SQL le langage relationnel

Structured Query Language

Définition

- Langage de 4ème génération
- Permet la description, l'interrogation et la manipulation des données
- Deux types d'utilisateurs :
  - les non informaticiens pour la mise en œuvre de questions simples
  - les informaticiens pour la création de BD, le développement de logiciels

Introduction

- 1979 : première version de SQL par Relational Software Inc. (Oracle)
- Standard de fait : norme américaine en 1986 (ANSI X3.135) et standard international en 1987 (ISO 9075) → SQL1 ou SQL/86
- Evolutions réalisées sous l'impulsion de SQL Access Group

Introduction

- Version suivante : SQL2 ou SQL92 → standard ISO/IEC 9075 :1992
- Aujourd'hui : version SQL3 → intégration
  - des concepts objet,
  - des fonctionnalités multimédia,
  - des fonctionnalités d'optimisation des performances,
  - de la gestion de la répartition,
  - de la gestion de grands volumes de données (petabytes = 10^{15} bytes).

Sous-langages

- Langage de Définition des Données (LDD) : permet la création, la modification ou la suppression des objets (tables, index, privilèges) et la gestion des contraintes d'intégrité
- Langage de Manipulation des Données (LMD) : permet la mise à jour de la base (ajout, modification et suppression de données)

Sous-langages

- Langage d'Interrogation des données (LID) : permet la recherche de l'information (représente la réelle mise en œuvre du langage relationnel)
- Langage de Contrôle et d'Administration des Données (LCD) : gère la confidentialité et la cohérence des données.

Exemple de schéma relationnel

Considérons un exemple simplifié d'une base de données d'une université.

Etudiant (N°SS_etu, nom_etu, adr_etu, nb_inscriptions)

Unite-valeur (N°UV, titre, responsable)

Enseignant (N°SS_ens, nom_ens, grade, salaire)

Inscription (N°SS_etu, N°UV, date_inscription)

Enseignement (N°SS_ens, N°UV)

Diplôme (N°SS_etu, N°UV)

L'instruction SELECT

- C'est l'instruction fondamentale du LID mais est aussi utilisée dans le LDD et la LMD
- Convention d'écriture :
  - Mots-clés sont écrits en majuscule
  - Options encadrées entre [ ]
  - Variables encadrées entre < >
  - Terminaison des instructions par ;

Présentation de l'interrogation

SELECT *

FROM <table>

[WHERE <condition>];

SELECT <att1>, <att2>, ….

FROM <table>

[WHERE <condition>];

Présentation de l'interrogation

- Le SELECT simple correspond à la projection
- L'utilisation du WHERE correspond à la sélection
- Exemple :
  - SELECT * FROM Etudiant; →affiche table entière
  - SELECT titre FROM Unite-valeur ;→ projection
  - SELECT nom_ens FROM Enseignant WHERE grade='MCF'; → projection + sélection

Variables paramétrées

- Une variable paramétrée est une variable qu'il faudra préciser lors de l'exécution de la requête.
- Leur nom commence par &
- Exemple

  SELECT * FROM &nomtab;

Variables paramétrées

- 2 utilisations possibles :
  - Utilisation directe :

    SQL>run

    Le système demande le nom de la variable

    ⇒enter value from Nomtab:

    L'utilisateur entre le nom réel de la table :

    ⇒Inscription
  - Utilisation indirecte

    SQL>start req1 Inscription

    La requête req1 est exécutée directement avec la valeur indiquée

Requêtes mono-tables

LANGAGE D'INTERROGATION DES DONNES

SELECT : formalisme général

```
SELECT [DISTINCT/ALL] /<colonne>/<expression>
FROM <NomTable>
[WHERE <condition>]
[GROUP BY <colonne>]
[HAVING <condition>]
[ORDER BY <colonne>/<expression> [ASC/DESC], …];
```

