REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
UNIVERSITE ALGER 1
BENYOUCEF BENKHEDDA

3EME ANNEE LICENCE INFORMATIQUE
SPECIALITÉ ISIL

Dr. AOUDIA

POLYCOPIE DU COURS BASE DE DONNEES AVANCEES

VERSION 1.0

ALGER, LE 3 SEPTEMBRE 2017

TABLE DES MATIÈRES

Page

CHAPITRE 1
INTÉGRITÉ DES BASES DE DONNÉES

1.1 Introduction

Une base de données est un ensemble d'information stocké d'une manière structurée sous une forme spécifique. Les données y figurant doivent représenter en tout temps les informations réelles, elles doivent aussi fournir l'information nécessaire à toute interrogation. Les données stockées dans une base de données ne sont pas toutes contenues dans une seule table, elles peuvent être réunies dans des tables différentes. Par ailleurs, plusieurs tables peuvent contenir une même donnée. De plus, toutes ces données peuvent subir des changements pendant la durée de vie de la base de données. En effet, garder la cohérence des données dans une base de données ne devient plus une tâche simple. Par conséquent, des mécanismes de gestions sont implémentés dans les SGBD (Systèmes de Gestion de Bases de Données) afin d'assurer l'intégrité de données.

Ce chapitre illustre ces systèmes de gestion pour l'assurance de l'intégrité de l'information. Notamment, les contraintes d'intégrité, les déclencheurs et les vues. Chacun de ces acteurs y sera détaillé et son rôle y sera illustré.

Le livre de gardarin a été une bonne source pour la matière présentée dans ce chapitre.

1.2 Contraintes d'intégrité

1.1.1 Définition

Une contrainte d'intégrité est une règle appliquée sur les données, spécifiée à la création de la base ou durant sa durée de vie. La contrainte d'intégrité garantie la cohérence et l'intégrité de données dans une base de données pour qu'elles restent conformes à la réalité.

Exemple :

```
create table EMPLOYE (
NOM Char(20) unique not null,
SAL Integer,
AGE Integer,
DIR Char(20) )
```

1.1.2 Typologie des contraintes d'intégrité

On entend par typologie des contraintes d'intégrité leurs types. L'auteur du livre (Gardarin, 2013) cite deux types de contraintes d'intégrité, à savoir, les **contraintes d'intégrité structurelles** et es **contraintes d'intégrité non structurelle.** Cette section va détailler chaque type.

1.2.1 Contraintes d'intégrité structurelles

"Contrainte d'intégrité spécifique à un modèle exprimant une propriété fondamentale d'une structure de données du modèle." (Gardarin, 2013)

Ces contraintes sont intégrées au modèle de données. Elles sont généralement statiques, , et s'expriment de manière explicite, et peuvent exprimer certaines propriétés des relations et des domaines des attributs.

Elles se composent de :

- **Contrainte d'unicité de clé :** Il s'agit de déterminer l'attribut ou l'ensemble d'attributs dans une table, non nuls, qui permettent de fournir un tuple unique pour cette table. Ces attributs sont la clé de la table.

- **Contrainte référentielle :** Cette contrainte est utilisée pour contrôler les valeurs d'un attribut existant dans plusieurs tables. Elle représente une association entre deux tables. L'une des deux tables est dite référencée, l'autre table est référençante. La table référencée représente l'entité, elle contient l'information commune comme des clés étrangères, et l'autre représente l'association et elle contient l'information commune comme clé primaire. La modification des tuples dans la table référençante, induit une mise à jours des tuples correspondants dans la table référencée en s'appuyant sur la valeur de la clé primaire et de la clé étrangère.

- **Contrainte de domaine :** Le rôle de cette contrainte est de réduire la plage de valeur d'un domaine. En effet, comme le domaine de valeur définie le type de valeur d'un attribut, la définition d'une contrainte de domaine induit que les valeurs de l'attribut doivent d'une part respecter le type de valeur définie et d'autre part respecter la plage de valeurs restreintes par la contrainte de domaine. À titre d'exemple, si l'âge d'un adulte est définie comme étant une valeur supérieure à 19, tous les attributs âge doivent respecter le type de la valeur qui doit être numérique et la valeur de l'attribut qui doit être supérieure à 19.

- **Contraintes de non nullité :** cela entend que la valeur de l'attribut doit être précisée et ne doit en aucun cas être laissée nulle.

