2021

SQL sous Oracle

Les bases du PL / SQL

Olivier D.

Table des matières

1 Data Definition Language (DDL) 3
2 Data Manipulation Language (DML) 4
2.1 Les Fonctions 4
2.2 Les instructions 4
2.3 Les transactions 4
2.4 Calcul d'agrégat 4
2.5 Les jointures 5
2.6 Tri 6
2.7 Les tables temporaires 6
2.8 Les séquences 6
2.9 Les synonymes 6
3 PL/SQL (Procedural Language / Simple Query Language) 7
3.1 Le bloc 7
3.2 Gestion des variables 7
3.3 Les chemins alternatifs 8
3.4 Les boucles répétitives 8
3.5 Les curseurs 9
3.6 La gestion des erreurs 10
3.7 Les blocs nommés 12
3.8 Les triggers 14

1 Data Definition Language (DDL)

Attention : jamais de SELECT sans FROM (utiliser la table DUAL quand il n'y a pas de table utilisée).

```
SELECT sysdate FROM dual;
```

Les types

`VARCHAR(x)` / `VARCHAR2(x)` : chaines de caractères (`x` est la longueur max de la chaine)

`NUMBER(x,y)` : nombre (`x` est la longueur max avant la virgule, `y` est la longueur max après la virgule)

Supprimer une table :

```
DROP|PURGE table [CASCADE CONSTRAINT]
```

- `CASCADE CONSTRAINT` : supprime les FOREIGN KEYS dans les autres tables
- `PURGE` : supprime physiquement, pas de ROLLBACK possible

Modifier une table : source pour Oracle12

```
ALTER TABLE <nom> MODIFY <type de champ> <nouveau parameter du champ> ;
```

Changer le nom d'une table

```
RENAME <nom> TO <nouveau_nom> ;
```

Retour en arrière

```
FLASHBACK TABLE table TO BEFORE DROP
```

- Récupère la structure de la table avant sa suppression
- Attention : les données doivent être présentes dans les redolog pour le faire (cas d'une base en NOARCHIVELOG)

Retour à une date antérieure

Dans Oracle, la plupart des commandes SQL ne sont visibles que dans la session utilisateur tant qu'elles ne sont pas commitées. Au moment du commit, les données sont écrites dans la base de données.

- Doit être `COMMIT` avant
- Penser à faire : `CREATE TABLE ( … ) ENABLE ROW MOVEMENT;`

```
FLASHBACK TO TIMESTAMP (22/08/2011 11:51:00, DD/MM/YYYY HH24:MI:SS);
```

Indexer un champ

```
CREATE INDEX champ ON table (colonne [DESC][,...]);
```

2 Data Manipulation Language (DML)

2.1 Les Fonctions

Concaténer

`||` ou `CONCAT(a,b)`

Position

```
INSTR(c1,c2,n,m)
```

- position de la $n^{ième}$ occurrence (DEFAULT 1) de c2 dans c1, à partir du $m^{ième}$ caractère

`INSTR('bonjour','o',1,3)` : renvoie 5

Partie de chaine

`SUBSTR('bonjour',4,[taille])` : renvoie 'jour'

`CURRENT_DATE` : renvoie la date

Conversions

`TO_CHAR` / `TO_NUMBER` / `TO_DATE` … : transforme un champ en un autre. TO_CHAR d'une date est très utilisé pour faire de la mise en forme des dates

`COALESCE()` : remplacer les NULL dans une colonne
`DECODE (nbEmploye,1,'un employé'[,2,'deux employés'] …)` : remplace une valeur spécifique par une autre.

Calcul de dates

`TRUNC` : Tronque la date : récupération jour, mois, etc. (RI p .58)

