Rappels SQL, Intro PL/SQL

Thibault Bernard

thibault.bernard@univ-reims.fr

Langage SQL

- SQL est un langage de manipulation de donnés non procédural, il permet :
  - La Définition de données
  - La Manipulation de données
  - Le contrôle de données
- Basé sur l'algèbre relationnel de Codd (1970)

Syntaxe Minimale

SELECT <liste d'attributs projetés> FROM <liste de relations>

- Syntaxe possiblement enrichie de très nombreuses clauses permettant d'exprimer
  - Projections
  - Restrictions
  - Jointures
  - Tris
  - …

Syntaxe étendue

SELECT <liste d'attributs projetés>

FROM <liste de relations> JOIN … ON <critères de jointure>

[WHERE < restrictions >]

[GROUP BY <liste d'attributs à partitionner>

[HAVING <condition>]]

[ORDER BY <attribut> [ASC/DESC] [<attribut> [ASC/DESC]] …]

Sous requêtes

- Where -> conditions par comparaisons sur des valeurs
- Expressions de conditions sur des relations ?
    - Sous requêtes: décrire des requêtes complexes permettant d'effectuer des opérations dépendant d'autres requêtes.
    - Utilisable derrière WHERE et HAVING
    - Sous requête renvoie
        - Une valeur unique : on utilise alors des opérateurs de comparaisons classiques
        - Un attribut => opérateurs IN, EXISTS, opérateurs de comparaisons classiques + ALL ou ANY

Mot Clé EXISTS

L'expression SQL EXISTS (SELECT… FROM…)

Est évalué à Vrai si et seulement si le résultat de l'évaluation du SELECT … FROM est non vide (le résultat donne au moins un tuple).

La requête « appelante » n'est évaluée que si le résultat de la sous requête est évaluée à vrai.

Division

- [Algèbre relationnelle]: La division de la relation R de schéma $R(A_1,\ldots,A_n)$ par la relation S de schéma $S(A_{p+1},\ldots,A_n)$ est la relation T de schéma $T(A_1,\ldots,A_p)$ formés de tous les tuples qui concaténées à chaque tuples de S donnent toujours un tuple de R.
- Notation R / S
- Opérateur type qui permet de répondre aux questions du type: Ç donner les docteurs qui soignent tous les patients

Division

- SQL n’offre pas la possibilité d’exprimer directement le quantificateur ∀. On utilise une négation du quantificateur ∃.
- La question « Donner le nom des fournisseurs livrant tous les produits » devient « Donner le nom des fournisseurs pour lesquels il n’existe aucun produit non livré »
- Procéder par étapes :

Division

- Division $R(A,B) / S(B) = T(A)$

  $= \{a \in R(A) / \forall\, b \in S(B), (a,b) \in R(A,B)\}$

  $= \{a \in R(A) / \nexists\, b \in S(B), (a,b) \notin R(A,B)\}$

- La traduction mot à mot donne

```
SELECT A FROM R AS R1 WHERE NOT EXISTS
        (SELECT B FROM S WHERE NOT EXISTS
        (SELECT A, B FROM R WHERE R1.A = A AND S.B = B ) )
```

Opérations ensemblistes

- UNION
- INTERSECT
- EXCEPT

s'intercalent entre deux SELECT

Requêtes complexes, exemple

- A l'aide du schéma relationnel suivant :
  - Vehicule (NumImmat, Modele, Marque, Couleur, PuissFisc, #NumAgenceProp)
  - Agence (NumAg, NomAg, AdresseAg, VilleAg, TelAg)
  - Locataire(NumLoc, NomLoc, PrénomLoc, AdresseLoc,VilleLoc, TelLoc)
  - Loue(NumLoc,NumImmat,NumAg,DateDebutL,DuréeL)
  - Assurance(NumAs, NomAs,AdresseAs,VilleAs, TelAs)
  - Assure(NumImmat,NumAg,NumAs,DateDebutA,DureeA)

Requêtes complexes, exemple

Formulez la requête suivante :

« Donner le nom des locataires ayant loué un véhicule à une agence qui assure ce même véhicule auprès d'une assurance située a Reims, et qui possède des véhicules gris ayant été loués à des locataires parisiens. »

Expressions Régulières et SQL

Définition :

Une expression régulière est une chaîne de caractères, qui décrit, selon une syntaxe précise, un ensemble de chaînes de caractères possibles.

