République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université A. Mira de Béjaïa

Faculté des Sciences Exactes
Département d'Informatique

Polycopié de cours

Administration de Bases de Données

Réalisé par:

Dr Mohammed KHAMMARI

Année 2019

Table des matières

Table des figures

Liste des acronymes

- **BD :** Base de données.
- **SGBD :** Système de Gestion de Base de Données.
- **SQL :** Langage d'interrogation structuré.
- **LDD :** Langage de définition des données.
- **LMD :** Langage de manipulation des données.
- **LCD :** Langage de contrôle des données.
- **DBA :** Administrateur de base de données.
- **E/A :** Le modèle Entité-Association.

Avant-propos

Ce polycopié est destiné aux étudiants de la troisième année du premier cycle universitaire (Licence) Informatique, à raison d'un cours et d'un TD par semaine pendant le $6^{\text{ème}}$ semestre, il est organisé comme suit :

- Le chapitre 1 est consacré au rappel sur les objectifs et l'architecture des SGBD (Création, Cohérence, Sécurité, etc.),
- Les rappels sur la modélisation et la conception de base de données sont présentés dans le chapitre 2,
- SQL avancé (LDD, LMD, LCD) est introduit dans le chapitre 3,
- Le chapitre 4 est consacré aux traitements de transactions,
- Le chapitre 5 abord, l'optimisation des requêtes,
- Le chapitre 6 traite l'administration d'une base de données (sauvegarde, restauration, recherches avancée, etc.).

Chapitre 1

Objectifs et architecture des SGBD

1.1 Définition d'une Base de Données

Une Base de données (BD) informatique est un ensemble de données structurées mémorisées sur un support permanent qui modélisent un univers réel, dans laquelle il est possible de stocker une collection de données organisées et structurées de manière à pouvoir facilement consulter et modifier leur contenu. Une BD est faite pour enregistrer des faits, des opérations au sein d'un organisme (administration, banque, université, hôpital, ...).

Mais il ne suffit pas que la base de données existe. Il faut aussi pouvoir la gérer, interagir avec cette base, donc il est nécessaire d'avoir également : un système permettant de gérer cette base ; un langage pour transmettre des instructions à la base de données (par l'intermédiaire d'un système de gestion de base de données).

1.2 Définition d'un SGBD

Un Système de Gestion de Base de Données (SGBD) est un logiciel (ou un ensemble de logiciels) permettant de manipuler les données d'une base de données. Manipuler, c'est-à-dire structurer, stocker, consulter, mettre à jour ou encore partager les informations par plusieurs utilisateurs simultanément en toute sécurité dans une base de données.

Les SGBD les plus répandus sont : Oracle, Microsoft SQL Server, MySQL, Access, etc.

1.2.1 Les Couches d'un SGBD

Un SGBD se compose en première approximation de trois couches emboîtées de fonctions, depuis les mémoires secondaires vers les utilisateurs :

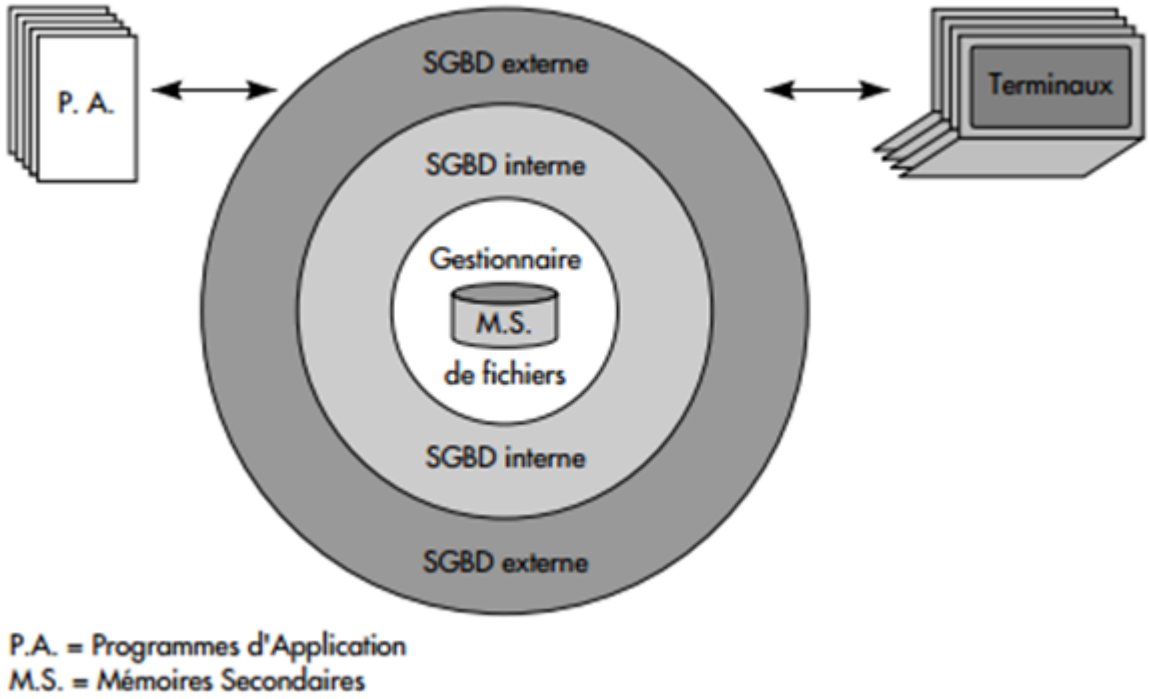

FIGURE 1.1 – Les Couches d'un SGBD

- Gestionnaire de fichiers : Gestion sur mémoire secondaire des données, fournit aux couches supérieures des mémoires secondaires adressables par objets et capables de faire le partage des données, la gestion de la concurrence d'accès, reprise après pannes.

- **SGBD interne :** Définition de la structure de données : Langage de Définition de Données (LDD). Consultation, Insertion, suppression et Mise à Jour des données : Langage de Manipulation de Données (LMD). Gestion de la confidentialité, Maintien de l'intégrité.
- **SGBD externe :** La mise en forme et la présentation des données aux programmes d'applications et aux utilisateurs interactifs.

1.3 Objectifs d'un SGBD

Que doit permettre un SGBD ?

Décrire l'information : Création des objets avec leurs contraintes indépendamment des applications. Modification des structures et des contraintes (Langage de Définition des Données).

Manipuler l'information : Manipulations des données par des utilisateurs sans décrire la manière de les retrouver ou de les mettre à jour, qui est propre à la machine. (Langage de Manipulation des Données).

Contrôler l'information : .

@ Intégrité : Respecter l'intégrité de l'information et vérifier les contraintes d'intégrité. Par exemple : le salaire doit être compris entre 20000 DA et 40000 DA.

@ Confidentialité : Autoriser la confidentialité des informations. Tout le monde ne peut pas voir et faire n'importe quoi : contrôle des droits d'accès, autorisation,etc (Langage de Contrôle des Données).

Partager l'information : une BD est partagée entre plusieurs utilisateurs en même temps (contrôle des accès concurrents). Notion de transaction : L'exécution d'une transaction doit préserver la cohérence de la BD. Notion de rôles et de privilèges : Droits et devoirs des utilisateurs.

Assurer la sécurité de l'information : reprise après panne, journalisation,etc.

Performances d'accès : index (hashage, arbres balancés,etc).

Indépendance physique : Pouvoir modifier les structures de stockage ou les index sans que cela ait de répercussion au niveau des applications. Les disques,

les méthodes daccès, les modes de placement, le codage des données ne sont pas apparents

Indépendance logique : Permettre aux différentes applications d'avoir des vues différentes des mêmes données. Permettre au DBA (Administrateur de base de données) de modifier le schéma logique sans que cela ait de répercussion au niveau des applications

1.4 Fonctions d'un SGBD

Un SGBD permet de décrire les données des bases, de les interroger, de les mettre à jour, de transformer des représentations de données, d'assurer les contrôles d'intégrité, de concurrence et de sécurité. Il supporte de plus en plus des fonctions avancées pour la gestion de procédures et d'événements.

- **DEFINITION DES DONNEES :** Langage de définition des données (LDD) (conforme à un modèle de données).
- **MANIPULATION DES DONNEES :** Interrogation, Mise à jour (insertion, suppression, modification). Langage de manipulation des données (LMD).
- **CONTRÔLE DES DONNEES :** Contraintes d'intégrité, Contrôle des droits d'accès, Gestion de transactions. Langage de contrôle des données (LCD).

1.5 Architecture typique d'un SGBD

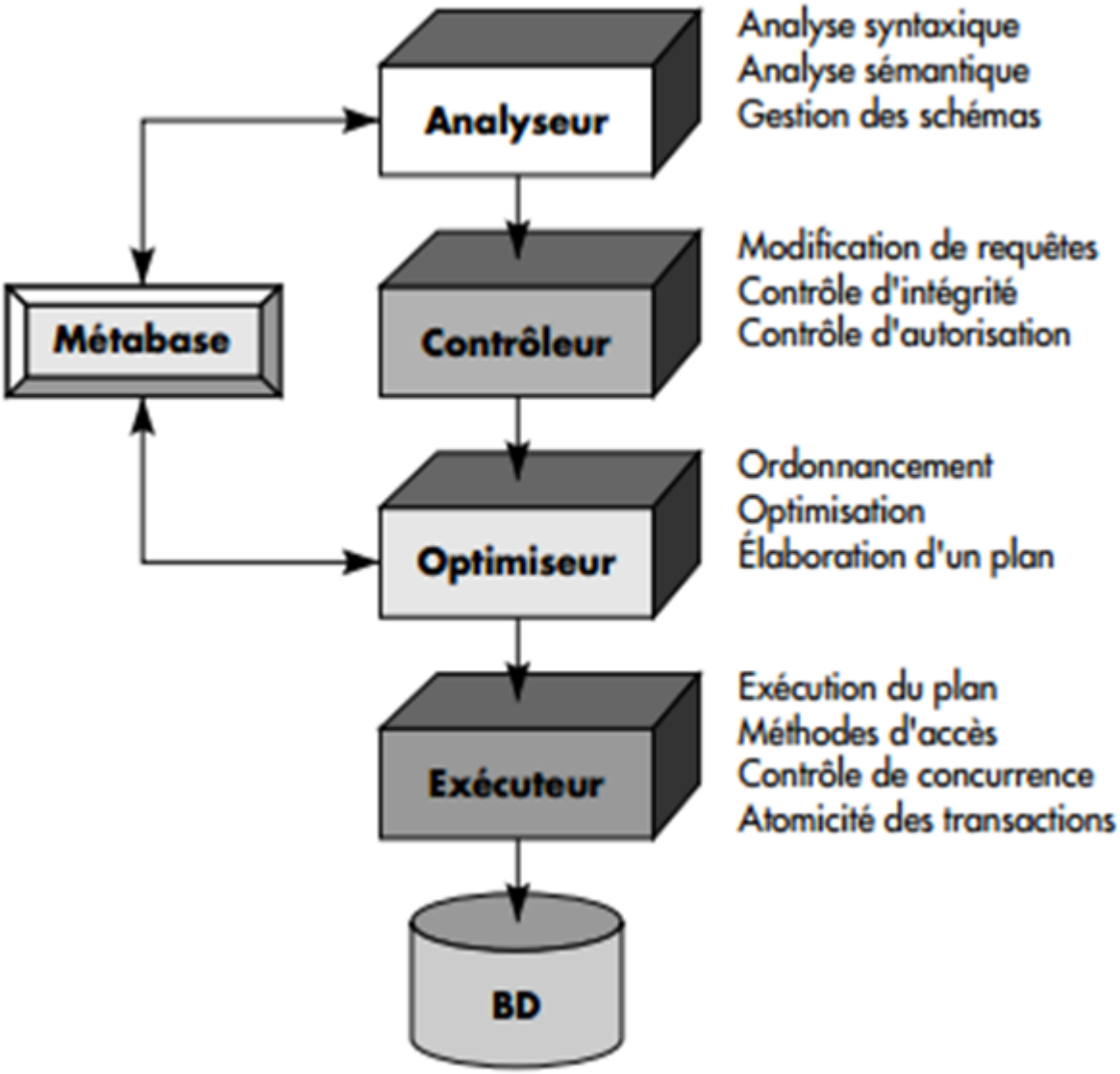

FIGURE 1.2 – Architecture typique d'un SGBD

a) Analyseur de requêtes : L'analyse syntaxique (conformité à la grammaire) et sémantique (conformité à la vue référencée ou au schéma) de la requête.

b) Contrôleur de requêtes : Consiste à changer la requête en remplaçant les références aux objets de la vue par leur définition en termes d'objets du schéma.

c) Optimiseur de requêtes : Elaborer un plan d'accès optimisé pour traiter la requête.

d) Exécuteur de plans : Exécuter le plan d'accès choisi et élaboré par l'optimiseur.

