Principes des systèmes de gestion des bases de données

Sylvain Bouveret, Paule-Annick Davoine
Scribe : Tom Cornebize

Ensimag, 2A
2015-2016

Table des matières

1 Introduction aux systèmes de gestion des bases de données **3**

1.1 Les bases de données . . . 3

1.2 Problèmes liés au stockage des données . . . 3

1.2.1 Accès aux données . . . 3

1.2.2 Cohérence de l'information . . . 4

1.2.3 Indépendance de la représentation . . . 4

1.3 Les systèmes de gestion de bases de données . . . 4

1.4 Modèles de données . . . 5

2 Modèle et algèbre relationnels **6**

2.1 Notions de base . . . 6

2.2 Contraintes d'intégrité . . . 7

2.2.1 Contraintes de valeur . . . 7

2.2.2 Clefs primaires . . . 7

2.2.3 Clefs étrangères . . . 7

2.3 Base de données relationnelle . . . 8

2.4 Algèbre relationnelle . . . 8

2.4.1 Opérateurs unaires . . . 8

2.4.2 Opérateurs dérivés du produit . . . 9

2.4.3 Opérateurs ensemblistes . . . 9

2.4.4 Division . . . 10

3 Dépendances fonctionnelles et normalisation **11**

3.1 Introduction . . . 11

3.2 Dépendances fonctionnelles . . . 11

3.2.1 Définition . . . 11

3.3 Conséquence logique . . . 12

3.4 Système déductif . . . 12

3.5 Fermeture transitive . . . 13

3.6 Couverture minimale . . . 13

3.7 Normalisation de relations 14
3.7.1 Premières formes normales 15
3.8 Décomposition et synthèse 15
3.8.1 Algorithme de décomposition 16
3.8.2 Algorithme de synthèse 17

4 Transactions **18**
4.1 Définitions et propriétés 18
4.2 Propriétés ACID en pratique 19
4.2.1 Atomicité 19
4.2.2 Cohérence 19
4.2.3 Isolation 19
4.2.4 Durabilité 20
4.3 En Oracle 20

A Oracle **22**

Chapitre 1

Introduction aux systèmes de gestion des bases de données

1.1 Les bases de données

But : archiver un ensemble de données, pour y accéder plus tard. Les besoins ont été accrus par l'avènement :

- des systèmes informatiques
- des réseaux (par exemple Internet)
- des dispositifs mobiles

On parle également de "déluge de données" : on produit maintenant plus de données que l'on ne peut en traiter.

Une base de données est un ensemble cohérent d'informations stockées sur un support à des fins de réutilisations ultérieures. Par exemple, la base de données de la scolarité à l'Ensimag (ensemble des élèves, des notes, etc.).

1.2 Problèmes liés au stockage des données

On souhaite stocker l'information de manière persistante. La garder en RAM ne suffit pas. On peut avoir une persistance en stockant l'information dans un fichier. Il reste quelques problèmes : modifications simultanées, taille, redondance, etc.

1.2.1 Accès aux données

1. Interrogation complexe.
   Par exemple, récupérer les adresses email des étudiants ayant moins de 10 à au moins une matière.
2. Accès optimisé.
   Comment répondre en quelque millisecondes sur des bases de données de plusieurs téra octets ?
3. Contrôle d'accès.
   Par exemple, qui a le droit de connaître (et de modifier) les informations de votre profil sur les réseaux sociaux ?

1.2.2 Cohérence de l'information

1. Contraintes sur les données.
   Par exemple, une note doit être entre 0 et 20.
2. Redondance.
   Par exemple, des informations sur les cours d'une filière apparaissent à deux endroits différents (description de la filière, notes des étudiants). Il y a un risque d'incohérence dans la base de données.
   Remarque : ne pas confondre la redondance subie et la redondance contrôlée, qui est utile pour ne pas perdre d'informations en cas de panne.
3. Concurrence.
   Par exemple, réservation d'un billet de train. On doit éviter que deux voyageurs réservent une même place.
4. Atomicité des mises-à-jour.
   Par exemple, un enseignant décide d'ajouter trois points à tous les étudiants de sa matière, mais le système plante au milieu, résultant en une injustice.
   On doit assurer qu'une mise-à-jour sera soit réalisée avec succès, soit ne sera pas réalisée du tout.

1.2.3 Indépendance de la représentation

1. Indépendance physique.
   Les concepts qu'il y a derrière les données doivent être indépendants de la manière dont sont stockées les données physiquement.
2. Indépendance logique.
   Il y a différentes vues possibles pour un même modèle.

1.3 Les systèmes de gestion de bases de données

Un système de gestion de bases de données (SGBD) est un système dédié au stockage et à l'interrogation de données, et résolvant tous les problèmes précédemment évoqués.

Repères historiques approximatifs :

Années 1960 première génération de SGBD "navigationnels".

Années 1970 modèle relationnel.

Années 1980 troisième génération, relationnel objet.

Années 2000 web, multimedia, information mal structurée, "NoSQL".

À propos de NoSQL

— NoSQL signifie "not only SQL" et non "no SQL".

— Il s'agit de SGBD basés sur des modèles plus simples que le modèle relationnel. Un modèle typique est le modèle clé-valeur, la base de donnée étant alors un simple tableau associatif.

— Ceci vient du fait que les SGBD relationnels sont lents et supportent mal le passage à un environnement distribué (théorème CAP : il est impossible d'assurer des transactions ACID dans un environnement distribué, voir `https://fr.wikipedia.org/wiki/Th%C3%A9or%C3%A8me_CAP`). De plus, on a rarement besoin de toute l'expressivité offerte par le modèle relationnel.

— Solution : NoSQL. Plus simple, plus performant, scalable facilement.

— NoSQL est typiquement utilisé pour des SGBD gérant de très grandes quantités de données. Exemples : BigTable (Google), HBase (Facebook), SimpleDB (Amazon), etc.

Plus d'informations sur Wikipedia : `https://fr.wikipedia.org/wiki/NoSQL`.

1.4 Modèles de données

Une base de données n'est rien d'autre qu'une représentation d'un fragment du "monde réel". Il nous faut un langage permettant de le représenter et tel qu'un ordinateur sot capable de comprendre.

Définition 1 (modèle de données)**.** Langage mathématique de représentation des données. Il contient des primitives permettant de représenter :

- *le schéma*, qui est la structure des données (types d'objets manipulés),
- *les données ou l'extension*, les données elles-mêmes.

On peut citer par exemple les modèles relationnels, hiérarchiques, réseau, graphe, etc.

Chapitre 2

Modèle et algèbre relationnels

Le modèle relationnel a été introduit par Codd au début des années 1970 et est à la base de nombreux SGBD historiques et actuels, et du standard SQL2.

2.1 Notions de base

Définition 2 (schéma de relation)**.** un schéma de relation est un ensemble nommé de couples (nom d'attribut, domaine), noté $S : \{att_1 : D_1, \ldots, att_n : D_n\}$.

