Systèmes de gestion de bases de données

Procédures stockées et triggers

P. Rigaux

Cnam, dépt. informatique

November 4, 2016

Contenu du cours

- Principes de la programmation avec BD
- Le langage PL/SQL
  - syntaxe
  - intégration avec SQL: types, clause INTO, valeurs à NULL, parcours d'un résultat de requête
- Les curseurs
- Les triggers

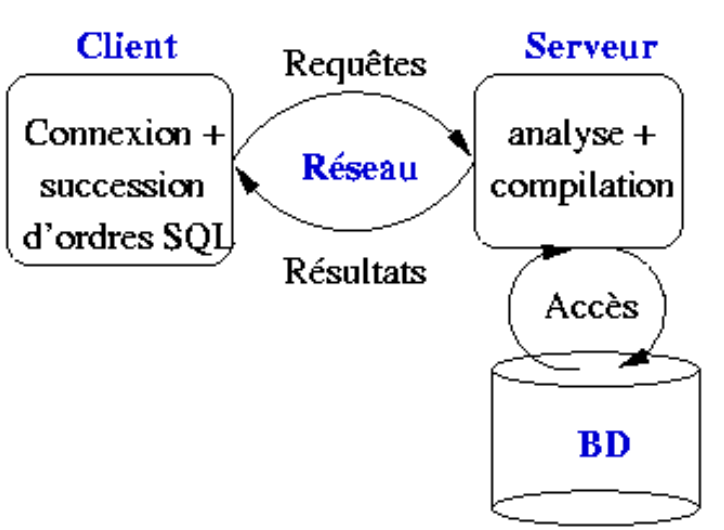
Cours rédigé disponible à http://sql.bdpedia.fr

Rôle des procédures stockées

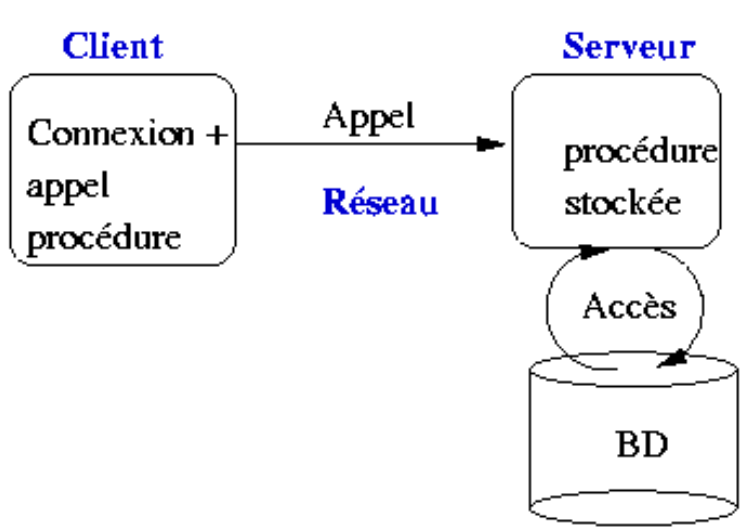
- SQL = langage
  - ensembliste
  - déclaratif (quoi, pas comment) :
  - pas de variables, d'itérations ou de tests, de fonctions, etc.
- *PL/SQL = Procedural SQL*

  Proposé par Oracle, existe avec des variantes dans d'autres systèmes.
- Qui : programmeur d'application, DBA

Architectures



Communications avec un programme externe



Communications avec une procédure stockée

PL/SQL, quand l'utiliser, à quoi ça sert ?

- Modes
  - Interactif : on exécute du code pour, par exemple, contrôler ou corriger des données
  - Stocké: sous forme de procédures ou de triggers
  - Programme : appel depuis langages généralistes
- But
  - stocker du code "BD", dans la base
  - factoriser certains traitements très proches des données (sécurité)
  - performances

Facilite grandement l'installation de la partie applicative.

Premier exemple, mode interactif

```
DECLARE
  -- Quelques variables
  v_nbFilms   INTEGER;
  v_nbArtistes INTEGER;

BEGIN
  -- Compte le nombre de films
  SELECT COUNT(*) INTO v_nbFilms FROM Film;
  -- Compte le nombre d'artistes
  SELECT COUNT(*) INTO v_nbArtistes FROM Artiste;

  -- Affichage des resultats
  DBMS_OUTPUT.PUT_LINE ('Nombre de films: ' || v_nbFilms);
  DBMS_OUTPUT.PUT_LINE ('Nombre d''artistes: ' || v_nbArtistes);

  EXCEPTION
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE ('Probleme rencontre');

END;
```

