la même ville que le pilote 100

```
SELECT  P.NUMPIL
FROM    PILOTE P, PILOTE P100
WHERE   P100.NUMPIL=100 AND P.VILLE=P100.VILLE
```

21.6 Jointures imbriquées

Une jointure imbriquée s'exprime avec deux blocs : le premier bloc et un bloc imbriqué (ou sous-requête).

21.6.1 La sous-requête renvoie une seule valeur

Exemple : numéro des pilotes qui gagnent plus que la moyenne

```
SELECT  NUMPIL
FROM    PILOTE
WHERE   SAL > ( SELECT  AVG(SAL)
                FROM    PILOTE   )
```

Le bloc imbriqué est exécuté en premier et une seule fois.

21.6.2 La sous-requête renvoie un seul tuple

Exemple : numéro des pilotes ayant la même adresse et le même salaire que le pilote 100

```
SELECT  NUMPIL
FROM    PILOTE
WHERE   (ADR,SAL)=(SELECT  ADR,SAL
                   FROM    PILOTE
                   WHERE   NUMPIL=100)
```

21.6.3 La sous-requête rend un ensemble de résultats

La comparaison se fait avec un des résultats du bloc imbriqué

Exemple : numéro des vols faits par un Airbus

```
SELECT  NUMVOL
FROM    VOL
WHERE   NUMAV=ANY (SELECT  NUMAV
                   FROM    AVION
                   WHERE   NUMAV LIKE 'A%')
```

Exemple : numéro des pilotes ayant le même salaire et la même adresse qu'un Dupont

```
SELECT  NUMPIL
FROM    PILOTE
WHERE   (SAL,ADR)=ANY (SELECT  SAL,ADR
                       FROM    PILOTE
                       WHERE   NUMPIL='Dupont')
```

La comparaison se fait avec tous les resultats du bloc imbriqué

Exemple : numéro des pilotes marseillais gagnant plus que tous les pilotes parisiens

```
SELECT  NUMPIL
FROM    PILOTE
WHERE   ADR='Marseille' AND SAL > ALL (SELECT  SAL
                                       FROM    PILOTE
                                       WHERE   ADR='Paris')
```

21.7 Opérateurs ensemblistes

Il faut que les clauses *SELECT* des blocs comportent le même nombre d'attributs, et que ces attributs soient compatibles.

Exemple : numéro des avions localisés à Paris ou faisant un vol au départ de Paris

```
SELECT  NUMAV
FROM    AVION
WHERE   LOC='Paris'
UNION
SELECT  NUMAV
FROM    VOL
WHERE   VILLE_DEP='Paris'
```

Exemple : numéro des avions faisant un vol au départ de leur localisation

```
SELECT  NUMAV, LOC
FROM    AVION
INTERSECT
SELECT  NUMAV, VILLE_DEP
FROM    VOL
```

Exemple : numéro des avions n'effectuant aucun vol

```
SELECT  NUMAV
FROM    AVION
MINUS
SELECT  NUMAV
FROM    VOL
```

Remarque avec les opérateurs ensemblistes, il y a élimination des doublons

21.8 Jointures externes

Une jointure classique entre r_1 et r_2 parcourt les tuples de r_1 et génère un résultat pour chaque tuple satisfaisant la condition de jointure. Une jointure externe procède de la même manière mais conserve les tuples de r_1 ne satisfaisant pas la condition de jointure.

Exemple : jointure externe entre PILOTE et VOL avec NUMPIL=NUMPIL

PILOTE	NUMPIL
	100
	101
	102
	103

VOL	NUMVOL	NUMPIL
	IT100	100
	IT200	102
	IT300	100
	IT400	102

JOINTURE	NUMPIL	NUMVOL	NUMPIL
	100	IT100	100
	100	IT300	100
	101	*NULL*	101
	102	IT200	102
	102	IT400	102
	103	*NULL*	103

Table 19 – Jointure externe

En SQL, la jointure externe est indiquée par un *(+)* derrière l'attribut de jointure qui pourra prendre des valeurs nulles :

```
SELECT  *
FROM    PILOTE,VOL
WHERE   PILOTE.NUMPIL=VOL.NUMPIL(+)
```

21.9 Test de non-existence de tuples

Pour ces requêtes, il faut vérifier qu'un tuple appartenant à un ensemble E_1 n'existe pas dans cet ensemble E_2. Par exemple, les pilotes ne faisant aucun vol. Il y a 5 formulations.