`ADD_MONTHS (d,n)` : Ajoute n mois à la date d

`MONTHS_BETWEEN (d1,d2)` : différence (en mois) entre 2 dates

2.2 Les instructions

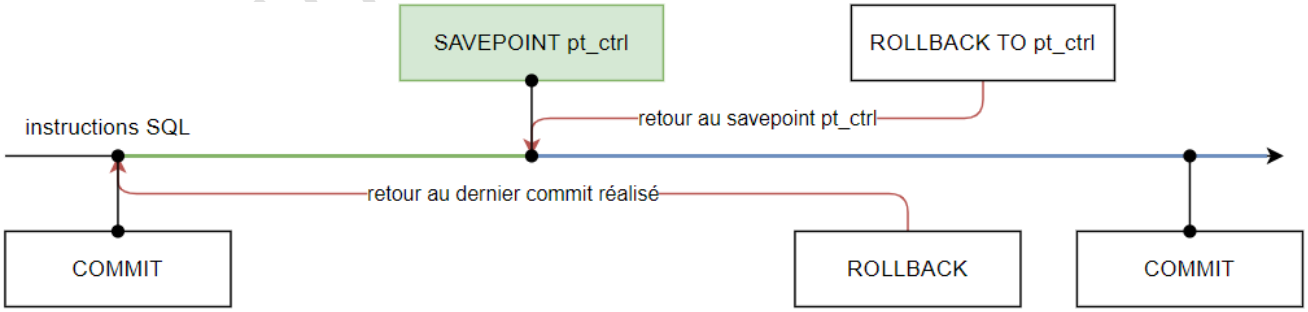
Suppression de données

`TRUNCATE` supprime toutes les lignes d'une table.

TRUNCATE est **irréversible**, immédiat et rapide !

2.3 Les transactions

Sous Oracle, toutes les instructions du DML se passent dans une transaction.



Fonctionnement de la commande Rollback

2.4 Calcul d'agrégat

```
COUNT *|champ , SUM , AVG , MIN , MAX [as "Libellé de colonne"]
GROUP BY ...
HAVING ...
```

Exemples : `select distinct grade, AVG(salaire) as "salaire moyen" from EMPLOYES group by grade HAVING grade in ('ETAM', 'CADRE', 'PRESTATAIRE');` ou `select count(name) from EMPLOYES;`

2.5 Les jointures

Jointures naturelles

```
SELECT * FROM EMPLOYES NATURAL JOIN SERVICES;
```

Les jointures naturelles sont peu utilisées dans la pratique

Jointures Internes / Externes

INNER JOIN classique :

```
SELECT CONGES.CODEEMP, CONGES.ANNEE,nbJoursAcquis, SUM(nbJoursPris)
FROM CONGES
        INNER JOIN CONGESMENS ON CONGES.CODEEMP = CONGESMENS.CODEEMP
        AND CONGES.ANNEE = CONGESMENS.ANNEE
GROUP BY CONGES.CODEEMP, CONGES.ANNEE, nbJoursAcquis;
```

INNER JOIN avec USING:

```
SELECT CODEEMP, ANNEE,nbJoursAcquis, SUM(nbJoursPris)
FROM CONGES
        INNER JOIN CONGESMENS USING (annee,codeemp)
GROUP BY CODEEMP, ANNEE, nbJoursAcquis;
```

Union, Intersection, Difference

Ces opérateurs permettent d'obtenir dans une requête des enregistrements provenant de requêtes différentes mais de même forme (même nombre/type/ordre des colonnes) :

a	
A	B
1	Albert
2	Bertrand

b	
A	B
1	Albert
2	Noemie
3	Louane

Tables utilisées dans les exemples suivants

```
SELECT A, B FROM a
   MINUS|UNION|INTERSECT
   SELECT A, B FROM b;
```

Différence :

Les enregistrements de a mais pas ceux de b : `MINUS`

a MINUS b	
A	B
2	Bertrand

Union :

Les enregistrements de a + les enregistrements de b : `UNION`

Nota : `UNION ALL` affiche les doublons

a UNION b	
A	B
1	Albert
2	Bertrand
1	Albert
2	Noemie
3	Louane

La deuxième ligne 1;Albert (en vert) n'apparait que dans le cas de UNION ALL car c'est un doublon

Intersection :

Les enregistrements communs à a et à b : `INTERSECT`

a INTERSECT b	
A	B
1	Albert

2.6 Tri

```
<requête select et clause where> ORDER BY … [DESC];
```

`<requête select et clause where> ORDER BY 2, 1 DESC;` : **2, 1** sont les numéros des colonnes

2.7 Les tables temporaires

```
CREATE VIEW nomVue AS SELECT …
WITH CHECK OPTIONS;
```

2.8 Les séquences

```
CREATE SEQUENCE sq_employes_codeEmp START WITH 3;
INSERT INTO Employes VALUES (sq_employes_codeEmp.NEXTVAL, 'Picasso', 'Pablo', );
```

2.9 Les synonymes

Chaque utilisateur (chaque personne qui utilise une session différente) utilise un schéma différent. Ce schéma préfixe de façon transparente le nom de chaque table, afin qu'il n'y ait pas de conflit si deux utilisateurs veulent créer et *modifier* leur table Employes sur une base de données existante.