1.2.2 Contraintes d'intégrité non structurelles

"*Contrainte d'intégrité exprimant une règle d'évolution que doivent vérifier les données lors des mises à jour."* (Gardarin, 2013)

Elle sont aussi appelées contraintes de comportement ou comportementales. Les contraintes comportementales ne sont pas inhérentes au modèle de données, elles contrôlent les données suite aux mises à jour de l'information.

Dans le modèle relationnel, et contrairement aux contraintes structurelles qui sont définies dans la commande CREATE TABLE, les contraintes non structurelles sont définies par la commande CREATE ASSERTION.

Les différentes contraintes comportementales sont les suivantes :

- **Les dépendances fonctionnelles :** un groupe d'attributs peut être définie via un autre groupe d'attributs en utilisant une fonction donnée.

    - **Exemple**

- Exprimer qu'un livre est identifié par son titre et son auteur
  titre, auteur → titre, auteur, year, type

- **Les dépendances multivaluées** : il s'agit d'une généralisation des contraintes de dépendances fonctionnelles où la fonction utilisée est une fonction multivaluée. Une fonction multivaluée est une fonction qui associe à chaque élément d'un ensemble, zéro, un ou plusieurs éléments d'un autre ensemble.

- **Les dépendances d'inclusion :** Elles assurent que les valeurs d'un ensemble d'attributs d'une table, restent incluses dans un autre ensemble d'attributs appartenant à une autre table de la base de donnée.

    - **Exemple,** les valeurs de la colonne département de la table *étudiant* doit être incluse dans les valeurs de la colonne département de la table *enseignant*

- **Les contraintes temporelles :** Cette contrainte permet d'introduire des expressions dans lesquelles intervient le temps. Elles permettent de comparer l'ancienne valeur d'un attribut à la nouvelle après mise à jour.

    - **Exemple,** la note globale d'un étudiant ne doit pas diminuée après l'ajout d'une note partielle.

- **Les contraintes équationnelles :** Ces contraintes servent à comparer des expressions arithmétiques calculées à partir des données des relations et de forcer l'égalité ou une inégalité.

    - **Exemple,** vérifier que la somme des étudiants admis (premier calcul) est égale au nombre des étudiants global moins le nombre des étudiants ajournés (2ème calcul).

1.1.3 Vérification des contraintes d'intégrité

1.2.3 Vérification de la cohérence des contraintes

«*Ensemble de contraintes non contradictoires, pouvant en conséquence être satisfait par au moins une base de données.* » (Gardarin, 2013)

Exemple :

- Les deux contraintes d'intégrités suivantes sont contradictoires :

```
CREATE ASSERTION
AFTER INSERT ON ADULTE CHECK AGE > 19 ;
```

et :

```
CREATE ASSERTION
AFTER INSERT ON ADULTE CHECK DEGRÉ < 10 ;
```

1.2.4 Vérification de non-redondance

"Ensemble de contraintes dont l'une au moins peut être déduite des autres."(Gardarin, 2013)

Exemple :

- Les deux contraintes d'intégrités suivantes sont redondantes, car la deuxième est déduite de la première :

```
CREATE ASSERTION
AFTER INSERT ON ADULTE CHECK AGE > 19 ;
```

et :

```
CREATE ASSERTION
AFTER INSERT ON ADULTE CHECK DEGRÉ > 10 ;
```

1.1.4 Définition de contraintes d'intégrité en SQL 1

SQL 1 permet la création des contraintes au moment de la création des tables, tel que le montre l'exemple ci-dessous