Exemples courants :

*.jpg

li*.*

Utilité :

Décrire un ensemble de chaînes de caractères par un motif.

Décrire une expression régulière

- Les symboles signifiant une opération particulière sont appelés opérateurs. Les caractères qui ne représentent qu'eux-mêmes sont des littéraux.

Opérateurs	Signification	Opérateurs	Signification
.	Un caractère quelconque	[^]	Génère n'importe quel caractère, exceptés ceux compris entre les crochets
()	Forme un groupe	\	Indique un littéral
{n}	Génère n occurrences de l'élément précédant	{n,m}	Génère au moins n occurrences de l'élément précédant et au plus m
*	Génère 0 ou plus occurrences de l'élément précédant	+	Génère 1 ou plus occurrences de l'élément précédant
|	Génère soit l'élément précédant soit l'élément suivant	[]	Génère un des éléments compris entre les crochets
?	Génère 0 ou 1 occurrence de l'élément précédant	[a-z]	Génère un des caractères donnés dans l'interval

Exemple d'expressions régulières

- Jean ou Marc ou Louis

  (Jean) | (Marc) | (Louis)
- Un code de 4 chiffres

  [0-9]{4}
- Un numéro de téléphone français

  (0|\(+33\) [1-9](*[0-9]{2}){4}
- Un pseudo commençant par une majuscule, suivie de lettres minuscules et terminant par un nombre à 2 ou 3 chiffres

  [A-Z] [a-z]* [0-9]{2,3}

Utilisation de regexp

- Donner le nom et le prénom des locataires ayant un numéro de téléphone via l'opérateur BuzzMobile

```
SELECT NomLoc, PrénomLoc FROM Locataire WHERE
        regexp_like(TelLoc,  ( 0 | \(+33\) ) 7501 [0-9] ([0-9]{2}){2},c)
```

- regexp_like prend 3 paramètres :
  - La chaine source
  - L'expression régulière à tester
  - Un paramètre de matching :
    - i non sensible à la casse
    - c sensible à la casse
    - …
- Autres fonctions pour la recherche d'expressions régulières : regexp_substr, regexp_replace, regexp_count, regexp_instr

Utilisation de regexp

- Donner le nom des locataires qui ont loué un véhicule à une agence dont la ville comporte la lettre ‘v’.

```
SELECT NomLoc FROM Locataire WHERE NumLoc IN
        (SELECT NumLoc FROM Loue WHERE NumAg IN
                (SELECT NumAg FROM Agence WHERE
                        regxep_like(VilleAg, .*v.*,i)
                )
        )
)
```

PL/SQL

Pourquoi ?

- SQL est un langage non procédural.
- Difficile d'écrire une requête complexe sans variables, structures de contrôle de programmation, …
- Nécessité de lier un langage procédural aux requêtes SQL.

Caractéristiques PL/SQL

- PL/SQL est un langage propriétaire de Oracle.
- Extension de SQL : une requête SQL cohabite avec des structures de contrôle (blocs, conditionnelles, boucles).
- Un programme est constitué de procédures et de fonctions.
- Des variables permettent l'échange d'information entre les requêtes SQL et le reste du programme.
- D'autres SGBQ utilisent des langages proches de PL/SQL.