21.9.1 Jointure imbriquée avec *NOT IN*

```
SELECT  NUMPIL
FROM    PILOTE
WHERE   NUMPIL NOT IN (SELECT  NUMPIL
                       FROM    VOL   )
```

21.9.2 Jointure imbriquée avec *<> ALL*

```
SELECT  NUMPIL
FROM    PILOTE
WHERE   NUMPIL<>ALL (SELECT  NUMPIL
                     FROM    VOL   )
```

21.9.3 Différence ensembliste

```
SELECT  NUMPIL
FROM    PILOTE
MINUS
SELECT  NUMPIL
FROM    VOL
```

21.9.4 Jointure externe et prédicat *IS NULL*

```
SELECT  *
FROM    PILOTE,VOL
WHERE   PILOTE.NUMPIL=VOL.NUMPIL(+)
        AND NUMVOL IS NULL
```

21.9.5 Bloc imbriquée et prédicat *NOT EXISTS*

Devant un bloc imbriquée, *NOT EXISTS* renvoie :
— vrai si le bloc imbriqué ne rend aucun résultat
— faux si le bloc imbriqué rend des résultats

```
SELECT  NUMPIL
FROM    PILOTE
WHERE   NOT EXISTS (SELECT *
                    FROM   VOL)
```

Avec *NOT EXISTS*, il faut une exécution corrélée des deux blocs : pour chaque tuple dans le premier bloc, il faut exécuter le bloc imbriqué. Pour cela, il faut exprimer une jointure entre les relations des deux blocs. Il faut donner un alias à la relation du premier bloc et faire la jointure avec cet alias dans le bloc imbriqué.

```
SELECT  NUMPIL
FROM    PILOTE P
WHERE   NOT EXISTS (SELECT *
                    FROM   VOL
                    WHERE  VOL.NUMPIL=P.NUMPIL)
```

21.10 Opérations de partitionnement

Partitionner une relation consiste à créer des classes d'équivalence (ou sous-ensembles) telles que :
— leur union soit égale à la relation de départ
— les classes d'équivalence soient disjointes deux à deux (pas de tuples en commun)

Le ou les critères permettant de créer ces classes sont les attributs de partitionnement. Par exemple, partitionner VOL sur NUMPIL consiste à créer trois classes d'équivalence (une par pilote). Chaque classe regroupe les vols d'un même pilote.

Avec un partitionnement, on peut appliquer les fonctions agrégatives sur les classes d'équivalence.

En SQL, le partitionnement se fait dans la clause *GROUP BY* dans laquelle on indique le ou les attributs de partitionnement. Cette clause suit le *WHERE*.

Exemple : donner pour chaque pilote, le nombre de vols

```
SELECT  NUMPIL, COUNT(*)
FROM    VOL
GROUP BYNUMPIL
```

Exemple : donner pour chaque pilote et chaque avion, le nombre de vols

```
SELECT  NUMPIL, NUMAV, COUNT(*)
FROM    VOL
GROUP BYNUMPIL, NUMAV
```

Remarque Avec un *GROUP BY*, on peut combiner un attribut atomique avec une fonction agrégative à condition que ce soit l'attribut de partitionnement.

Exemple : donner pour chaque pilote, le nombre de villes d'arrivées

```
SELECT  NUMPIL, COUNT(DISTINCT VILLE_ARR)
FROM    VOL
GROUP BYNUMPIL
```

Exemple : donner le salaire moyen et maximum des pilotes par ville de résidence, sauf Paris

```
SELECT  ADR, AVG(SAL), MAX(SAL)
FROM    PILOTE
WHERE   ADR<>'Paris'
GROUP BYADR
```

On peut exprimer des conditions sur des classes d'équivalence (conditions utilisant des fonctions agrégatives), on utilise la clause *HAVING* qui suit le *GROUP BY*.

