INSTITUT AFRICAIN D'INFORMATIQUE (IAI-TOGO)
Tel : 22 20 47 00 / 22 22 13 70 **e-mail** : iaitogo@iai-togo.com 07 **BP** : 12456 Lomé 07

PRATIQUE SQL | I.A.I-TOGO

IAI-TOGO

ANNEE ACADEMIQUE 2021-2022

LE LANGAGE SQL

1. Introduction

1.1. Langage de requêtes

En effet, les SGBD doivent implémenter deux langages :

- Un Langage de Définition de Données (LDD) qui permet de créer le schéma conceptuel de la base de données, les schémas externe, physique, ...
- Un Langage de Manipulation de Données (LMD) qui permet d'interroger la base de données bien sûr, mais aussi de la modifier (créer, supprimer ou modifier des enregistrements)

1.2. Langages de manipulation formels

Le langage SQL est basé sur des langages formels :

- L'algèbre relationnelle (vue dans le chapitre précédent)
- Le calcul relationnel (qui est un langage basé sur la logique des prédicats)

1.3. Langages de requêtes orientés utilisateur

L'ambition des langages de requêtes est d'être bien plus facile de manipulation que les langages de programmation par l'utilisateur. Plusieurs propositions de langage de requêtes ont été réalisées :

- Structured Query Language (SQL)
- QUEry Language (QUEL)
- Query By Example (QBE)

1.4. Lien avec les langages de programmation

SQL reste le seul langage d'interrogation et de manipulation des bases de données. Il est, en d'autres termes, impossible de manipuler les données contenues dans une base de données relationnelle sans envoyer une requête SQL au SGBD. Cependant SQL n'est pas un langage complet. Pour effectuer des traitements complexes sur les informations, il faut avoir recours à un langage de programmation classique. La solution est alors d'immerger du SQL dans un programme, approche autrement nommée "embedded-SQL". Techniquement, il est possible d'immerger du SQL dans pratiquement tous les langages de programmation (Pascal, C avec Pro*C par exemple, PHP,Java avec JDBC ...).

2. Présentation de SQL : fonctionnalités

2.1. Langage de définition de données

SQL est un langage qui permet :

- la définition de données au format relationnel
- le contrôle des données

IAI-TOGO

2.2. Langage de manipulation de données

SQL est un langage déclaratif. L'utilisateur énonce les propriétés du résultat souhaité mais ne décrit pas les différentes étapes à suivre pour construire ce résultat. Il s'agit donc là d'une démarche orthogonale aux langages de programmation, dits « procéduraux ». SQL est une implémentation de langages formels : il est emprunté à l'algèbre relationnelle et au calcul relationnel de tuples. Ceci garantit à ce langage des fondements théoriques solides.

2.3. Puissance du langage de manipulation

SQL peut être considéré comme de l'algèbre relationnelle à laquelle on a ajouté des fonctions d'agrégation ainsi que les tris.
Une requête SQL (sans fonction ni tri) est donc équivalente à une suite d'opérations de l'algèbre relationnelle.

2.4. Origine

Les premiers prototypes du langage SQL datent de 1974 (langage SEQUEL du prototype de SGBD relationnel SYSTEM/R (74-76), laboratoire de recherche IBM à San José), autrement dit une éternité en informatique. La robustesse des théories sous-jacentes et la simplicité du langage ont assuré sa longévité.

2.5. Normalisation ISO

Un langage normalisé assure que le code respectant la norme peut être compris quel que soit l'outil implémentant cette norme. Un code développé sous Oracle, se limitant à l'utilisation de la norme, peut donc être transposé dans n'importe quel autre SGBD Relationnel. Trois (03) normes concernent SQL :

- La norme SQL1 (1986, 1989) : cette norme fait 100 pages en 1986 puis 120 pages en 1989. Elle contient 94 mots-clés. L'évolution entre 1986 et 1989 concerne un meilleur contrôle de l'intégrité, en particulier l'intégrité référentielle. SQL1 ne prend pas en charge les instructions de mise à jour du schéma, le SQL dynamique dans "Embedded SQL", la description de la métabase. Certains éditeurs de SGBD ont anticipé les évolutions de la norme SQL1 vers SQL2 (Oracle, Ingres).
- La norme SQL2 (1992). Elle compte 520 pages, 224 mots-clés. Cette norme définit trois niveaux : "entry", "intermediate", "full".
    - entry : correction mineure de SQL1, remplacement de SqlCode par SqlState => plus précis, renommage de colonnes résultats, utilisation de mots-clés comme nom de relation, interfaces Embedded SQL pour Ada et C ...
    - intermediate : nouveaux types de données (Date, Time, ...), meilleure contrôle de l'intégrité, support complet de l'algèbre relationnelle (intersect, except, ..), ordres de modification de schéma (alter, drop), SQL dynamique, fonctions de scroll curseur (first, next, last, ..), structure des tables de la métabase, nouveaux jeux de caractères nationaux

IAI-TOGO

  - full : gestion de relations temporaires, support complet de l'intégrité, nouveaux types de données (chaîne de bits, …)
- Une nouvelle norme en préparation SQL3. Elle compte plus de 1000 pages. Cette norme intègre les extensions procédurales de SQL (procédures stockées). Elle tente de définir des fonctionnalités orientées objet (construction de nouveaux types de données, support de l'« héritage » entre tables) et ajoute des fonctionnalités déductives (opérateurs de fermeture transitives) et actives (support de règles déclenchées par des événements BD => mécanisme de "trigger").

