BASES DE DONNEES ET SQL

Enseignant: Robert GNENESSIO

Cel : 49 74 93 22 e_mail robert_gnenessio@yahoo.com

I. INTRODUCTION

1. Introduction aux bases de données

1.1. Qu'est-ce qu'une base de données?

1.1.1. Définition

Définition 1.1 Un ensemble structuré et organisé permettant le stockage de grandes quantités d'informations afin d'en faciliter l'exploitation (ajout, mise à jour, recherche de données).

Définition 1.2 Une base de données informatisée est un ensemble structuré de données enregistrées sur des supports accessibles par l'ordinateur, représentant des informations du monde réel et pouvant être interrogées et mises à jour par une communauté d'utilisateurs.

1.1.2. Modèles de base de données

Modèle hiérarchique : Une base de données hiérarchique est une forme de système de gestion de base de données qui lie des enregistrements dans une structure arborescente de façon à ce que chaque enregistrement n'ait qu'un seul possesseur (par exemple, une paire de chaussures n'appartient qu'à une seule personne).

I. INTRODUCTION

1. Introduction aux bases de données

1.1. Qu'est-ce qu'une base de données?

1.1.2. Modèles de base de données

Modèle réseau : Il peut établir des liaisons de type n-n, les liens entre objets pouvant exister sans restriction. Pour retrouver une donnée dans une telle modélisation, il faut connaître le chemin d'accès (les liens) ce qui rend les programmes dépendants de la structure de données.

Modèle relationnel : Une base de données relationnelle est une base de données structurée suivant les principes de l'algèbre relationnelle.

1.2. Système de gestion de base de données (SGBD)

1.2.1. Principes de fonctionnement

La gestion et l'accès à une base de données sont assurés par un ensemble de programmes qui constituent le Système de gestion de base de données (SGBD). Un SGBD doit permettre l'ajout, la modification et la recherche de données.

l'opération interrompue

I. INTRODUCTION

1. Introduction aux bases de données

1.2. Système de gestion de base de données (SGBD)

1.2.1. Principes de fonctionnement

Actuellement, la plupart des SGBD fonctionnent selon un mode client/serveur. Le serveur (sous entendu la machine qui stocke les données) reçoit des requêtes de plusieurs clients et ceci de manière concurrente. Le serveur analyse la requête, la traite et retourne le résultat au client.

1.2.2. Objectifs

Les objectifs sont les suivants :

Indépendance physique : La façon dont les données sont définies doit être indépendante des structures de stockage utilisées.

Indépendance logique : Un même ensemble de données peut être vu différemment par des utilisateurs différents.

Accès aux données : L'accès aux données se fait par l'intermédiaire d'un Langage de Manipulation de Données (LMD). Le LMD doit donc être optimisé, minimiser le nombre d'accès disques, et tout cela de façon totalement transparente pour l'utilisateur.

1. Introduction aux bases de données

1.2. Système de gestion de base de données (SGBD)

1.2.2. Objectifs

Les objectifs sont les suivants :

Administration centralisée des données (intégration) : Toutes les données doivent être centralisées dans un réservoir unique commun à toutes les applications.

Non redondance des données : Afin d'éviter les problèmes lors des mises à jour, chaque donnée ne doit être présente qu'une seule fois dans la base.

Cohérence des données : Les données sont soumises à un certain nombre de contraintes d'intégrité qui définissent un état cohérent de la base.

Partage des données : Il s'agit de permettre à plusieurs utilisateurs d'accéder aux mêmes données au même moment de manière transparente.

Sécurité des données : Les données doivent pouvoir être protégées contre les accès non autorisés.

Résistance aux pannes : Que se passe-t-il si une panne survient au milieu d'une modification, si certains fichiers contenant les données deviennent illisibles?

I. INTRODUCTION

1. Introduction aux bases de données

1.2. Système de gestion de base de données (SGBD)

1.2.3. Les grandes fonctions du SGBD

- *Définition et description des données*
- *Mémorisation des données*
- *Traitement de données*
- *Accès et diffusion des données*

1. Introduction aux bases de données

1.3. Système de gestion de base de données (SGBD)

1.2.4. Niveaux de description des données ANSI/SPARC

Le niveau externe **correspond à la perception de tout ou partie de la base par un groupe donné d'utilisateurs, indépendamment des autres. On appelle cette description le schéma externe ou vue.**

Le niveau conceptuel **décrit la structure de toutes les données de la base, leurs propriétés (i.e. les relations qui existent entre elles : leur sémantique inhérente), sans se soucier de l'implémentation physique ni de la façon dont chaque groupe de travail voudra s'en servir.**

Le niveau interne ou physique **s'appuie sur un système de gestion de fichiers pour définir la politique de stockage ainsi que le placement des données.**

1. Introduction aux bases de données

1.3. Système de gestion de base de données (SGBD)

1.2.4. Quelques SGBD connus et utilisés

Il existe de nombreux systèmes de gestion de bases de données, en voici une liste non exhaustive :

- ***PostgreSQL***
- ***MySQL***
- ***Oracle***
- ***IBM DB2***
- ***Microsoft SQL***
- ***Sybase***
- ***Informix***
- ***Accès***

II. CONCEPTION DES BASES DE DONNÉES

Voir Merise

III. LE LANGAGE SQL

3.1. Introduction

Le langage SQL (Structured Query Language) peut être considéré comme le langage d'accès normalisé aux bases de données.
Les instructions SQL sont regroupées en catégories en fonction de leur utilité et des entités manipulées. Nous pouvons distinguer cinq catégories, qui permettent :