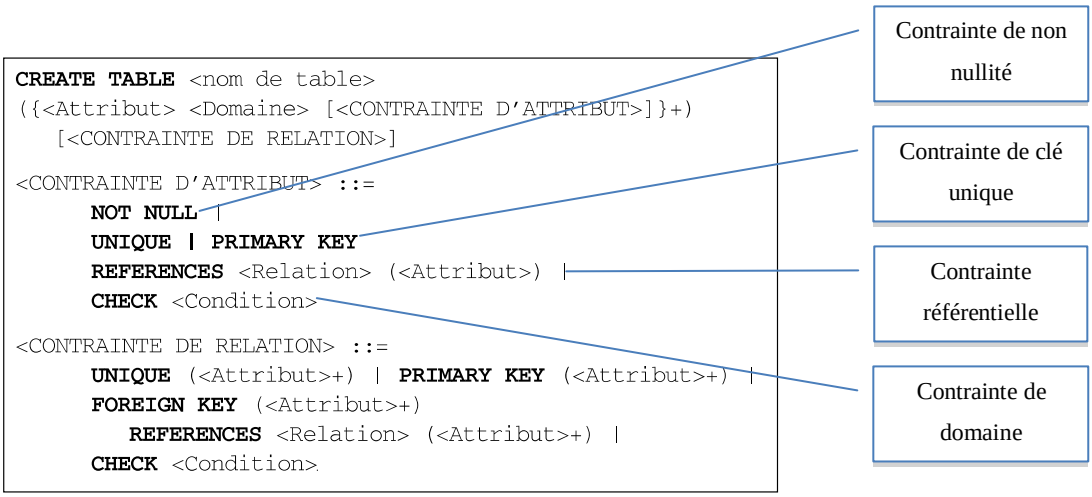

Figure. Création de contraintes d'intégrité en SQL1. (Gardarin, 2013)

Application :

Ici nous considérons trois tables, à savoir, la table ACHATS, la table ARTICLES et la table CLIENTS. La table ACHATS est celle que nous créons et c'est celle qui permet de définir certaines contraintes d'intégrité. On trouve la contrainte de nullité pour les attributs NC et NA, des contraintes référentielles pour l'attribut et pour la relation (table), contrainte de domaine (CHECK…) …etc.

```
CREATE TABLE ACHATS (
NC INT NOT NULL,
NA INT NOT NULL
REFERENCES ARTICLES(NA),
DATE DEC(6) CHECK BETWEEN 010117 AND 311217,
QUANTITE SMALLINT DEFAULT 1,
PRIMARY KEY(NC, NA, DATE),
FOREIGN KEY NB
REFERENCES CLIENTS,
CHECK (QUANTITE BETWEEN 1 AND 100))
```

Figure. Exemple de création de contraintes d'intégrité en SQL1.

1.1.5 Définition de contraintes d'intégrité en SQL 2

SQL 2 offre la possibilité de créer des contraintes d'intégrité tel qu'il est fait en SQL 1 et fournie aussi la possibilité de créer des contraintes d'intégrité en dehors de la création d'une table, grâce à la commande CREATE ASSERTION. Sa syntaxe est la suivante :

```
CREATE ASSERTION <nom de contrainte>
[{BEFORE COMMIT |
AFTER {INSERT|DELETE|UPDATE[OF(Attribut+)]} ON <Relation>}...]
CHECK <Condition>
[FOR [EACH ROW OF] <Relation>]
```

Figure. Création de contraintes d'intégrité en SQL2. (Gardarin, 2013)

Si la clause AFTER est choisie alors cette contrainte d'intégrité va être vérifiée après chaque insertion, suppression ou mise à jours d'un attribut. La clause AFTER peut être remplacée par la clause BEFORE COMMIT, dans ce cas là la vérification s'effectuera après chaque transaction. Par ailleurs, la vérification peut concerner chaque tuple, telle que le montre la figure ci dessus, en utilisant la clause FOR [EACH ROW OF] < nom de la table> ou bien peut concerner toute la table via la clause FOR < nom de la table>.

Application :

On considère les relations ACHATS (NC, NA, DATE, QUANTITE), CLIENTS (NC, nom, adresse), ARTICLES(NA, nom, prix).

1. On crée la contrainte d'intégrité SOMQTEACHETÉ, qui vérifie **après chaque transaction** que la **quantité totale acheté reste inférieure à 150 articles**. La vérification concerne **toute la table** ACHATS

```
CREATE ASSERTION SOMQTEACHETÉ
BEFORE COMMIT
CHECK (SELECT SUM(QUANTITE) FROM ACHATS GROUP BY NC) < 150
FOR ACHATS.
```

Figure. Exemple de création de contraintes d'intégrité en SQL2

2. On veut définir une contrainte d'intégrité qui permet de vérifier que chaque article coûte plus de 100 DA. L'assertion suivante répond à cette requête :

```
CREATE ASSERTION MINPRIX
BEFORE COMMIT
CHECK (SELECT MIN(prix) FROM ARTICLES > 100)
FOR ARTICLES;
```

1.3 Les triggers (les déclencheurs)

Les SGBD classique sont connus par leur caractéristique de passiveté, autrement dit, ces SGBD ne réagissent pas aux évènements et différents changements que connait une base de données, les modifications et les mise à jours s'effectueront d'une manière directes. Par contre, les SGBD actifs, permettent la modification indirecte de certaines informations suite à la modification d'autres données qui lui sont reliées. Pour ce faire, des éléments déclencheurs sont implantés dans la base de données et qui eux ont la capacité d'effectuer les changements nécessaires aux moments opportuns. Ans cette section nous allons étudier les triggers, ou les déclencheurs, qui sont les acteurs principaux de modifications actives dans les bases de données implantées dans les SGBD actifs.