Tous les SGBD relationnels devaient être conformes à SQL2 "entry" pour la fin de l'année 1993, à SQL2 "intermediate" pour la fin de l'année 1994. Ces normes actent l'évolution d'un simple langage d'accès à une BD relationnelle (SQL1) vers un langage de programmation BD (SQL3). SQL devient aussi un langage d'échange de données entre SGBD relationnels dans un contexte réparti. L'extension vers l'objet relationnel rend ce langage complexe et finalement non accessible à un utilisateur final, ce qui initialement était un but recherché.

3. Base de données exemple

Voici le schéma de la base de données *COOPERATIVE* qui servira de support pour illustrer les requêtes exprimées tout au long de ce cours :

```
VINS (NV, CRU, MIL, DEG)
VITICULTEURS (NVT, NOM, PRENOM, VILLE)
PRODUCTIONS (NV#, NVT#)
BUVEURS (NB, NOM, PRENOM, VILLE)
COMMANDES (NC, DATECDE, NV#, QTE, NB#)
EXPEDITIONS (NC#, DATEEXP, QTE)
```

4. Définition et contrôle des données

4.1. Introduction

Le langage de définition doit, entre autres, assurer les fonctionnalités suivantes :

- définition des schémas des relations ;
- définition de vues relationnelles ;
- définition de contraintes d'intégrité ;
- définition de droits ;
- validation d'un traitement ;
- définition du placement et des index *(non normalisé => SGBD dépendant !!)*

4.2. Domaines de base

Les domaines utilisés dans les bases de données sont conformes aux types classiquement utilisés dans les langages de programmation, à savoir :

- Entier : INTEGER ;

IAI-TOGO

- Décimal : DECIMAL (m, n) ou NUMBER (m, n) où m est le nombre de chiffres au total et n le nombre de chiffres après la virgule. Ainsi, DECIMAL (4,2) peut représenter des chiffres comme 99,99;
- Réel flottant : FLOAT ;
- Chaîne de caractères : CHAR (n) et VARCHAR(n). Les "CHAR" font systématiquement n caractères (les chaînes plus courtes sont complétées par des espaces, les VARCHAR sont taillés au plus juste dans la limite des n caractères.

Quelques types de bases de données méritent une attention particulière :

- Date : DATE *(dans la norme SQL2 !).* Ce type normalisé très tard possède une manipulation sensible. Il faut bien connaître le format d'expression de la date et sa langue. Le 1er janvier 2020 peut par exemple s'exprimer en '2020-01-01' ou '01-JAN-2020' ou '1/1/20' ...
- Il n'y a pas de type booléen !

Chaque SGBD possède d'autres domaines qui lui sont propres. Ayez bien conscience que toute utilisation d'élément non normalisé rend vos sources dépendant d'un produit !

4.3. Schéma d'une base de données

Il n'y a pas de standard pour la manipulation (création, modification, suppression) d'une base de données. Toutefois, certains SGBD autorisent les syntaxes exemples suivantes :

- Création d'une base de données

```
CREATE DATABASE COOPERATIVE
```

- Modification d'une base de données

```
ALTER DATABASE COOPERATIVE…
```

- Suppression d'une base de données

```
DROP DATABASE COOPERATIVE…
```

4.4. Schéma d'une relation

4.4.1. Création d'un schéma de relation

L'ordre de création de relation minimal suit la syntaxe de l'exemple ci-après :

```
CREATE TABLE vin
  (     nv    INTEGER,
        cru   CHAR(20),
        mil   INTEGER) ;
```

IAI-TOGO

4.4.2. Ajout d'un attribut

```
ALTER TABLE vin
   ADD COLUMN deg INTEGER ;
```

4.4.3. Suppression d'un schéma de relation (norme SQL2 !)

```
DROP TABLE vin ;
```

4.5. Contraintes d'intégrité

4.5.1. Définition

Une contrainte d'intégrité est une règle qui définit la cohérence d'une donnée ou d'un ensemble de données de la BD.

