SQL

Ch 9 – DDL
Vues, PL/SQL, procédures,...

I. Introduction

Les tables sont les objets les plus courants utilisés dans une base de données. Il existe cependant d'autres objets :

- Les vues (*anglais : views*), qui facilitent l'accès aux données et permettront d'en administrer l'accès,
- Les procédures stockées et fonctions, qui étendent les possibilités du langage SQL grâce aux apports de la programmation
- Les déclencheurs (*anglais : triggers*), qui complète les possibilités de ??? de l'intégrité des données d'une base de données.

Les ordres SQL utilisés dans les différents SGBD sont relativement conformes au standard en ce qui concerne les vues.
En ce qui concerne les langages de programmation intégrés aux SGBD, chaque éditeur a développé une syntaxe spécifique. Pour Oracle nous trouverons le langage PL/SQL (Procedural Language)

II. Les vues (anglais « views »)

On peut considérer les vues comme des « tables virtuelles » : leur contenu est constitué seulement au moment de la demande d'exécution de cette vue.

A. Créer une vue : CREATE VIEW

Créer une vue, c'est :
_ donner un nom à une requête qui pourra être utilisée ensuite comme toute autre table
_ la définition de la vue est stockée,
_ son contenu est constitué à chaque fois qu'on y accède

```
CREATE [OR REPLACE ] VIEW nomVue
    [(colonne1, colonne2,…)]
    AS
```

```
requête_SQL_SELECT ;
```

Toute requête d'interrogation (SELECT) peut être valable dans la création d'une vue.

Plusieurs intérêts aux vues (pour l'utilisateur) :

- Simplifier l'utilisation de tables complexes : supprimer la complexité des jointures, éviter les requêtes mal écrites et coûteuses,
- Sauvegarder les requêtes fréquemment utilisées,
- Cacher aux utilisateurs certaines informations sensibles.

1. Simplifier l'écriture des requêtes SQL

Soient les tables relationnelles suivantes :

- Pays (codePays, libPays)
- Client (idClient, raisonSociale, #codePays)

On souhaite fréquemment effectuer la requête suivante :

```
SELECT idClient, A.codePays, libPays, raisonSociale
    FROM client A
       INNER JOIN pays B ON A.codePays = B.codePays
    WHERE A.codePays = 'FR';
```

Créer une vue :

```
CREATE VIEW vueClientsFrancais
    AS
    SELECT idClient, A.codePays, libPays, raisonSociale
      FROM client A
         INNER JOIN pays B ON A.codePays = B.codePays;
      WHERE A.codePays = 'FR';
```

Utiliser une vue :

```
SELECT *
    FROM vueClientsFrancais ;
    ORDER BY raisonSociale ;
```

2. Protéger l'accès aux données « sensibles »

Soient les tables relationnelles suivantes :

- Service (codeService, nomService)
- Employe (numEmpl, nomEmpl, prenomEmpl, salaireEmpl, #codeService)

On souhaite protéger l'accès aux données et empêcher les utilisateurs non autorisés d'avoir accès au salaire de la fiche Employe (et on donnera des noms de colonnes plus simples) :

```
CREATE VIEW listeDuPersonnel (numero, nom, prenom, service)
    AS
    SELECT numEmpl, nomEmpl, prenomEmpl, nomService
    FROM Employe A
       INNER JOIN Service B ON A.codeService = B.codeService;
```

➔ On accordera les droits d'accès à la vue (plutôt qu'à la table entière) (cf ch.10)

Utiliser une vue :

```
SELECT *
    FROM listeDuPersonnel;
```

B. Supprimer une vue – DROP VIEW

La suppression de la vue n'entraîne rien d'autre que l'effacement de la requête qui lui est associée.

```
DROP VIEW nomVue;
```

➔ Supprimer la vue « vueClientsFrancais » :

```
DROP VIEW vueClientsFrancais ;
```