[DISTINCT/ALL] : permet l'élimination ou la conservation des tuples Identiques dans le résultat
<expression> : expression arithmétique, résultat de calcul ou fonction
WHERE <condition> : permet la sélection de tuples
[GROUP BY] : permet constitution de groupes de tuples ayant des valeurs Identiques
[HAVING <condition>] : permet d'effectuer une condition sur les groupes Précédents → attention : impossibilité d'utiliser un WHERE dans un GROUP BY
[ORDER BY] : permet l'affichage des tuples dans un ordre donné (tri)

Affichage des résultats

- Résultat affiché sous forme d'un tableau avec les noms de colonnes en en-tête.
- Pour présenter un tableau-résultat avec un nom de colonne différent, il y a 2 possibilités :
  - SELECT adr_etu adresse etudiant FROM Etudiant
  - SELECT adr_etu **AS** adresse etudiant FROM Etudiant

Conditions

- Forme générale :
  <opérande><opérateur><opérande>
- Opérandes :
  - Un nom de colonne
  - Une valeur : num, alphanum, date
  - Une expression : arithmétique (+ - * /) ou utilisant des fonctions
  - Une sous-requête

Conditions

- Opérateurs numériques (à n'utiliser qu'avec des colonnes ou des valeurs numériques) : + - * /
- Opérateurs de comparaison : = <> != > >= < <=

Conditions

- Opérateurs spécifiques
  - IN : le 1er opérande doit appartenir à une liste de valeurs donnée dans le 2ème opérande → couleur IN ('bleu', 'blanc', 'rouge')
  - [NOT] IN : le 1er opérande ne doit pas appartenir à la liste
  - IS [NOT] NULL : la condition est nulle si la valeur de l'attribut est [n'est pas] nulle

Conditions

- Opérateurs spécifiques
  - ALL : la condition est vraie si elle est satisfaite pour chacune des valeurs de la liste. Responsable <>(Riri, Loulou) → tous les responsables différents de Riri ou Loulou
  - ANY, SOME : La condition est vraie si elle est satisfaite par une ou plusieurs valeurs de la liste
  - Dans ces 2 cas, la liste est souvent fournie par une sous-requête.

Conditions

- Opérateurs spécifiques
  - [NOT] BETWEEN … AND … : la condition est vraie si le 1er opérande est compris (ou n'est pas compris) entre les 2 valeurs données (bornes comprises) :
    WHERE salaire BETWEEN 1200 AND 2000
  - [NOT] EXISTS <sous-requête> : la condition est vraie si la sous-requête retourne (ou ne retourne pas) au moins un tuple. *EXISTS est sémantiquement équivalente à l'opérateur ANY/SOME*

Exemples

```
SELECT n°SS_etu FROM etudiant
WHERE nom_etu like '_u%'

SELECT n°SS_ens FROM enseignant
WHERE grade='MCF' and salaire>=2100

SELECT n°SS_ens FROM enseignant
WHERE grade=ALL(MCF, professeur)

SELECT * FROM Unite-valeur
```

DISTINCT

DISTINCT élimine les valeurs dupliquées.

« Combien y a-t-il de responsables d'UV? ».

```
SELECT    COUNT (DISTINCT (RESPONSABLE))
FROM      UNITE-VALEUR
```

EXISTS

- L'opérateur EXISTS permet de construire un prédicat évalué à vrai si la sous-requête renvoie au moins une ligne.