Deuxième exemple, procédure stockée

```
CREATE OR REPLACE PROCEDURE InsereGenre (p_genre VARCHAR) AS
  v_genre_majuscules VARCHAR(20);
  v_count INTEGER;
  genre_existe EXCEPTION;
BEGIN
  -- On met le parametre en majuscules
  v_genre_majuscules := UPPER(p_genre);

  -- On verifie que le genre n'existe pas
  SELECT COUNT(*) INTO v_count
  FROM Genre WHERE code = v_genre_majuscules;

  -- Si on n'a rien trouve: on insere
  IF (v_count = 0) THEN
   INSERT INTO Genre (code) VALUES (v_genre_majuscules);
  ELSE
   RAISE genre_existe;
 END IF;

  EXCEPTION
    WHEN genre_existe THEN
     DBMS_OUTPUT.PUT_LINE('Le genre existe deja en ' ||
                          v_count || ' exemplaire(s).');
```

Troisième exemple, fonction

```
CREATE OR REPLACE FUNCTION MesActeurs(v_idFilm INTEGER)
    RETURN VARCHAR IS
  resultat VARCHAR(255);

BEGIN
  -- Boucle prenant tous les acteurs du films
  FOR art IN
    (SELECT Artiste.* FROM Role, Artiste
    WHERE idFilm = v_idFilm  AND idActeur=idArtiste)
  LOOP

    IF (resultat IS NOT NULL) THEN
      resultat := resultat || ', ' || art.prenom || ' ' || art.nom;
    ELSE
      resultat := art.prenom || ' ' || art.nom;
   END IF;

  END LOOP;

 return resultat;
END;
```

Utilisation

- En interactif

```
SQL> start StatsFilms.sql
```

- Avec l'ordre `execute`, placé dans un autre langage (C, Java, PHP)

```
execute insereGenre('Policier')
```

- Dans une requête SQL

```
SELECT titre, MesActeurs(idFilm)
FROM Film WHERE idFilm=5;
```

```
TITRE          MESACTEURS(IDFILM)
-------------- ------------------------------
Volte/Face     John Travolta, Nicolas Cage
```

Syntaxe PL/SQL: Les variables

Point à retenir : les types sont ceux de SQL, **y compris les lignes (`RECORD`) et les tables (`TABLE`).**

Favorise une intégration très facile avec les requêtes.

Exemple de déclaration d'un type `RECORD`.

```
TYPE adresse IS RECORD
(no          INTEGER,
 rue         VARCHAR(40),
 ville       VARCHAR(40),
 codePostal  VARCHAR(10)
);
```

On peut faire encore mieux en **dérivant** un type à partir du schéma de la base.

Dérivation de types depuis le schéma

Deux exemples pour illustrer.

- `Film.titre%TYPE` est le titre de l'attribut `titre` de la table `Film`;
- `Artiste%ROWTYPE` est un type `RECORD` correspondant aux attributs de la table `Artiste`.

Le même principe pour les **requêtes SQL** définies dans le cadre des curseurs (à suivre).

Beaucoup plus difficile avec un langage de programmation externe.

La section DECLARE

Toutes les déclarations sont rassemblées dans la section DECLARE.

```
DECLARE
  -- Constantes
  v_aujourdhui  CONSTANT DATE   := SYSDATE;
  v_pi          CONSTANT NUMBER(7,5) := 3.14116;

  -- Variables scalaires
  v_compteur    INTEGER NOT NULL := 1;
  v_nom         VARCHAR(30);

  -- Variables pour un n-uplet de la table Film et pour le resume
  v_film Film%ROWTYPE;
  v_resume Film.resume%TYPE;
```

Affectation des variables

PL/SQL propose un mécanisme d'affectation standard.

```
<nomVariable> := <expression>;
```

Mais surtout il est possible de placer dans des variables locales le résultat de requêtes SQL.

```
SELECT <nomAttribut1>, [<nomAttribut2>, ...]
INTO   <nomVariable1>, [<nomVariable2>, ... ]
FROM   [...]
```

Il doit y avoir correspondance des types (facile car...)