Utilisation

- PL/SQL peut être utilisé pour l'écriture des procédures stockées et des triggers
- Il convient aussi pour écrire des fonctions utilisateurs qui peuvent être utilisées dans les requêtes SQL (en plus des fonctions prédéfinies)
- Il est aussi utilisé dans des outils Oracle, Forms et Report en particulier

Structure d'un programme

- Un programme est structuré en blocs d'instructions de 3 types :
  - procédures anonymes
  - procédures nommées
  - fonctions nommées
- Un bloc peut contenir d'autres blocs.
- Structure d'un bloc :

```
DECLARE
        -- définitions de variables
BEGIN
        -- Les instructions à exécuter
EXCEPTION
        -- La récupération des erreurs
END;
```

- BEGIN et END sont obligatoires, « ; » à la fin d'une instruction ou d'un bloc.

Variables

- Identifiants:
  - 30 caractères au plus.
  - commence par une lettre.
  - peut contenir lettres, chiffres, _, $ et #.
- Pas sensible à la casse.
- Portée standard.
- Doivent être déclarées avant d'être utilisées.

Types et déclaration de variables

- Les types habituels correspondants aux types SQL2 ou Oracle : integer, varchar,…
- Types composites adaptés à la récupération des colonnes et lignes des tables SQL : %TYPE, %ROWTYPE
- Type référence : REF
- Déclaration d'une variable
  - identificateur [CONSTANT] type [:= valeur];
  - Exemples :
    - age integer;
    - nom varchar(30);
    - dateNaissance date;
    - ok boolean := true;
  - Déclarations multiples interdites !

Déclaration %TYPE/ %ROWTYPE

- On peut déclarer qu'une variable est de même type qu'une colonne d'une table ou d'une vue (ou qu'une autre variable) :

  nom relation.attribut.%TYPE;

- Une variable peut contenir toutes les colonnes d'une ligne d'une table
- employe emp%ROWTYPE; déclare que la variable employe contiendra une ligne de la table emp

Type RECORD

- Equivalent à struct du langage C
- TYPE nomRecord IS RECORD (

  champ1 type1,

  champ2 type2,

  …);

Affectation

- Plusieurs façons de donner une valeur à une variable :
  - :=
  - par la directive INTO de la requête SELECT
- Exemples :
  - dateNaissance := ’10/10/2004’;
  - select nome INTO nom from emp where matr = 509;

Exemple d’utilisation

```
employe emp%ROWTYPE;
nom emp.nome.%TYPE;
select * INTO employe from emp where matr = 900;
nom := employe.nome;
employe.dept := 20;
…
insert into emp values employe;
```

Conflits de noms

- Si une variable porte le même nom qu'une colonne d'une table, c'est la colonne qui prédomine

```
DECLARE
    nome varchar(30) := 'DUPOND';
BEGIN
        delete from emp where nome = nome;
```

- Pour éviter ça, le plus simple est de ne pas donner de nom de colonne à une variable !

Conditionnelles

```
IF condition THEN
        instructions;
END IF;

IF condition THEN
        instructions1;
ELSE
        instructions2;
END IF;
```

```
IF condition1 THEN
        instructions1;
ELSEIF condition2 THEN
        instructions2;
ELSEIF …
        …
ELSE
        instructionsN;
END IF;
```

Case of

CASE expression

WHEN expr1 THEN instructions1;

WHEN expr2 THEN instructions2;

…

ELSE instructionsN;

END CASE;

- expression peut avoir n'importe quel type simple (ne peut pas par exemple être un RECORD)

Boucle

```
LOOP
        instructions;
        EXIT [WHEN condition];
        instructions;
END LOOP;
```

Boucle Tant que

```
WHILE condition LOOP
        instructions;
END LOOP;
```

Boucle pour

```
FOR compteur IN [REVERSE] inf..sup LOOP
        instructions;
END LOOP;
```

Exemple :

```
for i IN 1..100 LOOP
        somme := somme + i;
end loop;
```

Extraction de données

- select expr1, expr2,… into var1, var2,… met des valeurs de la BD dans une ou plusieurs variables.
- Le select ne doit renvoyer qu'une seule ligne.
- Avec Oracle il n'est pas possible d'inclure un select sans « into » dans une procédure .
- Pour ramener des lignes, voir la suite du cours sur les curseurs.