Définition 3 (relation)**.** Une relation de schéma $S : \{att_1 : D_1, \ldots, att_n : D_n\}$ est un sous-ensemble du produit cartésien $D_1 \times \cdots \times D_n$.

Exemple. On considère un schéma Élèves : {prénom : String, nom : String, email : String, filière : {IF, ISI, ISSC, MMIS, SLE}}

prénom : String	**nom : String**	**email : String**	**filière : {IF, ISI, ISSC, MMIS, SLE}**
Luke	Skywalker	skywalker@imag.fr	MMIS
Dark	Vador	vador@imag.fr	IF
Han	Solo	solo@falcon.com	IF
Leia	Solo	princess@falcon.com	MMIS
Jabba	The Hut	jabba@imag.fr	ISSC

FIGURE 2.1 – Table *Élèves*

Le schéma est la première ligne (en gras) de la table (aussi appelé "intention"). Le reste de la table constitue les données (aussi appelé "extension de la relation").

Le schéma décrit ce que va contenir la relation. C'est la partie "statique" de la relation. L'ensemble des "lignes" constitue l'extension de la relation (les occurrences des données). Chaque ligne peut être vue comme un objet et est appelé tuple (ou n-uplet).

Exemple de tuple : $("Dark", "Vador", "vador@imag; fr", IF) \in String \times String \times String \times \{IF, ISI, ISSC, MMIS, SLE\}$.

Une relation est un ensemble :

- l'ordre sur les lignes n'a aucune importance,
- il ne peut pas y avoir deux lignes identiques,
- le nombre de tuples est appelé le cardinal de la relation.

Une colonne est appelée un attribut de la relation. Le nombre d'attributs est l'arité de la relation.

Remarque. Ce que l'on nomme ici "relation" est bien une relation au sens mathématique.

2.2 Contraintes d'intégrité

Le modèle relationnel permet d'exprimer des contraintes sur les données de la base. Ces contraintes sont définies par le concepteur de la base sur les schémas des relations, et ce sont les données qui doivent s'y conformer.

2.2.1 Contraintes de valeur

Définition 4 (contrainte de valeur)**.** Une contrainte de valeur est un prédicat logique sur un ou plusieurs attributs d'un schéma. Une telle contrainte est vérifiée si tous les tuples satisfont le prédicat logique.

Exemple. Dans la relation Notes, $note \in [0, 20]$.

2.2.2 Clefs primaires

Définition 5 (clef)**.** Soit R une relation. L'ensemble d'attributs K est une clef de R si et seulement si $\forall (t_1, t_2) \in R^2, t_1[K] = t_2[K] \Leftrightarrow t_1 = t_2$, où $t_i[K]$ désigne la restriction de t_i aux seuls attributs de K.

La notion de clef correspond à une notion d'identifiant : les attributs de K sont suffisants pour identifier un tuple sans ambiguïté.

Exemple. $\{email\}$ est une clef de Élèves, $\{nom, prenom\}$ est également une clef, mais $\{nom\}$ n'en est pas une.

Remarque. Il y a toujours au moins une clef, constituée de tous les attributs.

Définition 6 (clef primaire)**.** Une contrainte de clef primaire est un couple (S, K) où S est un schéma et K est un ensemble d'attributs de S. Une telle contrainte est vérifiée par une relation R si et seulement si K est une clef de R.

Exemple. Élèves : {prénom, nom, email, filière}

Cette notation signifie que {prénom, nom} est une clef primaire.

2.2.3 Clefs étrangères

Exemple. On considère un schéma Notes : {cours : String, prénom : String, nom : String, note : Int}

cours : String	prénom : String	nom : String	note : Int
sport	Dark	Vador	20
sport	Jabba	The Hut	3
pilotage	Han	Solo	15

FIGURE 2.2 – Table *Notes*

Plusieurs problèmes peuvent survenir :

- si l'on ajoute un tuple (pilotage, Obi-Wan, Kenobi, 16), on a une inconsistance, puisque cet élève n'existe pas dans la table Élèves,
- si l'on enlève l'attribut "prénom", on a une ambiguïté, à cause du nom "Solo".

Définition 7 (clef étrangère)**.** La séquence d'attributs K est une clef étrangère de la relation R référençant la séquence d'attributs K' de la relation R' si et seulement si :

- $\forall t \in R, \exists t' \in R', t[K] = t'[K']$
- K' est une clef de R'

Définition 8 (contrainte de référence)**.** Une contrainte de référence est un triplet (S, S', f) où S et S' sont deux schémas et f est une application injective d'un sous-ensemble K d'attributs de S vers un sous-ensemble K' d'attributs de S'.

Une telle contrainte est vérifiée par R et R' si et seulement si $(k_1, \ldots, k_n)$ est une clef étrangère de R référençant $(f(k_1), \ldots, f(k_n))$ de R', où $K = \{k_1, \ldots, k_n\}$.

Exemple. Notes : {cours, prénom*, nom*, note}

*(prénom, nom) référence Élèves (prénom, nom)

2.3 Base de données relationnelle

Définition 9 (base de données relationnelle (BDR))**.** Un schéma de bases de données relationnelles est constitué :

— d'un ensemble de schémas de relations,

— d'une et une seule contrainte de clef primaire par schéma de relation,

— d'un ensemble de contraintes de valeur et de référence.

Une BDR est un schéma de BDR accompagné de données vérifiant toutes ses contraintes.

2.4 Algèbre relationnelle

Nous allons maintenant doter le modèle relationnel d'un certain nombre d'opérateurs d'interrogation. Ces opérateurs sont des opérateurs internes transformant :

— une relation en une autre relation (opérateurs unaires),

— deux relations en une autre relation (opérateurs binaires).

On définit une algèbre au sens "informatique" du terme, i.e. un ensemble (l'ensemble des relations) doté d'opérateurs internes.

2.4.1 Opérateurs unaires

Définition 10 (projection)**.** La projection de R sur les attributs $\{att_1, \ldots, att_k\}$ est la restriction de R à ces seuls attributs.

$$\Pi_{\{att_1,\ldots,att_k\}}(R) = \{t\,[\{att_1, \ldots, att_k\}] \mid t \in R\}$$

Exemple. Projection de *Élèves* sur l'ensemble d'attributs $\{nom\}$.

nom : String
Skywalker
Vador
Solo
The Hut

FIGURE 2.3 – Table $\Pi_{\{nom\}}$*(Élèves)*

Remarque. Ici, on a perdu une ligne, à cause du nom "Solo".

Définition 11 (selection)**.** La sélection de R selon le prédicat P est la restriction de R aux seuls tuples vérifiant P.

$$\sigma_P(R) = \{t \in R \mid P(t)\}$$

Exemple. Sélection de *Notes* avec le prédicat $note < 10$.

cours : String	prénom : String	nom : String	note : Int
sport	Jabba	The Hut	3

FIGURE 2.4 – Table $\sigma_{note<10}(Notes)$

2.4.2 Opérateurs dérivés du produit

Définition 12 (produit)**.** Le produit cartésien de deux relations R_1 et R_2 est le produit constitué de toutes les concaténations possibles d'un tuple $t_1 \in R_1$ et d'un tuple $t_2 \in R_2$.

$$R_1 \times R_2 = \{t_1 \cdot t_2 \mid t_1 \in R_1 \wedge t_2 \in R_2\}$$

Exemple. $Notes \times Eleves$ a 8 attributs et 15 tuples.