1.5.1 Architecture d'un SGBD CLIENT-SERVEUR

Le mode de fonctionnement

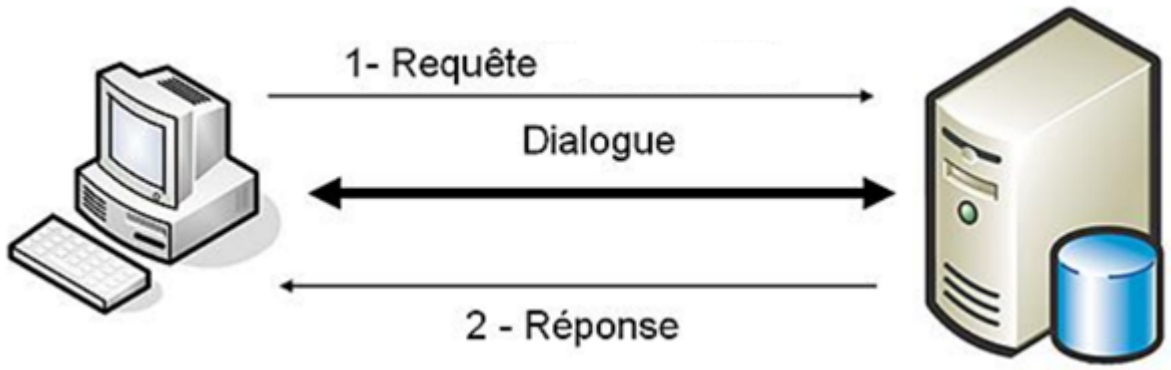

FIGURE 1.3 – Le mode de fonctionnement d'un SGBD CLIENT-SERVEUR

D'un point de vue opérationnel, un SGBD est un ensemble de processus et de tâches qui supportent l'exécution du code du SGBD pour satisfaire les commandes des utilisateurs.

L'architecture client-serveur inclut le noyau d'un SGBD, appelé DMCS (Description Manipulation and Control Sub-system), qui fonctionne en mode serveur. Autour de ce serveur s'articulent des processus attachés aux utilisateurs supportant les outils et les interfaces externes.

Chapitre 2

Rappels et compléments sur la modélisation et la conception de BDD

2.1 Notion de modélisation des données

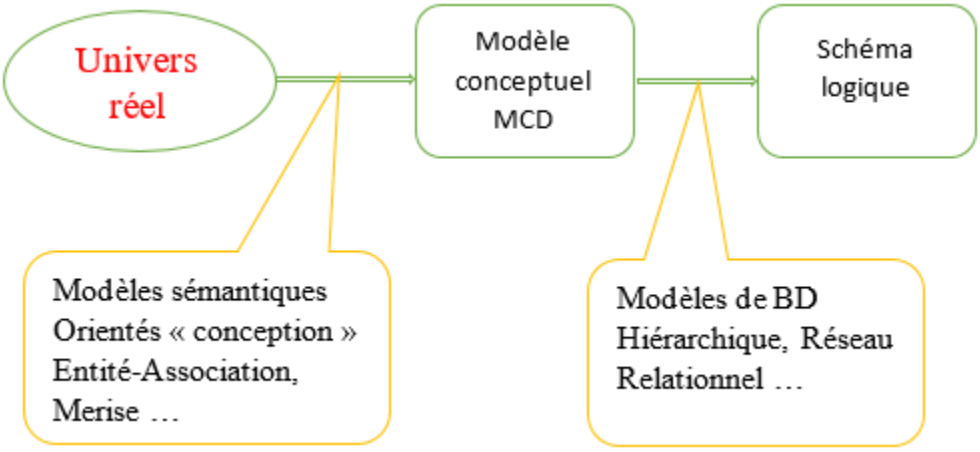

FIGURE 2.1 – Modélisation des données

2.1.1 Le modèle Entité-Association (E/A)

Le modèle E/A est un formalisme graphique pour la modélisation de données, les origines du modèle sont les travaux de Chen et Tardieu en 1975. Le succès du modèle E/A est dû aux spécificités suivantes : langage graphique, concepts simples : Choses (objets) -> entités, liens entre les choses (objets) -> association, regroupement des choses de

même nature : classes d'entités, classes d'association.

a) Entités : Une entité est un objet, un événement, un lieu, une personne, une chose, identifiable sans ambiguïté. Les entités peuvent être regroupées en types d'entités

Exemple :

le cinéma : «dar el baida» est une instance ou occurrence de l'entité cinéma, l'acteur : «Atemane Ariouate» est une instance ou occurrence de l'entité acteur, le film « dawria nahwa charke» est une instance ou occurrence de l'entité film.

b) Associations : Une association c'est un lien entre deux ou plusieurs entités

Exemple : Atemane Ariouate a joué dans "dawria nahwa charke"

c) Propriétés ou Attributs : Une donnée élémentaire que lon perçoit sur une entité ou une association

Exemple : Pour les entités :

— Nom, Prénom pour l'entité ACTEURS

— Titre, Metteur en scène pour l'entité FILMS

— Nom, Adresse pour l'entité CINEMA

Formalisme graphique pour les entités

FIGURE 2.2 – Formalisme graphique pour les entités

Pour les associations : Les propriétés que l'on met dans une association doivent obligatoirement relier les entités

Exemple :

Le Rôle dun acteur. Le rôle relie, un acteur et le film dans lequel il a joué
Formalisme graphique pour les associations

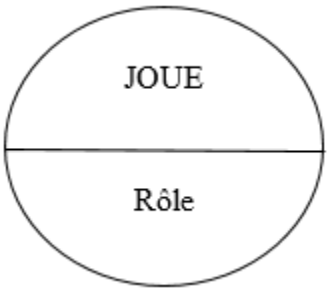

FIGURE 2.3 – Formalisme graphique pour les associations

d) L'identifiant : Propriété ou groupe de propriétés qui sert à identifier une entité. L'identifiant d'une entité est choisi par l'analyste de façon à ce que deux occurrences de cette entité ne puissent pas avoir le même identifiant

e) Les cardinalités : Une association permet de relier, une ou plusieurs entités. Le rôle détermine la façon dont les entités sont reliées.
Le rôle d'une association est défini par deux nombres (min, max) représentant le nombre de fois minimum et le nombre de fois maximum qu'une entité participe à une association. Les valeurs possibles sont : (0,1), (1,1), (0,N), (1,N)

- Min : Correspond à la réponse à la question : combien de fois au moins une entité de A est reliée à une entité de B
- Max : correspond à la réponse à la question : combien de fois au plus une entité de A est reliée à une entité de B

Remarque :
il faut les poser dans les deux sens de A vers B puis de B vers A.

FIGURE 2.4 – Les cardinalités

Exemple :

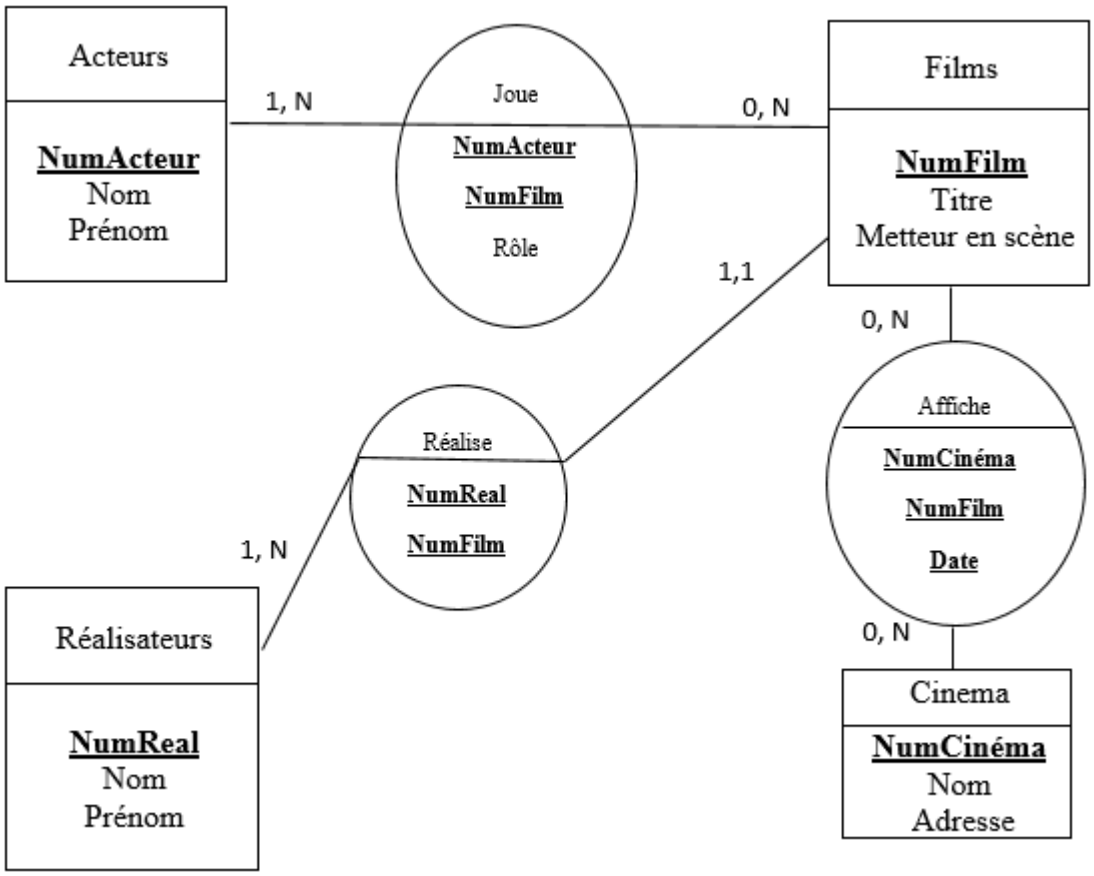

FIGURE 2.5 – Un exemple du modèle Entité-Association

Acteurs vers Films : le rôle de type 1,N

— (1) un acteur a joué dans au moins un film
— (N) un acteur peut avoir joué dans plusieurs films

De Films vers Acteurs 0,N :

— (0) : un film n'ayant pas d'acteurs, possible si cest un film documentaire
— (N) : un film peut avoir plusieurs acteurs

2.2 Notion de conception de bases de données

2.2.1 Le modèle relationnel

L'organisation des données au sein d'une BD a une importance essentielle pour faciliter l'accès et la mise à jour des données. Le modèle relationnel est fondé sur la notion mathématique de RELATION

— Introduit par Codd En 1970 (recherche IBM)
— Données organisées en tables (adressage relatif)
— Stratégie d'accès déterminée par le SGBD

Les règles de passage d'un schéma Entité/Association à un Schéma Relationnel :

(1) : Toute classe d'entités du diagramme entité/association est représentée par une relation dans le schéma relationnel équivalent. La clé de cette relation est l'identifiant de la classe d'entités correspondante.

(2) : Toute classe d'association est transformée en relation. La clé de cette relation est composée de tous les identifiants des entités participantes.

(3) : Toute classe d'associations reliée à une classe d'entités avec une cardinalité de type 0,1 ou 1,1 peut être fusionnée avec la classe d'entités. Dans ce cas, on déplace les attributs de la classe d'associations vers ceux de la relation traduisant la classe d'entités.

Exemple :
Acteurs (NumActeur, Nom, Prénom)
Films (NumFilm, Titre, MetteurEnScène, NumReal)

Cinema (NumCinéma, Nom, Adresse)
Realisateurs (NumReal,Nom, Prénom)
Joue (NumACteur, NumFilm , Rôle)
Affiche (NumFilm, NumCinema ,Date)

Les avantages du modèle relationnel

- Simplicité de présentation : représentation sous forme de tables,
- Opérations relationnelles : algèbre relationnelle,
- Indépendance physique : optimisation des accès, stratégie d'accès déterminée par le système,
- Indépendance logique : concept de VUES,
- Maintien de l'intégrité : contraintes d'intégrité définies au niveau du schéma.