  Vol (NumVol, VilDep, VilAr, HDep, HAr, JourAr)

  Affectation (NumVol, DateVol, NumPilote, NumAvion, NbPassagers)

  Pilote (NumPilote, Nom, Adr, Salaire, Commission, DateEmbauche)

  Avion (NumAvion, CodeType, AnneeMiseService, Nom, NbHVol)

Exemple

- Liste des vols ayant utilisé au moins une fois un Airbus A390

```
SELECT NomVol
FROM Affectation A
WHERE EXISTS (
        SELECT * FROM Avion
        WHERE A.NumAvion=NumAvion
        AND codeType='Airbus A380');
```

Fonctions

- On les rencontre :
  - Soit dans les expressions à afficher
  - Soit dans les opérandes des conditions. Il faut alors respecter la cohérence des différents opérandes
  - On les classe selon le type du paramètre :
    - Fonctions numériques (simples ou sur un ensemble de valeurs)
    - Fonctions sur les chaînes (fournissant une chaîne ou une valeur numérique)
    - Fonctions sur les dates

Fonctions numériques

- On peut citer :
  - Les fonctions logarithmiques et trigonométriques
  - Les fonctions de transformation
  - Les fonctions sur un ensemble de valeurs : ces fonctions s'appliquent sur l'ensemble des valeurs fournies par une requête ou une sous-requête
    - AVG(<n>) : moyenne des valeurs de n (valeurs nulles non comptées)
    - COUNT(*) : nombre de tuples renvoyés par la requête
    - COUNT(<n>) : nombre de valeurs non nulles
    - MAX (<n>) : valeur maximum de n
    - MIN (<n>) : valeur minimum de n
    - STDDEV(<n>) : écart-type de n (valeurs nulles non comptées)
    - VARIANCE(<n>) : variance de n (valeurs nulles non comptées)

Fonctions sur les chaînes de caractères

- On distingue :
  - Les fonctions retournant une chaîne
    - LENGTH(<c>) : longueur de la chaîne
  - Les fonctions retournant une valeur numérique
    - INITCAP (<c>) : la première lettre de chaque mot est mise en majuscule
    - LOWER (<c>) : conversion en minuscules
    - UPPER (<c>) : conversion en majuscules
    - REPLACE (<c>, <c1>, <c2>) : remplacement dans c de c1 par c2

Fonctions sur les dates

- Parmi ces fonctions, on peut citer :
  - ADD_MONTH(<d>,<n>) : ajouter n mois à la date d
  - MONTH_BETWEEN (<d1>, <d2>) : nombre de mois entre d1 et d2
  - SYSDATE : date et heure système
  - TO_CHAR(date, format) : convertit une date en chaîne de caractères.
  - TO_DATE (date, format) : convertit une chaîne en date.

Exemples

« Donner le nombre d'UV auxquelles est inscrit l'étudiant n°163.. »

```
SELECT          COUNT (N°UV)
FROM            INSCRIPTION
WHEREN°SS_étu = ‘ 163..’;
```

Afficher toutes les caractéristiques de l'enseignant ayant le plus gros salaire

```
SELECT *

FROM Enseignant

WHERE salaire=(SELECT max(salaire) FROM  Enseignant);
```

Exemples

« Donnez le salaire moyen des maîtres de conférences ».

```
SELECT      AVG (SALAIRE)
FROM        ENSEIGNANT
WHERE       grade = ‘ MAITRE DE CONFERENCES’;
```

« Donnez le nom des maître de conférences qui gagnent plus que la moyenne des salaires des enseignants ».

```
SELECT      Nom_ens
FROM        ENSEIGNANT
WHERE       grade = ‘ MAITRE DE CONFERENCES’ AND
            salaire > (SELECT AVG(salaire)FROM ENSEIGNANT);
```

Partitionnement d'une relation

- Syntaxe : GROUP BY … [HAVING]
- La clause GROUP BY permet de créer des sous-ensembles regroupant des tuples ayant une caractéristique commune. Le résultat de la requête est habituellement une fonction de groupe appliquée sur chaque sous-ensemble.
- La clause HAVING permet de ne prendre en compte que des sous-ensembles vérifiant une condition donnée.
- ATTENTION : on ne peut pas utiliser la clause WHERE dans un GROUP BY

Partitionnement d'une relation

« Donnez le salaire moyen des enseignants par grade. »

```
SELECT      grade, AVG (salaire)
FROM        ENSEIGNANT
GROUP BY    grade;
```

