Le langage de requêtes SQL

Plan

1. Introduction
2. Définition des données
3. Manipulation des données
4. Recherches
5. Mises à jour
6. Programmation
7. Normalisation et extensions

Introduction

SQL est un langage assertionnel

- L'utilisateur décrit le résultat à obtenir

SQL est non procédural

- C'est le SGBD qui détermine l'enchaînement

3

Normalisation et extensions

SQL est dérivé de l'algèbre relationnelle et de SEQUEL (System R 74)

Il a été intégré à SQL/DS et DB2.

Il existe trois versions normalisées :

- SQL-86 version minimale
- SQL-89 addendum (intégrité)
- SQL-92

Une version SQL-3 étendue est en préparation.

Peu de systèmes supportent SQL2 complet

4

Organisation du langage

SQL comprend quatre parties :

- Le langage de définition de schéma (Tables, Vues, Droits)
- Le langage de manipulation des données (Sélection et mises à jour)
- La spécification de modules appelables (Procédures et contrôles)
- L'intégration aux langages de programmation (Curseurs)

5

Possibilités du langage

- Définition et modification de schémas
- Définition de contraintes d'intégrité
- Définition de vues
- Définition d'autorisations
- Manipulation ensembliste
- Manipulation tuple à tuple
- Contrôle de transactions

6

Commandes de SQL1 - 86

- Définition des données
  - CREATE TABLE
  - CREATE VIEW
  - GRANT
- Manipulation des données
  - SELECT
  - INSERT
  - UPDATE
  - DELETE
- Programmation
  - OPEN
  - FETCH
  - CLOSE
- Contrôle des données
  - COMMIT
  - ROLLBACK

7

Définition des données

- Définition et modification de schéma
- Définition de contraintes d'intégrité
- Définition de vues
- Définition d'autorisations

8

Grammaire d'un langage

On utilise une notation basée sur la BNF (Backus Naur Form)

On définit la grammaire du langage à l'aide un ensemble de règles (clauses)

- Chaque règle a un nom noté entre < >

  <il pleut>

- La définition de la règle est indiquée par ::=

  <il pleut> ::= …

- Une règle peut être constante

  <il pleut> ::= "Il pleut"

- Une règle peut offrir un choix indiqué par |

  <il pleut> ::= "Il pleut" | "Il flotte"

9

Raccourcis syntaxiques

Partie optionnelle notée [ ]

- Définition : A [B] ::= A B | A

  <il pleut> ::= "Il pleut" ["beaucoup"]

Groupement noté { } ou [ ]

- Définition : A { B | C } ::= A B | A C

  <il pleut> ::= "Il" { "pleut" | "flotte" } ["un peu" | "beaucoup"]

Répétition notée { }…

- Définition : A … ::= · | A | A A | A A A | …

  <il pleut> ::= "Il pleut" { "Il pleut" }…

10

Dérivation

- On peut utiliser une règle dans la définition d'une autre règle

  <expr> ::= <terme> | <expr> { + | - } <terme>

  <terme> ::= <fact> | <terme> { * | / } <fact>

  <fact> ::= [+ | -] <prim>

- Les définitions peuvent être récursives

  <prim> ::= <const> | <var> | (<expr>)

Création du schéma

CREATE SCHEMA <clause d'autorisation>
[{ <élément de schéma> }…]

<clause d'autorisation> ::=
AUTHORIZATION <identifiant du propriétaire>

<élément de schéma> ::=
<définition de relation>
| <définition de vue>
| <définition de droit>

Création d'une relation

```
CREATE TABLE <nom de relation>
( <élément de relation> [ { , <élément de relation> }… ] )
```

SQL-86 :

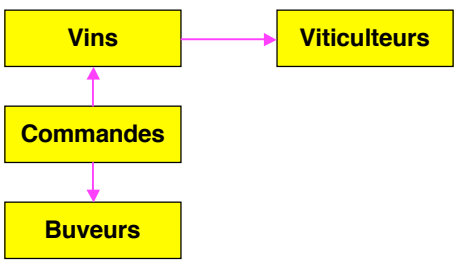
<élément de relation> ::=
    <définition d'attribut> I <définition de contrainte>