Exemple. On peut obtenir les emails des étudiants ayant moins de 10 à une matière en appliquant les opérateurs suivants :

$$\Pi_{\{email\}} \left(\sigma_{note<10} \left(\sigma_{Notes.prenom=Eleves.prenom \wedge Notes.nom=Eleves.nom} \left(Notes \times Eleves \right) \right) \right)$$

Définition 13 (jointure conditionnelle)**.** La jointure conditionnelle de R_1 et R_2 selon le prédicat P est la relation

$$R_1 \bowtie_P R_2 = \sigma_P(R_1 \times R_2)$$

Exemple. On peut obtenir les emails des étudiants ayant moins de 10 à une matière en appliquant les opérateurs suivants :

$$\Pi_{\{email\}} \left(Notes \bowtie_{Notes.prenom=Eleves.prenom \wedge Notes.nom=Eleves.nom \wedge note<10} Eleves \right)$$

Définition 14 (jointure naturelle)**.** La jointure naturelle de R_1 et R_2, notée $R_1 \bowtie R_2$, est le produit cartésien $R_1 \times R_2$ dans lequel :

- on ne garde que les tuples dont les valeurs coïncident sur les attributs communs (de même nom) de R_1 et R_2,
- on ne garde qu'une seule version de chaque attribut commun.

Exemple. $Eleves \bowtie Notes$ a 6 attributs et 3 tuples.

Exemple. On peut obtenir les emails des étudiants ayant moins de 10 à une matière en appliquant les opérateurs suivants :

$$\Pi_{\{email\}} \left(\sigma_{note<10} (Notes \bowtie Eleves) \right)$$

2.4.3 Opérateurs ensemblistes

Définition 15 (opérateurs ensemblistes)**.** Soient R_1 et R_2 deux relations de même schéma (même nombre et types d'attributs).

- L'union de R_1 et R_2 est $R_1 \cup R_2 = \{t \mid t \in R_1 \vee t \in R_2\}$.
- L'intersection de R_1 et R_2 est $R_1 \cap R_2 = \{t \mid t \in R_1 \wedge t \in R_2\}$.
- La différence de R_1 et R_2 est $R_1 \setminus R_2 = \{t \mid t \in R_1 \wedge t \notin R_2\}$.

Exemple. L'ensemble des élèves qui n'ont pas de notes est obtenu en appliquant les opérateurs suivants :

$$\Pi_{\{prenom,nom\}}(Eleves) \setminus \Pi_{\{prenom,nom\}}(Notes)$$

La liste des emails des élèves qui n'ont pas de notes est obtenue de cette façon :

$$\Pi_{\{email\}} \left(Eleves \bowtie \Pi_{\{prenom,nom\}}(Eleves) \setminus \Pi_{\{prenom,nom\}}(Notes) \right)$$

2.4.4 Division

Définition 16 (division)**.** Soient R_1 et R_2 deux relations telles que l'ensemble des attributs de R_2 soit inclut dans celui de R_1. On note Att_2 l'ensemble des attributs de R_2, $Att_1 = Att_2 \cup Att_3$ l'ensemble des attributs de R_1.

La division de R_1 par R_2 est l'ensemble des tuples t_3 sur Att_3 tels que pour chaque tuple $t_2 \in R_2$ (sur Att_2), le tuple $(t_2 \cdot t_3)$ (sur Att_1) existe dans R_1.

On note $R_1 \div R_2$.

Exemple. On considère les deux tables suivantes :

nom : String	**filière : {IF, ISI, ISSC, MMIS, SLE}**
Skywalker	MMIS
Vador	IF
Solo	IF
Solo	MMIS
The Hut	ISSC

FIGURE 2.5 – Table $R_1 = \Pi_{\{filiere,nom\}}$*(Élèves)*

filière : {IF, ISI, ISSC, MMIS, SLE}
MMIS
IF

FIGURE 2.6 – Table R_2

La division de R_1 par R_2 est :

nom : String
Solo

FIGURE 2.7 – Table $R_1 \div R_2$

Remarque. Pourquoi "division" ? C'est certainement l'opération inverse du produit.

nom : String	**filière : {IF, ISI, ISSC, MMIS, SLE}**
Solo	MMIS
Solo	IF

FIGURE 2.8 – Table $(R_1 \div R_2) \times R_2$

On n'a pas $(R_1 \div R_2) \times R_1 = R_1$, mais $(R_1 \div R_2) \times R_1 \subseteq R_1$.

Plus précisément, cela ressemble à la division euclidienne : on a $R_1 = \underbrace{(R_1 \div R_2)}_{"quotient"} \times R_2 \cup \underbrace{R_3}_{"reste"}$

En effet, la division euclidienne (sur $\mathbb{N}$) est : $n_1 \div n_2 = \max\{k \in \mathbb{N} \mid n_1 \geqslant k \times n_2\}$

La division relationnelle : $R_1 \div R_2 = \max_{\subseteq}\{R_3 \mid R_2 \times R_3 \subseteq R_1\}$

Remarque. La division peut s'exprimer à l'aide des autres opérateurs.

$$R_1 \div R_2 = \Pi_{Att(R_1)\setminus Att(R_2)}(R_1) - \Pi_{Att(R_1)\setminus Att(R_2)}\left(\Pi_{Att(R_1)-Att(R_2)}(R_1) \times R_2 - R_1\right)$$

Chapitre 3

Dépendances fonctionnelles et normalisation

3.1 Introduction

cours : String	volume : Int	prénom : String	nom : String	email : String	note : Int
sport	20	Dark	Vador	vador@imag.fr	20
sport	20	Jabba	The Hut	jabba@tatooine.ta	3
pilotage	40	Han	Solo	solo@falcon.com	15
sport	20	Han	Solo	solo@falcon.com	10

FIGURE 3.1 – Table *Notes*

Il y a de la redondance dans la table 3.1. On peut s'en apercevoir en regardant les données : le cours "sport" et le volume "20" semblent toujours associés, de même que le prénom "Han", le nom "Solo" et l'email "solo@falcon.com". Cependant, c'est bien notre connaissance du domaine (ce que représentent les attributs) qui nous permet de le dire. Cette redondance vient du fait, par exemple, qu'un même cours sera toujours associé à un même volume horaire. C'est un exemple d'une dépendance fonctionnelle.

En quoi cette redondance est-elle un problème ?

— Perte d'efficacité (temporelle et spatiale), l'information est stockée plusieurs fois.

— Risque d'incohérence.

— Risque de perte d'information. Par exemple, que se passe-t-il si l'on supprime la note de Han Solo au cours de pilotage ? On perd le volume horaire du cours de pilotage.