« Donnez le nom des enseignants responsables de plus de 3 UV »

```
SELECT      responsable
FROM        UNITE-VALEUR
GROUP BY    responsable
HAVING COUNT(N°UV)>3;
```

Partitionnement d'une relation

Soit la relation : Employé (NumEmp, nom, salaire) et la relation : Projet (NumEmp, NumProjet) et la requête :

« Donnez les noms des employés qui gagnent plus que la moyenne des autres employés et qui travaillent sur plus de trois projets ».

```
SELECT      NOM
FROM        EMPLOYE, PROJET
WHERE       Employe. NumEmp= Projet. NumEmp
AND         salaire>(SELECT AVG(salaire)FROM Employé)
GROUP BY    N°EMPLOYE
HAVING      COUNT(Projet)>3;
```

Les fonctions de tri

ORDER BY permet de présenter les résultats triés par ordre ascendant (**ASC**) [par défaut] ou descendant (**DESC**).

« Donnez le nom des professeurs par ordre alphabétique ».

```
SELECT      nom_ens
FROM        ENSEIGNANT
WHERE       grade = « PROFESSEUR »
ORDER BY    nom_ens ASC;
```

Requêtes multi-tables

LANGAGE D'INTERROGATION DES DONNES

Introduction

- Il existe trois types de requêtes multi-tables :
  - Celles qui utilisent des jointures
  - Celles qui sont basées sur les opérateurs ensemblistes (Union, Intersect, Minus)
  - Les requêtes imbriquées : utilisation de sous-requêtes dans les conditions Where ou Having.

Jointures

- L'opérateur de jointure permet de mettre en relation deux ou plusieurs tables (deux à deux).
- Une jointure constitue donc un produit cartésien suivi d'une sélection
- Dans une opération de jointure, la condition de sélection est appelée pivot de jointure.
- Plusieurs types de jointure sont définis selon la nature de la condition de sélection

Produit cartésien

Le produit cartésien s'exprime d'une façon très simple :

Il suffit de citer les deux relations qui participent au produit.

```
SELECT    *
FROM      R1, R2;
```

Thêta-jointure

- Rappel : jointure qui peut prendre les valeurs suivantes : >, <, >=, <=, <>, !=, BETWEEN, LIKE, IN
- Syntaxe (par ex) :

  SELECT…..

  FROM <table1>; <table2>

  WHERE <table1>.<col>!=<table2>.<col>;
- Exemple : Soit la relation suivante

  Etudiant (num_etu, nom, age)

  Enseignant (num_ens, nom, age)
- Donner les étudiants qui sont plus vieux que les enseignants :

  SELECT *

  FROM Etudiant, Enseignant

  WHERE Etudiant.age>Enseignant.age;

Jointure naturelle (ou équijointure)

- La jointure naturelle permet de réaliser une liaison logique entre 2 tables
- Dans la jointure naturelle, le pivot de jointure exprime l'égalité de valeurs entre la clé primaire de la première table et la clé étrangère de la seconde qui a pour référence la clé primaire de la première.
- Syntaxe :

  SELECT…..

  FROM <table1>; <table2>

  WHERE <table1>.<col>=<table2>.<col>;

Jointure naturelle

« Donnez le nom des étudiants inscrits à l'UV 10 »

```
SELECT      Nom_étu
FROM        ETUDIANT, INSCRIPTION
WHERE       ETUDIANT.N°SS_étu = INSCRIPTION.N°SS_étu
AND         N°UV= 10;
```

Autojointure (jointure interne)

- Lorsqu'une même table est utilisée 2 fois dans la même requête (autojointure), on peut lever l'ambiguïté en utilisant un synonyme local pour chacune.
- Exemple : Table X, Table Y → la sélection des champs se fait en préfixant le nom du synonyme de la table : X. idTable, Y.idTable.