<définition d'attribut> ::=
    <nom d'attribut> <type de donnée> [NOT NULL [UNIQUE]]

<définition de contrainte> ::=
    UNIQUE (<nom d'attribut> [{ , <nom d'attribut> }…])

Base de données exemple

VINS (NUMV, CRU, ANNEE, DEGRE, NUMVT, PRIX)
VITICULTEURS (NUMVT, NOM, PRENOM, VILLE)
BUVEURS (NUMB, NOM, PRENOM, VILLE)
COMMANDES (NUMC, NUMV, NUMB, DATE, QTE)



Exemple de création de relation

Les possibilités de SQL-86 sont très limitées

```
CREATE TABLE COMMANDES
    ( NUMC  INTEGER NOT NULL UNIQUE,
      NUMV  INTEGER NOT NULL,
      NUMB  INTEGER NOT NULL,
      DATE  CHARACTER (8) NOT NULL,
      UNIQUE  (NUMV, NUMB, DATE),
      QTE  DECIMAL (15, 2) )
```


Manipulation des données

Manipulation ensembliste

- Utilisation d'une expression logique spécifiant l'ensemble des tuples à insérer, rechercher, modifier ou détruire.

Manipulation tuple à tuple

- Insertion directe d'une valeur de tuple dans une relation.
- Déclaration et utilisation d'un curseur - un pointeur courant - sur un ensemble pour rechercher, modifier, détruire le tuple courant.

Manipulation ensembliste

SELECT : recherche de tuple(s) satisfaisant une condition (expression de qualification)

INSERT : insertion dans une relation de tuple(s) obtenus par recherche dans la base

DELETE : suppression de tuple(s) satisfaisant une condition

UPDATE : modification de tuple(s) satisfaisant une condition

Recherche

La syntaxe de base est fondée sur le bloc

```
SELECT  ...
FROM    ...
WHERE   ...
```

Les blocs peuvent être imbriqués

- Uniquement dans la clause where en SQL1

Au niveau externe des blocs peuvent être liés

- Uniquement par union en SQL1

Syntaxe d'une recherche simple

```
SELECT   [ DISTINCT | ALL ] <liste résultat>
FROM   <liste de références de relation>
[ WHERE   <expression de recherche> ]
```

<liste résultat> ::=
    * | <expression de valeur> [{ , <expression de valeur> }…]
<liste de références de relation> ::=
    <référence de relation> [{ , <référence de relation> }…]

19

Sélection de tuples (restriction)

Q1 : Donner les Beaujolais.

```
SELECT    *
FROM      VINS
WHERE     CRU  = 'Beaujolais'
```

Simplicité et productivité de SQL

20

Sélection d'attributs

Q2 : Donner les différents crus, années et degrés des vins.

```
SELECT   CRU, ANNEE, DEGRE
FROM     VINS
```

21

Utilisation de DISTINCT (projection)

Q3 : Donner les différents crus, années et degrés des vins sans doubles.

```
SELECT   DISTINCT CRU, ANNEE, DEGRE
FROM     VINS
```

22

Sélection sur plusieurs relations (jointure)

Q4 : Donner le nom des viticulteurs avec les crus qu'ils produisent.

```
SELECT    DISTINCT NOM, PRENOM, CRU
FROM      VITICULTEURS, VINS
WHERE     VITICULTEURS.NUMVT = VINS.NUMVT
```


Interprétation algébrique

```
SELECT  <liste des attributs projetés Ai>
FROM  <liste des relations touchées Rj>
WHERE   <expression de recherche E>
```

Expression algébrique équivalente pour une requête simple (en l'absence de sous-requête imbriquée) :

$PROJECTION_{Ai}$ (
  $RESTRICTION_{E}$ (
    PRODUIT (Rj))

Expression de recherche

- Comparaison (restriction ou jointure)
- Recherche par intervalle
- Recherche dans une liste
- Recherche textuelle
- Recherche sur valeur nulle
- Recherche quantifiée

Expression de recherche complexe

Q5 : Donner les numéros des vins de plus de 20 parmi les Beaujolais 80, 82 ou 83 de degré compris entre 11,0 et 12,4 ou alors parmi les vins de vigneron connu.

```
SELECT    NUMV
FROM      VINS
WHERE     PRIX > 20.00
AND  ( (  DEGRE BETWEEN 11.0 AND 12.4
     AND  ANNEE IN (1980, 1982, 1983)
     AND  CRU LIKE '%Beaujolais%'
          )
  OR NUMVT IS NOT NULL
     )
```