Chapitre 3

SQL Avancé (LDD,LMD et LCD)

3.1 Présentation de SQL

SQL signifie « Structured Query Language » c'est-à-dire « Langage d'interrogation structuré ». En fait SQL est un langage complet de gestion de bases de données relationnelles. Il est introduit par IBM dans les années 70, il est considéré comme l'évolution du langage SEQUEL, il est commercialisé tout d'abord par ORACLE. SQL est devenu le langage standard des systèmes de gestion de bases de données (SGBD) relationnelles (SGBDR).
C'est à la fois :

- Un langage d'interrogation de la base *(SELECT).*
- Un langage de manipulation des données *(UPDATE, INSERT, DELETE).*
- Un langage de définition des données *(CREATE, ALTER, DROP).*
- Un langage de contrôle de l'accès aux données *(GRANT, REVOKE).*
- Un langage de gestion de transactions *(COMMIT, ROLLBACK).*

SQL a été normalisé dès 1986 mais les premières normes, trop incomplètes, ont été ignorées par les éditeurs de SGBD : (ANSI : American National Standard Institute)

- SQL 1 initial : ANSI date de 1986.
- SQL 1 intégrité référentielle : ANSI date de 1989.
- SQL 2 ANSI date de 1992, extension de SQL1.
- SQL 3 ANSI date de 1999. (appelée aussi SQL99) est la nouvelle norme SQL avec l'intégration du modèle objet.

3.2 Définition de données

SQL est un Langage de Définition des Données (LDD), il permet de créer des tables dans une base de données relationnelle, ainsi que d'en modifier ou en supprimer.
Ordres pour la création et la suppression de la base de donnes :

Créer une Bases de données
CREATE DATABASE nom_bdd ;

Supprimer une Bases de données
DROP DATABASE nom_bdd ;

3.2.1 Création de tables

La commande CREATE TABLE crée la définition d'une table

Syntaxe :

CREATE TABLE nom_table (- - définition des colonnes
Col1 TYPE (taille) [NOT NULL [UNIQUE]]
[DEFAULT valeur]
[PRIMARY KEY]
[REFERENCES table]
[CHECK condition],
...,
- - contraintes de table
[PRIMARY KEY (liste de colonnes)],
[UNIQUE (liste de colonnes)],
... ,
[FOREIGN KEY (liste de colonnes) REFERENCES table
[ON DELETE {RESTRICT | CASCADE | SET NULL}]
[ON UPDATE {RESTRICT | CASCADE | SET NULL}],

... ,
[CHECK condition],
...
) ;

Principaux types de données

- le type CHAR pour les colonnes qui contiennent des chaînes de longueur constante (CHAR(n)).
- le type VARCHAR pour les colonnes qui contiennent des chaînes de longueurs variables (VARCHAR(n)).
- le type SMALLINT Nombres entiers sur 2 octets.
- le type INTEGER Nombres entiers sur 4 octets.
- le type DECIMAL(n,m) correspond à des nombres décimaux qui ont n chiffres significatifs et m chiffres après la virgule.
- le type DATE réserve 2 chiffres pour le mois et le jour et 4 pour l'année.
- le type TIME pour les heures, minutes et secondes.
- ...

Contraintes d'intégrité

- *NOT NULL* : valeur null impossible.
- *UNIQUE* : interdit qu'une colonne contienne deux valeurs identiques.
- *PRIMARY KEY* : définit la clé primaire de la table.
- *FOREIGN KEY* : indique que la colonne que l'on définit est une clé étrangère qui fait référence à la colonne de la table tableref (contrainte d'intégrité référentielle).
- *CHECK* : donne une condition que les colonnes de chaque ligne devront vérifier (plage ou liste de valeurs).
- *CASCADE* : cascader les suppressions ou Modifications.
- *SET NULL* : rendre nul les attributs référençant.
- *RESTRICT* : rejet de la mise à jour c'est l'option par défaut.

Les contraintes peuvent s'exprimer :

- Soit au niveau colonne (contraintes locales) : valables pour une colonne.

— Soit au niveau table (contraintes globales) : valables pour un ensemble de colonnes d'une table.

Les contraintes se définissent :

— Soit lors de la création des tables, dans l'ordre CREATE TABLE ;

— Soit après la création des tables, par l'ordre ALTER TABLE permettant certaines modifications de la structure des tables.

EXEMPLE :

```
CREATE TABLE Clients (
idClient CHAR(6) PRIMARY KEY ,
nom VARCHAR(30) NOT NULL,
adresse VARCHAR(30) ,
numéroTelephone INTEGER
) ;

CREATE TABLE Produit (
idProduit CHAR(6) PRIMARY KEY ,
nom VARCHAR(30) NOT NULL,
marque VARCHAR(30) NOT NULL ,
prix DECIMAL(6,2) ,
- - contrainte de table
CHECH (marque IN ( BMW, TOYOTA,PEUGEOT) )
) ;

CREATE TABLE Vente (
idVente CHAR(6) PRIMARY KEY ,
referenceProduit CHAR(6) ,
idClient CHAR(6) ,
date DATE NOT NULL ,
- - contrainte de table
FOREIGN KEY (referenceProduit) REFERENCES Produit(idProduit) ON DELETE
CASCADE,
FOREIGN KEY (idClient) REFERENCES Clients (idClient) ON DELETE CASCADE,
```

) ;

3.2.2 Modification du schéma

Il est possible de supprimer ou de modifier la structure d'une table à l'aide des commandes :
DROP TABLE et ALTER TABLE

Renommer une table
ALTER TABLE nom_table RENAME TO nouveau_nom ;

Ajout ou modification de colonne
ALTER TABLE nom_table {ADD/MODIFY} ([nom_colonne type [contrainte], ...]) ;

Renommer une colonne
ALTER TABLE nom_table RENAME COLUMN ancien_nom TO nouveau_nom ;

Supprimer une colonne
ALTER TABLE nom_table DROP COLUMN nom_colonne ;

Ajout d'une contrainte de table
ALTER TABLE nom_table ADD [CONSTRAINT nom_contrainte]contrainte ;

Supprimer des contraintes
ALTER TABLE nom_table DROP CONSTRAINT nomContrainte ;

Suppression de données uniquement
TRUNCATE TABLE nom_table ;

3.2.3 Création d'index

Afin d'améliorer les temps de réponses, SQL propose un mécanisme d'index. Un index est associé à une table, mais il est stocké à part. Un index peut ne porter que sur une colonne (index simple) ou sur plusieurs (index multiple). Chaque valeur d'index peut ne désigner qu'une et une seule ligne, on parlera alors d'index unique donc de clef primaire, dans le cas contraire on parlera d'index dupliqué.

Remarque :
Un index est automatiquement créé lorsqu'une table est créée avec la contrainte PRIMARY KEY.

Création
CREATE INDEX index_nom ON table ;

- *index sur une seule colonne :*
  CREATE INDEX index_nom ON table (colonne1) ;
- *index sur plusieurs colonnes :*
  CREATE INDEX index_nom ON table (colonne1, colonne2) ;

Suppression
DROP INDEX nom_index ;

3.3 Manipulation de données

SELECT, INSERT, UPDATE et DELETE sont les 4 commandes de manipulation des données en SQL.

3.3.1 Modifier une base de données

Les commandes INSERT, UPDATE et DELETE permet la modification dune base de données.

— *La commande INSERT :*

La commande INSERT permet d'ajouter de nouvelles lignes à une table

Syntaxe

INSERT INTO NomTable (colonne1,colonne2,colonne3,...)

VALUES (Valeur1,Valeur2,Valeur3,...) ;

Insertion par une sélection

INSERT INTO NomTable (colonne1,colonne2,...)

SELECT colonne1,colonne2,... FROM NomTable2

WHERE condition

Exemple :

INSERT INTO Clients (idClient, nom, adresse, numéroTelephone) VALUES (c214, hamiche, cité Hamadi N114, 0654874125) ;

Remarque :

Les valeurs inconnues prennent la valeur NULL

— *La commande UPDATE :*

La commande UPDATE permet de changer des valeurs d'attributs de lignes existantes.

Syntaxe

UPDATE NomTable SET Colonne = Valeur Ou Expression

[WHERE condition]

Exemple :

UPDATE Clients SET numéroTelephone = 05874521 WHERE idClient = c214 ;

Remarque :

L'absence de clause WHERE signifie que les changements doivent être appliqués à toutes les lignes de la table cible.

— *La commande DELETE :*

La commande DELETE permet d'enlever des lignes dans une table.

Syntaxe

DELETE FROM NomTable

[WHERE condition]

Exemple :

DELETE FROM Clients WHERE idClient = 'c214' ;

Remarque :

L'absence de clause WHERE signifie que toutes les lignes de la table cible sont enlevées

3.3.2 Interroger une base de données

La commande SELECT permet de rechercher des données à partir de plusieurs tables ; le résultat est présenté sous forme d'une table réponse.

Syntaxe de base
SELECT [ALL|DISTINCT] NomColonne1,... | *
FROM NomTable1,...
WHERE Condition

- ALL : toutes les lignes.
- DISTINCT : pas de doublons.
- * : toutes les colonnes.
- La clause AS : SELECT Compteur AS Ctp FROM Vehicule

Expression des restrictions
Les conditions peuvent faire appel aux opérateurs suivants :

- Opérateurs logiques : AND, OR, NOT, XOR
- Comparateurs de chaînes : IN, BETWEEN, LIKE
- Opérateurs arithmétiques : +, -, /, %
- Comparateurs arithmétiques : =, <, >, >=, <=, <>
- Fonctions numérique : abs, log, cos, sin, mod, power,...
- Fonctions sur chaine : length, concat,...

Exemple
SELECT * FROM produit WHERE (prix>1000) AND (prix <=3000)
SELECT * FROM produit WHERE prix BETWEEN 1000 AND 3000
SELECT * FROM produit WHERE Marque IN ('TOYOTA','BMW')

Différentes clauses de SELECT

Syntaxe
SELECT *
FROM table
WHERE condition
GROUP BY expression
HAVING condition
{ UNION | INTERSECT | EXCEPT }
ORDER BY expression
LIMIT count

— *GROUP BY*
GROUP BY est utilisé pour grouper plusieurs résultats sur un groupe de résultat.
Exemple
SELECT COUNT (*)
FROM produit
GROUP BY marque

— *HAVING*
La clause HAVING permet de spécifier une condition de restriction des groupes. Elle sert à éliminer certains groupes, comme WHERE sert à éliminer des lignes.
Exemple
SELECT COUNT (*)
FROM produit
GROUP BY marque
HAVING COUNT (*) $>$ 40

— *ORDER BY*
ORDER BY permet de trier les lignes dans un résultat d'une requête SQL. Il est possible de trier les données sur une ou plusieurs colonnes, par ordre ascendant ou descendant (ASC|DESC).
Exemple

SELECT *
FROM produit
ORDER BY marque DESC

— *UNION*

UNION permet de concaténer les résultats de deux requêtes ou plus. Pour l'utiliser il est nécessaire que chacune des requêtes à concaténer retournes le même nombre de colonnes, avec les mêmes types de données et dans le même ordre.

Exemple

SELECT marque FROM produit
UNION
SELECT marque FROM Vente

— *INTERSECT*

INTERSECT permet d'obtenir l'intersection des résultats de deux requêtes (récupérer les enregistrements communs à 2 requêtes). Cela permet de trouver s'il y a des données similaires sur 2 tables distinctes.

Exemple

SELECT marque FROM produit
INTERSECT
SELECT marque FROM Vente

— *Expression des jointures*

Le produit cartésien s'exprime simplement en incluant plusieurs tables après la clause FROM. La condition de jointure est exprimée après WHERE

Exemple

SELECT P.IdProduit, P.nom
FROM produit P , vente V
WHERE P.IdProduit = V.referenceProduit

— *Requête imbriquée*

SQL permet l'imbrication de sous-requêtes au niveau de la clause WHERE. Les sous-requêtes sont utilisées :

- dans des prédicats de comparaison : (=, <>, <, <=, >, >=)
- dans des prédicats : IN, EXISTS, ALL ou ANY.

Exemple

```
SELECT C.nom
FROM client C
WHERE idClient IN (
SELECT V.IdClient
FROM vente V
WHERE referenceProduit IN (
SELECT idProduit
FROM produit P
WHERE P.nom = 'p1' ))
```

— *Le prédicat EXISTS*

Il permet de tester si le résultat d'une sous-requête est vide ou non.

Exemple

```
SELECT P.nom
FROM produit P
WHERE NOT EXISTS ( SELECT *
FROM vente V
WHERE V.referenceProduit = P.IdProduit )
```

— *Le prédicat ALL ou ANY*

Ils permettent de tester si un prédicat de comparaison est vrai pour tous (ALL) ou au moins un (ANY) des résultats d'une sous-requête.

