LANGAGE SQL

Introduction aux Bases de Données

Résumé

Cours formé à partir des cours de PeiP (Semestre 4) et de SI3 (Semestre 5)

Antoine Steyer

TABLE DES MATIERES

I. INTRODUCTION

Un langage informatique utilisé pour communiquer avec une base de données.

Il permet d'exprimer des requêtes (Query) pour contrôler toutes les fonctions d'un Système de Gestion de Base de Données (SGBD par la suite).

- ➔ La définition des données (structure, organisation, etc…)
- ➔ La recherche
- ➔ La synthèse
- ➔ La manipulation
- ➔ Le contrôle d'accès
- ➔ Le partage
- ➔ L'intégrité

Ce langage est indépendant d'un "Distributeur / Editeur / Vendeur" de SGBD.

- ➔ Portabilité
- ➔ Norme soutenue par de grands éditeurs

SQL est un langage de haut niveau, proche du langage naturel.

Il se compose de 4 sous-langages :

LID (Langage d'Interrogation des Données)	**LDD** (Description des Données)	**LMD** (Manipulation des Données)	**LCD** (Contrôle des Données et des utilisateurs)	
SELECT	**CREATE** **DROP** **ALTER**	**INSERT** **DELETE** **UPDATE**	**GRANT** **REVOKE**	*Privilèges d'accès aux données*
			CONNECT **COMMIT** **ROLLBACK** **SET**	*Gestion des transactions*

II. EXEMPLE

(Texte souligné : Clés primaires ou étrangères)

FOURNISSEUR (N°F , NomF, Statut, Ville)

PRODUIT (N°P, NomP, Poids, Couleur, PrixHT)

USINE (N°U, NomU, Ville)

LIVRAISON (N°P , N°U , N°F, Qté)

III. LID (INTERROGATION)

III.1 FORMAT D'UNE REQUETE

SELECT liste des attributs sélectionnés -> projection, ne supprime pas les redondances
FROM liste des relations
WHERE conditions à vérifier par les n-uplets -> sélection

Ex : numéro des produits rouge

SELECT N°P
FROM PRODUIT
WHERE Couleur = Rouge

III.2 UN RESULTAT AVEC TRI

Ex : numéro de produits rouge triés

SELECT N°P
FROM PRODUIT
WHERE Couleur = Rouge
ORDER BY N°P **ASC**

Par défaut un ORDER BY est ascendant (ASC), mais il peut être descendant (DESC).

III.3 UN RESULTAT AVEC CALCUL

Ex : numéro Produits, prix HT et prix TTC des produits > 1 tonne
(On suppose la TVA à 20%)

SELECT N°P, prixHT, prixHT * 1,20 **AS** prixTTC -> Renommage
FROM PRODUIT
WHERE Couleur = Rouge

III.4 LES CONDITIONS DE RECHERCHE

- La comparaison : > , < , = , <> (différent de) , <= , >=
- Opérateurs : + , - , * , /
- La condition d'étendue : **BETWEEN**

```
SELECT N°P, prixHT
FROM PRODUIT
WHERE prixHT BETWEEN 1000 AND 3000
```

(Les produits dont le prix est compris entre 1000 et 3000 euros)

- La condition d'appartenance : **IN**

```
SELECT NomF
FROM FOURNISSEUR
WHERE ville IN ('Annecy','Voglans','Lyon')
```

- La condition de correspondance à un modèle : **LIKE**

```
SELECT NomF
FROM FOURNISSEUR
WHERE NomF LIKE 'Simon % SA'
```

% : une séquence de 0 ou plusieurs caractères

_ : 1 caractère

Ex : Simon SA **oui**
SimonSA **non**
Simonin SA **oui**
Simon SARL **non**

Caractère d'échappement : **ESCAPE**

```
WHERE NomP LIKE 'A$% B%' ESCAPE $
```

Ex : A% BC, A% BCB, etc…

Concaténations de chaînes: 'foo'||'bar' a pour valeur 'foobar'