Requête sur plusieurs relations

Q6 : Donner les noms des buveurs ayant commandé des Beaujolais.

```
SELECT   DISTINCT BUVEURS.NUMB, NOM,
         PRENOM
FROM     BUVEURS, COMMANDES, VINS
WHERE    BUVEURS.NUMB = COMMANDES.NUMB
AND      COMMANDES.NUMV = VINS.NUMV
AND      CRU  =  'Beaujolais'
```


Utilisation de variables

Q6' : Donner les noms des buveurs ayant commandé des Beaujolais.

```
SELECT   DISTINCT B.NUMB, B.NOM, B.PRENOM
FROM     BUVEURS B, COMMANDES C, VINS V
WHERE    B.NUMB = C.NUMB
AND      C.NUMV = V.NUMV
AND      V.CRU  =  'Beaujolais'
```

Plusieurs variables sur une même relation

Avec le schéma suivant :

EMPLOYE (NUM, NOM, PRENOM, …, RESP, …)

où RESP désigne un autre employé responsable

Q7 : Donner les noms et prénoms des employés dont le salaire est supérieur à celui de leur responsable.

```
SELECT    E.NOM, E.PRENOM
FROM      EMPLOYE E, EMPLOYE R
WHERE     E.RESP = R.NUM
AND       E.SALAIRE > R.SALAIRE
```


Requêtes imbriquées

Q8 : Donner le nom des viticulteurs qui produisent du Juliénas.

```
SELECT    NOM, PRENOM
FROM      VITICULTEURS
WHERE     NUMVT IN (
    SELECT    NUMVT
    FROM      VINS
    WHERE     CRU = 'Juliénas' )
```

Bloc imbriqué constant

Forme intuitive pour requête simple

Limitée à la puissance d'une semi-jointure

Forme "plate" équivalente

Q9 : Donner le nom des viticulteurs qui produisent du Juliénas.

```
SELECT    DISTINCT VT.NUMVT,
          VT. NOM, VT. PRENOM
FROM      VITICULTEURS VT,  VINS V
WHERE     VT.NUMVT  =  V.NUMVT
AND       V.CRU  =  'Juliénas'
```

Attention aux doubles !

31

Interprétation logique

SELECT < liste des attributs projetés V1.A1, …,Vp.Ap >
FROM < liste des relations touchées V1 □ R1, …, Vn □ Rn >
WHERE < expression de recherche Ei (V1, …, Vn) >

EXPRESSION LOGIQUE ÉQUIVALENTE

(Au nombre de doubles près)

{ (V1.A1, …,Vp.Ap) / V1 □ R1, …, Vp □ Rp,
□ Vp+1 □ Rp+1, …, □ Vn □ Rn,
Ei (V1, …, Vn) }

32

Quantificateur existentiel

Q10 : Donner les noms des crus commandés par au moins un buveur.

```
SELECT    DISTINCT CRU
FROM      VINS V
WHERE     EXISTS  (
      SELECT    *
      FROM      COMMANDES C
      WHERE     C.NUMV = V.NUMV  )
```

SQL comprend une expression de quantification existentielle explicite

33

Référence externe à un bloc

Q10 : Donner les noms des crus commandés par au moins un buveur.

```
SELECT    DISTINCT CRU
FROM      VINS V
WHERE     EXISTS  (
      SELECT    *
      FROM      COMMANDES C
      WHERE     C.NUMV = V.NUMV  )
```

Bloc imbriqué fonction de V

34

Forme existentielle implicite

Q10' : Donner les noms des crus commandés par au moins un buveur.

```
SELECT    DISTINCT V.CRU
FROM      VINS V, COMMANDES C
WHERE     C.NUMV = V.NUMV
```

- Dans Q7', la variable C ne figure pas dans le résultat
  - Elle est implicitement préfixée par un quantificateur ∃.
- Un SELECT DISTINCT est équivalent à une expression du calcul de tuple.