1. *La définition des éléments* d'une base de données (tables, colonnes, clés, index, contraintes, ...),
2. *La manipulation des données* (insertion, suppression, modification, extraction, ...),
3. *La gestion des droits d'accès* aux données (acquisition et révocation des droits),
4. *La gestion des transactions*,
5. *Le SQL intégré.*

3.1.1. Langage de définition de données

Les instructions du LDD sont : *CREATE, ALTER, DROP, AUDIT, NOAUDIT, ANALYZE, RENAME, TRUNCATE.*

3.1.2. Langage de manipulation de données

Les instructions du LMD sont : *INSERT, UPDATE, DELETE, SELECT, EXPLAIN, PLAN, LOCK TABLE*

III. LE LANGAGE SQL

3.1. Introduction

3.1.3. Langage de protections d'accès

Les instructions du DCL sont : *GRANT, REVOKE.*

3.1.4. Langage de contrôle de transaction

Les instructions du TCL sont : *COMMIT, SAVEPOINT, ROLLBACK, SET TRANSACTION*

3.1.5. SQL intégré

Le SQL intégré (Embedded SQL) permet d'utiliser SQL dans un langage de troisième génération (C, Java, Cobol, etc.) :

- Déclaration d'objets ou d'instructions;
- Exécution d'instructions;
- Gestion des variables et des curseurs;
- Traitement des erreurs.

Les instructions du SQL intégré sont : *DECLARE, TYPE, DESCRIBE, VAR, CONNECT, PREPARE, EXECUTE, OPEN, FETCH, CLOSE, WHENEVER.*

3.2. Langage de définition de données (LDD)

3.2.1. Créer une table : CREATE TABLE

Une table est un ensemble de lignes et de colonnes. La création consiste à définir le nom de ces colonnes, leur type, la valeur par défaut à la création de la ligne (*DEFAULT*) et les règles de gestion s'appliquant à la colonne (*CONSTRAINT*).

La syntaxe de création d'une table simple est la suivante :

CREATE TABLE nom_table (nom_col1 TYPE1, nom_col2 TYPE2, ...)

Quand on crée une table, il faut définir les contraintes d'intégrité que devront respecter les données que l'on mettra dans la table.

Les types de données :

- *INTEGER* : Entiers signés codés sur 4 octets.
- *BIGINT* : Entiers signés codés sur 8 octets.
- *REAL* : 6 chiffres significatifs codés sur 4 octets.
- *DOUBLE* PRECISION : 15 chiffres significatifs codés sur 8 octets.

3.2. Langage de définition de données (LDD)

- *NUMERIC*[(précision, [longueur])] : Données numériques à la fois entières et réelles avec une précision de 1000 chiffres significatifs. longueur précise le nombre maximum de chiffres significatifs et précision donne le nombre maximum de chiffres après la virgule.
- *CHAR*(longueur) : Chaînes de caractères de longueur fixe. longueur de 1 à 255.
- *VARCHAR*(longueur) : Chaînes de caractères de longueur de 0 à 4000.
- *DATE* : Données constituées d'une date.
- *TIMESTAMP* : Données constituées d'une date et d'une heure.
- *BOOLEAN* : Valeurs Booléenne.
- *MONEY* : Valeurs monétaires.
- *TEXT* : Chaînes de caractères de longueur variable.

3.2. Langage de définition de données (LDD)

Création avec Insertion de données On peut insérer des données dans une table lors de sa création par la commande suivante :

CREATE TABLE nom_table [(nom_col1, nom_col2, ...)] AS SELECT ...

Si les types des colonnes ne sont pas spécifiés, ils correspondront à ceux du SELECT. Il en va de même pour les noms des colonnes. Le SELECT peut contenir des fonctions de groupes mais pas d'ORDER BY.

3.2. Langage de définition de données (LDD)

4.2.3. Contraintes d'intégrité

Syntaxe A la création d'une table, les contraintes d'intégrité se déclarent de la façon suivante :

CREATE TABLE nom_table
(nom_col_1 type_1 [CONSTRAINT nom_1_1] contrainte_de_colonne_1_1 [CONSTRAINT nom_1_2]
contrainte_de_colonne_1_2 ... ... [CONSTRAINT nom_1_m] contrainte_de_colonne_2_m,
nom_col_2 type_2 [CONSTRAINT nom_2_1] contrainte_de_colonne_2_1 [CONSTRAINT nom_2_2]
contrainte_de_colonne_2_2 ... ... [CONSTRAINT nom_2_m] contrainte_de_colonne_2_m,
...
nom_col_n type_n [CONSTRAINT nom_n_1] contrainte_de_colonne_n_1 [CONSTRAINT nom_n_2]
contrainte_de_colonne_n_2 ... ... [CONSTRAINT nom_n_m] contrainte_de_colonne_n_m,
[CONSTRAINT nom_1] contrainte_de_table_1, [CONSTRAINT nom_2]
contrainte_de_table_2, ... [CONSTRAINT nom_p] contrainte_de_table_p)

3.2. Langage de définition de données (LDD)

4.2.3. Contraintes d'intégrité

Contraintes de colonne Les différentes contraintes de colonne que l'on peut déclarer sont les suivantes :

NOT NULL ou NULL : Interdit (NOT NULL) ou autorise (NULL) l'insertion de valeur NULL.

UNIQUE : Attribut comme clé secondaire. L'insertion de valeur NULL est toutefois autorisée.

PRIMARY KEY : La clé primaire est unique, elle ne peut apparaître qu'une seule fois dans l'instruction.

REFERENCES table [(colonne)] [ON DELETE CASCADE] : Contrainte d'intégrité référentielle pour l'attribut de la table en cours de définition.

CHECK (condition) : Vérifie lors de l'insertion de n-uplets que l'attribut réalise la condition condition.