Exemple : donner pour chaque pilote faisant au moins 5 vols, le nombre de vols

```
SELECT  NUMPIL
FROM    PILOTE
GROUP BYNUMPIL
HAVING  COUNT(*)>=5
```

Exemple : donner le numero des pilotes faisant autant de vols que le pilote 100

```
SELECT  NUMPIL
FROM    PILOTE
GROUP BYNUMPIL
HAVING  COUNT(*)>=(SELECT  COUNT(*)
                   FROM    VOL
                   WHERE   NUMPIL=100)
```

Remarque *GROUP BY* ne s'utilise pas avec *DISTINCT*, et pas sur une clef primaire.

21.11 Recherche dans des arborescences

21.11.1 Représentation des hiérarchies

Exemple : hiérarchie des lieux

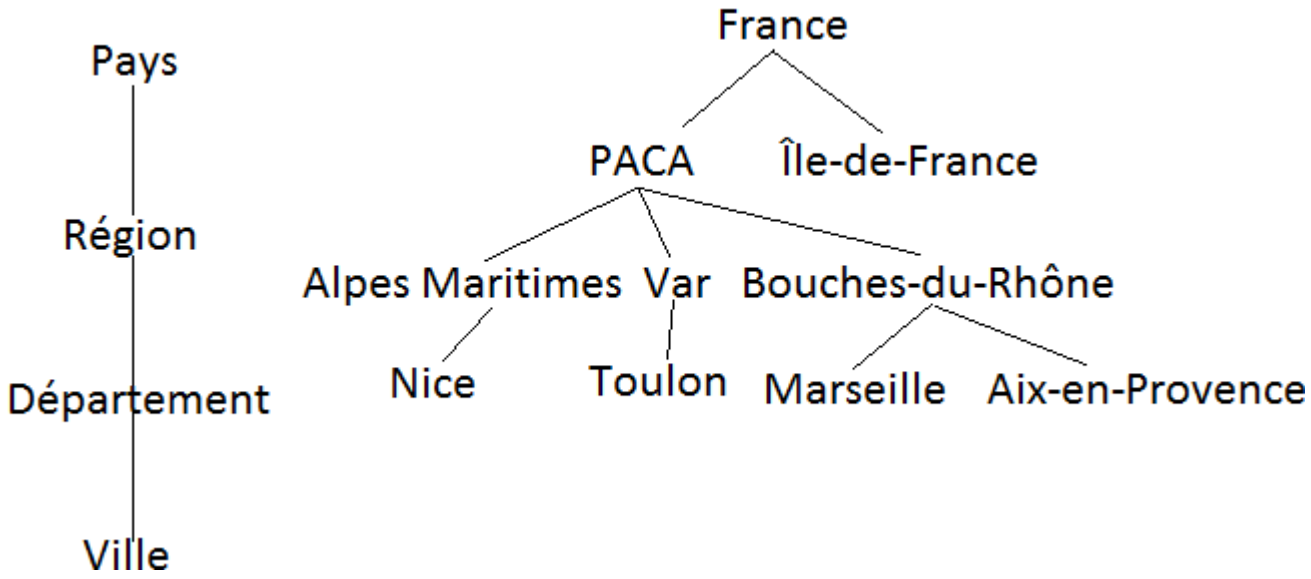

FIGURE 17 – Hiérarchie des lieux

Représentation avec le modèle entité/relation

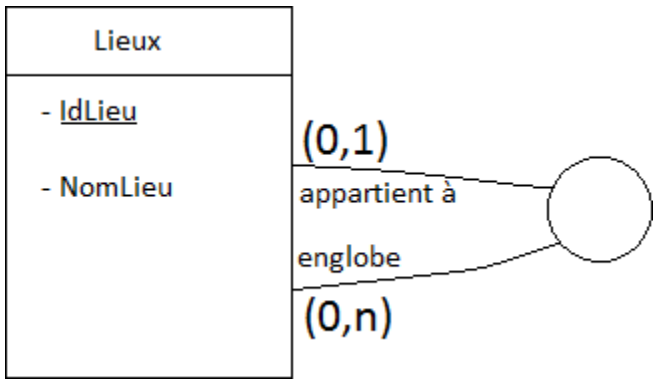

FIGURE 18 – Hiérarchie des lieux avec le modèle entité/relation

Remarque Ce TA est hiérarchique.