4.5.2. Contraintes définies en SQL1

Très peu de contraintes sont considérées dans la norme SQL 1 :

- non nullité des valeurs d'un attribut
- unicité de la valeur d'un attribut ou d'un groupe d'attributs
- valeur par défaut pour un attribut
- contrainte de domaine
- intégrité référentielle "minimale"

```
CREATE TABLE vin (
        nv    INTEGER UNIQUE NOT NULL,
        cru   CHAR(20),
        mil   INTEGER,
        deg   INTEGER BETWEEN 5 AND 15);
```

4.5.3. Contraintes définies en SQL2

La norme SQL2 prend en charge de nouvelles contraintes d'intégrités, plus complexes. Les plus remarquables sont :

- La contrainte de clé primaire
- Les contraintes d'intégrité référentielles avec gestion des suppressions, des mises à jour en cascade

NB : Nous parcourrons ces contraintes grâce au script qui accompagne ce cours.

IAI-TOGO

- **Se connecter à la base de donner**

mysql -h localhost -u root -p

- **Gestion des utilisateurs**

CREATE USER 'student'@'localhost' IDENTIFIED BY 'mot_de_passe';
GRANT ALL PRIVILEGES ON elevage.* TO 'student'@'localhost';

- **Utilisation d'une base de données**

pour pouvoir agir sur cette base, vous devez d'abord la sélectionner

USE elevage ;

- **liste les tables de la base de données**

SHOW TABLES;

- **liste les colonnes de la table avec leurs caractéristiques**

DESCRIBE Animal;

- **Création de table avec clé primaire :**

```
CREATE TABLE Animal (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    espece VARCHAR(40) NOT NULL,
    sexe CHAR(1),
    date_naissance DATETIME NOT NULL,
    nom VARCHAR(30),
    commentaires TEXT,
    PRIMARY KEY (id)
);
```

- **Définir une clé primaire après création de la** table :

```
ALTER TABLE Commande
ADD CONSTRAINT pk_client PRIMARY KEY (client);
```

- **Supprimer une clé primaire**

```
ALTER TABLE nom_table
DROP PRIMARY KEY
```

- **DECLARATION CLE ETRANGERE**

```
CREATE TABLE Commande (
    numero INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
```

IAI-TOGO

```
    client INT UNSIGNED NOT NULL,
    produit VARCHAR(40),
    quantite SMALLINT DEFAULT 1,
    CONSTRAINT fk_client_numero          -- On donne un nom à notre clé
      FOREIGN KEY (client)               -- Colonne sur laquelle on crée la clé
      REFERENCES Client(numero)          -- Colonne de référence
);
```

- **Après création de la table**

```
ALTER TABLE Commande
ADD CONSTRAINT fk_client_numero FOREIGN KEY (client)
REFERENCES Client(numero);
```

- **Ajouter une colonne**

```
ALTER TABLE nom_table
ADD [COLUMN] nom_colonne description_colonne;
```

- **Suppression une colonne**

```
ALTER TABLE Test_tuto
DROP COLUMN nom_colonne ;
```

- **Changement du nom de la colonne**

```
ALTER TABLE nom_table
CHANGE ancien_nom nouveau_nom description_colonne;
```

- **Changement du type de données**

```
ALTER TABLE nom_table
CHANGE ancien_nom nouveau_nom nouvelle_description;
ou
ALTER TABLE nom_table
MODIFY nom_colonne nouvelle_description;
```

- **Insérez des données**

```
INSERT INTO vin
VALUES (1, 'test', 75, 5);
```

- **Insertion en précisant les colonnes**

```
INSERT INTO vin (cru, mil, deg)
  VALUES ('test', 75, 5);
```

IAI-TOGO

- **Insertion multiple en précisant les colonnes**

```
INSERT INTO vin (cru, mil, deg)
    VALUES ('test', 75, 5),
                ('DOMAINE', 50, 8),
                ('SANGRIA', 70, 5),
                ('VIGNON', 100, 5);
```

- **Autre syntaxe d'insertion en précisant les colonnes**

```
INSERT INTO vin
SET cru= 'test', mil=75, deg=5;
```

Suppression de données

```
DELETE FROM nom_table
WHERE critères;
```

Mise à jour

```
UPDATE nom_table
SET nom_colonne=nouvelle valeur
Where id=valeur_identifiant
```

Vider la table

```
TRUNCATE TABLE `table`
```

IAI-TOGO

5. Langage d'Interrogation

5.1. Syntaxe générale

```
SELECT <liste d'attributs projetés>
FROM <liste de relations>
WHERE <liste de critères de restriction et de jointure>
```

Les clauses se renseignent dans l'ordre suivant :

1. La partie **FROM** décrit les relations qui sont utilisables dans la requête (c'est-à-dire l'ensemble des attributs que l'on peut utiliser). C'est par là que l'on doit commencer par écrire une requête ;
2. La partie **WHERE** exprime la (les) condition (s) que doivent respecter les attributs d'un tuple pour pouvoir être dans la réponse. Cette partie est optionnelle ;
3. La partie **SELECT** indique le sous-ensemble des attributs qui doivent apparaitre dans la réponse (c'est le schéma de la relation résultat). Bien entendu ces attributs doivent appartenir aux relations indiquées dans la partie FROM.