Autojointure

« Donnez les noms des assistants qui gagnent plus qu'un maître de conférence. »

```
SELECT    A.nom_ens
FROM      ENSEIGNANT A, ENSEIGNANT M
WHERE     A.salaire> M.salaire
AND       A.GRADE = « assistant »
AND       M.GRADE = « MDC »;
```

Jointure externe

SELECT ….

FROM <table1>, <table2>

WHERE <table1>.<col>[(+)]=<table2>.<col>[(+)];

- Le symbole (+) désigne une colonne qui sera complétée par des null dans l'opération de jointure externe.

Expression de la semi-jointure

On utilise le signe +

Soit la relation R

Professeur	Elève
Marc	Corinne
Marc	Maria
Antoine	Lise
Joseph	Corinne

Soit la relation R1

Elève	Classe
Corinne	S1
Maria	S2
Alice	S4

La requête :

```
SELECT professeur, élève, classe
FROM R, R1
WHERE R.élève = R1.élève(+);
```

Expression de la semi-jointure

Le résultat de la requête donne la relation R2

Professeur	Elève	Classe
Marc	Corinne	S1
Marc	Maria	S2
Antoine	Lise	
Joseph	Corinne	S1

Recherche des valeurs nulles

On utilise l'expression **IS [NOT] NULL**

IS NULL est vraie si et seulement si la valeur associée est la valeur nulle.

« Donnez les noms des professeurs qui enseignent à des élèves dont on ne connaît pas la classe ».

```
SELECT    PROFESSEUR
FROM      R2
WHERE     CLASSE  IS NULL
```

Opérateurs ensemblistes

- La syntaxe générale est la suivante :

  (<sous-requête1>) INTERSECT(<sous-requête2>)

  (<sous-requête1>) UNION(<sous-requête2>)

  (<sous-requête1>)MINUS(<sous-requête2>)

- Les résultats fournis par les sous-requêtes doivent être union-compatibles : même nombre de colonnes de même type.

Intersection

On utilise le verbe **INTERSECT**

« Quels sont les n° et les noms des étudiants qui sont aussi enseignants? »

(fournir les n°SS identiques dans les deux relations)

```
(SELECT     N°SS_étu, nom_étu
FROM        ETUDIANT)
INTERSECT
(SELECT     N°SS_ens, nom_ens
FROM        ENSEIGNANT);
```

Différence

On utilise le verbe **MINUS**.

« Liste des n°SS des étudiants et des n°UV auxquelles ils ont échoué »

```
(SELECT     *
FROM        INSCRIPTION)
MINUS
(SELECT     *
FROM        DIPLÔME);
```

Union

On utilise le verbe **UNION.**

« Liste des n°SS des personnes participant aux UV. »

```
(SELECT     N°SS_étu
FROM        INSCRIPTION)
UNION
(SELECT     N°SS_ens
FROM        ENSEIGNEMENT);
```

Requêtes imbriquées

- Une condition de sélection utilisée dans une clause WHERE ou HAVING s'exprime sous forme d'une comparaison entre la valeur d'un attribut et une valeur de référence.
- SQL étend cette notion de valeur de référence au résultat d'une requête.
- L'expression de la clause WHERE peut donc prendre la forme suivante : WHERE attribut opérateur (SELECT…);
- On dit que la requête qui sert de valeur de référence est une requête imbriquée ou une sous-requête.

Requêtes imbriquées

- Il est possible d'imbriquer plusieurs requêtes, le résultat de chaque requête imbriquée servant de valeur de référence dans la condition de sélection de la requête de niveau supérieur appelée requête principale.
- Le nombre de niveaux d'imbrication est théoriquement illimité.
- Plusieurs types de requêtes imbriquées :
  - La sous-requête renvoie une seule ligne
  - La sous-requête peut renvoyer plusieurs lignes
  - La sous-requête est synchronisée avec la requête principale.