Représentation avec le modèle relationnel

— Lieu(IdLieux, Nom, IdLieuPere#)

21.11.2 Manipulation de hiérarchies

En SQL classique, il faut faire des auto-jointures.

Exemple : liste des villes des Bouches-du-Rhône

```
SELECT  LV.NomLieu
FROM    LIEU LD, LIEU LV
WHERE   LD.NomLieu='Bouches-du-Rhône'
        AND LD.IdLIeu=LV.IdLieuPere
```

Exemple : liste des villes de PACA

```
SELECT  LV.NomLieu
FROM    LIEU LR, LIEU LD, LIEU LV
WHERE   LR.NomLieu='PACA'
        AND LR.IdLieu=LD.IdLieuPere
        AND LD.IdLieu=LV.IdLieuPere
```

En SQL*Plus d'Oracle, la clause *CONNECT BY* permet de parcourir des hiérarchies.

```
SELECT <liste_attributs>
FROM   <relation> -- Une seule relation
[WHERE <liste_conditions>]
CONNECT BY [PRIOR] <attribut_fils>=[PRIOR] <attribut_pere>
           [AND <condition_hierarchique>]
           [START WITH <condition_depart>]
           [ORDER BY LEVEL]
```

- *CONNECT BY* permet de donner le sens du parcours dans les hiérarchies. On met *PRIOR* devant l'attribut fils pour une recherche descendante, ou devant l'attribut père pour une recherche ascendante. Par exemple, *CONNECT BY PRIOR* `IdLieu=IdLieuPere` pour descendre dans la hiérarchie des lieux.
- *START WITH* permet dedonner une condition de départ pour la recherche.

Exemple : donner tous les lieux de la région PACA

```
SELECT  NomLieu
FROM   LIEU
CONNECT BY  PRIOR IdLieu=IdLieuPere
           START WITH NomLieu='PACA'
```

Remarques

- si plusieurs tuples satisfont la condition de départ, la recherche arborescente se fait pour chacun de ces tuples
- s'il n'y a pas de *START WITH*, Oracle part de toutes les racines pour un parcours descendant, ou de toutes les feuilles pour un parcours ascendant
- on peut mettre des jointures imbriquées dans la clause *START WITH*

Exemple : hiérarchie des localisations des villes de départ des vols

```
SELECT  NomLieu
FROM    LIEU
CONNECT BY IdLieu=PRIOR IdLieuPere
           START WITH NomLieu IN (SELECT Ville_Dep
                                  FROM   VOL      )
```

— *AND* permet d'éliminer de la recherche une partie de l'arbre. Si un nœud (tuple) ne satisfait pas cette condition, tous ses descendants sont éliminés (pour un parcours descendant), ou tous ses ascendants sont éliminés (pour un parcours ascendant).

Exemple : hiérarchie des lieux, sauf ceux de la région PACA

```
SELECT NomLieu
FROM   LIEU
CONNECT BY IdLieu=PRIOR IdLieuPere
           AND NomLieu<>'PACA'
```

Remarque Si on met `NomLieu<>'PACA'` dans le *WHERE*, la région PACA est éliminée du résultat mais pas ses départements ni ses villes.

— *ORDER BY* permet de trier les résultats.

Exemple : liste alphabétique des pilotes

```
SELECT NOMPIL
FROM   PILOTE
ORDER BY NOMPIL ASC
```

On peut utiliser *DESC* pour un tri inverse.

— *LEVEL* correspond au niveau. Le niveau 1 correspond aux premiers tuples sélectionnés.

21.12 Calculs imbriquées dans le *SELECT* et le *FROM*

Exemple : donner le numéro des pilotes, l'écart entre leur salaire et le salaire moyen

21.12.1 Bloc imbriqué dans le *SELECT*

Il faut que le bloc renvoie au plus un résultat.

```
SELECT NUMPIL,SAL-(SELECT AVG(SAL)
                  FROM  PILOTE  )
FROM   PILOTE
```

21.12.2 Bloc imbriqué dans le *FROM*

Le résultat est une relation temporaire. On peut donc l'imbriquer dans un *FROM*.

```
SELECT NUMPIL, SAL-SMOY
FROM   PILOTE, (SELECT AVG(SAL) SMOY
                FROM  PILOTE        )
```

Exemple : donner l'écart entre le salaire des pilotes et le salaire moyen de leur ville de résidence

```
SELECT NUMPIL, SAL-SMOY
FROM   PILOTE, (SELECT AVG(SAL) SMOY,ADR
                FROM  PILOTE
                GROUP BY ADR            ) PILSAL
WHERE  PILOTE.ADR=PILSAL.ADR
```

21.13 Expression des divisions

Il y a deux méthodes.