DEFAULT valeur : Permet de spécifier la valeur par défaut de l'attribut.

3.2. Langage de définition de données (LDD)

CHECK (condition) : Cette contrainte permet d'exprimer une condition qui doit exister entre plusieurs attributs de la ligne.

ON DELETE CASCADE : Demande la suppression des n-uplets dépendants, dans la table en cours de définition, quand le n-uplet contenant la clé primaire référencée est supprimé dans la table maître.

ON DELETE SET NULL : Demande la mise à NULL.

3.2.4 Supprimer une table

DROP TABLE Supprimer une table revient à éliminer sa structure et toutes les données qu'elle contient. Les index associés sont également supprimés. La syntaxe est la suivante :

DROP TABLE nom_table

3.2.5 Modifier une table

ALTER TABLE nom_table {ADD/MODIFY} ([nom_colonne type [contrainte], ...])

Ajout d'une contrainte de table

ALTER TABLE nom_table ADD [CONSTRAINT nom_contrainte] contrainte

3.2. Langage de définition de données (LDD)

Renommer une colonne

ALTER TABLE nom_table RENAME COLUMN ancien_nom TO nouveau_nom

Renommer une table

ALTER TABLE nom_table RENAME TO nouveau_nom

3.3. Langage de manipulation de données (LMD)

3.3.1. Insertion de n-uplets

La syntaxe est la suivante :

INSERT INTO nom_table(nom_col_1, nom_col_2, ...) VALUES (val_1, val_2, ...)

Si la liste des noms de colonne est omise, la liste des colonnes sera par défaut la liste de l'ensemble des colonnes de la table dans l'ordre de la création de la table.

Il est possible d'insérer dans une table des lignes provenant d'une autre table. La syntaxe est la suivante :

INSERT INTO nom_table(nom_col1, nom_col2, ...) SELECT ...

3.3. Langage de manipulation de données (LMD)

4.3.2 Modification de n-uplets

La commande UPDATE permet de modifier les valeurs d'une ou plusieurs colonnes, dans une ou plusieurs lignes existantes d'une table. La syntaxe est la suivante :

UPDATE nom_table SET nom_col_1 = {expression_1 | (SELECT ...) }, nom_col_2 = {expression_2 | (SELECT ...) }, ... nom_col_n = {expression_n | (SELECT ...) } WHERE predicat

Les valeurs des colonnes nom_col_1, nom_col_2, ..., nom_col_n sont modifiées dans toutes les lignes qui satisfont le prédicat predicat. En l'absence d'une clause WHERE, toutes les lignes sont mises à jour. Les expressions expression_1, expression_2, ..., expression_n peuvent faire référence aux anciennes valeurs de la ligne.

4.3.3 Suppression de n-uplets

La commande DELETE permet de supprimer des lignes d'une table. La syntaxe est la suivante :

DELETE FROM nom_table WHERE predicat

Toutes les lignes pour lesquelles predicat est évalué à vrai sont supprimées. En l'absence de la clause WHERE, toutes les lignes de la table sont supprimées.

3.4. Langage de manipulation de données (LMD): SELECT

4.3.1 Insertion de n-uplets

Permet d'insérer une ligne ou plusieurs lignes dans une table.

La syntaxe est la suivante :

INSERT INTO nom_table(nom_col_1, nom_col_2, ...) VALUES (val_1, val_2, ...)

La liste des noms de colonne est optionnelle. Si elle est omise, la liste des colonnes sera par défaut la liste de l'ensemble des colonnes de la table dans l'ordre de la création de la table.

Si une liste de colonnes est spécifiée, les colonnes ne figurant pas dans la liste auront la valeur NULL.

Il est possible d'insérer dans une table des lignes provenant d'une autre table. La syntaxe est la suivante :

INSERT INTO nom_table(nom_col1, nom_col2, ...) SELECT ...

III. LE LANGAGE SQL

3.4. Langage de manipulation de données (LMD): SELECT

4.3.1 Insertion de n-uplets

4.3.2 Modification de n-uplets

UPDATE La commande UPDATE permet de modifier les valeurs d'une ou plusieurs colonnes, dans une ou plusieurs lignes existantes d'une table. La syntaxe est la suivante :

UPDATE nom_table SET nom_col_1 = {expression_1 | (SELECT ...) }, nom_col_2 = {expression_2 | (SELECT ...) }, ... nom_col_n = {expression_n | (SELECT ...) } WHERE predicat

Les valeurs des colonnes nom_col_1, nom_col_2, ..., nom_col_n sont modifiées dans toutes les lignes qui satisfont le prédicat predicat. En l'absence d'une clause WHERE, toutes les lignes sont mises à jour. Les expressions expression_1, expression_2, ..., expression_n peuvent faire référence aux anciennes valeurs de la ligne.

4.3.3 Suppression de n-uplets : DELETE La commande DELETE permet de supprimer des lignes d'une table. La syntaxe est la suivante :

DELETE FROM nom_table WHERE predicat

Toutes les lignes pour lesquelles predicat est évalué à vrai sont supprimées. En l'absence de clause WHERE, toutes les lignes de la table sont supprimées

III. LE LANGAGE SQL

3.4. Langage de manipulation de données (LMD): SELECT

Exemple de base de données

Ville(Code, Nom)
Client(Code, Nom, Prenoms, Adresse, Contact, #CodeVille)
Matériel(Référence, Désignation, Prix)
Ventes(Numéro, #CodeClient, DateVente)
MaterielVendu(#NumVente, #RefMateriel, Quantité)