➔ Supprimer la vue « listeDuPersonnel » :

```
DROP VIEW listeDuPersonnel;
```

III.Langage PL/SQL, procédures et fonctions, déclencheurs

Les SGBD offrent des possibilités d'extension au déjà puissant langage SQL.
Ils mettent à disposition de l'informaticien un langage de programmation (pour Oracle, il s'appelle PL/SQL) et des différents objets qui permettent de l'utiliser :

- Les **procédures** et **fonctions** : de la même manière que tout langage programmation, on peut écrire des sous-programmes pour effectuer des actions trop complexes pour être réalisées en SQL
- Les **déclencheurs** : mécanisme qui permet l'exécution d'un bloc d'instructions lorsqu'un évènement se produit dans une table

Le code constituant ces blocs d'instructions est compilé.

A. Le langage Oracle PL/SQL

Ce langage est complet est comprend :

- Un ensemble de types de données : les types de base du SGBD, des types spécifiques pour manipuler les lignes des tables, etc.
- Des instructions de bases
- des structures de contrôles élaborées : structures conditionnelles, choix multiples, structures itératives
- une gestion d'exception avec des instructions permettant de gérer les anomalies

La structure d'un bloc PL/SQL est la suivante :

```
DECLARE
-- declaration des variables
BEGIN nomBloc
-- instructions à executer (peut inclure un nouveau bloc)
EXCEPTION
-- traitement des erreurs
END nomBloc;
```

Soit les tables relationnelles suivantes :

- Produit (refProduit, designation, prixCatalogue)
- Commande (idCde, #idClient)
- LigneCommande (idCde, numLigne, #refProduit, quantite, prixCde)

On souhaite vérifier qu'un client n'a plus de commandes avant de le supprimer d'une table :

```
declare
    idDesCommandes commande.idCde%type ;
begin
    SELECT idCde into idDesCommandes
      FROM commande
      WHERE idClient = 12 ;
exception
    when no_data_found then
      -- pas de commandes = suppression possible
      DELETE FROM client
        WHERE idClient = 1 ;
    when others then
      -- autres cas, commandes = suppression impossible
      raise_application_error(-20000,'err suppression') ;
end;
/
```

B. Procédures stockées

Une procédure stockée (*anglais : stored proc*) est un bloc nommé de PL/SQL stocké dans la base de données. On peut l'appeler directement par son nom pour l'exécuter en PL/SQL ou bien hrâce à la commande EXECUTE en SQL+.

Syntaxe de création (ou remplacement si la procédure existe déjà) :

```
CREATE OR REPLACE PROCEDURE nomProcedure
    [(nomParametre IN|OUT|IN OUT typeDeDonnees, … )]
    AS|IS
    Bloc PL/SQL;
```

➔ Créer une procédure :

```
CREATE OR REPLACE PROCEDURE suppLigneEtProd (numProd IN CHAR)
AS
begin
    DELETE FROM ligneCommande WHERE refProduit = numProd ;
    DELETE FROM produit WHERE refProduit = numProd ;
end ;
/
```

➔ Exécuter dans un autre bloc PL/SQL

```
declare
    unProduit CHAR;
begin
    unProduit := 'P001' ;
    suppLigneEtProd (unProduit) ;
```

```
end;
```

➔ Exécuter dans SQL*Plus

```
EXECUTE suppLigneEtProd ('P001') ;
```

Syntaxe de suppression :

```
DROP PROCEDURE nomProcedure;
```

C. Fonctions stockées

Une fonction stockée est un bloc nommé de PL/SQL stocké dans la base de données. Cette fonction renvoie une valeur. Elle peut être utilisée comme expression dans une requête SQL ou dans tout autre bloc de PL/SQL (procédure ou autre fonction), tout comme les autres fonctions Oracle.