Exemple

```
SELECT C.nom
FROM client C, vente V, produit P
WHERE C.idClient = V. idClient AND P.idProduit = V.referenceProduit AND
P.prix >= ALL
( SELECT PR.prix
FROM client CL, vente VT, produit PR
WHERE CL.idClient = VT. idClient AND PP.idProduit = VT.referenceProduit
AND PR.nom = 'c1' )
```

3.4 Contrôle de Données

3.4.1 Contrôle des accès concurrents

La notion de transaction :
Une transaction est une unité logique de traitement qui est soit complètement exécutée, soit complètement abandonnée. Une transaction fait passer la BD d'un état cohérent à un autre état cohérent. Une transaction est terminée : soit par COMMIT, soit par ROLLBACK.

La commande COMMIT
La commande COMMIT termine une transaction avec succès ; toutes les mises à jour de la transaction sont validées. On dit que la transaction est validée.

La commande ROLLBACK
La commande ROLLBACK termine une transaction avec échec ; toutes les mises à jour de la transaction sont annulées (tout se passe comme si la transaction n'avait jamais existé). On dit que la transaction est annulée.

3.4.2 Contrôle des droits d'accès

La commande GRANT
La commande GRANT permet de passer des droits d'accès à un utilisateur ou un groupe d'utilisateurs

Syntaxe
GRANT privilèges ON table TO bénéficiaire

[WITH GRANT OPTION]
Les privilèges qui peuvent être passés sont :

- soit ALL (tous les privilèges)
- soit une liste de privilèges parmi :
  - SELECT
  - INSERT
  - UPDATE [(liste de colonnes)] : l'omission de la liste de colonnes signifie toutes les colonnes
  - DELETE

  Le bénéficiaire peut être :

- soit PUBLIC (tous les utilisateurs)
- soit un utilisateur ou un groupe d'utilisateurs

L'option WITH GRANT OPTION permet de passer un privilège avec le droit de le transmettre.

La commande REVOKE
La commande REVOKE permet de retirer des droits à un utilisateur ou groupe d'utilisateurs.

Syntaxe
REVOKE privilèges ON table FROM bénéficiaire

3.4.3 Notion de sous-schéma

L'objet VUE
Une vue est une table virtuelle (aucune implémentation physique de ses données) calculée à partir des tables de base par une requête.

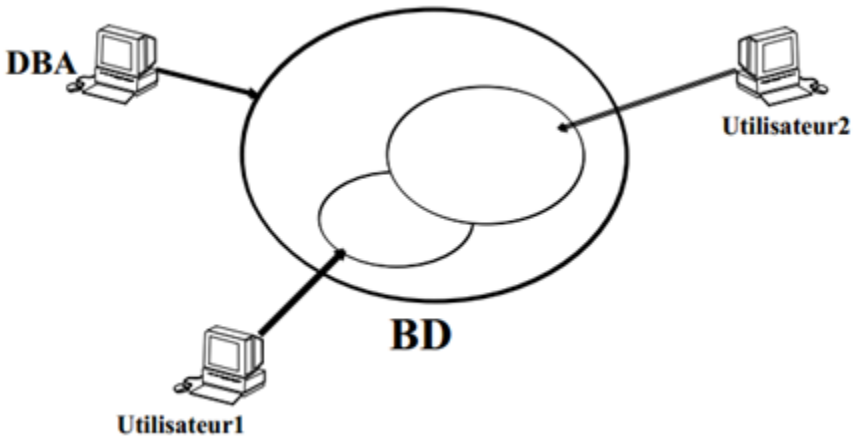

FIGURE 3.1 – Notion de Sous-Schéma

Une vue apparaît à l'utilisateur comme une table réelle, cependant les lignes d'une vue ne sont pas stockées dans la BD (La définition de la vue est enregistrée dans le DD). Les vues assurent l'indépendance logique, elles peuvent être utilisées pour cacher des données sensibles, ou pour montrer des données statistiques.

Création et suppression d'une VUE :

Création :

```
CREATE VIEW NomVue(NomColonne1,...) AS
SELECT NomColonne1,..
FROM NomTable
WHERE Condition
```

Suppression :

```
DROP VIEW nom_vue;
```

RENOME :

```
RENAME VIEW nom_vue TO nouveau_nom;
```

Intérêt des vues

— *Indépendance logique*
Le concept de vue permet d'assurer une indépendance des applications vis-à-vis des modifications du schéma (Assurer l'indépendance du schéma externe).

— *Simplification d'accès*
Les vues simplifient l'accès aux données en permettant par exemple une prédéfinition des jointures et en masquant ainsi à l'utilisateur l'existence de plusieurs tables (Création de résultats intermédiaires pour des requêtes complexes).
Exemple : La vue qui calcule les moyennes générales pourra être consultée par la requête :
SELECT * FROM Moyennes

— *Confidentialité des données*
Une vue permet d'éliminer des lignes sensibles et/ou des colonnes sensibles dans une table de base (éviter de divulguer certaines informations).

3.4.4 Rôles

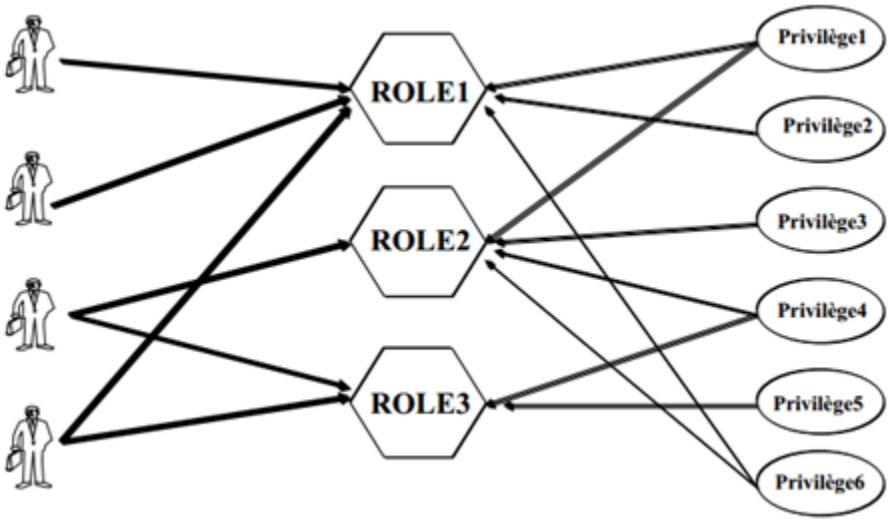

FIGURE 3.2 – Les Rôles

— Regroupement de privilèges pour des familles d'utilisateur

— Facilitent la gestion des autorisations des privilèges objet en évitant les ordres GRANT
— Un rôle par défaut est donné à un utilisateur
— Un utilisateur peut posséder plusieurs rôles mais il n'est connecté qu'avec un seul à la fois
— On peut donner un mot de passe pour certains rôles

Manipulation des rôles : Ordres

Création / Modification d'un rôle

{CREATE|ALTER} ROLE nom_role {NOT IDENTIFIED | IDENTIFIED {BY mot_de_passe|} EXTERNALLY} ;

Remplissage et attribution d'un rôle

GRANT {privilege1 | role1} TO nom_role ;
GRANT {privilege2 | role2} TO nom_role ;
GRANT ROLE nom_role TO user ;

Rôle par défaut ou activation

SET ROLE nom_role [IDENTIFIED BY mot_de_passe] ;

Suppression / Révocation d'un rôle

DROP ROLE nom_role ; REVOKE ROLE nom_role FROM user ;

Exemple

CREATE ROLE employeur ;
GRANT SELECT, INSERT,UPDATE(adresse, numéroTelephone) ON clients TO employeur ;

GRANT SELECT, INSERT ON produit TO employeur ;
GRANT ALL ON vente TO employeur ;
GRANT ROLE employeur TO Said, Salah, Karim ;

3.4.5 Contraintes évènementielles : Trigger

Tout comme les procédures stockées, les triggers servent à exécuter une ou plusieurs instructions. Mais à la différence des procédures, il n'est pas possible d'appeler un trigger : un trigger doit être déclenché par un événement. Un trigger est attaché à une table, et peut être déclenché par :

- Une insertion dans la table (INSERT).
- La suppression d'une partie des données de la table (DELETE) .
- La modification d'une partie des données de la table (UPDATE).

Par ailleurs, une fois le trigger déclenché, ses instructions peuvent être exécutées soit juste avant (BEFORE) l'exécution de l'événement déclencheur, soit juste après (AFTER).

Syntaxe
CREATE TRIGGER nom_trigger
{BEFORE | AFTER | INSTEAD OF}
{INSERT | DELETE | UPDATE [OF col1, col2,...]}
ON {nom_table | nom_vue}
[REFERENCING {NEW|OLD} AS <nom>]
[FOR EACH ROW]
[WHEN condition]
BLOC PL/SQL

- *BEFORE :* Exécution avant modification des données.
- *AFTER :* Exécution après modification des données.
- *INSTEAD OF :* Exécution à la place de l'ordre SQL envoyé.
- *INSERT, UPDATE, DELETE :* Action concernée par le déclencheur.

— *OLD :* représente les valeurs des colonnes de la ligne traitée avant qu'elle ne soit modifiée par l'événement déclencheur. Ces valeurs peuvent être lues, mais pas modifiées.
— *NEW :* représente les valeurs des colonnes de la ligne traitée après qu'elle a été modifiée par l'événement déclencheur. Ces valeurs peuvent être lues et modifiées.

Suppression des triggers

```
DROP TRIGGER nom_trigger;
```

Exemple 1 : Supprimer les produits correspondant au fournisseur supprimé (On suppose que chaque produit peut être livré par un seul fournisseur).

```
DELIMITER |
CREATE TRIGGER DeleteFournisseur
BEFORE DELETE ON Fournisseur
FOR EACH ROW
BEGIN
DELETE FROM Produit P WHERE P.idFournisseur = OLD.idFournisseur;
END |
DELIMITER;
```

Exemple 2 :

Contrôler l'existence d'un fournisseur lors de l'ajout d'un produit. Si pas de fournisseur, annuler la transaction.

```
DELIMITER |
CREATE TRIGGER InsertProduit
BEFORE INSERT ON Produit
FOR EACH ROW
BEGIN
WHEN ( NOT EXIST (
SELECT * FROM Fournisseur F WHERE F.IdFournisseur = NEW.IdFournisseur )
ABORT TRANSACTION;
END |
```

```
DELIMITER ;
```

Chapitre 4

Traitements de transactions

4.1 La notion de transaction

Une transaction est une unité atomique de traitement (séquence d'opérations qui doit être exécutée dans son intégralité) qui est : soit complètement exécutée soit complètement abandonnée. Une transaction fait passer la base de données d'un état cohérent à un autre état cohérent. Si une transaction ne va pas à son terme pour une raison ou pour une autre (erreur de syntaxe, violation de contrainte, bug SGBD, arrêt machine, etc), la base est restaurée dans l'état où elle se trouvait avant que la transaction ne démarre.

4.1.1 État cohérent

Définition : Une base de données est cohérente (ou dans un état cohérent) lorsque toutes les données qu'elle contient sont en accord avec les contraintes d'intégrité du schéma conceptuel.

4.1.2 État correct

Définition : Une base de données est dans un état correct lorsqu'elle est dans un état cohérent et que les valeurs des données reflètent exactement le résultat attendu des modifications effectuées.

Exemple :
CREATE TABLE Compte (numero INTEGER PRIMARY KEY, valeur INTEGER) ;

Requêtes d'insertion :
INSERT INTO Compte VALUES (1 , 500) ; INSERT INTO Compte VALUES (2 , 100) ;

Requêtes : Transférer du compte 1 vers le compte 2 la somme 300
R1 : UPDATE Compte SET valeur = valeur - 300 WHERE numero = 1 ;
Q1 : Est-ce que la base de données est dans un état cohérent ?
Q2 : Est-ce que la base de données est dans un état correct ?
R1 : UPDATE Compte SET valeur = valeur + 300 WHERE numero = 2 ;
Entre les deux mises à jour, la base de données est cohérente mais non correcte.

4.2 Les propriétés des transactions

Un système de gestion de transactions doit garantir les propriétés suivantes (ACID) :

a) Atomicité : Une transaction doit effectuer toutes ses mises à jour avec succès ; sinon ne rien faire du tout.

b) Cohérence : une transaction doit laisser la base dans un état cohérent.

c) Isolation : : les modifications effectuées par une transaction ne doivent être visibles aux autres transactions qu'une fois la transaction validée.

d) Durabilité : Dès qu'une transaction valide ses modifications, le système doit garantir que ces modifications seront conservées, même en cas de panne.