Exemple : `SELECT * FROM USER1.EMPLOYES;` est un objet différent de `SELECT * FROM USER2.EMPLOYES;`

Si USER5 est connecté, `SELECT * FROM EMPLOYES` fait implicitement référence à `USER5.EMPLOYES`

Un synonyme permet de rendre publique une table en lui donnant un alias :

`CREATE [PUBLIC] SYNONYM EMPL FOR USER98.EMPLOYES;` : PUBLIC rend le synonyme visible par tout le monde.

`SELECT * FROM EMPL;` fera référence à `USER98.EMPLOYES`

3 PL/SQL (Procedural Language / Simple Query Language)

- Pour SQL Server, le langage procédural est TRANSAC.
- Pour Oracle, le langage procédural est PL/SQL.

PL/SQL gère : tout SQL, les variables, les contrôles, les boucles, le traitement des erreurs

3.1 Le bloc

Un bloc est toujours organisé de la façon suivante :

```
DECLARE
-- déclaration des variables;
BEGIN
-- ensemble d'instructions SQL et/ou PL/SQL;
EXCEPTION
-- traitement des exceptions;
END;
```

3.2 Gestion des variables

Affichage de base : « valeur de vNb : 1 » :

```
SET SERVEROUTPUT ON;                    -- affiche les résultats à l'écran
DECLARE vNb INTEGER;
BEGIN
vNb :=1
DBMS_OUTPUT.PUT_LINE('valeur de vNb : ' || vNb);
END;
/                                       -- exécute la procédure avant de passer à la ligne suivante
```

Récupérer tous les champs d'une ligne d'une table :

```
SET SERVEROUTPUT ON;
DECLARE
lEmpEmployes%ROWTYPE;
BEGIN
SELECT * INTO lEmp FROM Employes WHERE codeEmp=1;          -- lEmp contient les données de la requete
DBMS_OUTPUT.PUT_LINE ( lEmp.nom || ' ' || lEmp.prenom);    -- lEmp.nom vaut la colonne nom
END;
/
```

Récupérer un enregistrement (scalaire) d'un champ d'une ligne :

```
SET SERVEROUTPUT ON;
DECLARE
lEmp VARCHAR2(80) | Employes.nom%TYPE;
BEGIN
SELECT nom INTO lEmp FROM Employes WHERE codeEmp=1;
DBMS_OUTPUT.PUT_LINE (lEmp);
END;
/
```

3.3 Les chemins alternatifs

IF / THEN / ELSE / END IF;

```
SET SERVEROUTPUT ON;
DECLARE
vNb NUMBER := 2;
BEGIN
IF vNb > 5 THEN
        DBMS_OUTPUT.PUT_LINE('plus grand que 5');
ELSE
DBMS_OUTPUT.PUT_LINE('plus petit ou égal à 5');
END IF;
END;
/
```

CASE / WHEN / ELSE / END CASE;

```
CASE
WHEN condition1 THEN action1;
WHEN condition2 THEN action2;
ELSE action3;
END CASE;
/
```

3.4 Les boucles répétitives

LOOP (complexe):

```
<<boucle>>
LOOP
        -- Instructions;
        -- Changement valeur de variable;
EXIT boucle WHEN condition;
END LOOP boucle;
/
```

FOR (pas de 1 ou -1 seulement) :

```
FOR vN IN [REVERSE] 1..10 LOOP       -- REVERSE : le pas d'incrément est –1 -- 1..10 va de 1 à 10
        -- Instructions;
END LOOP;
/
```

WHILE (à retenir) :

```
Variable := valeur_de_début
WHILE condition LOOP;
        -- Instruction;
        -- Changement valeur de variable;
END LOOP;
/
```

3.5 Les curseurs

Un curseur récupère les numéros d'enregistrements à partir d'une requête `SELECT`. Il est ensuite possible de passer tous les enregistrements les uns à la suite des autres.