III.5 LES OPERATEURS ENSEMBLISTES

Travaillent sur des relations de même schéma

UNION **MINUS / EXCEPT** **INTERSECT**

Nom des produits qui sont à la fois de couleur rouge et de couleur bleu

(**SELECT** NomP **FROM PRODUIT WHERE** Couleur = 'rouge')
INTERSECT
(**SELECT** NomP **FROM PRODUIT WHERE** Couleur = 'bleu')

(idem pour **UNION**)

Nom des produits qui ne sont pas rouge

(**SELECT** NomP **FROM PRODUIT**)
EXCEPT
(**SELECT** NomP **FROM PRODUIT WHERE** Couleur = 'rouge')

Remarques : Ces requêtes travaillent sur des ensembles et suppriment les redondances

III.6 LES CONDITIONS DE RECHERCHE COMPOSEES

AND, **OR**, **NOT**

Ex : **WHERE** poids < 3000
AND poids > 1000

équivaut à :
BETWEEN 1000 **AND** 3000

III. 7 LES EXPRESSIONS DE JOINTURE

(/!\ Ecriture obsolète / deprecated)

Ex : Liste des produits livrés en indiquant le nom du produit et les villes de leurs fournisseurs

SELECT NomP, ville
FROM FOURNISSEUR F, PRODUIT P, LIVRAISON L
WHERE P.N°P = L.N°P **AND** F.N°F = L.N°F

III.8 FONCTIONS D'AGREGATION

SQL permet d'agréger des données

- **COUNT** (nom attribut) -> Nb de valeurs différentes d'un attribut

  Ex : **SELECT COUNT**(**DISTINCT** Couleur)
  FROM PRODUIT
  (Le nombre de couleurs de produits)

- **COUNT(*)** -> Nb de n-uplets résultats d'une requête

  Ex : **SELECT COUNT**(*)
  FROM PRODUIT
  (Le nombre de produits de la relation PRODUIT)

- **AVG (** attribut **)** -> La moyenne des valeurs de l'attribut
- **MIN**
- **MAX**
- **SUM (** attribut **)** -> la somme des valeurs de l'attribut

  Ex : **SELECT SUM (** poids **)**
  FROM PRODUIT
  WHERE Couleur = 'rouge'
  (La somme des poids des produits rouge)

III.9 REQUETE SUR DES GROUPES DE N-UPLETS

GROUP BY , HAVING

Ex : **SELECT** N°F, **COUNT (DISTINCT** N°P **)**
FROM LIVRAISON
GROUP BY N°F

Combien de produits différents chaque fournisseur a livré, dès lors qu'il en a livré + de 1000 ?

SELECT N°F, **COUNT (DISTINCT** N°P **)**
FROM LIVRAISON
GROUP BY N°F
HAVING SUM (Qte **)** > 1000

HAVING porte sur chaque groupe du GROUP BY.

L'ORDRE DE CALCUL

1. S'il y a une condition **WHERE**, le SGBD commence par éliminer les n-uplets qui ne vérifient pas cette condition

2. Il fait le regroupement des n-uplets suivant l'attribut indiqué dans le **GROUP BY**

3. Il élimine tous les groupes qui ne vérifient pas la condition de la clause **HAVING**

4. Il évalue le résultat du **SELECT**

Remarque :

Dans la relation résultat d'une requête sur les groupes ne peuvent figurer que les fonctions sur les groupes et les attributs du GROUP BY.

Si la requête porte sur des groupes, la fonction d'agrégation s'applique aux groupes.

Ex : Recherche des couleurs de produits dont le prix moyen de vente des produits de ces couleurs est > 1000, ordonnés par couleur.

```
SELECT Couleur, AVG( prixHT )
FROM PRODUIT
GROUP BY Couleur
HAVING AVG( prixHT ) > 100
ORDER BY Couleur
```

III.10 LA DIVISION

L'opérateur de division n'existe pas en SQL.