1.1.6 Définitions

1.3.1 Trigger (Déclencheur)

L'auteur du livre (Gardarin, 2013) définie le trigger comme suit:

"Règle dont l'évaluation de type procédural est déclenchée par un événement, généralement exprimée sous la forme d'un triplet Événement -Condition –Action:
WHEN <Événement> IF <Condition sur BD> THEN <Action sur BD>."

En effet, un déclencheur est un exécuteur d'une action prédéfinie si lors d'une parvenue d'un évènement, certaines conditions sont réunies.

Notons que certains déclencheurs se composent seulement d'un événement et d'une action.

La notion de déclencheur, ou trigger n'est fournie que par SQL 3. De ce fait, on n'en a pas de représentation SQL 2 et SQL 1.

1.3.2 Evénement

En s'appuyant sur le livre (Gardarin, 2013) un événement se définie comme suit : *« Signal instantané apparaissant à un point spécifique dans le temps, externe ou interne, détecté par le SGBD.»*

Deux types d'évènements existent, à savoir, les évènements simples, ou primitifs, et les évènements composés.

Les **évènements primitifs** sont principalement :

- L'appel d'une modification de données via les commandes : BEFORE UPDATE, BEFORE DELETE ET BEFORE INSERT.
- La terminaison d'une modification de données via les commandes : AFTER UPDATE, AFTER DELETE ET AFTER INSERT.
- L'appel et la fin d'une recherche de données : BEFORE SELECT ET AFTER SELECT.
- Le début, la validation et l'annulation d'une transaction via les commandes : BEGIN, COMMIT ET ABORT.
- Un évènement temporel, via les commandes : At Time et In Time.
- Un évènement utilisateur, avant et après l'appel d'une procédure via les commandes BEFORE<PROCEDURE> ET AFTER <PROCEDURE>.

Par ailleurs, on définie les évènements composés comme un mélange d'évènements simples.

1.3.3 Condition

C'est une expression logique, pouvant avoir comme valeur vrai ou faux.

1.3.4 Action

C'est la modification à apporter sur l'information dans le cas où la condition est vérifiée, c'est à dire, elle a vrai comme valeur.

1.1.7 Expression en SQL3

1.3.5 Syntaxe

Rappelons que la définition des triggers ne se fait en SQL 1 et SQL 2. En revanche, SQL 3 offre la possibilité de définir des triggers. La syntaxe relative à la création d'un trigger est la suivante :

```
CREATE TRIGGER <nom>
// événement (avec paramètres)
{BEFORE | AFTER | INSTEAD OF}
{INSERT | DELETE | UPDATE [OF <liste de colonnes>]}
ON <table> [ORDER <valeur>]
[REFERENCING{NEW|OLD|NEW_TABLE|OLD_TABLE} AS <nom>]…
// condition
(WHEN (<condition de recherche SQL>)
// action
<Procédure SQL3>
// granularité
[FOR EACH {ROW | STATEMENT}])
```

L'analyse de cette syntaxe nous informe que l'événement est une mise à jours qui peut survenir, avant, après ou à la place d'une modification. La modification peut être une insertion d'un tuple dans une table, sa suppression ou sa mise à jours. La commande [OF <liste de colonnes>] est utilisée dans le cas où la modification touche seulement un ensemble des colonnes et non pas toute la table.

La valeur de la dernière ligne modifiée par l'événement est stockée dans une nouvelle variable d'identificateur <*nom*> créée grâce à la commande *REFERENCING NEW AS* <*nom*>. La valeur de cette ligne avant sa modification par l'évènement est stockée dans une nouvelle variable d'identificateur <nom> créée grâce à la commande *REFERENCING OLD AS* <*nom*>. L'identificateur <*nom*> est choisi par l'administrateur de la base de données.

L'exécution du déclencheur s'effectue soit pour chaque ligne modifiée (FOR EACH ROW), soit une fois pour la requête (FOR EACH STATEMENT).