Création des tables

Create Table [If not exists] Ville (Code Char(3) Not Null Primary Key, Nom Varchar(50) Not Null)
Create Table [If not exists] Client (Code Char(4) Not Null Primary Key, Nom Varchar(20) Not Null,
Prenoms Varchar(50), Adresse Varchar(80), Contact Varchar(20), CodeVille Char(4) Not Null Rferences Client On Delete Cascade On Update Cascade)
Create Table [If not exists] Matériel (Référence Char(10) Not Null Primary Key, Désignation Varchar(80) Not Null, Prix Numeric Default 0, CHECK (Prix > 0))
Create Table [If not exists] Ventes (Numéro Char(8) Not Null Primary Key, CodeClient Char(4) Not Null References Client On Delete Cascade On Update Cascade, DateVente Date Default 01/01/2018)
Create Table [If not exists] MaterielVendu (NumVente Char(10) Not Null References Vente On Delete Cascade On Update Cascade, RefMateriel Char(8) Not Null References Materiel On Delete Cascade On Update Cascade, Quantité Numeric, Primary Key(NumVente , RefMateriel))

Remarque : *Pour nommer chaque contrainte, on ajoute CONSTRAINT NomContrainte avant la contrainte*

III. LE LANGAGE SQL

3.4. LMD: SELECT

VILLE

Code	Nom
Y01	YAMOUSSOUKRO
G12	GAGNOA
G20	GUIGLO

CLIENT

Code	Nom	Prénoms	Adresse	Contact	ville
YK01	YOUSSOUF	KONE	BP 1093 YAKRO	01 25 01 01	Y01
GR01	GNON	RICHARD	BP 15 GAGNOA	33 01 45 54	G12
KA01	KOUASSI	ASSO	Face place publique	30 64 25 02	G20
KR01	KOTCHI	REMI	BP 155 GAGNOA	47 40 12 28	G12
KA02	KONAN	AMANI	BP 1093 YAKRO	02 01 02 36	Y01

MATERIELS

Référence	Désignation	Prix
HPO4501222	Ordinateur HP Elite X3 Core i7	475,000
DEI4501323	Ordinateur Dell Inspiron	385,000
HPL7521315	Imprimante HP LaserJet HP 1018	68,000
HPJ7521200	Imprimante HP Jet d'encre HP 1510	65,000
EPV9044210	Vidéo Projecteur Epson QLX 200	285,500

VENTES

Numéro	Client	Date
22011801	KR01	22/01/2018
22011802	KA01	22/01/2018
25011801	GR01	25/01/2018
01021820	KA02	01/02/2018
01021822	KA01	01/02/2018
02021808	KA02	02/02/2018
02021812	GR01	02/02/2018

MATERIELSVENDUS

Vente	Materiel	Quantité
22011801	HPO4501222	2
22011801	HPL7521315	2
22011802	HPO4501222	5
22011802	HPL7521315	1
25011801	HPJ7521200	1
01021820	HPL7521315	2
01021820	HPO4501222	5
01021820	DEI4501323	2
01021822	HPJ7521200	1
02021808	HPO4501222	3
02021808	DEI4501323	3
02021808	EPV9044210	1
02021808	HPJ7521200	1
02021812	DEI4501323	2
02021812	HPL7521315	1

III. LE LANGAGE SQL

3.5. LMD: SELECT

3.5.1. Introduction

La commande SELECT constitue, à elle seule, le langage permettant d'interroger une base de données. Elle permet de :

- Sélectionner certaines colonnes d'une table (projection);
- Sélectionner certaines lignes d'une table en fonction de leur contenu (sélection);
- Combiner des informations venant de plusieurs tables (jointure, union, intersection, différence et division);
- Une requête est une combinaison d'opérations portant sur des tables (relations) et dont le résultat est lui-même une table dont l'existence est éphémère (le temps de la requête).

Une requête se présente généralement sous la forme :

*SELECT [ALL | DISTINCT] { * | attribut [, ...] } FROM nom_table [, ...] [WHERE condition]*

- SELECT permet de spécifier les attributs que l'on désire voir apparaître dans la requête; (*) récupère tous les attributs de la table générée par la clause FROM de la requête;
- FROM spécifie les tables sur lesquelles porte la requête;
- WHERE, facultatif, énonce une condition que doivent respecter les n-uplets sélectionnés.

3.5. LMD: SELECT

3.5.1. Introduction

Pour afficher l'ensemble des n-uplets de la table Client:

*SELECT * FROM Client*

La clause *SELECT* permet de réaliser la projection, la clause *FROM* le produit cartésien et la clause *WHERE* la sélection.

Pour spécifier une chaîne de caractères, il faut l'entourer d'apostrophes. Par exemple, pour sélectionner les Clients dont la ville a pour code G12 , on utilise la requête :

*SELECT * FROM Client WHERE CodeVille ='G12'*

Idem pour les dates : '01/01/2005'. Pour représenter une appostrophe dans une chaîne, il faut la dédoubler (exemple : *'l''arbre'*), ou la faire précéder d'un antislash (exemple : *'l\'arbre'*).

III. LE LANGAGE SQL

3.5. LMD: SELECT

3.5.2. Traduction des opérateurs algébriques

Projection Π(A1, …An)(relation)

SELECT DISTINCT A_1, …, A_n FROM relation

DISTINCT permet de ne retenir qu'une occurrence de n-uplet dans le cas où une requête produit plusieurs n-uplets identiques

Sélection σ(prédicat)(relation)

*SELECT * FROM relation WHERE prédicat*

Produit cartésien

*SELECT * FROM relation_1, relation_2*

Equi-jointure relation1 A1, A2relation2

*SELECT * FROM relation_1, relation_2 WHERE relation_1.A_1 = relation_2.A_2*

III. LE LANGAGE SQL

3.5. LMD: SELECT

3.5.3. Syntaxe générale de la commande SELECT

*SELECT [ALL | DISTINCT] { * | expression [AS nom_affiché] } [, ...]*
FROM nom_table [[AS] alias] [, ...]
[WHERE prédicat]
[GROUP BY expression [, ...]]
[HAVING condition [, ...]]
[{UNION | INTERSECT | EXCEPT [ALL]} requête]
[ORDER BY expression [ASC | DESC] [, ...]]