5.2. Requêtes mono-relation

Les requêtes mono-relation ne concernent par définition qu'une seule relation. Les opérateurs de l'algèbre relationnelle exprimés sont donc la projection et la sélection. Nous vous proposons une série de requêtes avec le résultat de leur exécution illustrant les facettes principales de SQL pour ces deux opérateurs.

1. Donner les vins de cru "Chablis"

```
SELECT nv, mil, deg
FROM vins
WHERE cru = 'Chablis';
```

2. Donner tous les vins

```
SELECT *
FROM vins ;
```

3. Donner les crus des vins de millésime 1985 et de degré supérieur à 9 triés par ordre croissant

```
SELECT cru
FROM vins
WHERE mil=1985 AND deg>9
ORDER BY cru;
```

4. Donner la liste de tous les crus, avec élimination des doubles

```
SELECT DISTINCT cru
FROM vins;
```

IAI-TOGO

Comparer avec la requête suivante

```
SELECT cru
FROM vins;
```

5. Donner les vins de degré compris entre 8 et 12

```
SELECT *
FROM vins
WHERE deg >= 8 AND deg <= 12;
```

```
SELECT *
FROM vins
WHERE deg BETWEEN 8 AND 12;
```

IAI-TOGO

```
SELECT *
FROM VINS
WHERE DEG IN (8, 9, 10, 11, 12);
```

NB : Cette dernière requête ne donnera de bons résultats que si tous les degrés sont des entiers. Mais elle est mise dans ce cours à but pédagogique pour vous faire connaître l'usage de la clause IN.

6. Donner les vins dont le cru commence par la lettre 'B' ou 'b'

```
SELECT *
FROM vins
WHERE cru LIKE 'B%' OR cru LIKE 'b%';
```

5.3. Expression de jointure

Dès lors qu'une requête concerne plusieurs relations, il faut exprimer une jointure afin de « mettre **correctement** en correspondance » les tuples.
A priori la solution est de désigner plusieurs relations dans la clause FROM. Cependant ceci n'exprime pas vraiment la jointure mais le produit cartésien. Le passage du produit cartésien à la jointure se fait en utilisant la clause WHERE. Par exemple, la requête suivante calcule le produit cartésien entre la relation VINS et la relations PRODUCTIONS :

```
SELECT V.*, P.*
FROM vins V, productions P;
```

NB : Les V, P sont appelés des alias. Ils peuvent être utilisés pour ne pas à avoir à réécrire les noms complets des relations à chaque fois qu'on les appelle.

Avez-vous remarqué le schéma de la relation résultat ? Il est constitué de l'union des schémas des relations présentes dans la clause FROM. Les tuples résultats n'ont aucun sens, il faut les filtrer pour obtenir une sémantique correcte. Par exemple, en cherchant les tuples donnant pour chaque numéro de vin, le numéro du producteur. La requête devient alors :

```
SELECT V.nv, P.nvt
FROM vins V, productions P
WHERE V.nv=P.nv ;
```

Avez-vous remarqué la différence de taille des résultats (en nombre de tuples bien sûr) ?

En résumé, une jointure s'exprime généralement en SQL en exprimant dans la clause FROM l'ensemble des relations que l'on veut manipuler et dans la clause WHERE l'expression de jointure (condition que doit vérifier un tuple résultat pour faire partie de la jointure). La clause WHERE peut contenir d'autres conditions si en plus de la jointure on veut rajouter d'autres critères de sélection (dans l'exemple suivant la sélection sur le cru de Bordeaux).

IAI-TOGO

Exemple : Donner les noms des viticulteurs produisant du Bordeaux

```
SELECT DISTINCT NOM
FROM VITICULTEURS VT, VINS V, PRODUCTIONS P
WHERE VT.NVT=P.NVT AND P.NV = V.NV
AND V.CRU = 'Bordeaux' ;
```

5.3.1. Expression de jointure de manière procédurale : les blocs select imbriqués

La même requête que précédemment peut être exprimées avec des blocs SELECT imbriqués :

```
SELECT DISTINCT nom
FROM viticulteurs VT
WHERE nvt IN
   (SELECT nvt
    FROM productions P
    WHERE nv IN
          (SELECT nv
           FROM vins V
           WHERE V.cru = 'Bordeaux'
          )
   ) ;
```

5.3.2. Expression d'une auto-jointure

Définition :

Une auto-jointure est une jointure d'une relation avec elle-même.

L'expression d'un auto-jointure nécessite l'utilisation de synonymes pour les relations.