Cas d’erreurs

- Si le select renvoie plus d’une ligne, une exception « TOO_MANY_ROWS » (ORA-01422) est levée
- Si le select ne renvoie aucune ligne, une exception « NO_DATA_FOUND » (ORA-01403) est levée

Exemple

```
DECLARE
        v_nom emp.nome%TYPE;
        v_emp emp%ROWTYPE;
BEGIN
        select nome into v_nom from emp where matr = 500;
        select * into v_emp from emp where matr = 500;
```

Modification de données

- Les requêtes SQL (insert, update, delete,…) peuvent utiliser les variables PL/SQL.
- Les commit et rollback doivent être explicites : aucun n'est effectué automatiquement à la sortie d'un bloc.

Ajout

```
DECLARE
        v_emp emp%ROWTYPE;
        v_nom emp.nome%TYPE;
BEGIN
        v_nom := 'Dupond';
        insert into emp (matr, nome) values (600, v_nom);
        v_emp.matr := 610;
        v_emp.nome := 'Durand';
        insert into emp (matr, nome)
        values(v_emp.matr, v_emp.nome);
        commit;
END;
```

Curseurs - Fonctionnalités

- Toutes les requtes SQL sont associées à un curseur.
- Ce curseur représente la zone mémoire utilisée pour parser et exécuter la requête.
- Le curseur peut être implicite (pas déclaré par l'utilisateur) ou explicite.
- Les curseurs explicites servent à retourner plusieurs lignes avec un select.

Attributs des curseurs

- Tous les curseurs ont des attributs que l'utilisateur peut utiliser
  - %ROWCOUNT : nombre de lignes traitées par le curseur
  - %FOUND : vrai si au moins une ligne a été traitée par la requête ou le dernier fetch
  - %NOTFOUND : vrai si aucune ligne n'a été traitée par la requête ou le dernier fetch
  - %ISOPEN : vrai si le curseur est ouvert (utile seulement pour les curseurs explicites)
- Les curseurs implicites sont tous nommés SQL

Exemple de curseur implicite

```
DECLARE
        nb_lignes integer;
BEGIN
        delete from emp where dept = 10;
        nb_lignes := SQL%ROWCOUNT;
```

Curseur explicite

- Pour traiter les select qui renvoient plusieurs lignes.
- Ils doivent être déclarés.
- Le code doit les utiliser explicitement avec les ordres OPEN, FETCH et CLOSE.
- Le plus souvent on les utilise dans une boucle dont on sort quand l'attribut NOTFOUND du curseur est vrai.

Exemple (déclaration)

```
DECLARE
        CURSOR salaires IS select sal from emp where dept = 10;
        salaire numeric(8, 2);
        total numeric(10, 2) := 0;
```

Exemple Utilisation

```
BEGIN
        open salaires;
        loop
                fetch salaires into salaire;
                exit when salaires%notfound;
                if salaire is not null then
                        total := total + salaire;
                        DBMS_OUTPUT.put_line(total);
                end if;
        end loop;
        close salaires; -- Ne pas oublier
        DBMS_OUTPUT.put_line(total);
END;
```

Type «row » associé à un curseur

- On peut déclarer un type « row » associé à un curseur
- Exemple :

```
declare
        cursor c is
                select matr, nome, sal from emp;
        employe c%ROWTYPE;
begin
        open c;
        fetch c into employe;
        if employe.sal is null then …
```

Boucle FOR pour un curseur

- Elle simplifie la programmation car elle évite d'utiliser explicitement les instruction open,fetch, close
- En plus elle déclare implicitement une variable de type « row » associée au curseur

Exemple

```
declare
        cursor c is select dept, nome from emp where dept = 10;
begin
        FOR employe IN c LOOP
                dbms_output.put_line(employe.nome);
        END LOOP;
end;
```