Le but est de pouvoir faire des manipulations sur les lignes de cette requête

```
CURSOR cReq IS SELECT * FROM EMPLOYES;
```

Résultat de la requête			
valeur de cReq	codeEmp	nom	...
1	0005	Erlin	
2	0001	Albert	
3	0003	Chaoleau	
4	0004	Dupont	
n	xxxx	...	

Fonctionnement des curseurs

Ecriture de la procédure

```
SET SERVEROUTPUT ON;
DECLARE
        CURSOR cEmp IS SELECT codeEmp, nom, prenom FROM Employes;
        lEmpcEmp%ROWTYPE;
BEGIN
        OPEN cEmp;
        FETCH cEmp INTO lEmp;
        DBMS_OUTPUT.PUT_LINE('employé : ' || lEmp.nom || ' ' || lEmp.prenom);
        CLOSE cEmp;
END;
/
```

Explications

`CURSOR cEmp IS <requête_select>;` : Déclaration et définition des valeurs du curseur

`lEmpcNon%ROWTYPE;` : `lEmp` est formaté pour correspondre aux types des champs de la requête de `cEmp`

`OPEN cEmpl` et `CLOSE cEmp;` : Ouverture / fermeture de l'utilisation du curseur `cEmp`

`FETCH cEmp INTO lEmp` : `FETCH` passe à la valeur suivante de `cEmp`. La première utilisation de `FETCH` (après un `OPEN`) envoie à la première valeur trouvée de `cEmp`

Curseur dans une boucle FOR

C'est la façon la plus simple de l'utiliser. Passe en revue les enregistrements de la requête l'un après l'autre jusqu'au dernier.

Le but est de faire des mises à jour sélectives de tables/champs.

```
DECLARE
        CURSOR cEmp IS SELECT codeEmp, nom, prenom FROM Employes;
        lEmpcEmp%ROWTYPE;
BEGIN
        FOR lEmp IN cEmp LOOP
                DBMS_OUTPUT.PUT_LINE('employé : '||lEmp.nom||' '||lEmp.prenom);
        END LOOP;
END;
/
```

Attention à l'ordre : … FETCH cEmp INTO lEmp … est différent de … FOR lEmp IN cEmp LOOP …

3.6 La gestion des erreurs

Il existe deux types d'erreurs :

- Les erreurs prédéfinies dans Oracle (trop de champs par rapport au résultat attendu, aucun enregistrement retourné, etc.). Ces erreurs sont le plus souvent préfixées par ORA-nnnnn,
- Les erreurs définies par le développeur.

Les erreurs prédéfinies dans Oracles

```
SET SERVEROUTPUT ON;
DECLARE
vNomEmployes.nom%TYPE;
BEGIN
SELECT nom INTO vNom FROM Employes WHERE codeDepartement='COMPT';
DBMS_OUTPUT.PUT_LINE('L''unique employé est ' || vNom);
SELECT nom INTO vNom FROM Employes;
EXCEPTION
WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('Il y a plus d''un employé dans ce service');
WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Il n''y a aucun employé dans ce service');
WHEN OTHERS
        DBMS_OUTPUT.PUT_LINE('Erreur autre');
END;
/
```

Construction :

```
WHEN nom_de_lerreur THEN
-- Instructions jusqu'au prochain WHEN … ou END;
```

Quitte ensuite le bloc. Il n'est pas possible de revenir à la procédure après une erreur.

Les erreurs définies par le développeur (erreurs personnalisées)

```
DECLARE
trop EXCEPTION;
vNbJoursPris NUMBER;
BEGIN
vNbJoursPris := 47;
        IF vNBJoursPris > 31 THEN
        RAISE trop;
        END IF;
INSERT INTO CongesMens VALUES (1,2011,10, vNbJoursPris);
EXCEPTION
WHEN trop THEN
        INSERT INTO CongesMens VALUES (1,2011,10, 31);
END;
/
```

Construction :

On définit un nom d'erreur (son type est EXCEPTION). Pour envoyer à la résolution de l'erreur on RAISE l'erreur, ce qui amène à la section EXCEPTION.