Comment faire pour résoudre ces problèmes ? Il faut probablement "découper" la relation (mais pas n'importe comment).

Objectifs du chapitre :

1. Identifier les redondances potentielles (anomalies de représentation). Dépendances fonctionnelles.
2. "Découper" de manière correcte.

Question de la conception du schéma relationnel.

3.2 Dépendances fonctionnelles

3.2.1 Définition

Définition 17 (dépendance fonctionnelle)**.** Soit S un schéma de relation. Une dépendance fonctionnelle (DF) sur S est un couple (X, Y) noté $X \to Y$, où X et Y sont deux sous-ensembles d'attributs de S.

Une relation R de schéma S vérifie la dépendance fonctionnelle $X \to Y$ si et seulement si la condition suivante est satisfaite :

$$\forall (t_1, t_2) \in R^2, t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$$

Exemple. La relation *Notes* vérifie les dépendances fonctionnelles :

- $cours \to volume$
- $prenom, nom \to email$
- $email \to prenom, nom$
- $cours, prenom, nom \to email, note$
- etc.

Mais elle ne vérifie pas $prenom, nom \to cours$.

Si R vérifie $X \to Y$, on dira que X détermine Y dans R et on notera $R \models (X \to Y)$.

Pour un ensemble F de dépendances fonctionnelles, on notera $R \models F$ si et seulement si $\forall f \in F, R \models f$.

Remarque. Pourquoi dépendance fonctionnelle ? Si $R \models (X \to Y)$, alors $\Pi_{X,Y}(R)$ définit une fonction sur $X \times Y$. En effet, $\Pi_{X,Y}(R)$ est un sous-ensemble de $X \times Y$ tel que tout $x \in X$ a une unique image dans cette relation.

Remarque. Les dépendances fonctionnelles sont extraites de la connaissance du domaine d'application, et non des données elles-mêmes. Par exemple, $(note, cours)$ n'est pas une dépendance fonctionnelle, même si dans notre exemple les notes sont uniques.

Remarque.
- On a vu dans un exercice que c et $NB \to ND$. A-t-on automatiquement $NT \to ND$? Oui, car $NT \to ND$ est une conséquence logique de $\{NB \to ND, NT \to ND\}$.
- On a $NE \to ND$ et $NE \to NP$. A-t-on $NE \to ND, NP$? Oui, $NE \to ND, NP$ est une conséquence logique de $\{NE \to ND, NE \to NP\}$.

À partir d'un ensemble de dépendances fonctionnelles, on peut en dériver plein d'autres, qui sont logiquement équivalentes. Lequel de ces ensembles va-t-on utiliser pour l'identification des redondances et le découpage des tables ?

3.3 Conséquence logique

Définition 18 (conséquence logique)**.** Soit S un schéma et F un ensemble de dépendances fonctionnelles sur S. La dépendance fonctionnelle f est une conséquence logique de F si et seulement si, pour toute relation R de schéma S, $R \models F \Rightarrow R \models f$.

Exemple. $\{(email \to prenom, nom), (prenom, nom, cours \to note)\}$ a pour conséquence logique $email, cours \to note$.

3.4 Système déductif

Pour montrer qu'une dépendance fonctionnelle f est conséquence logique d'un ensemble F, on va utiliser le système déductif suivant.

Définition 19 (système formel d'Armstrong)**.** Le système d'Armstrong est défini par les trois règles de base suivantes :

Réflexivité : si $X \subseteq Y$, alors $Y \to X$.

Augmentation : si $X \to Y$ alors $X, Z \to Y, Z$.

Transitivité : si $X \to Y$ et $Y \to Z$ alors $X \to Z$.

Ainsi que les trois règles dérivées suivantes :

Décomposition : si $Z \subseteq Y$ et $X \to Y$ alors $X \to Z$.

Union : si $X \to Y$ et $X \to Z$ alors $X \to Y, Z$.

Pseudo-transitivité : si $X \to Y$ et $W, Y \to Z$ alors $W, X \to Z$.

Exemple. Pour démontrer notre précédent exemple, on peut appliquer la pseudo-transitivité avec $X = \{prenom, nom\}$, $Y = \{email\}$, $W = \{cours\}$ et $Z = \{note\}$.

Proposition 1. *Toute conséquence logique peut être obtenue à partir de l'ensemble initial par applications successives des trois règles du système d'Armstrong.*

3.5 Fermeture transitive

Définition 20 (fermeture transitive)**.** La fermeture transitive d'un ensemble F de dépendances fonctionnelles, notée F^+, est l'ensemble des conséquences logiques de F :

$$F^+ = \{f \mid F \models f\}$$

Remarque. Cet ensemble existe et est unique.

Définition 21 (équivalence logique)**.** Deux ensembles de dépendances fonctionnelles F et G sont logiquement équivalents si et seulement si $F^+ = G^+$.

3.6 Couverture minimale

Définition 22 (forme canonique)**.** Une dépendance fonctionnelle $X \to Y$ est sous forme canonique si et seulement si Y ne contient qu'un seul élément.

Exemple. $email \to prenom, nom$ n'est pas sous forme canonique.

Remarque. L'ensemble $\{X \to Y_1, Y_2, \ldots, Y_n\}$ est logiquement équivalent à l'ensemble $\{X \to Y_1, X \to Y_2, \ldots, X \to Y_n\}$ (preuve : utiliser les règles de décomposition et d'union).

Définition 23 (dépendance fonctionnelle élémentaire)**.** Une dépendance fonctionnelle $(X \to Y) \in F$ est élémentaire par rapport à F si et seulement si :

- elle est sous forme canonique,
- $Y \not\subseteq X$ (on dit qu'elle est non triviale),
- $\nexists X' \subsetneq X$ tel que $F \models (X' \to Y)$ (on dit qu'elle est minimale).

Définition 24 (couverture minimale)**.** La couverture minimale (ou irredondante) d'un ensemble F de dépendances fonctionnelles, notée $irr(F)$, est un ensemble tel que :

- $\forall f \in irr(F)$, f est élémentaire par rapport à F,
- $(irr(F))^+ = F^+$ ($irr(F)$ et F sont équivalents),
- $\forall f \in irr(F)$, $(irr(F) \setminus \{f\})^+ \neq F^+$ (minimalité).

Remarque. Tout ensemble a au moins une couverture minimale, qui en général n'est pas unique.

On donne maintenant un algorithme de calcul d'une couverture minimale.