Ex : Quels sont les entreprises capables de servir tous les rayons ? (Poly 3 pour l'exemple, on possède les tables T_ENTREPOT (VILLE_ETP, RAYON_RYN) et T_RAYON (RAYON_RYN))

Pour le traduire, je cherche :
Les entrepôts tels qu'il n'existe pas de rayon qu'ils ne puissent pas servir.

```
SELECT DISTINCT VILLE_ETP
FROM T_ENTREPOT AS ETP1
WHERE NOT EXISTS
      ( SELECT *
        FROM T_RAYON RYN
        WHERE NOT EXISTS
```

```
( SELECT *
  FROM T_ENTREPOT AS ETP2
  WHERE ETP1.VILLE_ETP = ETP2.VILLE_ETP
  AND (ETP2.RAYON_RYN = RYN.RAYON_RYN) ) )
```

III.11 OPERATEURS SUR LES RELATIONS

Les opérateurs qui s'appliquent aux relations et qui produisent un booléen sont :

- **EXISTS** R : vrai ssi R n'est pas vide
- s **IN** R : vrai ssi s est égal à un des tuples de R
- s **> ALL** R : vrai ssi s est plus grand que toutes les valeurs de la relation unaire R
- s **> ANY** R : vrai ssi s est plus grand qu'au moins une des valeurs de la relation unaire R

On peut aussi utiliser les comparateurs <, <>,<=,>= avec ALL et ANY

Les opérateurs EXISTS,IN, ALL et ANY peuvent être niés:

- **NOT EXISTS** R: vrai ssi R est vide
- **NOT** s **> ANY** R: vrai ssi s est la valeur minimale de R
- s **NOT IN** R : vrai ssi s n'est égal à aucun des tuples de R

IV. LE LDD (DESCRIPTION)

IV.1 CREATION DE TABLE

CREATE TABLE < nom de la relation > (< attributs >*) (* -> Plusieurs)

Ex :

CREATE TABLE FOURNISSEUR (N°F **INT**, NomF **VARCHAR(20)**, Statut **CHAR(7)**, Ville **VARCHAR(20)**)

IV.2 TYPES DE DONNEES

Caractères

- **CHARS**(n) : chaîne de longueur fixe (n caractères) (chaînes plus courtes sont complétées par des blancs)
- **VARCHARS**(n) : chaîne d'au plus n caractères

Numérique

- **INT** (INTEGER) et **SHORT INT**
- **FLOAT** (ou REAL), DOUBLE PRECISION
- **DECIMAL**(n,d) : n chiffres et un point décimal à d positions de la droite (0123.45 sera du type DECIMAL(6,2))

Booléen : **BIT**(n), **BIT VARYING**(n)

IV.3 EXPRESSION DES CONTRAINTES D'INTEGRITE

```
CREATE TABLE LIVRAISON (   N°P  INT  NOT NULL ,   <- Valeur obligatoire
                           N°U  INT  NOT NULL ,
                           N°F  INT  NOT NULL ,
                           PRIMARY KEY ( N°P, N°U, N°F ) ,
                           FOREIGN KEY  N°P REFERENCES PRODUIT.N°P ,
                                        N°U REFERENCES USINE.N°U ,
                                        N°F REFERENCES FOURNISSEUR.N°F CHECK
                                                    (Qte BETWEEN 1 AND 100 000) );
```

Il est possible de créer des tables de relations temporaires.

CREATE TEMP TABLE

IV.4 SUPPRESSION D'UNE RELATION

DROP TABLE < nom relation >

Penser à supprimer les relations temporaires après leur utilisation.

IV.5 DEFINITION DES VUES

Une vue = une relation virtuelle calculée à partir d'une requête sur les relations de la base. Est utile pour donner une vue partielle sur des données de la base.

Ex : **CREATE VIEW** PRODUITS_CHERS (N°P , NomP, PrixHT)
AS (**SELECT** N°P , NomP , PrixHT
FROM PRODUITS
WHERE PrixHT > 1000) ;

IV.6 MODIFIER LA STRUCTURE D'UNE RELATION

Ex : rajouter l'attribut 'produitdangereux?' à la relation PRODUIT

ALTER TABLE < nom relation >
ADD (att1 type1,
att2 type2,
… **)** ;

Va rajouter les attributs à tous les n-uplets existants

Remarque : on peut modifier un attribut existant (délicat)

V. LE LMD (MANIPULATION)

V.1 INSERTION DE N-UPLETS

INSERT INTO < nom relation > **:** < n-uplet >

Insère n-uplet par n-uplet dans la relation.