21.13.1 Avec *GROUP BY* et comptage

Exemple : numéro des pilotes qui conduisent tous les avions (pilotes qui conduisent autant d'avions qu'il en existe)

```
SELECT NUMPIL
FROM   VOL
GROUP BY NUMPIL
HAVING COUNT(DISTINCT NUMAV)=(SELECT COUNT(*)
                              FROM   AVION   )
```

Exemple : numéro des pilotes qui conduisent tous les types d'appareils

```
SELECT NUMPIL
FROM   VOL, AVION
WHERE  VOL.NUMAV=AVION.NUMAV
GROUP BY NUMPIL
HAVING COUNT(DISTINCT NOMAV)=(SELECT COUNT(DISTINCT NOMAV)
                              FROM   AVION                )
```

21.13.2 Avec un double *NOT EXISTS*

Exemple : numéro des pilotes qui conduisent tous les avions (numéro des pilotes tels qu'il n'existe aucun avion qui ne soit pas conduit par ces pilotes)

```
SELECT NUMPIL
FROM   PILOTE P
WHERE  NOT EXISTS (SELECT *
                   FROM   AVION A
                   WHERE  NOT EXISTS (SELECT *
                                      FROM   VOL
                                      WHERE  VOL.NUMAV=A.NUMAV
                                      AND VOL.NUMPIL=P.NUMPIL))
```

Exemple : numéro des pilotes qui conduisent au moins les mêmes avions que le pilote 100 (numéro des pilotes tels qu'il n'existe aucun avion non conduit par le pilote 100 qui ne soit pas conduit par eux)

```
SELECT NUMPIL
FROM   PILOTE P
WHERE  NOT EXISTS (SELECT *
                   FROM   VOL V
                   WHERE  NUMPIL=100
                          AND NOT EXISTS (SELECT *
                                          FROM   VOL
                                          WHERE  P.NUMPIL=VOL.NUMPIL
                                                 AND V.NUMAV=VOL.NUMAV))
```

22 SQL comme langage de contrôle des données (LCD)

22.1 Concept de vue

UNe vue est une table virtuelle. On peut la voir comme une relation temporaire résultant d'une requête. On crée une vue ainsi : *CREATE VIEW* <nomvue> *AS* <requete>.

Exemple : vue des pilotes habitant à Marseille

```
CREATE VIEW PIL_MARSEILLE AS
SELECT NUMPIL, NOMPIL
FROM   PILOTE
WHERE  ADR='Marseille'
```

22.2 Premier rôle des vues : limiter la consultation des données

Les vues permettent de limiter l'accès aux données à certains utilisateurs. L'utilisateur aura accès à une vue, mais ne pourra pas voir les attributs et tuples qui ne sont pas dans la vue. L'utilisateur peut manipuler les vues comme des relations.

22.3 Deuxième rôle des vues : écrire des requêtes complexes

```
CREATE VIEW VOL_HORAIRE AS
SELECT NUMPIL, SUM(HEURE_ARR-HEURE_DEP)
FROM   VOL
GROUP BY NUMPIL
```

Pour mettre à jour une vue, il ne faut pas mettre de *GROUP BY*, *CONNECT BY*, *DISTINCT*, fonctions agrégatives, opérateurs ensemblistes, dans le premier bloc.