{ V.CRU | V ∈ VINS, ∃C ∈ COMMANDES, (C.NUMV = V.NUMV) }

35

Forme existentielle négative

Q11 : Donner les noms des crus qui n'ont été commandés par aucun buveur.

```
SELECT    DISTINCT CRU
FROM      VINS V
WHERE     NOT  EXISTS  (
      SELECT     *
      FROM       COMMANDES C
      WHERE      C.NUMV =   V.NUMV  )
```

36

Quantificateur universel

Q12 : Donner les noms des buveurs qui ont commandé tous les vins de la base.

```
SELECT   NOM, PRENOM
FROM     BUVEURS  B
WHERE    NOT  EXISTS  (
      SELECT     *
      FROM       VINS  V
      WHERE      NOT  EXISTS  (
            SELECT   *
            FROM     COMMANDES  C
            WHERE    B.NUMB = C.NUMB
            AND      C.NUMV = V.NUMV  )  )
```


Fonctions d'agrégation

Q13 :

Donner la moyenne des degrés des Juliénas.

```
SELECT   AVG (DEGRE), COUNT (*)
FROM     VINS
WHERE    CRU = 'Juliénas'
```

SELECT : forme générale

```
SELECT   [ DISTINCT  I  ALL ]  <liste résultat>
FROM <liste de références de relation>
[ WHERE  <expression de recherche> ]
[ GROUP BY  <réf. d'attribut>  [ { , <réf. d'attribut> }… ]
[ HAVING  <expression de recherche> ]
[ ORDER BY <ref colonne> [ { , <ref colonne> }… ]
```

39

Groupement des agrégations

Q14 : Calculer le degré moyen pour chaque cru.

```
SELECT      CRU,  AVG (DEGRE), COUNT (*)
FROM        VINS
GROUP BY    CRU
```

40

Sélection de groupe

Q15 : Calculer le degré moyen pour tous les crus pour lesquels il y a au moins 10 vins.

```
SELECT      CRU, AVG (DEGRE)
FROM        VINS
GROUP BY    CRU
HAVING      COUNT (*) >= 10
```

- Clause WHERE --> sélection des tuples
- Clause HAVING --> sélection des groupes entiers

Expression de valeurs

- Références d'attributs
- Calculs arithmétiques
  + - * /
- Fonctions agrégats
  Compte, somme, moyenne, min et max
- Manipulation de chaînes de caractères
  Expressions régulières

Agrégation et requête imbriquée

Q16 : Donner le nom des buveurs ayant commandé plus de 10 litres d'alcool pur en 1988.

```
SELECT      NOM
FROM        BUVEURS
WHERE       NUMB  IN  (
      SELECT      NUMB
      FROM        COMMANDES, VINS
      WHERE       DATE  LIKE  '%88'
      AND         COMMANDES.NUMV  =  VINS.NUMV
      GROUP BY    NUMB
      HAVING      SUM  (QTE * DEGRE / 100)  >  10 )
```

43

Mises à jour

INSERT : insertion dans une relation de tuple(s) obtenus par recherche dans la base

DELETE : suppression de tuple(s) satisfaisant une condition

UPDATE : modification de tuple(s) satisfaisant une condition

44

Insertion

INSERT INTO <nom de relation> [(<liste d'attributs cibles>)]
{ VALUES (<liste de valeurs à insérer>) | <spécif. de recherche> }

<liste d'attributs cibles> ::=
<nom d'attribut> [{ , <nom d'attribut> }...]
<liste de valeurs à insérer> :=
<valeur à insérer> [{ , <valeur à insérer> }...]
<valeur à insérer> ::=
NULL | <spécif. de valeur>

45

Exemples d'insertion

Insertion (partielle) d'un vin particulier

```
INSERT INTO VINS (NUMV, CRU, ANNEE)
VALUES (112, 'Juliénas', NULL)
```

Les viticulteurs Dijonnais sont des buveurs

```
INSERT INTO BUVEURS
        SELECT *
        FROM VITICULTEURS
        WHERE VILLE = 'Dijon'
```

46

Modification

```
UPDATE  <nom de relation>
SET  <affectation>  [ { , <affectation> }... ]
[ WHERE <expression de recherche> ]
```