Ex : **INSERT INTO** PRODUIT **: (** 101 , 'Roulemen' , 50 , 'gris' , 5 **)** ;

Insérer un ensemble de n-uplets :

On a **CATALOGUE (**N°P, nomP, Couleur, Poids, PrixHT**)**

INSERT INTO PRODUIT **: (SELECT** * **FROM** CATALOGUE **) ;**

V.2 SUPPRIMER DES N-UPLETS

DELETE FROM < nom relation > [**WHERE** < condition >] -> [] : optionnel

Ex :

DELETE FROM PRODUIT **WHERE** Poids > 10000 ;

(Supprime les produits de poids supérieur à 10 tonnes)

V.3 METTRE A JOUR

UPDATE < nom relation >
SET nomAtt1 **=** < expression qui définit une valeur pour Att1 >
…
[**WHERE** < condition >] ;

Ex : **UPDATE PRODUIT**
SET Couleur = 'vert d eau'
WHERE Couleur = 'vert' ;

VI. LE LCD (CONTROLE)

Deux clauses : **GRANT … ON** pour donner des droits — **REVOKE … ON** pour supprimer des droits

Ex : **GRANT** ALL_PROVILEGES
ON TABLE PRODUIT
TO Dubois ;

-> / ! \ Dubois pourra tout faire

Ex : **GRANT UPDATE (** PrixHT **)**
ON TABLE PRODUIT
TO Dubois ;

-> Dubois ne pourra que modifier le prixHT

VII. POUR ALLER PLUS LOIN

VII.1 JOINTURES

CROSS JOIN

Correspond au produit cartésien

SELECT < att1, … > **FROM** < nom relation > **CROSS JOIN** < nom relation >

INNER JOIN

Ex : Liste des commandes associées aux utilisateurs

SELECT *
FROM UTILISATEUR U, COMMANDE C
WHERE U.id = C.idUtilisateur -> ici = est obsolète

Solution avec un **INNER JOIN**:

SELECT *
FROM UTILISATEUR
INNER JOIN COMMANDE **ON** UTILISATEUR.id = COMMANDE.idUtilisateur

LEFT / RIGTH / FULL JOIN

Jointure externe, permet de retrouver tous les enregistrements de la table de 'gauche' / 'droite' , même si la condition n'est pas vérifiée.

SELECT *
FROM …
LEFT JOIN … **ON** …

Un **FULL JOIN** remplace les valeurs différentes dans une des tables par null.

VII.2 TRIGGERS

- ➔ Une fonction déclenchée par le SQBD quand une opération précise est faite
- ➔ Base de donnée active
- ➔ Evènement – Condition – Action

Sur ORACLE

EMPLOYE (id **INT** , Nom **VARCHAR** , Salaire **FLOAT**)

CREATE TRIGGER SalaireCroissant
BEFORE UPDATE OF Salaire **ON** EMPLOYE
REFERENCING OLD **AS** O, NEW **AS** N
(**WHEN** O.Salaire > N.Salaire
SIGNAL.SQLState '7005' ("Les salaires ne peuvent pas décroître"))

Sur POSTGRE

CREATE TRIGGER name
{ **BEFORE** | **AFTER** } { event [**OR** …] }
ON table
[**FOR** [**EACH**] { **ROW** | **STATEMENT** }]
EXECUTE PROCEDURE funcname (arguments)

VII.3 INDEX

CREATE [**UNIQUE**] **INDEX** nomindex **ON** table (col1, col2 , …)

Accès rapide aux valeurs d'une colonne

VII.4 LES SEQUENCES

Pseudo attribut permettant de générer automatique un id.

CREATE SEQUENCE nom_sequence
[**INCREMENT BY** entier1]
[**START WITH** entier2]

Ex : **CREATE SEQUENCE** seqdept
INCREMENT BY 10
START WITH 10

- **CURVAL** retourne la valeur courante
- **NEXTVAL** incrémente la séquence et retourne la nouvelle valeur

Ex : **INSERT INTO** dept (dept, nomD)
VALUES (seqdept.NEXTVAL, 'Finances')