1. Mettre sous forme canonique.
2. Mettre sous forme élémentaire.
3. Éliminer les redondants.

Exemple. $F = \{(a \to b, c), (a, b \to d), (d \to c)\}$

1. Mise sous forme canonique.
   $F' = \{(a \to b), (a \to c), (a, b \to d), (d \to c)\}$

Fonction minimalCover(F)

foreach $f \in F$ **do**
  mettre f sous forme canonique
foreach $X \to Y \in F$ **do**
  foreach $x \in X$ **do**
    if $F \models (X \setminus \{x\} \to Y)$ **then**
      remplacer $X \to Y$ par $X \setminus \{x\} \to Y$ dans F
foreach $f \in F$ **do**
  if $F \setminus \{f\} \models f$ **then**
    enlever f de F

FIGURE 3.2 – Algorithme de calcul d'une couverture minimale d'un ensemble F

2. Mise sous forme élémentaire. La seule dépendance fonctionnelle potentiellement non élémentaire est $a, b \to d$.
   Éliminons b. A-t-on $F' \models \{a \to d\}$? Oui. $a \to b$ donc $a \to a, b$ (augmentation), or $a, b \to d$ donc $a \to d$ (transitivité).
   On a donc l'ensemble $F'' = \{(a \to b), (a \to c), (a \to d), (d \to c)\}$
3. Élimination des redondances.
   $a \to b$ n'est pas redondante, car b n'apparaît pas ailleurs en partie droite.
   $a \to c$ est redondante, car on peut la retrouver par transitivité. On l'élimine.
   $a \to d$ n'est pas redondante, même argument.
   $d \to c$ n'est pas redondante.
   Finalement, la couverture minimale est l'ensemble $F'' = \{(a \to b), (a \to d), (d \to c)\}$

3.7 Normalisation de relations

L'idée sous-jacente à la normalisation est de pouvoir caractériser très précisément le degré d'anomalie d'un schéma, grâce (surtout) aux dépendances fonctionnelles. Cette caractérisation passe par une échelle qualitative, les formes normales.

Définition 25 (clef)**.** Soit S un schéma, et F un ensemble de dépendances fonctionnelles. L'ensemble des attributs K est une clef de (S, F) si et seulement si :
— $F \models (K \to att(S))$
— $\nexists K' \subset K, F \models (K' \to att(S))$

On appellera attribut-clelf tout attribut faisant partie d'au moins une clef.

Exemple. — $nom, prenom \to email$
— $email \to nom, prenom$
— $cours, email \to note$
— $cours \to volume$

Clefs : $\{\{cours, email\}, \{cours, nom, prenom\}\}$.

Attributs clefs : $\{cours, email, nom, prenom\}$.

Attributs non-clefs : $\{volume, note\}$.

Définition 26 (dépendance pleine et partielle)**.** Soient S un schéma, F un ensemble de dépendances fonctionnelles, et X et Y deux sous-ensembles d'attributs de S tels que $F \models (X \to Y)$. On dit que Y est pleinement dépendant de X si $\forall y \in Y\ (X \to y)$ est une dépendance fonctionnelle élémentaire par rapport à F. Sinon, Y est partiellement dépendant de X.

Exemple. $\{prenom, nom\}$ est partiellement dépendant de $\{cours, email\}$ car $\{prenom, nom\}$ est pleinement dépendant d'un sous ensemble strict : $\{email\}$.

3.7.1 Premières formes normales

Définition 27 (1FN)**.** Un schéma est en première forme normale (noté 1FN, 1NF, FN1, NF1) si et seulement si tout attribut est de type atomique.

Atomicité : pas une liste d'éléments. Par exemple, si l'on avait un attribut *emails* (dénottant une liste de plusieurs emails), ce ne serait pas un attribut atomique. Solution dans ce cas : créer une table *Emails*, avec pour attributs *prénom*, *nom*, *email*. On aurait alors un attribut par adresse email.

Définition 28 (2FN)**.** Un couple schéma ensemble de dépendances fonctionnelles (S, F) est en deuxième forme normale (2FN, FN2, 2NF, NF2) si et seulement si S est en 1FN et tout les attributs non-clefs sont pleinement dépendants de chacune des clefs.

Exemple. On reprend le schéma *Notes*.

(S, F) n'est pas en 2FN car $F \models (cours \rightarrow volume)$. Or, $\{cours\}$ est un sous-ensemble strict de la clef $\{cours, email\}$, donc $volume$ ne dépend que partiellement de $\{cours, email\}$.

Définition 29 (3FN)**.** Un couple schéma ensemble de dépendances fonctionnelles (S, F) est en troisième forme normale (3FN, FN3, 3NF, NF3) si et seulement si (S, F) est en 2FN et aucun attribut non clef ne dépend d'un autre attribut non-clef (on dit aussi que tout attribut est pleinement (2FN) et directement (3FN) dépendant des clefs).

Exemple. On reprend le schéma *Notes*, dans lequel on enlève l'attribut $volume$.

(S, F) est en 2FN, car *note* dépend pleinement des clefs.

C'est également en 3FN, car *notes* est le seul attribut non clef.

Exemple. Même schéma (sans $volume$), mais en ajoutant un attribut booléen $rattrapage$, et la dépendance fonctionnelle $note \rightarrow rattrapage$.

(S, F) est 2FN, mais pas 3FN.

Définition 30 (3FNBCK)**.** Un couple schéma ensemble de dépendances fonctionnelles (S, F) est en troisième forme normale de Boyce-Codd-Kent (3FNBCK, 3BCKNF, FNBCK, BCKNF, BCNF, FNBC) si et seulement si (S, F) est en 3FN et toute dépendance fonctionnelle $X \rightarrow Y$ non triviale (i.e. $Y \not\subset X$) conséquence logique de F contient une clef en partie gauche.

Exemple. Même schéma (sans $volume$ ni $rattrapage$).

(S, F) est en 3FN.

Ce n'est pas 3FNBCK, car $F \models (\{prenom, nom\} \rightarrow email)$ et $\{prenom, nom\}$ n'est pas une clef.

3.8 Décomposition et synthèse

Nous allons maintenant tâcher de répondre à la question suivante. Si (S, F) n'est pas en 3FN(BCK), comment le décomposer en "sous-schémas", tous bien normalisés.

Cette décomposition devra être :

- sans perte d'information (SPI),
- sans perte de dépendance fonctionnelle (SPD).

Définition 31 (décomposition sans perte d'information)**.** Un couple (S, F) est décomposable sans perte d'information en $(S_1, F_1), (S_2, F_2)$ si et seulement si, pour toute relation R de S satisfaisant toutes les dépendances fonctionnelles de F, on a

$$R = \Pi_{att(S_1)}(R) \bowtie \Pi_{att(S_2)}(R)$$

Définition 32 (décomposition sans perte de dépendance fonctionnelle)**.** Un couple (S, F) est décomposable sans perte de dépendance fonctionnelle en $(S_1, F_1), (S_2, F_2)$ si et seulement si :

— $att(S) = att(S_1) \cup att(S_2)$

— $(F_1 \cup F_2)^+ = F^+$

Remarque. L'une de ces décompositions n'implique pas l'autre de ces décompositions.

Nous allons introduire deux approches pour décomposer.

— Décomposition : on part du schéma universel initial (qui contient tous les attributs) et on découpe jusqu'à obtenir des schémas 3FN.

— Synthèse : on part des dépendances fonctionnelles et on s'en sert pour synthétiser des schémas.