Un exemple complet

```
CREATE OR REPLACE FUNCTION TitreEtMES(v_idFilm INTEGER)
    RETURN VARCHAR IS

  -- Variables
  v_titre Film.titre%TYPE;
  v_idMES Film.idMES%TYPE;
  v_mes  Artiste%ROWTYPE;

BEGIN
  -- Recherche du film
  SELECT  titre, idMES
  INTO v_titre, v_idMES
  FROM Film
  WHERE idFilm=v_idFilm;

  -- Recherche du metteur en scene
  SELECT * INTO v_mes FROM Artiste WHERE idArtiste = v_idMES;

  return v_titre  || ', realise par ' || v_mes.prenom
                  || ' ' || v_mes.nom;
END;
```

Les exceptions avec un exemple

```
CREATE OR REPLACE PROCEDURE TitreEnMajuscules (p_idFilm INT) AS
  v_titre Film.titre%TYPE;
  deja_fait EXCEPTION;
BEGIN
  -- Recherche du film. Exception si on ne trouve rien
  SELECT titre INTO v_titre
  FROM Film WHERE idFilm = p_idFilm;

  -- Si le titre est en majuscule, autre exception
  IF (v_titre = UPPER(v_titre)) THEN
      RAISE deja_fait;
  END IF;

  -- Mise en majuscules du titre
   UPDATE Film SET titre=UPPER(v_titre) WHERE idFilm=p_idFilm;

  EXCEPTION
    WHEN NO_DATA_FOUND THEN
     DBMS_OUTPUT.PUT_LINE('Ce film n''existe pas');
    WHEN deja_fait THEN
     DBMS_OUTPUT.PUT_LINE('Le titre est en majuscules');
    WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE('Autre erreur...');
END;
```

Exemple d'exécution

Illustration des levées d'exception.

```
SQL> execute TitreEnMajuscules(900);
Le film n'existe pas

SQL> execute TitreEnMajuscules(5);

SQL> execute TitreEnMajuscules(5);
Le titre est en majuscules
```

Les curseurs

Curseur = **mécanisme de base pour parcourir le résultat d'une requête SQL**.

Permet de charger une ligne à la fois. Evite le chargement d'une très grosse table si résultat volumineux.

Un curseur se déclare. Exemple PL/SQL.

```
CURSOR MonCurseur IS
SELECT * FROM Film, Artiste
WHERE idMES = idArtiste;

-- Une variable pour le curseur
v_monCurseur MonCurseur%ROWTYPE;
```

Les curseurs paramétrés

Autre exemple, avec paramètre

```
CURSOR MonCurseur (p_artiste INTEGER) IS
 SELECT prenom || nom AS nomMES, anneeNaiss, COUNT(*) AS nbFilms
 FROM Film, Artiste
 WHERE idMES = p_artiste

-- Une variable pour le curseur
v_realisateur MonCurseur%ROWTYPE;
```

Exécution d'un curseur

Un curseur est **toujours** exécuté en trois phases

- ouverture du curseur (ordre OPEN) ;
- parcours du résultat en itérant des ordres FETCH autant de fois que nécessaire ;
- fermeture du curseur (CLOSE).

Bien comprendre : (1) on calcule, (2) on récupère le résultat **ligne à ligne**; (3) on libère les ressources.

Un exemple complet (1)

```
CREATE OR REPLACE PROCEDURE CurseurFilms (p_annee INT) AS

  -- Curseur
  CURSOR MonCurseur (v_annee INTEGER) IS
   SELECT idFilm, titre, prenom, nom
   FROM Film, Artiste
   WHERE idMES = idArtiste
   AND   annee = v_annee;

  -- Variable du curseur
  v_monCurseur MonCurseur%ROWTYPE;
  -- Variable pour la liste des acteurs
  v_mesActeurs VARCHAR(255);
BEGIN
  -- Ouverture du curseur
  OPEN MonCurseur(p_annee);
```

Un exemple complet (2)

```
-- On prend le premier n-uplet
  FETCH MonCurseur INTO v_monCurseur;
  -- Boucle sur les n-uplets
  WHILE (MonCurseur%FOUND) LOOP
    -- Recherche des acteurs avec la fonction MesActeurs
    v_mesActeurs := MesActeurs (v_monCurseur.idFilm);

    DBMS_OUTPUT.PUT_LINE('Ligne ' || MonCurseur%ROWCOUNT ||
          ' Film:  ' || v_monCurseur.titre ||
          ', de ' || v_monCurseur.prenom || ' ' ||
          v_monCurseur.nom || ', avec ' || v_mesActeurs);
    -- Passage au n-uplet suivant
    FETCH MonCurseur INTO v_monCurseur;
  END LOOP;
  -- Fermeture du curseur
  CLOSE MonCurseur;

  EXCEPTION
    WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE('Message : ' || sqlerrm);
END;
```

Un exemple complet (3)

Exemple de l'exécution de cette procédure.

```
SQL> set serveroutput on
SQL> execute CurseurFilms(1992);
Ligne 1 Film:  Impitoyable, de Clint Eastwood, avec
         Clint Eastwood, Gene Hackman, Morgan Freeman
Ligne 2 Film:  Reservoir dogs, de Quentin Tarantino, avec
          Quentin Tarantino, Harvey Keitel, Tim Roth, Chris Penn
```

On a "encapsulé" un traitement de données complet, que l'on peut ensuite exécuter dans n'importe quel contexte.