En fait l'ordre SELECT est composé de 7 clauses dont 5 sont optionnelles :
SELECT : Spécifie les attributs que l'on désire voir apparaître dans le résultat
FROM : Spécifie les tables sur lesquelles porte la requête
WHERE : Fltre les n-uplets en imposant une condition à remplir
GROUP BY : Définit des groupes
HAVING : Spécifie un filtre (condition de regroupement des n-uplets) portant sur les résultats.
UNION, INTERSECT et EXCEPT : Effectue des opérations ensemblistes entre plusieurs résultats de requête (i.e. entre plusieurs SELECT).
ORDER BY : Trie les n-uplets du résultat.

3.5. LMD: SELECT

3.5.4. L'opérateur étoile (*)

Il permet de récupérer automatiquement tous les attributs de la table générée par la clause FROM de la requête

*SELECT * FROM Elèves*

3.5.5. Les opérateurs DISTINCT et ALL

Le mot clef DISTINCT permet d'éliminer les doublons dans la réponse. Par exemple, pour afficher la liste des noms, sans doublon, des Elèves, on a la requête :

SELECT DISTINCT Nom FROM Elèves

3.5.6. Les opérations mathématiques de base

Il est possible d'utiliser les opérateurs mathématiques de base (i.e. +, -, * et /). Pour afficher le nom, le prénom et le salaire annuel des salariés, on peut utiliser la requête :

*SELECT Nom, Prénoms, Salaire * 12 FROM Salarié*

III. LE LANGAGE SQL

3.5. LMD: SELECT

3.5.7. L'opérateur *AS*

Permet de renommer une colonne. Pour afficher le nom, le prénom et le salaire annuel des employés, on peut utiliser la requête :

SELECT Nom AS NomElève, Prénoms AS PrénomsElève, Salaire 12 AS Salaire FROM Salarié*

Permet aussi de renommer une table.

FROM nom_de_table AS nouveau_nom
FROM nom_de_table nouveau_nom

*SELECT * FROM nom_de_table_1 AS t1, nom_de_table_2 AS t2 WHERE t1.A_1 = t2.A_2*
*SELECT * FROM nom_table AS t WHERE nom_table.a > 5*

3.5.8. L'opérateur de concaténation

L'opérateur || (double barre verticale) permet de concaténer des champs de type caractères.

*SELECT Nom || ' ' || Prénoms AS Nom , Salaire * 12 AS Salaire FROM Salarié*

3.5. LMD: SELECT

3.5.9. Sous-requête

Les tables mentionnées dans la clause FROM peuvent correspondre à des tables résultant d'une requête, spécifiée entre parenthèses.
Il faut toujours nommer les tables correspondant à des sous-requêtes en utilisant l'opérateur AS:
*SELECT * FROM table_1, table_2*
*SELECT * FROM (SELECT * FROM table_1) AS t1, table_2*

4.5.10. La clause ORDER BY

ORDER BY permet de trier les n-uplets du résultat et sa syntaxe est la suivante :
ORDER BY expression [ASC | DESC] [, ...]
expression désigne soit une colonne, soit une opération mathématique de base.
ASC spécifie l'ordre ascendant et DESC l'ordre descendant du tri. *ASC par défaut.*
Quand plusieurs expressions sont mentionnées, le tri se fait d'abord de la première à la dernière.

III. LE LANGAGE SQL

3.6. LMD: LES OPERATEURS

Opérateurs	Significations
=	Egal
!=	Différent de
<	Inférieur à
<=	Inférieur ou égal à
>	Supérieur à
>=	Supérieur ou égal à
~	Comme défini par l'expression régulière
~*	Comme LIKE mais sans tenir compte de la casse
!~	Non décrit par l'expression régulière
!~*	Comme NOT LIKE mais sans tenir compte de la casse

Expression régulière peut être LIKE ou SIMILAR TO

III. LE LANGAGE SQL

3.6. LMD: LES OPERATEURS

Opérateurs	Significations
expr IS NULL	Test sur l'indétermination de expr
expr IN (expr_1 [, ...])	Comparaison de expr à une liste de valeurs
expr NOT IN (expr_1 [, ...])	Test d'absence d'une liste de valeurs
expr IN (requête)	Même chose, mais la liste de valeurs est le résultat d'une expr
expr NOT IN (requête)	NOT IN (requête) sous-requête qui doit impérativement retourner une table ne contenant qu'une colonne
EXIST (requête)	Vraie si la sous-requête retourne au moins un n-uplet
expr operateur ANY (requête)	Vraie si au moins un n-uplet de la sous-requête vérifie la comparaison « expr opérateur n-uplet »; la sous-requête doit impérativement retourner une table ne contenant qu'une colonne; IN est équivalent à = ANY
expr operateur ALL (requête)	Vraie si tous les n-uplets de la sous-requête vérifient la comparaison « expr opérateur n-uplet »; la sous-requête doit impérativement retourner une table ne contenant qu'une colonne

3.6. LMD: LES OPERATEURS

Si troiuver les motifs situés en début de ligne, on utilise le symbole ^. Pour chercher toutes les chaînes qui commencent par « voiture », on utilise '^voiture'.