4.3 Les ordres des transactions

a) BEGIN TRAN : début de la transaction, en l'absence de cette commande, toute instruction SQL est une transaction implicite.

b) SAVE TRAN : Cette instruction permet de définir des points d'arrêt, et donc donne la possibilité d'annuler une partie de la transaction en cours.

c) COMMIT TRAN : Cette instruction permet de mettre fin avec succès à une transaction.

d) ROLLBACK TRAN : permet d'annuler une partie ou la totalité de la transaction (fin avec échec).

Exemple :
BEGIN TRAN Tr1
- - Ajouter un client
INSERT INTO clients(nom, prenom, telephone) VALUES ('Ali', 'Bal', '021364897') ;
- - Poser un point d'arrêt
SAVE TRAN P1 ;
- - Compter les clients
SELECT COUNT(*) AS NbreClients FROM clients ; NbreClients = 201
DELETE FROM clients ;
SELECT COUNT(*) AS NbreClients FROM clients ; NbreClients = 0
- - Annuler la suppression
ROLLBACK TRAN P1 ;
SELECT COUNT(*) AS NbreClients FROM clients ; NbreClients = 201
- - Valider le reste de la transaction
COMMIT TRAN Tr1 ;
- - Compter les clients
SELECT COUNT(*) AS NbreClients FROM clients ; NbreClients = 201

4.4 Les problèmes de concurrence d'accès

Plusieurs utilisateurs peuvent lancer des transactions en même temps ⇒ Concurrence d'accès. Des transactions exécutées concurremment peuvent interférer et mettre la base de données dans un état incohérent.

Exemple - Problème des accès concurrents

- Considérons deux transactions T1 et T2 qui s'intéressêt à un même objet A
- Les deux seules opérations possibles sur A, sont : lire et écrire

Quatre possibilités :

1. Lecture Lecture
2. Ecriture - Ecriture

3. Ecriture Lecture
4. Lecture Ecriture

1. Lecture-Lecture (Partage)
   — Aucun conflit
   — un même objet peut toujours être partagé en lecture
2. Ecriture Ecriture (Perte de données)
   — T2 peut écraser la valeur de A, par une autre écriture, celle effectuée par T1 (perte de données).

temps	Transaction T1	Transaction T2	Etat de la base
t1	Lire A		A = 20
t2		Lire A	
t3	A := A +50		
t4		A := A +10	
t5	Ecrire A		A = 70
t6		Ecrire A	A = 30

   Résultat : A = 30, Résultat correct : A = 80
3. Ecriture Lecture (Lecture impropre)
   — Une transaction lit des données écrites par une transaction concurrente non validée.
   — T2 lit une valeur modifiée par T1 et ensuite T1 est annulée.

temps	Transaction T1	Transaction T2	Etat de la base
t1	Lire A		A = 20
t2	A := A +50		
t3	Ecrire A		A = 70
t4		Lire A	
t5	Annulation		

   Résultat : A = 70, Résultat correct : A = 20
4. Lecture Ecriture (Lecture non reproductible)
   — T1 modifie la valeur de A entre deux lectures de T2.

temps	Transaction T1	Transaction T2	Etat de la base
t1	Lire A		A = 20
t2		Lire A	
t3	A := A +50		
t4	Ecrire A		A = 70
t5		Lire A	

4.4.1 Sérialisation des transactions

L'état final d'une BD après exécutions en parallèle de transactions doit être identique à une exécution en série des transactions.

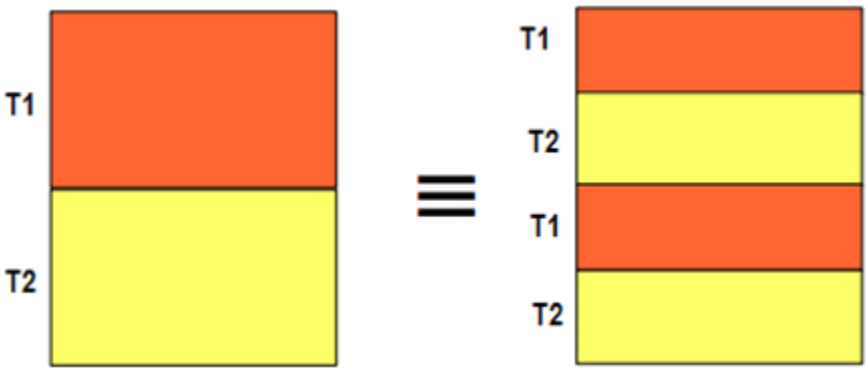

FIGURE 4.1 – Une exécution en parallèle et une exécution en série des transactions

Exemple : Succession : T1,T2

T1	(1) : A := A - 5
	(2) : A := A + 2
T2	(3) : B := B + 3
	(4) : B := B - 6

(2) et (3) sont permutables car elles n'agissent pas sur le même granule (un tuple ou bien une table). On peut transformer en :

(1) : A := A - 5
(2) : B := B + 3
(3) : A := A + 2
(4) : B := B - 6

Cette exécution est sérialisable.

Le problème du contrôle de concurrence consiste pour le système de ne générer que des exécutions sérialisables.

4.4.2 Graphe de précédence

— La notion de précédence de transactions peut être représentée par un graphe.
— Un graphe dont les nuds représentent les transactions et dans lequel il existe un arc Ti vers Tj si Ti précède Tj dans l'exécution analysée.

Exemple :

T1 Lire A ; T2 Ecriture A ; T2 Lire B ; T3 Lire A ; T1 Ecriture B
Pour chaque granule :
A : {LT1, ET2, LT3}
B : {LT2, ET1} (Pas darc entre L et L car on a pas de conflit)
Condition suffisante de sérialisabilité : le graphe de précédence est sans circuit.

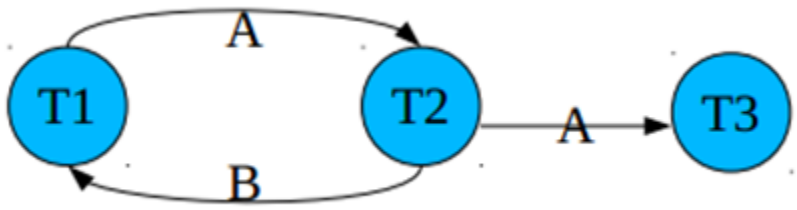

FIGURE 4.2 – Graphe de précédence

— De nombreuses solutions ont été proposées pour traiter le problème des accès concurrents. Deux principales techniques pour garantir la sérialisabilité des transactions :
  — Le verrouillage
  — L'estampillage

4.4.3 Verrouillage

La technique de contrôle des accès concurrents consistent à verrouiller les objets au fur et à mesure des accès par une transaction et à relâcher les verrous seulement après obtention de tous les verrous.

Il repose sur les deux actions :

— Acquérir un verrou sur l'objet A : *verrouiller (A)*
— Relâcher le verrou sur l'objet A : *libérer (A)*

Pour garantir l'isolation des mises à jour, les verrous sont généralement relâchés en fin de transaction. Si une transaction demande un verrouillage d'un objet déjà verrouillé, cette transaction est mise en attente jusqu'à relâchement de l'objet.
un objet A est un granule (un tuple ou bien une table) de la BD.

Deux types de verrous :

— Verrous exclusifs *(X locks)* ou verrous d'écriture
— Verrous partagés *(S locks)*ou verrous de lecture

Verrou demandé	Verrou déjà accordé pour une autre transaction	
	S-lock	X-lock
S-lock	Accordé	Attente de libération
X-lock	Attente de libération	Attente de libération

Protocole d'accès aux données

1. Aucune transaction ne peut effectuer une lecture ou une mise à jour d'un objet si elle n'a pas acquis au préalable un verrou S ou X sur cet objet
2. Si une transaction a un verrou S sur un granule, elle peut demander le verrou X (upgrade).
3. Si une transaction ne peut obtenir un verrou déjà détenu par une autre transaction T2, alors elle doit attendre jusqu'à ce que le verrou soit libéré par T2.
4. Les verrous X sont conservés jusqu'à la fin de la transaction (COMMIT ou ROLLBACK)
5. En général les verrous S sont également conservés jusqu'à cette date.

Phénomènes indésirables

1. **La privation :** .
   — Une transaction risque d'attendre un objet indéfiniment si à chaque fois que cet objet est libéré, il est pris par une autre transaction.
   — Pour traiter ce problème, on peut organiser sur chaque verrou une file d'attente avec une politique "première arrivée", "première servie".

2. **L'inter-blocage (ou verrou mortel) :** Ti attend Tj , Tj attend Ti : il y a inter-blocage.

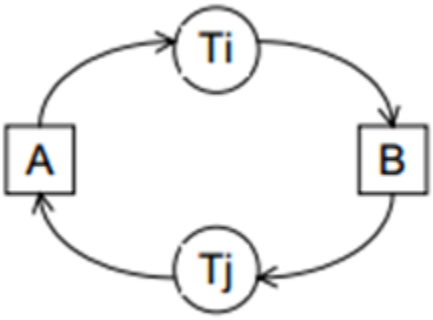

FIGURE 4.3 – L'inter-blocage

Traiter le problème d'inter-blocage

1. **Prévention des inter-blocages :** — Lorsqu'une demande d'acquisition de verrou ne peut être satisfaite on fait passer un test aux deux transactions impliquées, à savoir celle qui demande le verrou, Ti, et celle qui le possède déjà, Tj.
   — Si Ti et Tj passent le test alors Ti est autorisée à attendre Tj, sinon l'une des deux transactions est annulée pour être relancée par la suite.

2. **Détection des inter-blocages :** — Les inter-blocages sont détectés en construisant le graphe "qui attend quoi" et en y recherchant les cycles.
   — Lorsqu'un cycle est découvert l'une des transactions est choisie comme victime, elle est annulée de manière à faire disparaître le cycle.

Syntaxe (MYSQL) : verrous de table

LOCK TABLES nom_table *[READ | WRITE]*;

— En utilisant READ, un verrou de lecture sera posé ; c'est-à-dire que les autres sessions pourront toujours lire les données des tables verrouillées, mais ne pourront plus les modifier.
— En utilisant WRITE, un verrou d'écriture sera posé. Les autres sessions ne pourront plus ni lire ni modifier les données des tables verrouillées.

UNLOCK TABLES ; - - On relâche les verrous

Syntaxe (MYSQL) : verrous de ligne

a- Verrou partagé : Pour poser un verrou partagé, on utilise LOCK IN SHARE MODE à la fin de la requête SELECT.
Exemple de Code SQL
SELECT * FROM WHERE idProduit = 10 *LOCK IN SHARE MODE*;

b- Verrou exclusif : Pour poser un verrou exclusif, on utilise FOR UPDATE à la fin de la requête SELECT.

Exemple de Code SQL

SELECT * FROM WHERE idProduit = 10 *FOR UPDATE*;

Un verrou de ligne est donc lié à la transaction dans laquelle il est posé. Dès que l'on fait un COMMIT ou un ROLLBACK de la transaction, le verrou est levé.

Exemple

On considère l'ordonnancement de T1, T2, T3 suivant :

T1	T2	T3
Lire(A)		
		Ecrire(A)
	Lire(A)	
Ecrire(B)		
	Lire(B)	
		Ecrire(B)

Q1 : Est-ce cet ordonnancement est sérialisable ?

Q2 : Décrivez comment le mécanisme d'accès par verrouillage a deux phases sérialise cet ordonnancement ?

R1 : Graphe de précédence :

T1L(A), T3E(A), T2L(A), T1E(B), T2L(B), T3E(B)

A :{LT1, ET3, LT2}

B :{ET1, LT2, ET3}

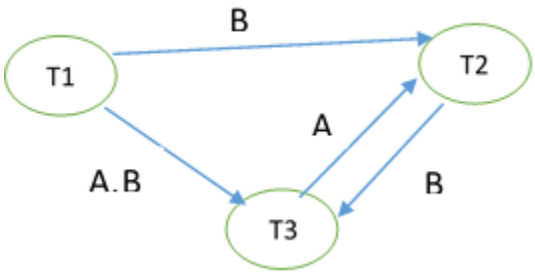

FIGURE 4.4 – Le graphe de précédence de l'exemple

Le graphe avec circuit donc cette exécution peut être sérialisable ou non

R2 : le mécanisme d'accès par verrouillage a deux phases

	T1	T2	T3
t1	S(A)		
t2			Attente X(A)
t3		S(A)	
t4	S(A) X(B)		
t5	Libérer S(A) Libérer X(B)		
t6		S(A) S(B)	
t7		Libérer S(A) Libérer S(B)	
t8			X(A) X(B)
t9			Libérer X(A) Libérer X(B)

Le nouvel ordre dexécution : LT1(A), LT2(A), ET1(B), LT2(B), ET3(A), ET3(B).