<affectation> ::=

<nom d'attribut> = NULL | <expression de valeur>

47

Exemples de modification

Les Juliénas 1994 font 12 degré.

```
UPDATE      VINS
SET         DEGRE = 12
WHERE       CRU = 'Juliénas'
AND         ANNEE = 1994
```

10 % gratuit sur les commandes de Volnay 1993.

```
UPDATE      COMMANDES
SET         QTE = QTE * 1.1
WHERE       NUMV IN (
      SELECT    NUMV
      FROM      VINS
      WHERE     CRU = 'Volnay'
      AND       ANNEE = 1993 )
```

48

Suppression

```
DELETE FROM  <nom de relation>
[ WHERE <expression de recherche> ]
```


Exemple de suppression

Supprimer les commandes de vins de degré inconnu.

```
DELETE FROM  COMMANDES
WHERE    NUMV  IN  (
                SELECT NUMV
                FROM    VINS
                WHERE  DEGRE  IS  NULL  )
```

Programmation

- Les langages relationnels manipulent les données par ensembles
- Les langages de programmations manipulent les données par boucle de programme
- Il y a incompatibilité d'humeur
- Solution SQL les curseurs

Un curseur décrit une position courante dans un ensemble de tuples décrit par une expression de recherche SQL

Curseurs

Un curseur est manipulé par des commandes spécifiques

- DECLARE <nom de curseur>
  CURSOR FOR <expression de recherche
- OPEN <nom de curseur>
- CLOSE <nom de curseur>

Plusieurs commandes permettent d'accéder au tuple courant et de modifier la position du curseur

- FETCH <nom de curseur> INTO .....
- UPDATE <nom de relation> SET .....
  WHERE CURRENT OF <nom de curseur>
- DELETE FROM <nom de relation>
  WHERE CURRENT OF <nom de curseur>

Utilisation de SQL depuis un LP

Programme PL/1-SQL

```
...
EXEC SQL BEGIN DECLARE SECTION ;
  DCL VAR1 CHAR(20) ;
  DCL VAR2 INT ;
EXEC SQL END DECLARE SECTION ;
...
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT ......
  FROM ......
  WHERE ......  :VAR1  ..... ;
...
EXEC SQL OPEN C1 ;
DO WHILE SQLCODE = 0
  BEGIN
    ...
    EXEC SQL FETCH C1 INTO  :VAR2 ;
  END
...
```


Normalisation et extensions

SQL est dérivé de l'algèbre relationnelle et de SEQUEL (System R 74)

Il a été intégré à SQL/DS et DB2.

Il existe trois versions normalisées :

- **SQL-86 version minimale**
- **SQL-89 addendum (intégrité)**
- **SQL-92**

Une version SQL-3 étendue est en préparation.

Peu de systèmes supportent SQL2 complet

SQL2 (SQL-92)

SQL2, trois niveaux sont distingués :

- SQL2 entrée

  SQL-89 + corrections

- SQL2 intermédiaire

  Compléments relationnels

- SQL2 complet

  Gadgets en plus

55

SQL2 intermédiaire

- Type de données dates avec opérateurs correspondants
- Cascade des mises à jour par intégrité référentielle
- Différents alphabets et ordres lexicographiques
- Possibilité de créer des domaines
- Jointure externe
- Union [externe], intersection, différence

56

SQL2 complet

- Extension des dates et du temps
- Expressions étendues avec correspondances de colonnes
- Possibilité de SELECT en argument d'un FROM
- Vues concrètes (implémentées)
- Contraintes d'intégrité multi-tables
- Contrôles d'intégrité différés

57

SQL3

Fonctionnalités objets

- Procédures externes
- Types de données abstraits
- Sous-tables et généralisation

Fonctionnalités déductives

- Union récursive
- Déclencheurs
- Parallélisme
- Requêtes asynchrones au programme

58

Conclusion

Un standard de plus en plus complet et de plus en plus suivi

- Attention aux approximations et imitations fausses
- Tout existe dans les propositions SQL2/3
- Une référence pour implémenter / utiliser chaque aspect des BD

Le langage de communication inter-système

- SQL - CLI (protocole client / serveur)
- RDA (remote data access, protocole BD distante)
- TP (transaction processing, exécution répartie)

Le langage universel sur lequel s'appuient les progiciels