Exemple : Donner les paires de noms de buveurs habitant la même ville

```
SELECT B1.nom, B1.ville, B2.nom
FROM buveurs  B1, buveurs B2
WHERE B1.ville = B2.ville ;
```

Il y a peut-être beaucoup de réponses n'est-ce pas ? Alors regardons ce que donne cette requête :

```
SELECT B1.NOM, B1.VILLE, B2.NOM
FROM BUVEURS B1, BUVEURS B2
WHERE B1.VILLE = B2.VILLE
AND B1.NB != B2.NB ;
```

Encore un petit effort et l'on va arriver à la bonne réponse !

```
SELECT B1.NOM, B1.VILLE, B2.NOM
FROM BUVEURS B1, BUVEURS B2
WHERE B1.VILLE = B2.VILLE
AND B1.NB > B2.NB ;
```

Syntaxe ANSI SQL2

La norme SQL2 a introduit une nouvelle syntaxe pour la jointure plus proche de l'algèbre relationnelle. Elle permet de définir la jointure directement dans la clause FROM. Cette syntaxe n'est pas forcément supportée par tous les SGBD, néanmoins

IAI-TOGO

elle est assez répandue (Oracle depuis la version 9, MySQL, DB2, SQL Server par exemple).

Voici les différentes formes syntaxiques introduites :

- Produit cartésien :

```
SELECT V.*, P.*
FROM vins V CROSS JOIN productions P ;
```

- Jointure naturelle : elle est utilisable lorsque les deux relations à joindre ont (au moins) un attribut commun de même nom sur lequel doit s'effectuer la jointure

```
SELECT V.nv, P.nvt
FROM vins V NATURAL JOIN productions P ;
```

Si on veut expliciter le ou les attributs communs sur lesquels faire la jointure (lorsque l'on ne veut pas utiliser tous les attributs communs comme expression de jointure) :

```
SELECT V.nv, P.nvt
FROM vins V JOIN productions P USING (nv) ;
```

- Jointure classique : dans ce cas, on explicite la condition de jointure (quelconque) avec une clause ON :

```
SELECT V.nv, P.nvt
FROM vins V JOIN productions P ON (V.nv = P.nv) ;
```

5.4. Opérateurs ensemblistes

Trois opérateurs ensemblistes sont définis dans l'algèbre relationnelle : l'union, l'intersection et la différence. Il faut être très vigilant lors de la manipulation de ces opérateurs : les deux relations en entrée doivent impérativement avoir le même schéma.

5.4.1. UNION (norme SQL1)

L'opérateur UNION élimine automatiquement les doublons.

Exemple : Quels sont les numéros de viticulteurs et de buveurs ?

```
SELECT nvt FROM viticulteurs
UNION
SELECT nb FROM buveurs ;
```

5.4.2. INTERSECT (norme SQL2 !)

Exemple : Quels sont les numéros qui sont à la fois des numéros de viticulteurs et de buveurs ?

```
SELECT nvt FROM viticulteurs
INTERSECT
SELECT nb FROM buveurs ;
```

NB : MySQL n'implémente pas cet opérateur

IAI-TOGO

5.4.3. EXCEPT (norme SQL2 !)

Exemple : Quels sont les numéros de buveurs qui ne correspondent pas à des numéros de viticulteurs ?

```
SELECT nb FROM buveurs
EXCEPT
SELECT nvt FROM viticulteurs;
```

NB : MySQL n'implémente pas cet opérateur. Sous Oracle, cet opérateur se nomme MINUS !
En cas d'ambiguïté, il faut parenthéser les requêtes.

5.5. Fonctions - Agrégats

5.5.1. Définition

Les agrégats sont des fonctions de calcul. Elle s'applique sur l'ensemble des valeurs prises par un attribut et renvoie une valeur unique. Cinq agrégats sont définis : COUNT, SUM, AVG, MIN, MAX. La fonction COUNT (*), un peu particulière, compte les tuples d'une relation.

Exemple : Donner le nombre de tuples de VINS

```
SELECT Count (*)
FROM vins  ;
```

5.5.2. Fonctions dans le SELECT

1. Donner la moyenne des degrés de tous les vins

```
SELECT Avg(DEG)
FROM vins;
```

2. Donner la quantité totale commandée par le buveur Bac"

```
SELECT Sum (qte)
FROM commandes, buveurs
WHERE buveurs.nom = 'Bac' AND buveurs.nb = commandes.nb ;
```

Remarque : Attention à ne pas mélanger dans un SELECT un agrégat avec un attribut classique (par exemple SUM(QTE) avec NOM dans l'exemple précédent). Un agrégat renvoie un résultat unique alors qu'un attribut renvoie un ensemble de résultat. Le seul cas possible d'utilisation d'un tel mélange est lorsqu'il y a une clause ***GROUP BY****.*

5.5.3. Fonctions dans le WHERE

On ne peut utiliser un agrégat directement dans une clause WHERE. Par contre, il peut apparaitre dans la clause SELECT d'une sous-requête et donc appartenir (en un sens) à une clause WHERE.