A :{LT1, LT2, ET3}

B :{ET1, LT2, ET3}

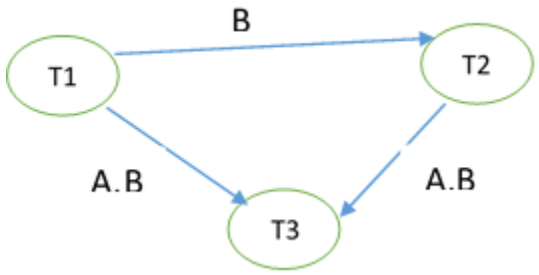

FIGURE 4.5 – Le nouveau graphe de précédence de l'exemple

Chapitre 5

Optimisations de requêtes

5.1 Généralités sur l'optimisation des requêtes

Objectif de l'évaluation des requêtes :

- Évaluateur traduit une requête SQL en un programme appelé plan d'exécution
- Plan d'exécution se compose :
  - Des opérations de l'algèbre relationnelle avec quelques extensions (tri, agrégat)
  - D'opérations de transfert de données
  - De primitives de synchronisation
- Facteurs d'optimisation (Optimiser quelles ressources) :
  - CPU, nombre d'E/S, taille mémoire nécessaire, coût de transfert, gain dû au parallélisme

Etapes de l'évaluation

1. Traduction de la requête SQL en algèbre relationnelle
2. Tracer l'arbre algébrique
3. Déduire les plans d'exécutions pour l'arbre algébrique
4. Calculer les coûts des plans
5. Sélectionner un plan

5.2 Traduction de la requête SQL en algèbre relationnelle

Rappel de vocabulaire : Algèbre relationnelle

1. La projection π : $\pi_{Nom,adresse}$.
2. La sélection σ : $\sigma_{adresse='Bejaia'}$.
3. Le produit cartésien x : R x S.
4. La jointure ∞ : Client $\infty_{numero=no\_client} Vente$.
5. L'union $\cup$: $R \cup S$.
6. La déférence - : R - S.
7. Intersection, division, etc

Exemple : Requête SQL
SELECT Nom, Bureau
FROM Employe, Departement
WHERE Employe.numoD = Departement.numoD AND Employe.salaire > 1000

Traduction en algèbre relationnelle

$$\pi_{Nom,Bureau}(\sigma_{E.salaire>1000}(Employe \infty_{E.numD=D.numD} Departement))$$

5.3 Tracer l'arbre algébrique

— Visualiser graphiquement une requête relationnelle traduite en une expression équivalente de l'algèbre relationnelle.
— *Feuilles :* tables utilisées dans la requête
— *Noeuds intermédiaires :* opérations algébriques
— *Noeud racine :* dernière opération algébrique avant le retour du résultat

— L'opération d'un noeud intermédiaire est déclenchée quand ses opérandes sont disponibles.

On remplace alors le noeud par la relation produite par l'opération.
Exemple :

$$\pi_{Nom,Bureau}(\sigma_{E.salaire>1000}(Employe \infty_{E.numD=D.numD} Departement))$$

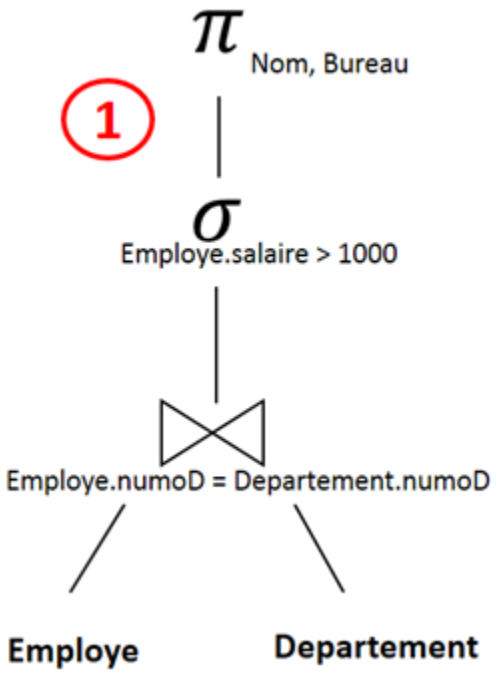

FIGURE 5.1 – l'arbre algébrique

5.4 Déduire les plans d'exécutions pour l'arbre algébrique

Le principe général de l'optimisation repose sur le constat suivant :

1. Les opérations unaires produisent des tables plus petites que la table d'origine.

2. Les opérations binaires produisent des tables plus grandes que la table d'origine. Autrement dit, ce sont les produits cartésiens des jointures qui accroissent la taille des tables intermédiaires.
   ⇒ *Idée :* Supprimer un maximum de lignes et de colonnes avant de faire les jointures et faire les jointures avant les produits cartésiens.

5.4.1 Règles de transformation : (La base mathématique)

— *Règle 1 :* Commutativité et associativité de ∞ et x :

$$E_1 \infty E_2 = E_2 \infty E_1$$
$$(E_1 \infty E_2) \infty E_3 = E_1 \infty (E_2 \infty E_3)$$

Si $(E_1 \infty E_2)$ est plus grand que $(E_2 \infty E_3)$,choisir $E_1 \infty (E_2 \infty E_3)$

$$E_1 \text{ x } E_2 = E_2 \text{ x } E_1$$
$$(E_1 \text{ x } E_2) \text{ x } E_3 = E_1 \text{ x } (E_2 \text{ x } E_3)$$

— *Règle 2 :* Cascade de projections

$$\pi_{A_1,...,A_n}(\pi_{B_1,...,B_m}(E)) = \pi_{A_1,...,A_n}(E)$$

— *Règle 3 :* Cascade de sélections

$$\sigma_{F_1}(\sigma_{F_2}(E)) = \sigma_{F_1 \wedge F_2}(E)$$

— *Règle 4 :* Commutativité de la sélection avec la projection

Si F ne porte que sur $A_1, ..., A_n$

$$\pi_{A_1,...,A_n}(\sigma_F(E)) = \sigma_F(\pi_{A_1,...,A_n}(E))$$

Si F porte aussi sur $B_1, ..., B_m$

$$\pi_{A_1,...,A_n}(\sigma_F(E)) = \pi_{A_1,...,A_n}(\sigma_F(\pi_{A_1,...,A_n,B_1,...,B_m}(E)))$$

— *Règle 5 :* Commutativité de la sélection avec x, $\cup$, -, ∞

$$\sigma_F(E_1 OP E_2) = \sigma_F(E_1) OP \sigma_F(E_2)$$

— *Règle 6 :* Commutativité de la projection avec $\cup$, x

$$\pi_{A_1,...,A_n}(E_1 OP E_2) = \pi_{A_1,...,A_n}(E_1) OP \pi_{A_1,...,A_n}(E_2)$$

L'optimisation intuitive se résume à :

1. Faire toutes les restrictions spécifiques pour limiter le nombre de tuples.
2. Faire toutes les projections mono-tables possibles pour limiter le nombre d'attributs.
3. Faire les jointures et après chaque jointure, les projections possibles.

4. Faire les "distinct", les "tris", les "group by", les fonctions de groupe.

En utilisant les règles de transformations précédentes, on arrive à l'algorithme suivant :

1. Séparer les restrictions comportant plusieurs prédicats (R3).
2. Faire descendre les restrictions le plus bas possible (R4, R5).
3. Regrouper les restrictions successives portant sur une même relation (R3).
4. Séparer les projections comportant plusieurs prédicats (R2).
5. Faire descendre les projections le plus bas possible (R4, R6).
6. Regrouper les projections successives (R2).

Exemple :

$$\pi_{Nom,Bureau}(\sigma_{E.salaire>1000}(Employe \infty_{E.numD=D.numD} Departement))$$

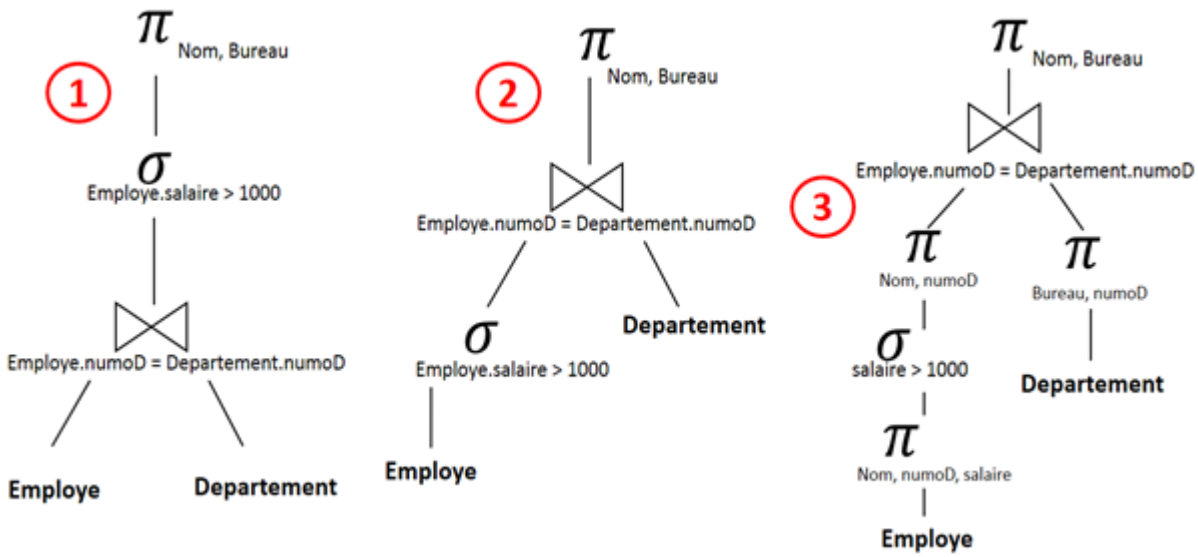

FIGURE 5.2 – (1)l'arbre algébrique (Plan initial) (2) Plan optimisé 1 (3) Plan optimisé 2

5.5 Calculer les coûts des plans

Le coût d'un plan dépend des algorithmes du calcul du coût des opérateurs algébriques. Il existe plusieurs algorithmes, donc, le coût peut changer d'un algorithme à l'autre.
On s'intéresse au coût des opérateurs ∞ et σ en terme d'Entrées/Sortie
Algorithme :

- coût de la sélection : $\sigma_{condition}(R)$
  - coût (Entrée) = Lecture de tous les tuples- Tuples (R).
  - coût (Sortie) = Les tuples qui vérifient la condition.

- coût de la jointure : $R_1 \infty R_2$
  - coût (Entrée) = (Chargement) Lecture de R1, puis lecture de R2 = Tuples(R1)+(Tuples(R1)x Tuples(R2))
  - coût (Sortie) = Les tuples qui vérifient la condition : <= Tuples(R1)x Tuples(R2)

coût Total = $\sum_{i=1}^{n} OP_i$

Exemple :

$$\pi_{Nom,Bureau}(\sigma_{E.salaire>1000}(Employe \infty_{E.numD=D.numD} Departement))$$

Hypothèses :

1. Il y a 300 lignes dans "Employe"
2. Il y a 12 lignes dans "Departement"
3. On suppose qu'il n'y a que 20% d'employés ayant un salaire > 1000