Le signe $ (dollar) indique qu'on souhaite trouver les motifs en fin de ligne. Ainsi : 'voiture$' permet de trouver toutes les chaînes finissant par « voiture ».

Le symbole . (point) remplace n'importe quel caractère. Pour trouver toutes les occurrences du motif composé des lettres vo, de trois lettres quelconques, et de la lettre e, on utilise : 'vo...e'. Cette commande permet de trouver des chaînes comme : voyagent, voyage, voyager, voyageur.

III. LE LANGAGE SQL

3.7. LMD: LES JOINTURES

3.7.1. Le produit cartésien

Prenons une opération de jointure entre deux tables R1 et R2 selon une expression logique E. En algèbre relationnelle, cette opération se note :

$R1 \bowtie_E R2$

La jointure est un produit cartésien suivi d'une sélection. Le produit cartésien n'est rien d'autre qu'une jointure dans laquelle l'expression logique E est toujours vraie : $R1 \times R2 = R1 \bowtie_{true} R2$

VILLE

Code	Nom
Y01	YAMOUSSOUKRO
G12	GAGNOA
G20	GUIGLO

CLIENT

Code	Nom	Prénoms	Adresse	Contact	ville
YK01	YOUSSOUF	KONE	BP 1093 YAKRO	01 25 01 01	Y01
GR01	GNON	RICHARD	BP 15 GAGNOA	33 01 45 54	G12
KA01	KOUASSI	ASSO	Face place publique	30 64 25 02	G20
KR01	KOTCHI	REMI	BP 155 GAGNOA	47 40 12 28	G12
KA02	KONAN	AMANI	BP 1093 YAKRO	02 01 02 36	Y01

III. LE LANGAGE SQL

3.7. LMD: LES JOINTURES

3.7.1. Le produit cartésien

VILLE

Code	Nom
Y01	YAMOUSSOUKRO
G12	GAGNOA

CLIENT

Code	Nom	Prénoms	Adresse	Contact	ville
YK01	YOUSSOUF	KONE	BP 1093 YAKRO	01 25 01 01	Y01
GR01	GNON	RICHARD	BP 15 GAGNOA	33 01 45 54	G12
KR01	KOTCHI	REMI	BP 155 GAGNOA	47 40 12 28	G12
KA02	KONAN	AMANI	BP 1093 YAKRO	02 01 02 36	Y01

VILLE | **CLIENT**

Code	Nom	Code	Nom	Prénoms	Adresse	Contact	ville
Y01	YAMOUSSOUKRO	YK01	YOUSSOUF	KONE	BP 1093 YAKRO	01 25 01 01	Y01
Y01	YAMOUSSOUKRO	GR01	GNON	RICHARD	BP 15 GAGNOA	33 01 45 54	G12
Y01	YAMOUSSOUKRO	KR01	KOTCHI	REMI	BP 155 GAGNOA	47 40 12 28	G12
Y01	YAMOUSSOUKRO	KA02	KONAN	AMANI	BP 1093 YAKRO	02 01 02 36	Y01
G12	GAGNOA	YK01	YOUSSOUF	KONE	BP 1093 YAKRO	01 25 01 01	Y01
G12	GAGNOA	GR01	GNON	RICHARD	BP 15 GAGNOA	33 01 45 54	G12
G12	GAGNOA	KR01	KOTCHI	REMI	BP 155 GAGNOA	47 40 12 28	G12
G12	GAGNOA	KA02	KONAN	AMANI	BP 1093 YAKRO	02 01 02 36	Y01

VILLE | **CLIENT**

Code	Nom	Code	Nom	Prénoms	Adresse	Contact	ville
Y01	YAMOUSSOUKRO	YK01	YOUSSOUF	KONE	BP 1093 YAKRO	01 25 01 01	Y01
Y01	YAMOUSSOUKRO	KA02	KONAN	AMANI	BP 1093 YAKRO	02 01 02 36	Y01
G12	GAGNOA	GR01	GNON	RICHARD	BP 15 GAGNOA	33 01 45 54	G12
G12	GAGNOA	KR01	KOTCHI	REMI	BP 155 GAGNOA	47 40 12 28	G12

III. LE LANGAGE SQL

3.7. LMD: LES JOINTURES

3.7.2. Syntaxe générale des jointures

Trois syntaxes possibles de l'expression d'une jointure dans la clause *FROM* :

table_1 { [INNER] { LEFT | RIGHT | FULL } [OUTER] } JOIN table_2 ON predicat [...]

table_1 { [INNER] { LEFT | RIGHT | FULL } [OUTER] } JOIN table_2 USING (colonnes) [...]

table_1 NATURAL { [INNER] { LEFT | RIGHT | FULL } [OUTER] } JOIN table_2 [...]

ON : La clause ON correspond à la condition de jointure la plus générale. Le prédicat *predicat* est une expression logique de la même nature que celle de la clause WHERE décrite avant.

USING : Notation abrégée correspondant à un cas particulier de la clause ON. Les deux tables, doivent posséder toutes les colonnes qui sont mentionnées, en les séparant par des virgules. La condition de jointure sera l'égalité des colonnes au sein de chacune des paires de colonnes.

NATURAL : Il s'agit d'une notation abrégée de la clause USING dans laquelle la liste de colonnes est implicite et correspond à la liste des colonnes communes aux deux tables participant à la jointure. Tout comme dans le cas de la clause USING, les colonnes communes n'apparaissent qu'une fois dans la table résultat.

III. LE LANGAGE SQL

3.7. LMD: LES JOINTURES

3.7.2. Syntaxe générale des jointures

INNER et OUTER : La jointure est interne ou externe. En effet, le comportement par défaut est INNER.