Curseur paramétré

- Un curseur paramétré peut servir plusieurs fois avec des valeurs des paramètres différentes.
- On doit fermer le curseur entre chaque utilisation de paramètres différents (sauf si on utilise « for » qui ferme automatiquement le curseur).
- Exemple :

```
declare
        cursor c(p_dept integer) is
        select dept, nome from emp where dept = p_dept;
begin
        for employe in c(10) loop
                dbms_output.put_line(employe.nome);
        end loop;
        for employe in c(20) loop
                dbms_output.put_line(employe.nome);
        end loop;
end;
```

Ligne courante d'un curseur

- La ligne courante d'un curseur est déplacée à chaque appel de l'instruction fetch.
- On est parfois amené à modifier la ligne courante pendant le parcours du curseur.
- Pour cela on peut utiliser la clause « where current of » pour désigner cette ligne courante dans un ordre LMD (insert, update, delete).
- Il est nécessaire d'avoir déclaré le curseur avec la clause FOR UPDATE pour que le bloc compile.

FOR UPDATE

- FOR UPDATE [OF col1, col2,…].
- Cette clause bloque toute la ligne ou seulement les colonnes spécifiées.
- Les autres transactions ne pourront modifier les valeurs tant que le curseur n'aura pas quitté cette ligne.
- Exemple :

```
DECLARE
    CURSOR c IS
        Select matr, nome, sal from emp where dept = 10 FOR UPDATE OF emp.sal;
        …
        if salaire is not null then
                total := total + salaire;
        else -- met 0 à la place de null
                update emp set sal = 0
                where current of c;
        end if;
```

Exceptions

- Une exception est une erreur qui survient durant une exécution.
- 2 types d'exception :
  - prédéfinie par Oracle.
  - définie par le programmeur.
- Une exception ne provoque pas nécessairement l'arrêt du programme si elle est saisie par un bloc (dans la partie « EXCEPTION »).
- Une exception non saisie remonte dans la procédure appelante (où elle peut être saisie).

Exceptions prédéfinies

- NO_DATA_FOUND
- TOO_MANY_ROWS
- VALUE_ERROR (erreur arithmétique)
- ZERO_DIVIDE
- …

Traitement des exceptions

```
BEGIN
…
EXCEPTION
WHEN NO_DATA_FOUND THEN
. . .
WHEN TOO_MANY_ROWS THEN
. . .
WHEN OTHERS THEN -- optionnel
. . .
END;
```

Exceptions utilisateur

- Elles doivent être déclarées avec le type EXCEPTION.
- On les lève avec l'instruction RAISE.

```
DECLARE
        salaire numeric(8,2);
        salaire_trop_bas  EXCEPTION;
BEGIN
        select sal into salaire from emp where matr = 50;
        if salaire < 300 then
                raise salaire_trop_bas;
        end if;
        -- suite du bloc
EXCEPTION
        WHEN salaire_trop_bas  THEN . . .;
        WHEN OTHERS THEN dbms_output.put_line(SQLERRM);
END;
```

Création d'une procédure

```
create or replace
PROCEDURE(<liste params>) IS
        -- déclaration des variables
BEGIN
        -- code de la procédure
END;
```

- Pas de DECLARE ; les variables sont déclarées entre IS et BEGIN.
- Si la procédure ne nécessite aucune déclaration, le code est précédé de « IS BEGIN ».

Création d'une fonction

```
create or replace
FUNCTION(<liste params>)
RETURN <type retour> IS
        -- déclaration des variables
BEGIN
        -- code de la procédure
END;
```

Passage des paramètres

- Dans la définition d'une procédure on indique le type de passage que l'on veut pour les paramètres :
  - IN pour le passage par valeur.
  - INOUT pour le passage par référence.
  - OUT pour le passage par référence mais pour un paramètre dont la valeur n'est pas utilisée en entrée.
- Pour les fonctions, seul le passage par valeur (IN) est autorisé.