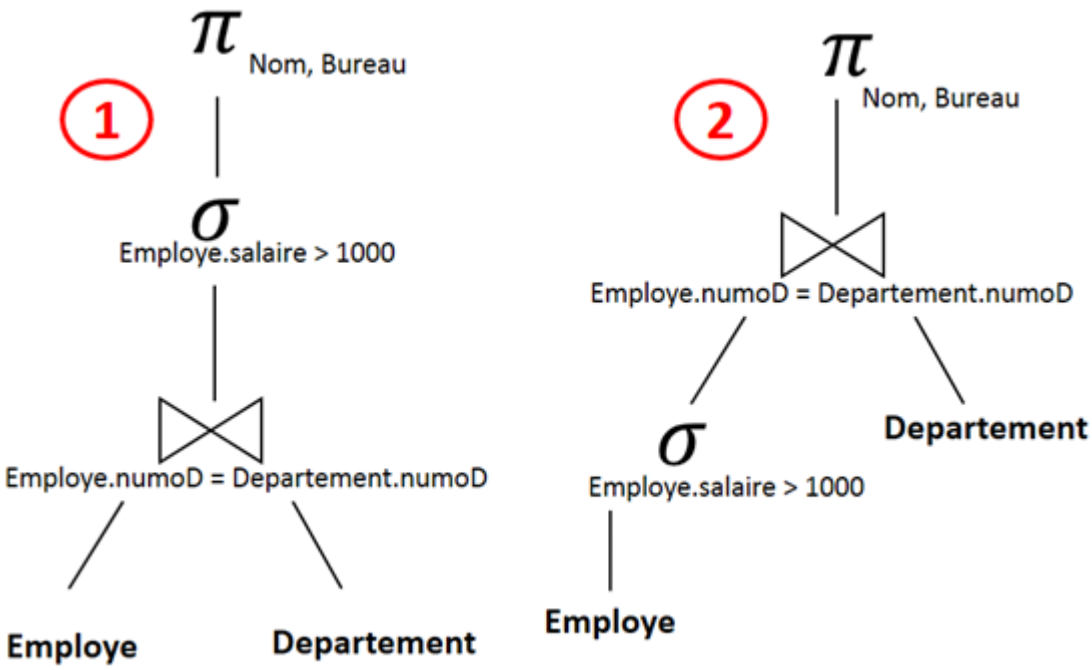

FIGURE 5.3 – (1) Plan initial (2) Plan optimisé

Coût plan initial

- Coût de la jointure :
  - coût (Entrée) = 300 + (300*12) = 3900E
  - coût (Sortie) = 300 S (pour chaque employé, un seul Département)
  - Total = 3900 + 300 = 4200 E/S
- Coût de la sélection :
  - coût (Entrée) = 300 E
  - coût (Sortie) = 300 * 0.2 = 60 S
  - Total = 300 + 60 = 360 E/S
- Coût Total = 4200 E/S + 360 E/S = 4560 E/S

Coût plan optimisé

- Coût de la sélection :
  - coût (Entrée) = 300 E
  - coût (Sortie) = 300 * 0.2 = 60 S
  - Total = 300 + 60 = 360 E/S
- Coût de la jointure :
  - coût (Entrée) = 60 + (60*12) = 780E
  - coût (Sortie) = 60 S (pour chaque employé, un seul Département)
  - Total = 780 + 60 = 840 E/S
- Coût Total = 360E/S + 840 E/S = 1200 E/S

5.6 Sélectionner un plan

Le principe général pour sélectionner un plan parmi d'autres consiste à sélectionner celui qui permet de limiter le nombre d'accès disques, soit, en première approximation, le nombre de tuples parcourus.

Chapitre 6

Administration d'une base de données (sauvegarde, restauration)

6.1 La sauvegarde

6.1.1 Pourquoi sauvegarder ?

Les sauvegardes, apparaissent comme des actions incontournables pour l'ensemble des données critiques (Base de données) qu'il ne faudrait surtout pas perdre. Avec des sauvegardes valides, vous pouvez récupérer vos données suite à de nombreuses défaillances, par exemple :

Les causes humaines : mauvaises manipulation.

Les causes naturelles : inondation ou incendie.

Les causes matérielles : défaillance de matériel informatique (Crash du disque dur,etc)

La sauvegarde est le seul moyen de protéger vos données.

6.1.2 Comment sauvegarder ?

Il existe de nombreuses solutions informatiques pour sauvegarder vos informations :

La sauvegarde interne : Les données restent au sein de l'entreprise. Elles sont juste copiées sur un autre ordinateur, un disque dur externe ou encore un deuxième serveur (Durée de vie entre 1 et 5 ans).

La sauvegarde externe (ou Cloud) : Les données sont protégées sur un serveur en ligne. Elles sont ainsi accessibles de partout et récupérables rapidement (Durée de vie indéterminé).

6.1.3 Quand sauvegarder ?

Tout dépend du type de la base de données. Généralement, on se fonde sur plusieurs critères pour établir une stratégie de sauvegarde. En voici quelques-uns :

— L'Indice de volatilité des données, qui indique si les données sont fréquemment mises à jour ou non. (Mise à jour fréquente ⇒ sauvegarde fréquente).
— L'Espace alloué au journal des transactions (Un journal de faible taille ⇒ des sauvegardes fréquentes).
— La qualité des supports de sauvegarde.
— La période où est possible de faire une sauvegarde (Ex : La nuit).

De manière générale, sauvegardez vos bases de données toutes les semaines,

6.1.4 Les types de sauvegarde

Ils existent différents types de sauvegardes :

Sauvegarde complète ou totale

Une sauvegarde totale réalise une copie des données à sauvegarder sur un autre support. C'est le type de sauvegarde le plus rapide, le plus simple, et le plus précis pour restaurer les données sans erreurs.

Sauvegarde incrémentale

La sauvegarde incrémentale ne copie que les éléments modifiés depuis la sauvegarde précédente.

Ce type de sauvegarde est plus performant qu'une sauvegarde totale car elle permet de se focaliser uniquement sur les fichiers modifiés avec un espace de stockage plus faible, mais nécessite en contrepartie de posséder les sauvegardes précédentes pour reconstituer la sauvegarde complète et demande un temps de restauration plus long.

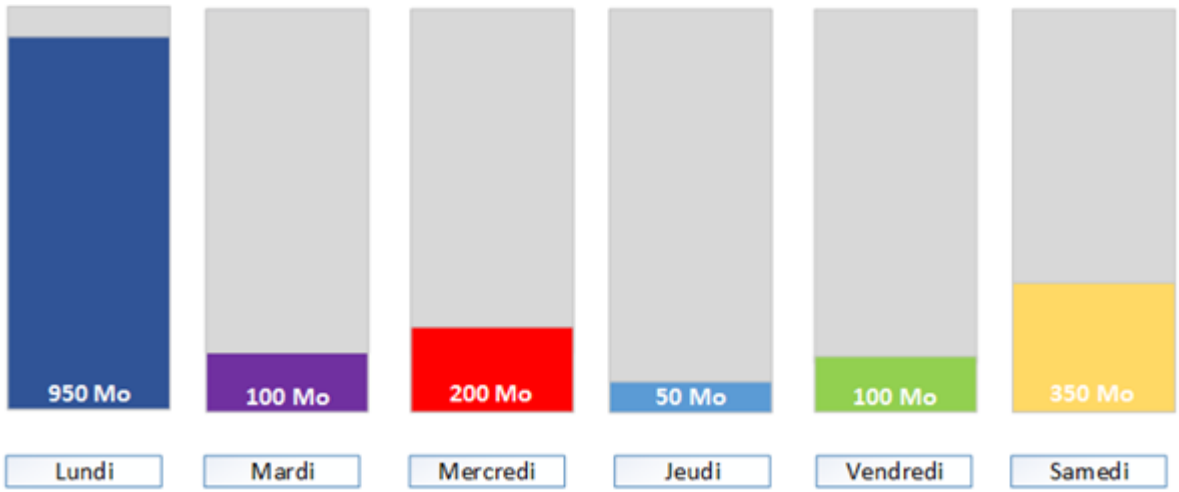

FIGURE 6.1 – Sauvegarde incrémentale

Sauvegarde différentielle :

La sauvegarde différentielle s'occupe uniquement des fichiers modifiés depuis la dernière sauvegarde complète.
Cette sauvegarde demande plus d'espace de stockage qu'une sauvegarde incrémentale mais elle est également plus fiable car seule la sauvegarde complète est nécessaire pour reconstituer les données sauvegardées.

Il existe différents logiciels capables d'effectuer des sauvegardes de type complète, incrémentielle ou différentiel, Exemple sur Windows : AOMEI Backupper / Acronis Backup / Symantec Backup Exec.
Pour sauvegarder une base de données avec MySQL :

mysqldump - -user=mon_ user - -password=mon_ password - -databases nom_ de_ la_ base > fichier_ destination.sql

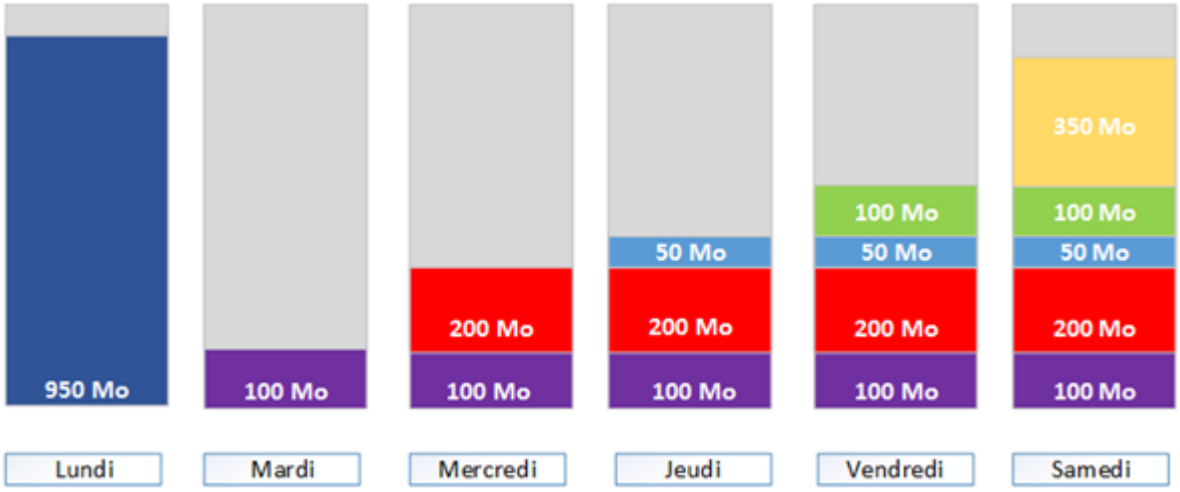

FIGURE 6.2 – Sauvegarde différentielle

6.2 La restauration

La restauration de la base de données restitue le contenu complet de la base de données. Elle ne doit donc pas être en exploitation lors de la restauration.
Pour restaurer une base de données avec MySQL :

mysql - -user=mon_ user - -password=mon_ password nom_ de_ la_ base < fichier_ source.sql

Chapitre 7

Exercices d'application

7.1 Série de TD N 01

Exercice 1 :

Donner la requête SQL pour chacune des situations suivantes :

1. Créer une base de données "Universite".
2. Supprimer la base de données "Universite".
3. Créer une base de données Univ (Si elle n'existe pas).
4. Créer une table utilisateur avec les informations : id, nom, prenom, email, dateNaissance, ville.
5. Ajouter une colonne adresse à la table utilisateur.
6. Supprimer la colonne ville de la table utilisateur.
7. Insérer un utilisateur : id = id1, nom =nom1, prenom= per1, email = nom1@gmail.com, dateNaissance= 12-01-2000, ville =bejaia.
8. Changer la ville de l'utilisateur ayant le nom 'Hamadi' à Alger.
9. Supprimer les utilisateurs qui habitent à Oran.
10. Supprimer la table utilisateur.

Exercice 2 :

Soit la base de données relationnelle avec le schéma :

- Atelier(NumA, NomA, VilleA)
- Fournisseur(NumF, NomF, Statut, VilleF)
- Produit(NumP, NomP, Couleur, Poids)
- Livraison(#NumP, #NumA, #NumF, Quantité)

Avec :

Atelier : Un atelier de fabrication est décrit par son numéro NumA, son nom NomA et la ville VilleA.

Produit : Décrit par son numéro NumP, son nom NomP, sa couleur et son poids.

Fournisseur : Décrit par son numéro NumP, son nom NomF, son statut (sous-traitant, client, etc) et la ville VilleF où il est domicilié.

Livraison : le produit de numéro NumP a été délivré à l'atelier de numéro NumA par le fournisseur de numéro NumF.

Exprimer en SQL :

1. Donner le numéro, le nom, la ville de tous les Ateliers.
2. Donner le numéro, le nom, la ville de tous les Ateliers de Setif.
3. Supprimer tous les produits de couleur Jaune et de numéros compris entre 100 et 150.
4. Donner les numéros des fournisseurs qui approvisionnent l'atelier de numéro 2 en produit de numéro 10.
5. Donner (par deux requêtes différentes) les numéros des fournisseurs qui approvisionnent l'atelier de numéro 2 en un produit rouge.
6. Donner les noms des fournisseurs qui approvisionnent un atelier de Bejaia ou d'Annaba en produit bleu.
7. Donner les numéros des ateliers qui ont au moins un fournisseur qui n'est pas de la même ville.
8. Donner le numéro des ateliers qui ne reçoivent aucun produit rouge d'un fournisseur d'Alger.

9. Donner les numéros des produits qui sont livrés à tous les ateliers de Bejaia.
10. Donner les numéros des fournisseurs qui approvisionnent tous les ateliers avec un même produit.