Autre syntaxe

Il existe une syntaxe encore plus simple.

```
CREATE OR REPLACE PROCEDURE CurseurMAJ  AS

  -- Curseur
  CURSOR CurseurGenre IS
   SELECT * FROM Genre  FOR UPDATE;
BEGIN
  -- Boucle FOR directe: pas de OPEN, pas de CLOSE

  FOR v_genre IN CurseurGenre LOOP
   IF (v_genre.code IS NULL) THEN
     DELETE FROM Genre WHERE CURRENT OF CurseurGenre;
   ELSE
     UPDATE Genre SET code=UPPER(code)
     WHERE CURRENT OF CurseurGenre;
   END IF;
  END LOOP;
END;
```

On retrouve ce type d'expression dans des langages intéprétés (construction de la variable au moment de l'exécution). Python, PHP...

Les *triggers*

Un **trigger** est une procédure stockée qui se déclenche automatiquement sur certains événements.

Les possibilités offertes par les triggers sont très intéressantes. Citons:

- la gestion des redondances; l'enregistrement automatique de certains événements (auditing);
- la spécification de contraintes complexes liées à l'évolution des données (exemple: le prix d'une séance ne peut qu'augmenter);
- toute règle liée à l'environnement d'exécution (restrictions sur les horaires, les utilisateurs, etc.).

Inconvénient (fort): les *triggers* s'exécutent de manière cachée, et peuvent mener à des cycles sans fin.

Un exemple simple

Voici un trigger qui maintient la capacité d'un cinéma à chaque mise à jour sur la table `Salle`.

```
CREATE TRIGGER CumulCapacite
AFTER UPDATE ON Salle
FOR EACH ROW
WHEN (new.capacite != old.capacite)
BEGIN
  UPDATE Cinema
  SET capacite = capacite - :old.capacite  + :new.capacite
  WHERE  nom = :new.nomCinema;
END;
```

Pour garantir la validité du cumul, il faudrait créer des triggers sur les événements UPDATE et INSERT.

Les composants d'un trigger

Le modèle d'exécution des triggers est basé sur la séquence événement-Condition-Action (ECA) que l'on peut décrire ainsi:

- un trigger est déclenché par un évènement, spécifié par le programmeur, qui est en général une insertion, destruction ou modification sur une table;
- la première action d'un trigger est de tester une condition: si cette condition ne s'évalue pas à TRUE, l'exécution s'arrête;
- enfin l'action proprement dite peut consister en toute ensemble d'opérations sur la base de données, effectuée si nécessaire à l'aide du langage procédural supporté par le SGBD (PL/SQL).

Un trigger peut manipuler simultanément les valeurs ancienne et nouvelle de la donnée modifiée, ce qui permet de faire des tests sur l'évolution de la base.

Syntaxe générale

```
CREATE [OR REPLACE] TRIGGER <nomTrigger>
  {BEFORE | AFTER}
  {DELETE | INSERT | UPDATE [of column, [, column] ...]}
     [ OR {DELETE | INSERT | UPDATE [of column, [, column] ...]}] ...
  ON <nomTable> [FOR EACH ROW]
[WHEN <condition]
   <blocPLSQL>
```

- **Evénement**, peut être BEFORE ou AFTER, suivi de DELETE, UPDATE ou INSERT séparés par des OR.
- **Condition**, FOR EACH ROW est optionnel. En son absence le trigger est déclenché une fois pour toute requête modifiant la table, et ce sans condition.
- **Action** est une procédure PL/SQL. Elle peut contenir des ordres SQL mais pas de mise à jour de la table courante.

Attention avec les triggers!

On peut des triggers qui **ralentissent** tout. Exemple

```
CREATE TRIGGER CumulCapaciteGlobal
AFTER UPDATE OR INSERT OR DELETE ON Salle
BEGIN
  UPDATE Cinema C
  SET capacite = (SELECT SUM (capacite)
                  FROM   Salle S
                  WHERE  C.nom = S.nomCinema);
END;
```

Attention avec les triggers!

On peut des triggers qui **bloquent** tout. Exemple d'un trigger à la fois idiot et assassin.

```
CREATE TRIGGER MAJSalle
AFTER UPDATE OR INSERT OR DELETE ON Cinema
BEGIN
  UPDATE Salle C
  SET capacite = (SELECT :new.capacite / SUM (capacite)
                  FROM   Salle S
                  WHERE  C.id = :new.id);
END;
```

Boucle infinie garantie, et très difficile à comprendre....