1.3.6 Exemple d'application

1. **Contrôle d'intégrité temporelle :** Création d'un trigger pour vérification de la saisie d'un salaire

```
CREATE TRIGGER SalaireCroissant

// événement (avec paramètres)
BEFORE UPDATE OF Salaire ON EMPLOYE
REFERENCING OLD AS O, NEW AS N

// condition
 (WHEN O.Salaire > N.Salaire

// action
SIGNAL.SQLState '7005' ("Les salaires ne peuvent décroître")

// granularité
FOR EACH ROW);
```

1.4 Les Vues

1.1.8 Définition

Une vue est une table virtuelle conçue via une requête d'interrogation d'une table d'une base de données. Son contenu est dynamique, autrement dit, aucun stockage d'information n'est effectué sur le disque dur, son contenu est plutôt calculé à chaque interrogation. Une vue permet à l'utilisateur de voir l'information selon ou plus proche de son besoin. De plus, elle permet de garantir l'intégrité de la base de données en évitant les mises à jour directes sur les tables de la base de données. Sans oublier le volet sécurité, où l'utilisateur de la base de données ne pourra voir et accéder qu'aux données dont il possède le droit d'accès, il ne pourra donc voir que la partie de la table ou des tables qui lui sont autorisées de consulter et/ou modifier.

1.1.9 Syntaxe d'une vue

La syntaxe d'une vue est la suivante :

```
CREATE VIEW <nom_vue>(<liste d'attributs>)
        AS
        <requête >
        [CHECK OPTION]
```

Exemples :

1- Créer une vue d'une table

```
CREATE VIEW VueEmployee
        AS SELECT *
        FROM employee
```

2- Créer une vue pour les employés de la faculté centrale d'Alger

```
CREATE VIEW VueEmployee
        AS SELECT enum, ename
        FROM employee
        WHERE address="02 rue Didouche Mourad, Alger"
```

3- Créer une vue pour les employés de la faculté centrale d'Alger avec changement de noms d'attributs

```
CREATE VIEW VueEmployee(num,nom)
        AS SELECT enum, ename
        FROM employee
        WHERE address="02 rue Didouche Mourad, Alger"
```

4- Créer une vue pour une catégorie de clients d'une table client : représentation limité d'une table

```
CREATE VIEW client_c1 (nom, adresse, localite, compte)
        AS SELECT nomClient, adresse, localite, compte
    FROM client
        WHERE categorie = 'C1'
```

1.1.10 Utilisation des vues pour l'interrogation

1.4.1 Consultation

L'utilisateur de la base de donnée considèrera l'interrogation de la base de données au travers les vues comme l'interrogation classique effectuée avec des tables de la base.

Exemple :

Rappelons l'exemple de création de la vue client client_c1 citée dans la section ci dessus :

```
CREATE VIEW client_c1 (nom, adresse, localite, compte)
        AS SELECT nomClient, adresse, localite, compte
    FROM client
    WHERE categorie = 'C1'
```

La consultation de cette vue s'effectue comme suit :

```
SELECT *
FROM client_c1
```

1.4.2 Mise à jour

La mise à jours des vues est un processus très délicat. En fait, Il n'est pas évident de mettre à jour un tuple d'une vue s'il ne correspond pas à un et un seul tuple des tables de la base de données.

La solution triviale à ce problème est d'instaurer une restriction quant à l'ensemble des vues mettables à jours. En effet, ce ne sont pas toute les vues qui sont mettable à jour avec SQL, certaines conditions s''imposent, notamment :

1. La vue est définie comme une sélection (pas d'union, ni jointure)
2. DISTINCT dans le SELECT, GROUP BYet HAVING ne doivent pas figurer dans la vue.
3. "Les colonnes modifiées dans la vue doivent faire référence directement aux données sous-jacentes se trouvant dans les colonnes des tables. Elles ne peuvent être dérivées de quelque autre façon, telle que :
    a. d'une fonction d'agrégation (AVG, COUNT, SUM, MIN, MAX, GROUPING) ;
    b. d'un calcul ; la colonne ne peut pas être calculée à partir d'une expression qui utilise d'autres colonnes." Réf.https://msdn.microsoft.com/fr-fr/library/ms180800(v=sql.90).aspx
4. La clause FROM doit contenir une référence à une seule table
5. Cette table peut être soit une table de base soit une vue qui peut être mise à jour
6. On ne doit pas utiliser la clause WHERE contenant une sous-requête dont la clause FROM fait référence à la table en question