Exercice 3 :

Soit le schéma suivant :

— **coureur** (numLicence, nom, prenom, dateNaissance)
— **course** (numCourse, codePostal, ville)
— **resultat** (#numCourse, #numLicence, rang)

1. Donner le nombre de coureurs dans chaque course.
2. Donner le nom et prenom du coureur le plus âgé ayant couru dans les courses qui se sont déroulées à Béjaia.
3. Donner la (ou les) villes ayant organisé le plus grand nombre de courses.
4. Ajouter des contraintes de clés étrangères sur les deux colonnes numCourse et numLicence de la table resultat qui font référence respectivement à numCourse de la table course et numLicence de la table coureur.
5. Donner les noms et prénoms des coureurs ayant participé à au moins deux courses à Alger.
6. Donner le nom, prénom et classement du coureur le plus jeune dans la course numéro 15.

7.2 Série de TD N 02

Exercice 1 :

Soit le schéma suivant :

- Client (NClient, Nom, Prenom, email, telephone, Adresse)
- Produit(NProduit, nomProduit, prix, quantiteReelle)
- Commande(NCommande, dateCommande, #NClient)
- LigneCommande(#NProduit, #NCommande, quantiteCommandee)

1. Créer un index unique sur les colonnes Nom et Prenom de la table Client.
2. Créer une vue ProduitCmd regroupant la quantité globale commandée de chaque produit.
3. Créer une vue LigneCmd (NP, NC, QC) sur les produit ayant une quantité commandée globale qui dépasse les 500 unités.
4. Sélectionner toutes les lignes de la vue LigneCmd.
5. Insérer les valeurs : (15, 121, 600) dans la vue LigneCmd.
6. Sélectionner la liste des produits en indiquant : 'stock épuisé' si quantiteReelle = 0, 'stock plein' si quantiteReelle = 5000 et 'produit disponible' sinon.

Exercice 2 :

1. Accorder le droit de SELECT sur la table GENRE à un utilisateur user1.
2. Accorder les droits de INSERT et UPDATE sur GENRE à un utilisateur user1.
3. Accorder tous les droits sur la table DISQUE à un utilisateur user1.
4. Créer une vue relative au titre, à l'artiste (prénom+nom+groupe) et à l'année des disques. Puis accorder les droits de SELECT sur la vue à un utilisateur user1
5. Créer une vue relative à toutes les informations (titre de la chanson, titre du disque, artiste du disque, position sur le disque, année du disque) des chansons interprétées par David BOWIE. Puis accorder tous les droits sur la vue à un utilisateur user1.
6. Retirer le droit INSERT à l'utilisateur user1, sur la table GENRE.

7. Créer un rôle DroitEnseignant
8. Accorder le privilège de création de table et de vue pour ce rôle.
9. Créer un utilisateur user1 et lui octroyer le rôle DroitEnseignant.

Exercice 3 :

Soit le schéma de l'exercice 1.

1. Créer un trigger (Déclencheur) qui met à jour la quantité réelle (quantiteReelle) d'un produit à chaque nouvelle commande sur le produit en question.
2. Créer un trigger (Déclencheur) qui supprime automatiquement les lignes commandes lors de la suppression d'une commande.
3. Que fait le trigger suivant :
   CREATE TRIGGER supLC
   AFTER DELETE ON LigneCommande
   FOR EACH ROW
   BEGIN
   UPDATE Produit p SET p.quantiteReelle = p.quantiteReelle + OLD.quantiteCommandee
   WHERE p.NProduit = OLD.NProduit
   END

7.3 Série de TD N 03

Exercice 1 :

1. Commenter et donner le résultat de l'exécution des deux transactions ci-dessous ?

BEGIN TRANSACTION TRAN1 ; CREATE TABLE T1 (A INTEGER) ; CREATE TABLE T2 (A INTEGER) ; CREATE TABLE T3 (A INTEGER) ; INSERT INTO T1 VALUES (1) ; INSERT INTO T2 VALUES (1) ; INSERT INTO T3 VALUES (1) ; ROLLBACK ;	BEGIN TRANSACTION Tr1 ; CREATE TABLE T1 (A INTEGER) ; CREATE TABLE T2 (A INTEGER) ; SAVE TRAN P1 ; CREATE TABLE T3 (A INTEGER) ; INSERT INTO T1 VALUES (1) ; INSERT INTO T2 VALUES (1) ; INSERT INTO T3 VALUES (1) ; ROLLBACK TRAN P1 ; COMMIT ;

2. Soit la table Film : Film(idF, entrees) définie en relationnel permettant d'enregistrer le nombre d'entrées des films identifiés par leur idF.

Temps	Transaction Tr1	Transaction Tr2
t0	BEGIN TRANSACTION Tr1	
t1		BEGIN TRANSACTION Tr2
t2	UPDATE Film SET entrees=entrees+1 WHERE idF ='3'	
t3		UPDATE Film SET entrees=entrees+1 WHERE idF ='3'
t4	COMMIT	
t5		COMMIT

   a. Du point de vue de la transaction Tr1, puis Tr2, de combien les entrées du film 3 ont-t-elles été augmentées à t2, t3, t4 et t5.

   b. Après exécution des deux transactions, la base de données est-elle dans un état cohérent ? est-elle dans un état correct ? expliquer ?.

Exercice 2 :

1. Soit une table Gardien ayant initialement 0 tuples. Commenter chaque action dans les deux transactions ci-dessous, et dire quel sera le nombre final des tuples après leur exécution ?

BEGIN TRAN ;	BEGIN TRAN ;
INSERT INTO GARDIEN VALUES	INSERT INTO GARDIEN VALUES
(80,'Durand','aa','16/01/1990') ;	(80,'Ali','aa','16/01/1990') ;
SELECT * FROM GARDIEN ;	SELECT * FROM GARDIEN ;
ROLLBACK ;	COMMIT ;
SELECT * FROM GARDIEN ;	SELECT * FROM GARDIEN ;

2. Ecrire une transaction contenant un BEGIN TRAN, puis 2 insertions dans Gardien (avec nom= Nom1 et Nom2), suivies d'un COMMIT, suivi d'une insertion (nom=Nom3), une mise à jour (sur date de naissance) et une destruction d'un tuple, suivie d'un BEGIN TRAN, suivi d'une sélection complète sur la table, suivi d'un point d'arrêt P1, suivi d'une mise à jour (sur date de naissance), suivie d'une insertion d'un tuple, suivi d'une sélection complète sur la table, suivie d'un ROLLBACK P1, suivi d'une sélection complète sur la table.

   a. Donner le nombre de tuples dans la table Gardien à chaque sélection ?

   b. Donner le nombre final des tuples dans la table Gardien ?

   c. Donner le nombre de mises à jour validées.

Exercice 3 :

On considère 04 transactions concurrentes qui veulent mettre à jour deux tables Tab1 et Tab2.

T0	UPDATE Tab1 SET nb1 = 15
	UPDATE Tab2 SET nb2 = 20
T1	UPDATE Tab1 SET nb1 = nb1 + 1
	UPDATE Tab2 SET nb2 = nb2 - 1
T2	UPDATE Tab1 SET nb1 = nb1 + 3
	UPDATE Tab2 SET nb2 = nb2 - 3
T3	SELECT nb1 FROM Tab1
	SELECT nb2 FROM Tab2

1. Donner les valeurs finales de nb1 et nb2 dans le cas d'exécution des transactions succession T0, T1, T2 puis T3.

   On envisage trois schémas d'exécutions Exécution 1, Exécution 2 et Exécution 3 :

Exécution 1	**Exécution 2**	**Exécution 3**
T0 écriture Tab1	T0 écriture Tab1	T0 écriture Tab1
T0 écriture Tab2	T1 lecture Tab1	T0 écriture Tab2
T1 lecture Tab1	T1 écriture Tab1	T3 lecture Tab1
T1 écriture Tab1	T0 écriture Tab2	T3 lecture Tab2
T2 lecture Tab1	T2 lecture Tab1	T1 lecture Tab1
T2 écriture Tab1	T2 écriture Tab1	T1 écriture Tab1
T3 lecture Tab1	T1 lecture Tab2	T2 lecture Tab2
T1 lecture Tab2	T2 lecture Tab2	T2 écriture Tab2
T1 écriture Tab2	T1 écriture Tab2	T1 lecture Tab2
T2 lecture Tab2	T2 écriture Tab2	T1 écriture Tab2
T2 écriture Tab2	T3 lecture Tab1	T2 lecture Tab1
T3 lecture Tab2	T3 lecture Tab2	T2 écriture Tab1

2. Lesquels de ces exécutions sont sérialisables par rapport à la succession T0, T1, T2, T3 ?
3. Donnez les graphes de précédences de ces exécutions et retrouvez les résultats précédents.
4. Intéressons-nous aux exécutions non sérialisables. Est-ce qu'elles peuvent être exécutées par une procédure de verrouillage à deux phases (2PL) utilisant deux

modes : S-lock (lecture partagée) et X-lock (écriture exclusive).

Exercice 4 :

Un ordonnanceur avec verrouillage à 2 phases reçoit l'exécution ci-dessous, de trois transactions T1, T2 et T3, sur deux variables différentes x et y (L = Lecture, E = Ecriture, R = relâcher tous les verrous détenus par la transaction).

Exe : L1[x] L2[y] E3[x] E1[y] E1[x] E2[y] R2 L3[y] L1[y] R1 E3[y] R3

Indiquer l'ordre d'exécution établi par l'ordonnanceur, en considérant que les opérations bloquées en attente d'un verrou seront exécutées en priorité dès que le verrou devient disponible, dans l'ordre de leur blocage. On suppose que les verrous d'une transaction sont relâchés au moment du Commit.

7.4 Série de TD N 04

Exercice 1 :

Soit la requête suivante :

SELECT Film.réalisateur
FROM Film, Séance
WHERE Seance.heure-debut > 14 AND Film.film = Seance.film

1. Donner l'expression algébrique
2. Donner l'arbre algébrique
3. Donner un plan d'exécution optimisé
4. Supposant que la table Film comporte 200 tuples, Séance 100 tuples et que 50% des films ont des séances et 30% des séances sont après 14h.
   a. Calculer le cout (en Entrée/Sortie) des opérateurs algébriques des deux plans (initial et optimisé)
   b. Donner le cout total (en Entrée/Sortie) des deux plans

Exercice 2 :

Soit la requête suivante :

SELECT nomArtiste, titre
FROM Artiste, Chanson
WHERE Artiste.numArtiste = Chanson.numArtiste AND dateSortie > '01-01-2019'

1. Donner l'expression algébrique
2. Donner l'arbre algébrique
3. Donner un plan d'exécution optimisé
4. Supposant que la table Artiste comporte 20 tuples, Chanson 300 tuples et que 80% des artistes ont des chansons et 50% des chanson sont après '01-01-2019'.

a. Calculer le cout (en Entrée/Sortie) des opérateurs algébriques des deux plans (initial et optimisé)

b. Donner le coût total (en Entrée/Sortie) des deux plans

Bibliographie

[1] TONY ARCHAMBEAU. *Polycopié de cours : Cours SQL.* Institut Universitaire de Technologie de Villetaneuse de Nice, France.

[2] Nacer BOUDJLIDA. *livre : Gestion et administration des bases de données.* Dunod. ISBN-13 : 978-2100058471, 2003.

[3] OLIVIER CURE. *Polycopié de cours : Concurrence et transactions.* Université de Marne la vallée de Paris, France.

[4] GARDARIN GEORGES. *livre : Bases de données : Objet et relationnel.* Eyrolles. ISBN-13 : 978-2212092837, 2001.

[5] CHANTAL GRIBAUMONT. *livre : Administrez vos bases de données avec MySQL.* OpenClassrooms. ISBN-13 : 979-1090085671, 2014.

[6] RICHARD GRIN. *Polycopié de cours : Langage SQL.* Institut Universitaire de Technologie de Villetaneuse de Nice, France.

[7] MAXIMILIEN LAURENT. *Polycopié de cours : Concepts et langages des Bases de Données Relationnelles.* Institut Universitaire de Technologie de Villetaneuse de Nice, France.

[8] ELISABETTA DE MARIA. *Polycopié de cours : Cours de Base de Données.* Institut Universitaire de Technologie de Villetaneuse de Nice, France.

[9] ANDREAS MEIER. *livre : Introduction pratique aux bases de données relationnelles.* Springer. ISBN-13 : 978-2287252051, 2010.

[10] PHILIPPE RIGAUX. *Polycopié de cours : Cours de bases de données.* conservatoire national des arts et métiers, France.

[11] PHILIPPE RIGAUX. *livre : Pratique de MySQL et PHP.* Dunod. ISBN-13 : 978-2100523368, 2009.