1. Donner les vins dont le degré est supérieur à la moyenne des degrés de tous les vins

```
SELECT *
FROM vins
WHERE deg > (SELECT Avg (deg) FROM vins);
```

IAI-TOGO

2. Donner les numéros de commande où la quantité commandée a été totalement expédiée

```
SELECT C.nc
FROM    commandes C
WHERE C.qte =
        (SELECT Sum (E.qte)
         FROM expeditions E
         WHERE E.nc = C.nc
        ) ;
```

Attention : Cette requête n'est pas aussi simple qu'elle en a l'air ! En effet, si on regarde de plus près, on s'aperçoit que la sous-requête portant sur EXPEDITIONS utilise dans sa clause WHERE la variable n-uplet C qui est définie dans la requête externe portant sur COMMANDES. La requête externe revient à faire une boucle sur la relation COMMANDES avec la variable C qui prend successivement chaque valeur de COMMANDES et la requête interne traite chaque valeur de C.

Comparons avec la requête suivante :

```
SELECT C.nc
FROM commandes  C
WHERE C.nc = 1 AND C.qte =
                    (SELECT     Sum (E.qte)
                     FROM       expeditions E
                     WHERE      E.nc = 1
                    ) ;
```

Question : Qu'est-ce qu'elle calcule ?

5.6. Partitionnement de relation : la clause GROUP BY

5.6.1. Principe

Le but est de pouvoir appliquer les fonctions ou agrégats de manière dynamique à plusieurs sous-ensembles d'une relation. Le chiffre d'affaires total pour un prestataire de service est une information importante. Mais ce prestataire peut souhaiter analyser plus finement ce chiffre en étudiant le chiffre d'affaires par client. La fonction SUM() devra alors être appliquée autant de fois qu'il y a de clients. La requête doit pouvoir être écrite sans connaître a priori les clients.
Le partitionnement horizontal d'une relation est réalisé selon les valeurs d'un attribut ou d'un groupe d'attributs qui est spécifié dans la clause GROUP BY. La relation est (logiquement) fragmentée en groupes de tuples, où tous les tuples de chaque groupe ont la même valeur pour l'attribut (ou le groupe d'attributs) de partitionnement.
Il est possible alors d'appliquer des fonctions à chaque groupe. Enfin, il est possible d'exprimer l'équivalent des restrictions mais cette fois-ci sur les groupes obtenus. Cela se fait au moyen de la clause HAVING.

5.6.2. Exemples

1. Donner pour le cru Jurançon la moyenne des degrés de ses vins

```
SELECT AVG (deg)
FROM vins
WHERE cru='Jurançon' ;
```

IAI-TOGO

Attention, la requête suivante est syntaxiquement incorrecte (parce que son schéma n'est plus défini conformément au modèle relationnel) :

```
SELECT cru, AVG(deg)
FROM vins
WHERE cru ='Jurançon' ;
```

On se pose la même question mais pour tous les crus et non pas seulement le Jurançon. Si on connait tous les crus de la base de données, on peut lancer une requête par cru sur le modèle précédent. L'inconvénient est qu'il va falloir pas mal de requêtes. Si on ne connait pas les crus, cette solution ne fonctionne pas. Il faut alors absolument recourir au groupement (c'est donc là son principal intérêt) :

```
SELECT cru, AVG (deg)
FROM vins
GROUP BY cru ;
```

Comment se calcule la partition ?
Si on reprend la requête précédente, le calcul peut se détailler selon les étapes suivantes :

- Trier la relation selon les attributs de groupement

Regardons la requête suivante :

```
SELECT cru, deg
FROM vins
ORDER BY cru ;
```

- Créer des sous-relations

Créer des sous-relations (ensemble de tuples ou partitions) pour chaque paquet ayant même valeur sur l'attribut CRU (de façon générale sur l'ensemble des attributs de groupement)

- Appliquer les projections

Appliquer la clause SELECT sur chaque partition (dans notre exemple la valeur de CRU et la moyenne des degrés sur la partition). Par définition, un tuple au plus est construit par partition (s'il y a une clause HAVING toutes les partitions ne sont pas forcément dans la réponse). Le nombre de tuples-résultats est donc majoré par le nombre de partitions.
Cette clause SELECT ne peut être formée que d'un sous ensemble des attributs de groupement ainsi que n'importe quel agrégat exprimé sur la partition.

5.6.3. Restriction sur les groupes

Les clauses WHERE et HAVING ont des rôles différents bien précis.
La clause WHERE permet d'exprimer des restrictions sur les tuples d'une relation. La condition de restriction est donc toujours exprimée sur des valeurs d'attributs. Il n'y a jamais de condition exprimée sur un agrégat dans la clause WHERE !
La clause HAVING permet d'exprimer des restrictions sur les groupes d'une relation obtenus par la clause GROUP BY. La condition de restriction est exprimée sur des

IAI-TOGO

agrégats. Il n'y a jamais d'expression de restriction sur les valeurs des attributs dans la clause HAVING !