3.8.1 Algorithme de décomposition

Fonction decomposition((U, F))

$res \leftarrow \{(U, F)\}$
while $\exists (S, F) \in res$ *tel que* (S, F) *n'est pas 3FNBCK* **do**
  Chercher dans (S, F) une DF $X \rightarrow Y$ non triviale telle que X ne contient pas de clef
  Remplacer (S, F) dans *res* par $(S_1, F_1), (S_2, F_2)$ tels que
    — $att(S_1) = X \cup Y$ et $att(S_2) = att(S) \setminus Y$
    — F_1, F_2 correspondent respectivement à la projection de F^+ sur S_1 et S_2

FIGURE 3.3 – Algorithme de décomposition

Cet algorithme assure une décomposition SPI, mais pas SPF. Tous les couples résultants sont en 3FN (et même 3FNBCK ?).

Exemple. On reprend l'exemple précédent : $Notes = \{cours, volume, prenom, nom, email, note\}$ et $F = \{(cours \rightarrow volume), (email \rightarrow prenom, nom), (prenom, nom \rightarrow email), (cours, email \rightarrow note)\}$.

$(Notes, F)$ n'est pas en 2FN, à cause de $cours \rightarrow volume$.

— $res \leftarrow \{(Notes, F)\}$

— $cours \rightarrow volume$ est "fautive", on s'en sert pour décomposer.
  — (S_1, F_1) avec $S_1 = \{cours, volume\}$, $F_1 = \{(cours \rightarrow volume)\}$
  — (S_2, F_2) avec $S_2 = \{cours, prenom, nom, email, note\}$, $F_2 = \{(email \rightarrow prenom, nom), (prenom, nom \rightarrow email), (cours, email \rightarrow note)\}$

— $(S_1, F_1) : clefs = \{\{cours\}\}$, $att\_clefs = \{cours\}$, $att\_non\_clefs = \{volume\}$, donc c'est bien 3FNBCK.

— $(S_2, F_2) : clefs = \{\{cours, email\}, \{cours, prenom, nom\}\}$, $att\_clefs = \{cours, email, prenom, nom\}$, $att\_non\_clefs = \{note\}$. C'est bien 2FN et 3FN, mais pas 3FNBCK, à cause par exemple de $email \rightarrow prenom, nom$.

— On se sert de $email \rightarrow prenom, nom$ pour décomposer S_2, F_2.
  — (S_3, F_3) avec $S_3 = \{email, prenom, nom\}$, $F_3 = \{(email \rightarrow prenom, nom), (prenom, nom \rightarrow email)\}$
  — (S_4, F_4) avec $S_4 = \{email, cours, note\}$, $F_4 = \{(cours, email \rightarrow note)\}$

— $(S_3, F_3) : clefs = \{\{email\}, \{prenom, nom\}\}$. C'est bien 3FNBCK.

— $(S_4, F_4) : clefs = \{\{cours, email\}\}$. C'est bien 3FNBCK.

Fonction synthese((U, F))

Calculer la couverture minimale $im(F)$
Partitionner $im(F)$ en regroupant toutes les DF ayant la même partie gauche
Pour chaque classe d'équivalence, construire un couple (S, F) où F contient toutes les DF de cette classe d'équivalence et S tous les attributs correspondants
Fusionner les (S, F) si besoin (par exemple, s'il existe $(S, F), (S', F')$ avec $S' = S$)
S'il n'existe pas de (S, F) tel que S contient une clef des DF initiales, ajouter un tel schéma

FIGURE 3.4 – Algorithme de synthèse

3.8.2 Algorithme de synthèse

Exemple. On reprend $(Notes, F)$.

- $im(F) = \{(cours \to volume), (email \to prenom), (email \to nom), (prenom, nom \to email), (cours, email \to note)\}$
- Classes d'équivalences :
  - $\{(cours \to volume)\}$
  - $\{(email \to prenom), (email \to nom)\}$
  - $\{(prenom, nom \to email)\}$
  - $\{(cours, email \to note)\}$
- Schémas :
  - $(S_1, F_1) = (\{cours, volume\}, \{(cours \to volume)\})$
  - $(S_2, F_2) = (\{email, prenom, nom\}, \{(email \to prenom), (email \to nom)\})$
  - $(S_3, F_3) = (\{prenom, nom, email\}, \{(prenom, nom \to email)\})$
  - $(S_4, F_4) = (\{cours, email, note\}, \{(cours, email \to note)\})$
- On fusionne (S_2, F_2) et (S_3, F_3)
- On ne rajoute pas de schéma.

L'algorithme de synthèse garantit une décomposition SPD, et SPI uniquement si la dernière étape est appliquée.

Tous les couples (S, F) créés sont en 3FN au moins.

Chapitre 4

Transactions

Exemple. Considérons un système de gestion de bases de données gérant des données bancaires. On souhaite effectuer un virement de 42k€ du compte c_1 au compte c_2.

Une exécution possible est (en terme de requêtes) :

1. SELECT solde FROM compte WHERE noCompte = "c_1"
2. Si solde > 42000€
3. UPDATE compte SET solde = solde - 42000 WHERE noCompte = "c_1"
4. UPDATE compte SET solde = solde + 42000 WHERE noCompte = "c_2"

Que se passe-t-il si :

— Un autre utilisateur vide le compte c_1 en parallèle entre les requêtes 1 et 3 ? La banque n'est pas contente.

— Le système plante entre les requêtes 3 et 4 ? Les clients de la banque ne sont pas contents.

Il faut que le SGBD nous protège de ces situations. C'est le rôle des transactions.

4.1 Définitions et propriétés

Définition 33 (transaction)**.** Une transaction est une séquence de requêtes vue comme une unité logique de traitement (une fonctionnalité).

Les transactions sont caractérisées par les propriétés ACID.

A Atomicité.

C Cohérence.

I Isolation.

D Durabilité.

Nous allons détailler ces propriétés par la suite. En pratique, pour définir une transaction, on utilise trois mots clefs SQL.

BEGIN Débuter une transaction.

COMMIT Valider la transaction (la terminer et appliquer les changements).

ROLLBACK Annuler la transaction (la terminer sans appliquer les changements).

4.2 Propriétés ACID en pratique

4.2.1 Atomicité

Cette propriété stipule qu'une transaction s'exécute en totalité ou pas du tout.

En pratique, deux mécanismes :

— Travail sur une copie temporaire des données.

— Stockage des états intermédiaires dans des fichiers journaux (log). Au redémarrage, le SGBD remonte les logs jusqu'au dernier BEGIN.

4.2.2 Cohérence

Cette propriété stipule qu'une transaction amène la base de données d'un état cohérent (au sens des contraintes d'intégrité) vers un autre état cohérent.

En pratique, en SQL, on peut ajouter le mot-clef DEFERRABLE (resp. IMMEDIATE) après la définition d'une contrainte, stipulant que cette contrainte ne sera vérifiée qu'après chaque COMMIT (resp. après chaque requête individuelle).