Fonctions

- Les fonctions peuvent aussi être utilisées dans les requêtes SQL

```
create or replace
function euro_to_fr(somme IN number)
        RETURN number IS
        taux constant number := 6.55957;
begin
        return somme * taux;
end;
```

Utilisation

- Utilisation dans un bloc anonyme

```
declare
        -cursor c(p_dept integer) is
                select dept, nome, sal from emp where dept = p_dept;
begin
        for employe in c(10) loop
                dbms_output.put_line(employe.nome || ' gagne ‘|| 
euro_to_fr(employe.sal) || ' francs');
        end loop;
end;
```

- Utilisation dans une requte SQL

```
        select nome, sal, euro_to_fr(sal) from emp;
```

Procédure stockée

- Une procédure stockée est en fait une série d'instructions SQL désignée par un nom.
- Lorsque de sa création, on l'enregistre dans la base de données.
- Il est possible d'appeler celle-ci, par son nom. Les instructions de la procédure sont alors exécutées.
- Pour déclencher l'exécution d’une procédure stockée, il faut utiliser CALL, suivi du nom de la procédure appelée, puis de parenthèses (avec éventuellement des paramètres).
- Suppression d'une procédure : DROP

Triggers

- Les triggers (ou déclencheurs) sont des objets de la base de données.
- Attachés à une table, ils se déclenchent l'exécution d'une instruction, ou d'un bloc d'instructions, lors d'un événement d'insertion, de suppression ou de modification (INSERT, DELETE, UPDATE).
- Un trigger exécute un traitement pour chaque ligne insérée, modifiée ou supprimée par l'événement déclencheur.

À quoi sert un trigger ?

- Contraintes et vérifications de données (Vérifier les valeurs des données lors d’ajout/modification)
- Intégrité des données (verification clé étrangère par exemple)
- Historisation des actions (pour verification de qui a modifié quoi)
- Mise à jour d'informations qui dépendent d'autres données

Création des triggers

- Pour créer un trigger, on utilise CREATE :

```
CREATE TRIGGER nom_trigger moment_trigger evenement_trigger
ON nom_table FOR EACH ROW
corps_trigger;
```

- nom est le nom du trigger
- nom_table est la table à laquelle est attaché le trigger
- FOR EACH ROW signifie que le traitement sera effectué pour chaque ligne
- corps_trigger c'est les instructions
- moment_trigger (BEFORE/AFTER)
- evenement_trigger (INSERT/UPDATE/DELETE)

Exemple de création

- Pour créer un trigger sur la table *Animal*, déclenché par une insertion, et s'exécutant après ladite insertion, on utilisera la syntaxe suivante :

```
CREATE TRIGGER after_insert_animal AFTER INSERT
ON Animal FOR EACH ROW
… ;
```

- Il ne peut exister qu'un seul trigger par combinaison.

Ecriture d'un trigger

- Dans le corps du trigger, SQL met à disposition deux mots-clés : OLD et NEW
- Exemple
  - Contraintes et vérification des données :

    Dans notre table *Animal* se trouve la colonne *sexe*. Cette colonne accepte tout caractère, ou NULL. Or, seuls les caractères "M" et "F" ont du sens.

Exemple

```
DELIMITER |
CREATE TRIGGER before_update_animal BEFORE UPDATE
ON Animal FOR EACH ROW
BEGIN
IF
        NEW.sexe IS NOT NULL   -- le sexe n'est ni NULL
        AND NEW.sexe != 'M'      -- ni  Mâle
        AND NEW.sexe != 'F'      -- ni  Femelle
        AND NEW.sexe != 'A'      -- ni  Autre
THEN
        SET NEW.sexe = NULL;
END IF;
END |
DELIMITER ;
```

- **TEST**

```
UPDATE Animal SET sexe = 'A' WHERE id = 20;  -- l'animal 20 est Balou, un mâle
SELECT id, sexe, date_naissance, nom FROM Animal WHERE id = 20;
```

Restrictions

- Il est impossible de travailler avec des transactions à l'intérieur d'un trigger.
- Les requêtes préparées ne peuvent pas non plus être utilisées.
- Une suppression ou modification de données déclenchée par une clé étrangère ne provoquera pas l'exécution du trigger correspondant.