Rosine CICCHETI
Lotfi LAKHAL
Sebastien NEDJAR

Huitième partie

Mémento des ordres SQL*Plus

A SQL comme langage de définition de données (LDD)

Types syntaxiques des attributs : `VARCHAR2(n) CHAR[(n)] NUMBER[(n[,m)]] DATE LONG`

A.1 Création de relation

```
CREATE TABLE  <nom_table>
              (<nom_colonne1> <type1> [DEFAULT <expression1>]
             [, <nom_colonne2> <type2> [DEFAULT <expression2>]]
             [, <contrainte1>[, <contrainte2>]])
```

où :

```
<contraintei> = CONSTRAINT <nom_contrainte> <spec_contrainte> [<etat>]
<spec_contrainte> =  PRIMARY KEY(<attribut1[, <attribut2>, ...])
                     |  FOREIGN KEY(<attribut1>[, <attribut2>,...])
                        REFERENCES <nom_relation_associée>(<attribut1>[, <attribut2>,...])
                        [ON DELETE <CASCADE|SET NULL>]
                     | CHECK(<nom_attribut | expression> <condition>)
<etat> = ENABLE | DISABLE


CREATE TABLE <nom_relation> [(liste_attributs>, <liste_contraintes>)]
AS <requete>
```

A.2 Ajout d'attributs et de contraintes dans une relation

```
ALTER TABLE <nom_table>
ADD ([nom_colonne1> <type1>] [DEFAULT <expression1>] [NOT NULL] [UNIQUE]
     [,<nom_colonne2> <type2> [DEFAULT <expression2>] [NOT NULL] [UNIQUE]...]
     [,<contrainte1> ...])
```

où

```
<contraintei> = CONSTRAINT <nom_contrainte> <spec_contrainte> [<etat>]
<spec_contrainte> =  PRIMARY KEY(<attribut1[, <attribut2>, ...])
                     |  FOREIGN KEY(<attribut1>[, <attribut2>,...])
                        REFERENCES <nom_relation_associée>(<attribut1>[, <attribut2>,...])
                        [ON DELETE <CASCADE|SET NULL>]
                     | CHECK(<nom_attribut | expression> <condition>)
<etat> =  ENABLE | DISABLE | VALIDATE | NOVALIDATE | ENABLE VALIDATE | ENABLE NOVALIDATE
          | DISABLE VALIDATE | DISABLE NOVALIDATE
```

A.3 Modification de la définition d'un attribut

```
ALTER TABLE <nom_table>
MODIFY       [(]<nom_colonne1> [<nouveau_type1>] [DEFAULT <expression1>] [NOT NULL]
             [,<nom_colonne2> [<nouveau_type2>] [DEFAULT <expression2>] [NOT NULL] ... [)]
```

A.4 Modification de l'état d'une contrainte

```
ALTER TABLE <nom_table>
MODIFY CONSTRAINT <nom_contrainte> <etat_contrainte>
```

A.5 Suppression de contrainte dans une relation

```
ALTER TABLE <nom_table> DROP CONSTRAINT <nom_contrainte> [CASCADE]

ALTER TABLE <nom_table> DROP UNIQUE(<nom_attribut>) [CASCADE]

ALTER TABLE <nom_table> DROP PRIMARY KEY [CASCADE]
```

A.6 Suppression d'attribut dans une relation

```
ALTER TABLE <nom_table> SET UNUSED COLUMN <nom_attribut>

ALTER TABLE <nom_table> SET UNUSED(<nom_attribut1>[, <nom_attribut2> ...])

ALTER TABLE <nom_table> DROP COLUMN <nom_attribut> [CASCADE CONSTRAINTS]

ALTER TABLE <nom_table> DDROP(<nom_attribut1>[, <nom_attribut2>, ...]) [CASCADE CONSTRAINTS]

ALTER TABLE <nom_table> DROP UNUSED COLUMNS
```

A.7 Suppresion de relation

```
DROP TABLE <nom_table> [CASCADE CONSTRAINTS]
```

A.8 Création/suppression de synonyme et changement du nom d'une relation

```
CREATE [PUBLIC] SYNONYM <nom_synonyme> FOR <nom_objet>

DROP SYNONYM <nom_synonyme>

RENAME <ancien_nom> TO <nouveau_nom>
```