4.2.3 Isolation

Cette propriété stipule qu'une transaction s'exécute comme si elle était seule à utiliser le SGBD. Les effets observés par une transaction sont les mêmes, qu'elle soit seule sur le système ou qu'il y ait d'autres transactions en parallèle.

Exemple. Considérons deux transactions T_1 et T_2 exécutées en parallèle.

T_1 :

1. $t \leftarrow lire(A)$
2. $t \leftarrow 100 + t$
3. $ecrire(A, t)$
4. $t \leftarrow lire(B)$
5. $t \leftarrow 100 + t$
6. $ecrite(B, t)$

T_2 :

1. $s \leftarrow lire(A)$
2. $s \leftarrow 2s$
3. $ecrire(A, s)$
4. $s \leftarrow (A, s)$
5. $s \leftarrow lire(B)$
6. $s \leftarrow 2s$
7. $ecrire(B, s)$

Initialement, $A = 25$ et $B = 25$.

En pratique, T_1 et T_2 peuvent être mélangées. Que signifie l'isolation dans ce contexte ? Les mêmes effets sont observés au cours d'une exécution que si T_1 et T_2 s'exécutaient en séquence (donc seules). Les deux seules exécutions séquentielles possibles sont T_1, T_2 ou T_2, T_1. T_1, T_2 mène à $A = 250, B = 250$ et T_2, T_1 mène à $A = 150, B = 150$.

Notons A_1 (resp. A_2) les trois premières requêtes de T_1 (resp. T_2). et B_1 (resp. B_2) les trois dernières requêtes de T_1 (resp. T_2).

L'exécution parallèle A_1, A_2, B_1, B_2 satisfait la propriété d'isolation, car le résultat final est $A = 250, B = 250$, qui correspond à une exécution séquentielle.

L'exécution parallèle A_1, A_2, B_2, B_1 ne satisfait pas la propriété d'isolation, car le résultat final est $A = 250, B = 150$, ne correspond à aucune exécution séquentielle. On peut observer qu'il y a eu une exécution parallèle, donc il n'y a pas d'isolation.

Cette notion d'isolation, très forte, correspond à la sérialisabilité.

Définition 34 (sérialisabilité)**.** Un ordonnancement de requêtes issues de plusieurs transactions concurrentes est sérialisable si son exécution produit le même effet qu'un ordonnancement séquentiel des transactions.

Solution : utiliser des verrous (c'est le même problème qu'en programmation concurrente).

En pratique, exiger la sérialisabilité est trop fort (cela produit une trop importante baisse de performances). En SQL, on peut relâcher ce niveau d'isolation et en choisir un plus faible, au risque d'être confronté à trois types de problèmes.

Définition 35 (lecture sale)**.** Une transaction T_2 lit des données d'une transaction T_1 non validée.

Exemple. T_1 écrit A, T_2 lit A, T_1 abandonne, T_2 exploite la valeur incorrecte de A.

Définition 36 (lecture non reproductible)**.** Une transaction T_2 relit des données qu'elle a déjà lues et trouve une valeur changée du fait d'une autre transaction T_1 validée entre temps.

Exemple. T_2 lit A, T_1 écrit A, T_1 valide, T_2 relit A (nouvelle valeur validée).

Définition 37 (lecture fantôme)**.** Une transaction T_2 réexécute une requête sur une table donnée et trouve un nombre de lignes modifié du fait d'une autre transaction T_1 validée entre temps.

Exemple. T_2 compte le nombre de tuples de R, T_1 insère un tuple dans R, T_1 valide T_2 recompte le nombre de tuples dans R.

SQL propose quatre modes d'isolation.

	lecture sale	lecture non reproductible	lecture fantôme	contrôle de concurrence
READ UNCOMMITTED	×	×	×	$ecriture \rightarrow ecriture$
READ COMMITTED	✓	×	×	$+ecriture \rightarrow lecture$
REPEATABLE READ	✓	✓	×	$+lecture \rightarrow ecriture$
SERIALIZABLE	✓	✓	✓	$+lecture \rightarrow insertion$

✓ : protégé contre.

× : non protégé contre.

$X \rightarrow Y$: l'opération Y ne peut pas s'effectuer après l'opération X sur la même donnée. Par exemple, pour $ecriture \rightarrow ecriture$ signifie que si T_1 écrit A, alors aucune autre transaction n'a le droit d'écrire A avant que T_1 valide.

4.2.4 Durabilité

Cette propriété stipule que les effets d'une transaction validée sont permanents (notion de persistance).

Remarque. En SQL, on peut définir des points de sauvegarde intermédiaires avec *SAVEPOINT <nom>* et y revenir avec *ROLLBACK <nom>*.

4.3 En Oracle

Il y a deux niveaux d'isolation : READ COMMITTED et SERIALIZABLE.

Il y a deux mécanismes.

- Copie locale. Chaque transaction travaille sur une “copie locale” des données.
- Mécanisme de verrouillage à deux niveaux.
  - Lorsqu’une transaction accède à une donnée en lecture, elle doit obtenir un verrou sur sa copie locale.
  - Lorsqu’une transaction accède à une donnée en écriture, elle doit obtenir un verrou sur toute les copies de toutes les transactions.

Dans le mode READ COMMITED, la copie locale est synchronisée dès que n’importe quelle transaction valide.

Dans le mode SERIALIZABLE, la copie locale est synchronisée à la validation de cette transaction locale.

Annexe A

Oracle

Syntaxe simplifiée de Oracle

ALTER SEQUENCE [<schéma>] <séquence> (
INCREMENT BY <entier> |
(**MAXVALUE** <entier> | **NOMAXVALUE**) |
(**MINVALUE** <entier> | **NOMINVAMUE**))*;

ALTER SESSION SET nls_date_format = <valeur nls>;

ALTER TABLE [<schéma>.]<table> (
(**ADD ((**<colonne> <type> [<contrainte de colonne>]) |
(**DEFAULT** <expression>) |
<contrainte de table>)[,])*)) |
(**MODIFY ((**<colonne> <type> [<contrainte de colonne>]) |
(**DEFAULT** <expression>) |
<contrainte de table>)[,])*)) |
(**DROP** <clause drop>) |
(**HIDE** [**COLUMN**] <colonne>) |
(**ATTACH JAVA** (**CLASS** | **SOURCE**) « <nom de fichier> » **IN** (**DATABASE** | '<chemin>'
[**WITH CONSTRUCTOR ARGS (**<liste de colonnes>**)**]) |
(**DETACH** [**AND DELETE**] **JAVA CLASS** <nom de classe>);

<contrainte de colonne> : [**CONSTRAINT** <nom de contrainte>] (([**NOT**] **NULL**) |
UNIQUE | **PRIMARY KEY** |
(**REFERENCES** [<schéma>.]<table> **(**<colonne>**)** [**ON DELETE CASCADE**]) |
CHECK (<condition>**)**)