Exemples

La vue suivante est mettable à jour

```
CREATE VIEW client_c1 (nom, adresse, localite, compte)
        AS SELECT nomClient, adresse, localite, compte
    FROM client
    WHERE categorie = 'C1'
```

La vue suivante n'est pas mettable à jours

```
CREATE VIEW Moyenne_Heures_Pilote
        AS SELECT compagnie,
        AVG(nbHVol) moyenne
        FROM Pilote
        GROUP BY compagnie;
```

À savoir que la table Pilote est définie comme suit : Pilote (Matricule, Nom, Prénom, nnHVol, Compagnie). La vue créée est définie comme suit : Moyenne_Heures_Pilote (compagnie, moyenne). Cette vue contient une clause GROUP BY et un attribut obtenu par calcul(moyenne). Par conséquent, cette vue devient non mettable à jour.

1.4.3 Suppression d'une vue

La suppression d'une vue se fait via la commande DROP <nom de la vue>

Exemples

1. DROP client_c1
2. DROP Moyenne_Heures_Pilote

1.1.11 Matérialisation des vues.

Parmi les avantages qu'offre l'utilisation des vues dans une base de données est l'accélération des différentes opérations effectuées sur la base de données étant donné qu'elle permet de fournir une vue partielle de la table interrogée et de fournir une réponse sur mesure à la requête de l'utilisateur. Toutefois, la vue étant virtuelle, elle sera calculée à chaque sollicitation et ne sera nullement stockée sur le disque dur. De ce fait, une vue fréquemment utilisée consommera des ressources considérables en raison des calculs effectués, et sa performance sera mise en cause du moment qu'elle va freiner la rapidité d'exécution du système de gestion de la base de donnée. À partir de là, on a pensé à une solution efficace à ce problème, notamment, la matérialisation de la vue.

La matérialisation d'une vue consiste à la sauvegarde physique d'une vue dans une table. Ces vues matérialisées jouent un rôle particulièrement important dans les systèmes de gestion des bases de données, dans la mesure où elles réduisent le temps de réponse à des requêtes très souvent complexes.

Notons que ce ne sont pas toutes les vues qui seront matérialisées, mais certaines seulement.

Pourquoi on ne peut pas matérialiser toutes les vues ?

La raison est qu'on dispose d'un espace de stockage limité et de capacités de calcul finies. De plus, on doit garder la base de données à jour.

Quels sont les critères de sélection des vues à matérialiser?

Des critères de sélection de vue à matérialiser existent pour choisir celles qui seront matérialisées. Généralement, on matérialise les vues

1. qui sont fréquemment utilisées;
2. dont le volume total est petit ;
3. dont le calcul est dispendieux.

Quelles sont les solutions de sélection de vues à matérialiser ?

Selon les auteurs de l'article "Sélection de vues par matérialisation dans les dépôts de donné" (Samiha Brahimi & Kholladi Mohamed-Khireddine, MISC Lab, Mentouri University of Constantine)

Les algorithmes de sélection des vues à matérialiser se dévisent en deux catégories, à savoir :

1. Algorithmes dirigés par la contrainte d'espace.
2. Algorithmes dirigés par le temps de maintenance [01].

Les même auteurs rapportent que trois solutions existent dans la revue de littérature pour le traitement de cette problématique qui est la sélection des vues à matérialiser, notamment, les algorithmes de glouton , les algorithmes génétiques et l'assimilation du problème de la sélection des vues à matérialiser au problème du sac à dos

1.5 Conclusion

Ce chapitre a résumé certaines notions importantes pour la gestion de bases de données, à savoir, les contraintes d'intégrités, les trigger et les vues en SQL. Pour la partie « contraintes d'intégrité » les deux types, structurelles et non structurelles ont été présentés

avec exemples pour faciliter leur compréhension. Les trigger ont été entamé par la suite afin de les définir, d'illustrer leur mode d'exécution ainsi que d'enseigner la façon de créer un trigger en SQL3 via des exemples détaillés. La partie finale consiste à l'étude des vues en SQL à travers la présentation de leur définition, syntaxe, leur utilisation pour l'interrogation de la base de données, ainsi que leur matérialisation. Une série d'exercices a été incluse à la fin de ce chapitre pour faciliter la compréhension de la matière enseignée.

Le chapitre suivant traitera La conception et l'optimisation du schéma relationnel.

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

1. Livre Gardarin
2. Cours LOG660 - Bases de données de haute performance, ÉTS de Montréal