A.9 Gestion de séquences

```
CREATE SEQUENCE <nom_sequence>  [START WITH <valeur_initiale>]
                                [INCREMENT BY <valeur_increment>]
                                [MAXVALUE <valeur_maximale> | NOMAXVALUE]
                                [MINVALUE <valeur_minimale> | NOMINVALUE]
                                [CYCLE | NOCYCLE]


DROP SEQUENCE <nom_sequence>


ALTER SEQUENCE <nom_sequence>  [INCREMENT BY <valeur_increment]
                               [MAXVALUE <valeur_maximale> | NOMAXVALUE]
                               [MINVALUE <valeur_minimale> | NOMINVALUE]
                               [CYCLE | NOCYCLE]
```

A.10 Index sur les relations

```
CREATE [UNIQUE | BITMAP] INDEX <nom_index>
ON <nom_table> ([<nom_colonne1>[, <nom_colonne2>, ...])

ALTER TABLE <nom_index> RENAME TO <nouveau_nom>

DROP INDEX <nom_index>
```

A.11 Principales tables systèmes Oracle

```
ALL_CONS_COLUMNS  (OWNER, CONSTRAINT_NAME, TABLE_NAME, COLUMN_NAME,...)
ALL_CONSTRAINTS   (OWNER, CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME, SEARCH_CONDITION,...)
ALL_INDEXES       (OWNER, INDEX_NAME, INDEX_TYPE, TABLE_OWNER,
                   TABLE_NAME, TABLE_TYPE, UNIQUENESS, COMPRESSION)
ALL_OBJECTS       (OWNER, OBJECT_NAME, OBJECT_ID, DATA_OBJECT_ID, OBJECT_TYPE, CREATED, ...)
ALL_SEQUENCES     (SEQUENCE_OWNER, SEQUENCE_NAME, MIN_VALUE, MAX_VALUE, INCREMENT, CYCLE_FLAG)
ALL_SYNONYMS      (OWNER, SYNONYM_NAME, TABLE_OWNER, TABLE_NAME)
ALL_TAB_COLUMNS   (OWNER, TABLE_NAME, COLUMN_NAME, DATA_TYPE, DATA_LENGHT)
ALL_TABLES        (OWNER, TABLE_NAME, TABLESPACE_NAME)
ALL_VIEWS         (OWNER, VIEW_NAME, TEXT_LENGTH, TEXT,...)
```

Les tables de même nom préfixées par `USER_` ont la même structure hormis l'attribut `OWNER`, et décrivent seulement les composants du schéma de l'utilisateur.

Pseudo-colonnes

<nom_sequence>.*CURRVAL* , <nom_sequence>.*NEXTVAL* , *LEVEL* , *ROWID* , *ROWNUM* , *USER*

B SQL comme langage de manipulation de données (LMD)

B.1 Requêtes

```
<requete> =  SELECT <liste_resultat | *>
             FROM <liste_relations>
             [WHERE <liste_conditions>]
             [GROUP BY <liste_attributs_de_partitionnement>
             [HAVING <liste_conditions_de_partitionnement>]]
             [ORDER BY <liste_attributs_a_trier>]
```

où :

```
liste_resultat =  [DISTINCT] <attribut1 | expr1 | requete1> [<alias1>]
                  [,<attribut2 | expr2 | requete2> [<alias2>], ...]
liste_relations =  <relation1 | vue1 | requete1> [alias1]
                   [, <relation2 | vue 2 | requete2> |alias2]...
liste_conditions = [NOT] <condition1> [AND | OR <condition2> ...]
```

Condition de sélection :

```
<condition1> = <attribut [(+)] | expression> <comparateur | predicat_conditionnel> <constante>
```

```
<predicat_conditionnel> =  IS NULL | IN | BETWEEN ... AND | LIKE
                           | IS NOT NULL | NOT IN | NOT BETWEEN | NOT LIKE
```

Condition de jointure prédicative :

```
<conditionj> = <attribut1[(+)] | expr1> <comparateur> <attribut2[(+)] | expr2 >
```

Condition de jointure imbriquée :

```
<conditionji> =  <expression1>[, <expression2>,...] θ (<requete>)
                 | <expression1>[, <expression2>,...] θ ANY | IN (<requete>)
                 | <expression1>[, <expression2>,...] θ ANY (<requete>)
```