<contrainte de table> : [**CONSTRAINT** <nom de contrainte>] ((**CHECK (**<condition>**)**) |
((**UNIQUE** | **PRIMARY KEY**) **(**(<colonne> [,])*)) |
(**FOREIGN KEY ((**<colonne>[,])*) **REFERENCES** [<schéma>.]<table>
((<colonne>[,])*) [**ON DELETE CASCADE**])

<clause drop> : **DROP** (**PRIMARY KEY** | **COLUMN** <colonne> |
UNIQUE ((<colonne>[,])*) |
CONSTRAINT <contrainte>) [**CASCADE**]

ALTER USER <utilisateur> **IDENTIFIED BY** <mot de passe>;

ALTER VIEW [<schéma>] <vue> **COMPILE;**

COMMIT [**WORK**];

CREATE DATABASE <base> [**DATABASE_ID** <id>] [**DATABASE_SIZE** <taille max>]
[**EXTENT_SIZE** <nb pages>];

CREATE [**OR REPLACE**] **FUNCTION** [<schéma>.]<fonction>
((<arg> [**IN** | **OUT** | **IN OUT**] <type> [,])*)
RETURN <type> (**IS** | **AS**) [<droits>]
[**LANGUAGE JAVA NAME** '<chaîne>'] ;

CREATE [**UNIQUE**] **INDEX** [<schéma>.]<index> **ON**
[<schéma>.]<table> **(**(<colonne> [**ASC** | **DESC**] [,])*);

CREATE [**OR REPLACE**] [**AND** (**RESOLVE** | **COMPILE**) **NOFORCE**] **JAVA**
((**RESOURCE NAMED** [<schéma>.]<nom>) |
(**CLASS** [**SCHEMA** <schéma>])) [<droits>]
[**RESOLVER (((**<modèle> [,] (<schéma> | -)))*)]
USING BFILE (<chemin>, <fichier>**);**

CREATE [**OR REPLACE**] **PROCEDURE** [<schéma>.]<procédure>
[((<arg> [**IN** | **OUT** | **IN OUT**] <type> [,])*)]
[<droits>] (**IS** | **AS**) **LANGUAGE JAVA NAME** '<chaîne>'] ;

CREATE SCHEMA <schéma> (**CREATE TABLE** <commande>)*;

CREATE SEQUENCE [<schéma>.]<séquence> ((**INCREMENT BY** <entier>) |
(**MAXVALUE** <entier>) | **NOMAXVALUE** |
(**MINVALUE** <entier>) | **NOMINVALUE** |
START WITH <entier>)*;

CREATE [**PUBLIC**] **SYNONYM** [<schéma>] <synonyme> **FOR** [<schéma>] <objet>**;**

CREATE TABLE [<schéma>**.**]<table> **(**<colonne> <type> [**DEFAULT** <expression>]
[(<clause de contrainte>)*****] [<contrainte de table>]**;**

CREATE [**OR REPLACE**] **TRIGGER** [<schéma>] <trigger>
(**BEFORE** | **AFTER**) (**INSERT** | **DELETE** |
(**UPDATE** [**OF** <liste de colonnes>]) [**OR**])*****
ON [<schéma>**.**]<table> **FOR EACH ROW** <procédure> [**(**<args>**)**]**;**

CREATE USER <utilisateur> **IDENTIFIED BY** <mot de passe>**;**

CREATE [**OR REPLACE**] **VIEW** [<schéma>] <vue> [**(**<alias>[**,**]**)**] **AS** <requête>**;**

DELETE FROM [<schéma>] (<table> | <vue>) [**WHERE** <condition>]**;**

DROP FUNCTION [<schéma>**.**]<fonction>**;**

DROP IN**D**EX [<schéma>] <index>**;**

DROP JAVA (**CLASS** | **RESSOURCE**) [<schéma>**.**]<objet>**;**

DROP PROCEDURE [<schéma>**.**]<procédure>**;**

DROP SCHEMA <schéma> [**CASCADE** | **RESTRICT**]**;**

DROP SEQUENCE [<schéma>] <séquence>**;**

DROP [**PUBLIC**] **SYNONYM** [<schéma>] <synonyme>**;**

DROP TABLE [<schéma>**.**]<table> [**CASCADE** | **CASCADE CONSTRAINT** | **RESTRICT**]**;**

DROP TRIGGER [<schéma>] <trigger>**;**

DROP USER <utilisateur> [**CASCADE**]**;**

DROP [<schéma>] **VIEW** <vue> [**CASCADE** | **RESTRICT**]**;**

EXPLAIN PLAN <requête>**;**

GRANT (<rôle> | (<liste de privilèges> **ON** <objet>)) **TO** <liste d'utilisateurs>**;**

INSERT INTO [<schéma>] (<table> | <vue>) [**((**<colonne>[**,**]**)*)**]
((**VALUES** [**((**<expr>[**,**])***)**]) | <requête>)**;**

REVOKE (<rôle> | (<liste de privilèges> **ON** <objet>)) **FROM** <liste d'utilisateurs>**;**

ROLLBACK [**WORK** | **TO** <point de sauvegarde>]**;**

SAVEPOINT <point de sauvegarde>**;**

SELECT [**DISTINCT** | **ALL**] (**.*** | (([<schéma>**.**](<table> | <vue>)**.***) |
(<expr> [[**AS**] <alias>])[**,**])*****
FROM [<schéma>**.**](**(**<requête>**)** | <table> | <vue>) [<alias>]
[**WHERE** <condition>]
[([**START WITH** <condition>] **CONNECT BY** <condition>) |
(**GROUP BY** (<expr>[**,**])***** [**HAVING** <condition>])]
[(**INTERSECT** | **UNION** | **UNION ALL** | **MINUS**) **SELECT** <clause>]
[**ORDER BY** ((<expr> | <position> | <alias>) [**ASC** | **DESC**] [**,**])*****]
[**FOR UPDATE** [**OF** ([<schéma>**.**](<table> | <vue>) <colonne> [**,**])*****]**;**

SELECT [<schéma>**.**]<table**.**procédure> [<args>] **FROM DUAL;**

SELECT [<schéma>**.**<table**.**procédure> [<args>] **FROM** [<schéma>**.**]<table> **WHERE** <condition>**;**

SET TRANSACTION ISOLATION LEVEL
(**READ COMMITTED** | **REPEATABLE READ** | **SERIALIZABLE** | **SINGLE USER**)**;**

UPDATE [<schéma>] (<table> | <vue>) [<alias>] **SET**
(<colonne> **=** (<expr> | <requête>) [**,**])*****
[**WHERE** <condition>]**;**

Bibliographie

[CB05] Thomas Connolly and Carolyn Begg. *Systèmes de bases de données*. Reynald Goulet, 2005.

[Dat90] Christopher Date. *An Introduction to Database Systems*. Addison Wesley, 1990.

[Gar03] Georges Gardarin. *Bases de Données Objet et Relationnel*. Eyrolles, 2003.

[UW08] Jeffrey Ullman and Jennifer Widom. *A First Course in Database Systems*. Prentice-Hall, 2008.