Sous-requête indépendante renvoyant une seule ligne

- On utilise une sous-requête de ce type lorsque la valeur de référence de la condition de sélection doit être unique.
- La sous-requête est évaluée entièrement avant la requête principale. Tout se passe comme si on exécutait la sous-requête et qu'on utilise son résultat pour exécuter la requête principale.
- Pour que la requête s'exécute correctement, il faut que la sous-requête retourne une seule ligne. A défaut, SQL génère une erreur.

Exemple

- Donner la liste des étudiants inscrits la même année que l'étudiant E8

```
SELECT n°SS_etu, nom_etu
FROM Etudiant, Inscription
WHERE Etudiant.n°SS_etu=Inscription.n°SS_etu
AND date_inscription =(
        SELECT date_inscription
        FROM Inscription
        WHERE n°SS_etu='E8');
```

Extension : sous-requête avec plusieurs colonnes

- Donner la liste des étudiants inscrits la même année que l'étudiant E8 et ayant les mêmes UV

```
SELECT n°SS_etu, nom_etu
FROM Etudiant, Inscription
WHERE Etudiant.n°SS_etu=Inscription.n°SS_etu
AND (date_inscription, n°UV) =(
        SELECT date_inscription, n°UV
        FROM Inscription
        WHERE n°SS_etu='E8');
```

Autre exemple

- Donner le nom des maîtres de conférences qui gagnent plus que la moyenne des salaires des enseignants.

```
SELECT      Nom_ens
FROM        ENSEIGNANT
WHERE       grade = ‘ MAITRE DE CONFERENCES’ AND
            salaire > (SELECT AVG(salaire)FROM ENSEIGNANT);
```

Sous-requête indépendante pouvant renvoyer plusieurs lignes

- Une sous-requête de ce type s'utilise lorsque la condition de sélection fait référence à une liste de valeurs.
- La condition de sélection utilise alors l'opérateur IN ou un opérateur simple (=, !=, <>, <, <=, >, >=) précédé de ALL ou ANY.
- Rappel :
  - IN → la condition est vraie si elle est vérifiée pour une des valeurs renvoyées par la sous-requête
  - ANY → la comparaison est vraie si elle est vraie pour au moins un des valeurs renvoyées par la sous-requête
  - ALL → la comparaison est vraie si elle est vraie pour chacune des valeurs renvoyées par la sous-requête.

Remarques / Exemple

- WHERE <col> IN (SELECT)

  → l'opérateur IN est équivalent à l'opérateur =ANY

- WHERE <col> NOT IN (SELECT)

  → l'opérateur NOT IN est équivalent à l'opérateur !=ANY

- Exemple : Liste des étudiants qui n'ont pas réussi

```
SELECT n°SS_etu, nom_etu
FROM Etudiant
WHERE n°SS_etu NOT IN (
        SELECT n°SS_etu
        FROM Diplôme);
```

Sous-requête synchronisée avec la requête principale

- SQL sait traiter une sous-requête faisant référence à une colonne de la table de l'interrogation principale.
- Le traitement est plus complexe car il faut évaluer la sous-requête pour chaque ligne traitée par la requête principale.
- La sous-requête est évaluée pour chaque ligne de la requête principale.

Exemple

- Soit l'exemple suivant :

  Vol (NumVol, VilDep, VilAr, HDep, HAr, JourAr)

  Affectation (NumVol, DateVol, NumPilote, NumAvion, NbPassagers)

  Pilote (NumPilote, Nom, Adr, Salaire, Commission, DateEmbauche)

- Liste des vols ayant un pilote qui habite la ville de départ du vol. Editer le numéro du vol, la ville de départ, la ville d'arrivée, le nom du pilote

Exemple

```
SELECT vol, vildep, vilar, nom
FROM  Vol V, Affectation A, Pilote P
WHERE  V.numVol=A.numVol
AND   A.numPilote=P.numPilote
AND   vildep=(
                SELECT Adr
                FROM Pilote
                WHERE A.numPilote=P.numPilote);
```