B.2 Calculs verticaux (fonctions aggrégatives)

```
<nom_fonction> ([DISTINCT] <nom_colonne>)
```

où :

```
<nom_fonction> = SUM | AVG | COUNT | MAX | MIN | STDDEV | VARIANCE
```

B.3 Tri des résultats

```
ORDER BY <expression1> [ASC | DESC] [NULLS FIRST | NULLS LAST] [,<expression2> ...]
```

B.4 Opérateurs ensemblistes

```
<requete1>
UNION | INTERSECT | MINUS
<requete2>
```

B.5 Test d'absence ou d'existence de données

```
SELECT  <liste_attributs>
FROM    <relation1> [<alias1>] [,<relation2> [<alias2>] ...]
WHERE   [<liste_conditions> AND | OR] [NOT] EXISTS (<sous_requete>)
```

B.6 Classifications ou partitionnement

```
ORDER BY  <colonne1> [, <colonne2>,...]
HAVING    <liste_conditions_classe>
```

B.7 Recherche dans une arborescence

```
SELECT  <colonne1> [, <colonne2>, ...]
FROM    <table> [<alias>]
[WHERE  <liste_conditions>]
CONNECT BY [PRIOR] <colonne1> = [PRIOR] <colonne2>
           [AND <condition_hierarchique>]
           [START WITH <condition_depart>]
           [ORDER BY LEVEL]
```

B.8 Mise à jour des données

```
UPDATE  <nom_table>
SET     <nom_colonne1> = <expression1> [, <nom_colonne2> = <expression2>,...]
[WHERE  <condition_selection>]


UPDATE  <nom_table>
SET     (<nom_colonne1>[, <nom_colonne2>, ...]) = (SELECT  <colonne1>[,<colonne2>,...]
                                                   FROM  ...
                                                   WHERE ...                          )
[WHERE  <condition_selection>]


INSERT INTO  <nom_table> [(<liste_attributs>]
VALUES       (<valeur1>[,<valeur2>,...])


INSERT INTO <nom_table> [(<liste_attributs>)] <requête>

DELETE FROM <nom_table> WHERE <condition>
```

C SQL comme langage de contrôle des données (LCD)

C.1 Gestion des transactions

```
COMMIT, ROLLBACK
```

C.2 Création et suppression de roles et d'utilisateurs

```
CREATE ROLE <nom_role> [IDENTIFIED BY <mot_de_passe>]

ALTER ROLE <nom_role> [IDENTIFIED BY <nouveau_mot_de_passe>]

DROP ROLE <nom_role>

CREATE USER  <nom_utilisateur> [IDENTIFIED BY <mot_de_passe>]
DEFAULT TABLESPACE <nom_tablespace>
QUOTA        <taille> PROFILE <nom_profil>


ALTER USER <nom_utilisateur> [IDENTIFIED BY <nouveau_mot_de_passe>]

DROP USER <nom_utilisateur>
```

C.3 Attribution ou suppression de privilèges

```
GRANT  <systeme_privileges | ALL [PRIVILEGES]>
TO     <liste_role_utilisateur | PUBLIC>
[WITH ADMIN OPTION]
```

où :

```
<systeme_privilege> =  CREATE ROLE | CREATE SEQUENCE | CREATE SESSION | CREATE SYNONYM
                       | CREATE PUBLIC | CREATE TABLE | CREATE USER | CREATE VIEW
```

```
GRANT  <liste_droits>
TO     <nom_composant>
[WITH GRANT OPTION]
```

où :

```
<liste_droits> = SELECT | INSERT | UPDATE [nom_colonne1,...] | DELETE | ALTER
                 | INDEX | REFERENCES | ALL [PRIVILEGES]
```

```
GRANT  <liste_roles_attribues>
TO     <liste_roles_utilisateurs>
[WITH ADMIN OPTION]
```

C.4 Gestion de vues

```
CREATE [OR REPLACE] [[NO] FORCE]
VIEW <nom_vue> [(alias1[,alias2,...])]
AS <requete>
[WITH CHECK OPTION | WITH READ ONLY]

ALTER VIEW <nom_vue> COMPILE

DROP VIEW <nom_vue>
```

Rosine CICCHETI
Lotfi LAKHAL
Sebastien NEDJAR

Index

Table des matières

Liste des tableaux

Table des figures

Liste des mots-clefs SQL