INNER JOIN : La table résultat est constituée de toutes les juxtapositions possibles d'une ligne de la table *table_1* avec une ligne de la table table_2 qui satisfont la condition de jointure.

LEFT OUTER JOIN : Dans un premier temps, une joint ure interne (i.e.detype INNER JOIN) est effectuée. Ensuite, chacune des lignes de la table table_1 qui ne satisfait pas la condition de jointure avec aucune des lignes de la table table_2 (i.e. les lignes de table_1 qui n'apparaissent pas dans la table résultat de la jointure interne) est ajoutée à la table résultats. Les attributs correspondant à la table table_2, pour cette ligne, sont affectés de la valeur NULL. *Ainsi, la table résultat contient au moins autant de lignes que la table table_1.*

RIGHT OUTER JOIN : Même scénario que pour l'opération de jointure de type LEFT OUTER JOIN, mais en inversant les rôles des tables table_1 et table_2.

FULL OUTER JOIN : La jointure externe bilatérale est la combinaison des deux opérations précédentes afin que la table résultat contienne au moins une occurrence de chacune des lignes des deux tables impliquées dans l'opération de jointure.

III. LE LANGAGE SQL

3.8. LMD: LES CLAUSES GROUP BY ET HAVING

3.8.1. Les clauses GROUP BY et HAVING

La syntaxe d'une requête faisant éventuellement intervenir des fonctions d'agrégation, une clause GROUP BY et une clause HAVING est la suivante :

SELECT expression_1, [...,] expression_N [, fonction_agrégation [, ...]]
FROM nom_table [[AS] alias] [, ...]
[WHERE prédicat]
[GROUP BY expression_1, [...,] expression_N]
[HAVING condition_regroupement]

La commande GROUP BY permet de définir des regroupements (i.e. des agrégats) qui sont projetés dans la table résultat (un regroupement correspond à une ligne) et d'effectuer des calculs statistiques, définis par les expressions fonction_agrégation [, ...], pour chacun des regroupements.

La liste d'expressions expression_1, [...,] expression_N correspond généralement à une liste de colonnes colonne_1, [...,] colonne_N.

La liste de colonnes spécifiée derrière la commande SELECT doit être identique à la liste de colonnes de regroupement spécifiée derrière la commande GROUP BY.

III. LE LANGAGE SQL

3.8. LMD: LES CLAUSES GROUP BY ET HAVING

A la place des noms de colonne il est possible de spécifier des opérations mathématiques de base sur les colonnes. Dans ce cas, les regroupements doivent porter sur les mêmes expressions.

Un SELECT avec une clause GROUP BY produit une table résultat comportant une ligne pour chaque groupe.

3.8.2. Les fonctions d'agrégation

AVG([DISTINCT | ALL] expression) : Calcule la moyenne des valeurs de *expression*.
*COUNT(* | [DISTINCT | ALL] expression)* : Compte le nombre de lignes du résultat. Si *expression* est présent, ne compter que les lignes pour lesquelles cette *expression* n'est pas NULL.
MAX([DISTINCT | ALL] expression) : Retourne la plus grande des valeurs de *expression*.
MIN([DISTINCT | ALL] expression) : Retourne la plus petite des valeurs de *expression*.
STDDEV([DISTINCT | ALL] expression) : Calcule l'écart-type des valeurs de *expression*.
SUM([DISTINCT | ALL] expression) : Calcule la somme des valeurs de *expression*.
VARIANCE([DISTINCT | ALL] expression) : Calcule la variance des valeurs de *expression*.

DISTINCT indique à la fonction de groupe de ne prendre en compte que des valeurs distinctes.

3.8. LMD: LES CLAUSES GROUP BY ET HAVING

3.8.2. Les fonctions d'agrégation

DISTINCT indique à la fonction de groupe de ne prendre en compte que des valeurs distinctes. ALL indique à la fonction de groupe de prendre en compte toutes les valeurs, c'est la valeur par défaut. Aucune des fonctions de groupe ne tient compte des valeurs NULL à l'exception de COUNT(*).

Ainsi, SUM(col) est la somme des valeurs non NULL de la colonne col.

De même AVG est la somme des valeurs non NULL divisée par le nombre de valeurs non NULL.

Il est tout à fait possible d'utiliser des fonctions d'agrégation sans clause GROUP BY. Dans ce cas, la clause SELECT ne doit comporter que des fonctions d'agrégation et aucun nom de colonne. Le résultat d'une telle requête ne contient qu'une ligne.

III. LE LANGAGE SQL

3.9. TRAVAUX PRATIQUES

Ville(Code, Nom)
Client(Code, Nom, Prenoms, Adresse, Contact, #CodeVille)
Matériel(Référence, Désignation, Prix)
Ventes(Numéro, #CodeClient, DateVente)
MaterielVendu(#NumVente, #RefMateriel, Quantité)

Création des tables

```
Create Table [If not exists] Ville
(
    Code Char(3) Not Null Primary Key,
    Nom Varchar(50) Not Null
)
```

3.9. TRAVAUX PRATIQUES

```
Create Table [If not exists] Client
(
    Code Char(4) Not Null Primary Key,
    Nom Varchar(20) Not Null,
    Prenoms Varchar(50),
    Adresse Varchar(80),
    Contact Varchar(20),
    CodeVille Char(4) Not Null Rferences Client On Delete Cascade On Update Cascade
)
Create Table [If not exists] Matériel
    (Référence Char(10) Not Null Primary Key,
    Désignation Varchar(80) Not Null,
    Prix Numeric Default 0, CHECK (Prix > 0)
)
```