Syntaxe de création (ou remplacement si la procédure existe déjà) :

```
CREATE OR REPLACE FUNCTION nomFonction
    [(nomParametre IN|OUT|IN OUT typeDeDonnées, … )]
    RETURN typeDeDonnees
    AS|IS
    Bloc PL/SQL incluant une instruction RETURN;
```

➔ Créer une fonction :

```
CREATE OR REPLACE FUNCTION calculTTC (montantHT IN NUMBER)
RETURN NUMBER
AS
begin
    return (montantHT * 1.196);
end;
```

➔ Exécuter dans une requête SQL (sans accéder à une table) :

```
SELECT calculTTC(12.50)
    FROM DUAL ;
```

➔ Exécuter dans une requête SQL :

```
SELECT SUM(calculTTC(quantite *prixCde))
    FROM ligneCommande ;
```

Syntaxe de suppression :

```
DROP FUNCTION nomFonction;
```

D. Les déclencheurs - TRIGGERS

Un déclencheur est un bloc de code PL/SQL associé à une table. Son exécution va être déclenchée (*anglais : fired*) lorsqu'une instruction DML (INSERT, UPDATE, DELETE) sera exécutée sur la table.

Les triggers offrent un moyen

- de définir des contraintes d'intégrité complexes (en plus des contraintes d'intégrité référentielles)
- de mettre en œuvre des journaux de mises à jour de certaines informations sensibles.

```
CREATE [OR REPLACE] TRIGGER nomDeclencheur
  BEFORE|AFTER    INSERT|UPDATE|DELETE
  ON nomTable [FOR EACH ROW]
  [WHERE condition ]
  Bloc PL/SQL;
```

Le code PL/SQL d'un déclencheur manipule les données des lignes qui vont être ajoutées, mises à jour ou supprimées. Pour cela des mots-clefs sont définis pour accéder aux nouvelles valeurs à ajouter (:NEW) et aux anciennes (:OLD).

Soit la table relationnelle suivante (*en général, on acceptera que les tables d'audits, soient dépourvues de clef primaire, d'une part, car il n'y a pas de besoins d'y retrouver une ligne en particulier, d'autre part pour des raisons de performances : la gestion d'une clef prend du temps...*) :

- AuditEmploye (utilisateur, dateAudit, typeMaj, numEmpl, nomAV, prenomAV, salaireMensuelAV, codeServiceAV, nomAP, prenomAP, salaireMensuelAP, codeServiceAP)

On souhaite conserver une trace des actions sur la table des employés :

```
CREATE OR REPLACE TRIGGER  auditer_employes
  AFTER INSERT OR DELETE OR UPDATE
        ON employe
        FOR EACH ROW
DECLARE
    typeMaj CHAR(3) := '';
    numEmploye INT := 0;
BEGIN
  IF INSERTING THEN
       typeMaj := 'cre';
       numEmploye := :NEW.numEmpl;
  END IF;
  IF DELETING THEN
       typeMaj := 'sup';
       numEmploye := :OLD.numEmpl;
  END IF;
  IF UPDATING THEN
       typeMaj := 'mod';
       numEmploye := :OLD.numEmpl;
  END IF;
  DBMS_OUTPUT.PUT_LINE('action :'||typeMaj);
  INSERT INTO auditEmploye VALUES (USER, SYSDATE, typeMaj, numEmploye ,
:OLD.nom,  :OLD.prenom,  :OLD.salaireMensuel,  :OLD.codeService,  :NEW.nom,
:NEW.prenom, :NEW.salaireMensuel, :NEW.codeService) ;
END;
/
```

Chaque insertion, mise à jour ou suppression de ligne da,s la table « employe » va ajouter une ligne dans la table « auditEmploye ».

En cas d'erreur de compilation sur un déclencheur (lister les dernières erreurs) :

```
SELECT * FROM USER_ERRORS WHERE TYPE = 'TRIGGER';
```

Ou bien :

```
SHOW ERRORS;
```

Syntaxe de suppression :

```
DROP TRIGGER nomTrigger;
```