Exemple : Donner les crus et les moyennes de degré des vins associés, si aucun vin du cru considéré n'a de degré supérieur ou égal à 12

```
SELECT cru, AVG(deg)
FROM vins
GROUP BY cru
HAVING MAX(deg)< 12 ;
```

Que l'on peut comparer à la requête suivante :

```
SELECT cru, AVG(deg)
FROM vins
WHERE deg<12
GROUP BY cru ;
```

qui donne pour chaque cru la moyenne des degrés des vins de degré inférieur à 12 (elle correspond à la requête initiale mais travaillant non pas sur la relation VINS toute entière, mais seulement les tuples de degré inférieur à 12).

Exemple de requête erronée

```
SELECT cru, nv, AVG(deg)
FROM vins
GROUP BY cru;
```

5.7. Quantificateurs : prédicats ALL, ANY, EXISTS

5.7.1. Prédicat ALL

Le prédicat teste si la valeur d'un attribut satisfait un critère de comparaison avec tous les résultats d'une sous-question.

Exemple : Numéro et nom du (ou des) buveurs ayant fait la plus grosse commande

```
SELECT B.nb, B.nom
FROM buveurs B, commandes C
WHERE B.nb = C.nb
AND C.qte >= ALL ( SELECT qte FROM commandes ) ;
```

Cette requête peut s'écrire sans utiliser de quantificateurs :

```
SELECT B.nb, B.nom
FROM buveurs B, commandes C
WHERE B.nb = C.nb
AND C.qte >= ( SELECT MAX(qte) FROM commandes ) ;
```

5.7.2. Prédicat ANY

Le prédicat ANY teste si la valeur d'un attribut satisfait un critère de comparaison avec au moins un résultat d'une sous-question.

IAI-TOGO

```
SELECT B.nb, B.nom
FROM buveurs B, commandes C
WHERE B.nb = C.nb
AND C.qte > ANY ( SELECT qte FROM commandes );
```

Question : Que calcule cette requête ?

5.7.3. Prédicat d'existence "EXISTS"

Le prédicat EXISTS teste si la réponse à une sous-question est vide.

Exemple : Viticulteurs ayant produit au moins un vin

```
SELECT VT.*
FROM viticulteurs VT
WHERE EXISTS
        (SELECT P.*
         FROM productions P
         WHERE P.nvt = VT.nvt);
```

Cette requête peut également s'écrire sans utiliser de quantificateur :

```
SELECT VT.*
FROM viticulteurs VT
WHERE nvt IN (SELECT nvt FROM productions) ;
```

5.8. Traduction de la division

La division, définie comme opérateur de l'algèbre relationnelle, n'a pas de traduction directe en SQL. Il existe alors deux solutions : le double NOT EXISTS ou le partitionnement.

5.8.1. Division avec double NOT EXISTS

Exemple : Quels sont les viticulteurs ayant produit tous les vins (ceux connus de la base de données) ?

Pour parvenir à exprimer correctement la requête en SQL, l'idée est de paraphraser la requête en français avec une double négation : *"Un viticulteur est sélectionné s'il n'existe aucun vin qui n'ait pas été produit par ce producteur"*

La requête SQL est alors :

```
SELECT VT.*
FROM viticulteurs VT
WHERE NOT EXISTS
        (SELECT V.*
         FROM vins V
         WHERE NOT EXISTS
                    (SELECT P.*
                     FROM productions P
                     WHERE P.nvt=VT.nvt
                     AND P.nv=V.nv )
) ;
```

IAI-TOGO

5.8.2. Division avec partitionnement

L'idée ici est de compter le nombre d'éléments dans un ensemble de référence et de le comparer à chacun des nombres d'éléments pour une valeur donnée. Un viticulteur produit tous les vins si le nombre de vins qu'il produit est égal au nombre de vins dans la relation des vins.

```
SELECT  VT.*
FROM    viticulteurs  VT
WHERE   VT.nvt IN
        (SELECT nvt
         FROM productions
         GROUP BY nvt
         HAVING COUNT(*) =
                    (SELECT COUNT(*) FROM vins)) ;
```

Attention, cette écriture n'est correcte que parce que l'on est sûr (par définition) que l'ensemble des vins produits par un viticulteur est inclus (ou égal) dans l'ensemble des vins de la base de données.

5.9. Synthèse

5.9.1. Expression des conditions de restriction

Condition de recherche :

- Sélection de tuples (clause WHERE) et sélection de groupes (clause HAVING)
- Composition de conditions élémentaires à l'aide des connecteurs logiques : NOT, AND, OR
- Evaluée à Vrai ou Faux (ou Inconnu)

Conditions élémentaires :

- Exprimée à l'aide d'un prédicat
- Evaluée à Vrai ou Faux (ou Inconnu)
- Liste des prédicats SQL
- Comparaison : =, <, <=, >=, >, <> entre un attribut et une valeur ou entre deux attributs
- Intervalle : BETWEEN
- Comparaison de chaîne : LIKE
- Test de nullité : IS NULL
- Appartenance : IN
- Quantification: EXISTS, ANY, ALL