3.9. TRAVAUX PRATIQUES

```
Create Table [If not exists] Ventes
(
    Numéro Char(8) Not Null Primary Key,
    CodeClient Char(4) Not Null References Client On Delete Cascade On Update Cascade,
    DateVente Date Default 01/01/2018
)
Create Table [If not exists] MaterielVendu
(
    NumVente Char(10) Not Null References Vente On Delete Cascade On Update Cascade,
    RefMateriel Char(8) Not Null References Materiel On Delete Cascade On Update Cascade,
    Quantité Numeric,
    Primary Key(NumVente , RefMateriel )
)
```

3.9. TRAVAUX PRATIQUES

Donner le nombre de fois qu'un client a fait des achats

```
SELECT Numéro, Nom, CodeClient COUNT(*)
FROM Client NATURAL JOIN Vente ON CodeClient GROUP BY CodeClient , Nom;
SELECT Numéro, Nom, CodeClient COUNT(*) AS Nombre
FROM Client, Vente WHERE Code = CodeClient GROUP BY CodeClient , Nom;
```

Pour connaître la date du premier et du dernier achat, on utilise :

```
SELECT Numéro, Nom, CodeClient COUNT(*), MIN(DateVente), MAX(DateVente)
FROM Client NATURAL JOIN Vente GROUP BY CodeClient, Nom;
SELECT Numéro, Nom, CodeClient COUNT(*) AS Nombre, MIN(DateVente),
MAX(DateVente)
FROM Client, Vente WHERE Code = CodeClient GROUP BY CodeClient, Nom;
```

3.9. TRAVAUX PRATIQUES

Pour connaître le nombre total d'achats effectués par les clients de Gagnoa

```
SELECT COUNT(*) AS "Total vente"
FROM Ville V, Client C, Vente Vn
WHERE V.Code = C.CodeVille AND C.Code = Vn.CodeClient
AND V.Nom = 'Gagnoa';
```

La clause HAVING

Il est possible, dans un SELECT comportant une fonction de groupe, de sélectionner certains groupes par la clause HAVING.

Celle-ci se place après la clause GROUP BY.

3.9. TRAVAUX PRATIQUES

Le prédicat dans la clause HAVING suit les mêmes règles de syntaxe qu'un prédicat figurant dans une clause WHERE. Cependant, il ne peut porter que sur des caractéristiques du groupe: fonction d'agrégation ou expression figurant dans la clause GROUP BY.

On peut avoir à la fois le WHERE et le HAVING. WHERE sera d'abord appliquée, puis les groupes seront constitués, les fonctions de groupe seront ensuite évaluées et la clause HAVING sera enfin appliquée.

Exemples Pour connaître le nombre de fois qu'un client a fait des achats en ne s'intéressant qu'aux clients ayant plus de 2 achats, on utilise la requête :

```
SELECT CodeClient, Nom, COUNT(*)
FROM Client , Vente
WHERE Code = CodeClient
GROUP BY CodeClient, Nom
HAVING COUNT(*) > 2;
```

3.9. TRAVAUX PRATIQUES

Si en plus, on ne s'intéresse qu'aux Clients de Gagnoa, il faut ajouter WHERE :

```
SELECT V.Code, Vn.CodeClient, C.Nom, COUNT(*)
FROM Client C, Ville V, Vente Vn
WHERE C.CodeVille =  V.Code AND C.Code = Vn.CodeClient
AND C.Nom = 'Gagnoa'
GROUP BY V.CodeVille, V.Nom
HAVING COUNT(*) > 2;
```

III. LE LANGAGE SQL

3.10. Vues

Les vues sont des tables virtuelles qui « contiennent » le résultat d'une requête SELECT. L'un des intérêts de l'utilisation des vues vient du fait que la vue ne stocke pas les données, mais fait référence à une ou plusieurs tables d'origine à travers une requête SELECT, requête qui est exécutée chaque fois que la vue est référencée. De ce fait, toute modification de données dans les tables d'origine est immédiatement visible dans la vue dès que celle-ci est à nouveau référencée dans une requête. Les utilisations possibles d'une vue sont multiples :

- Cacher aux utilisateurs certaines colonnes ou certaines lignes en mettant à leur disposition des vues de projection ou de sélection. Ceci permet de fournir un niveau de confidentialité et de sécurité supplémentaire.
- Simplifier l'utilisation de tables comportant de nombreuses colonnes, de nombreuses lignes ou des noms complexes, en requêtes créant des vues avec des structures plus simples et des noms plus intelligibles.
- Nommer des fréquemment utilisées pour simplifier et accélérer l'écriture de requête y faisant référence.

III. LE LANGAGE SQL

3.10. Vues

Syntaxe de définition Voici la syntaxe de création d'une vue :
CREATE [OR REPLACE] VIEW nom [(nom_colonne [, ...])] AS requête
CREATE VIEW : définit une nouvelle vue.
CREATE OR REPLACE VIEW : définit une nouvelle vue, ou la remplace si une vue du même nom existe déjà.

Syntaxe de suppression
DROP VIEW nom [, ...] [CASCADE | RESTRICT]
DROP VIEW : Supprime une vue existante. nom : Le nom de la vue à supprimer (qualifié ou non du nom du schéma).
CASCADE : Supprime automatiquement les objets qui dépendent de la vue (comme par exemple d'autres vues).
RESTRICT : Refuse de supprimer la vue si un objet en dépend. Ceci est la valeur par défaut.

4.9.4 Schémas
Les schémas sont des espaces dans lesquels sont référencés des éléments (tables, vues, index...). La notion de schéma est très liée à la notion d'utilisateur ou de groupe d'utilisateurs.
Syntaxe de définition
CREATE SCHEMA nom_schéma
CREATE SCHEMA crée un nouveau schéma dans la base de données en cours. Le nom du