IAI-TOGO

5.9.2. Syntaxe complète d'une requête de recherche SQL

(6) SELECT <liste attributs Aj et/ou expressions sur attributs Ap>	6) *Projection* de l'ensemble obtenu en (5) sur les attributs Aj, avec *calcul des fonctions* appliquées aux *groupes (s'il y en a)*
(1) FROM <liste relations Ri>	(1) *Produit cartésien* des relations Ri
(2) WHERE <condition sur tuples C1>	(2) *Sélection des tuples* de (1) vérifiant C1
(3) GROUP BY <liste attributs Ak>	(3) *Partitionnement* de l'ensemble obtenu en (2) suivant les valeurs des Ak
(4) HAVING <condition sur groupes C2>	(4) *Sélection des groupes* de (3) vérifiant C2
(5) ORDER BY <liste attributs Al>	(5) *Tri des groupes* obtenus en (4) suivant les valeurs des Al

5.9.3. Exemple complet

Question : Donnez par ordre croissant le nom et la somme des quantités commandées par des buveurs bordelais, uniquement si chaque commande est d'une quantité strictement supérieure à 20 litres.

Requête SQL

```
SELECT B.nom, Sum(C.qte)
FROM buveur B, commandes C
WHERE B.nb=C.nb AND B.ville = 'Bordeaux'
GROUP BY B.nb, B.nom
HAVING MIN(C.qte) > 20
ORDER BY B.nom ;
```

Attention, ici on groupe par numéro puis par nom de buveur, parce que la clé de BUVEURS est le numéro (et non pas le nom) et donc que l'on peut rajouter n'importe quel attribut dans la partition après la clé sans changer les partitions construites. Si on ne groupe que par le nom, le résultat peut être erroné (rien ne garantit qu'il n'y a pas deux buveurs de même nom) et si on groupe par numéro seulement, on ne pourra pas sélectionner le nom qui ne sera pas dans les attributs de partitionnement.

6. Langage de mise à jour

Le langage de mise à jour couvre trois fonctionnalités :

- l'insertion de tuples (un seul tuple ou un ensemble de tuples) ;
- la suppression d'un ensemble de tuples ;
- la modification d'un ensemble de tuple.

6.1. Insertion de tuples

6.1.1. Insertion d'un seul tuple

Si tous les attributs sont renseignés, il est possible de ne pas spécifier les noms des attributs. Dans ce cas, il faut renseigner les attributs dans l'ordre de leur création (ordre utilisé au CREATE TABLE).

IAI-TOGO

```
INSERT INTO vins VALUES (100, 'Jurançon', 1979, 12) ;
```

Si quelques attributs seulement sont renseignés, il faut préciser leur nom. Les attributs non renseignés prennent alors la valeur NULL.

```
INSERT INTO vins (nv, cru) VALUES (200, 'Gamay') ;
```

6.1.2. Insertion de plusieurs tuples

Il est possible d'insérer plusieurs tuples dans la même requête.

```
INSERT INTO vins VALUES
   (100, 'Jurançon', 1979, 12),
   (200, 'Gamay', 1989, 22) ;
```

6.1.3. Insertion d'un ensemble de tuples

Il est possible d'insérer tous les tuples résultat d'une requête au moyen d'une seule requête :

```
CREATE TABLE bordeaux
   (nv INTEGER, mil INTEGER, deg INTEGER) ;

INSERT INTO Bordeaux
   SELECT nv, mil, deg
   FROM vins
   WHERE cru= 'Bordeaux' ;
```

6.2. Suppression de tuples

Supprimer tous les tuples de VINS

```
DELETE FROM vins ;
```

Supprimer le vin de numéro 150

```
DELETE FROM vins
WHERE nv = 150 ;
```

Supprimer les vins de degré <9 ou >12

```
DELETE FROM vins
WHERE deg < 9 OR deg > 12 ;
```

Supprimer les commandes passées par Dupond

```
DELETE FROM commandes
WHERE nb IN
   (SELECT nb
    FROM buveurs
    WHERE nom= 'Dupond') ;
```

6.3. Modification des valeurs de tuples

Modifier les valeurs d'un attribut d'un tuple donné.

Exemple : Mettre la ville du viticulteur 150 à Bordeaux

IAI-TOGO

```
UPDATE VITICULTEURS
SET VILLE = 'Bordeaux'
WHERE NVT = 150 ;
```

Modifier les valeurs d'un attribut pour un ensemble de tuples.

Exemple : Augmenter de 10% le degré des Gamay

```
UPDATE VINS
SET DEG = DEG * 1.1
WHERE CRU = 'Gamay' ;
```

Que fait la requête suivante ?

```
UPDATE COMMANDES
SET QTE = QTE+10
WHERE NB IN
   (SELECT     NB
   FROM  BUVEURS
   WHERE NOM='Bac') ;
```

IAI-TOGO

TABLE DES MATIERES

IAI-TOGO

IAI-TOGO