Alternative

Il est aussi possible, sans ajouter l'erreur à la section EXCEPTION, de définir l'erreur ainsi :

```
RAISE_APPLICATION_ERROR (numero-de-lerreur, description);
```

Exemple :

`RAISE_APPLICATION_ERROR (-20100, 'le nombre de jours est supérieur à 30');` : renvoit une ORA-20100

Le numéro d'erreur doit être compris entre -20000 et -20999

3.7 Les blocs nommés

- Procédures : ensemble d'instructions, ne retourne pas de valeur
- Fonctions : ensemble d'instructions, retourne une seule valeur à la fin

Les procédures

Construction :

```
CREATE [OR REPLACE] PROCEDURE nom [(paramètres)] AS|IS
Bloc PL/SQL (sans le mot DECLARE, finit par END;)
```

Les paramètres :

```
[(nomParam {IN|OUT|IN OUT} type* [DEFAULT NULL**](,nomparam2 …)]
```

*sans la taille – exemple : VARCHAR, NUMBER …

** pour un paramètre optionnel

Appel :

Appel depuis un bloc PL/SQL

```
BEGIN
  nomProcedure(parametres);
END;
```

Appel direct

```
SET SERVEROUTPUT ON;
EXECUTE nomProcedure(valeur_parametre);
```

Exemple :

```
/* Création d'une procédure */
CREATE PROCEDURE bonjour(nom IN VARCHAR2) AS
  BEGIN
  DBMS_OUTPUT.PUT_LINE('Bonjour ' || nom || ', comment allez vous ?');
END;
/
/* Appel depuis un bloc */
SET SERVEROUTPUT ON;
DECLARE
vNom VARCHAR2(30) := 'Olivier Dehecq';
BEGIN
bonjour(vNom);
END;
/
/* Appel direct */
SET SERVEROUTPUT ON;
EXECUTE bonjour('John Doe');
```

Les fonctions (renvoit UNE SEULE valeur à la fin)

Construction :

```
CREATE [OR REPLACE] FUNCTION nom [(paramètres)] RETURN type* AS|IS
Bloc PL/SQL (sans le mot DECLARE, finit par END;)
RETURN (variable);
```

* le type de la valeur renvoyée

Les paramètres :

```
[(nomParam [IN] type* [DEFAULT NULL**](,nomparam2 …)]
```

*sans la taille – exemple : VARCHAR, NUMBER …

** pour un paramètre optionnel

Appel :

Appel direct

```
SELECT nomFonction(valeurs_param) FROM DUAL;
```

Exemples :

```
/* Creation d'unefonction */
CREATE OR REPLACE FUNCTION addition(a NUMBER, b IN NUMBER) RETURN NUMBER AS
BEGIN
  RETURN (a+b);
END;
/

/* Appel d'une fonction */
SELECT addition(112,78) FROM DUAL;

/* Création d'une fonction plus longue */
CREATE OR REPLACE FUNCTION departement(NumDep NUMBER) RETURN VARCHAR2 AS
vNomDep VARCHAR2(30);
BEGIN
  CASE numDep
    WHEN 1 THEN vNomDep := 'AIN';
    WHEN 2 THEN vNomDep := 'AISNE';
    WHEN 3 THEN vNomDep := 'ALLIER';
    ELSE vNomDep := 'Autredépartement';
  END CASE;
RETURN (vNomDep);
END;
/

/* Appel d'une fonction */
SELECT departement(44) FROM DUAL;
```

3.8 Les triggers

- Un TRIGGER est un bout de code qui s'enclenche automatiquement lors d'un INSERT / UPDATE / DELETE.
- Il se déclenche soit avant, soit après la mise à jour de la table.
- Il peut s'exécuter une seule fois pour plusieurs mises à jour, ou bien à chaque enregistrement modifié.

Mise en place

Trigger validant ou non la modification de la table :

```
CREATE [OR REPLACE] TRIGGER bf_insert_employe
{BEFORE|AFTER} {INSERT|UPDATE|MODIFY} ON Table
[FOR EACH ROW]
Bloc complet (avec DECLARE);
```

Trigger modifiant les données avant la mise à jour de la table :

```
CREATE [OR REPLACE] TRIGGER bf_insert_employe
INSTEAD OF {INSERT|UPDATE|MODIFY} ON Table
[FOR EACH ROW]
Bloc complet (avec DECLARE);
```

Exemples :

```
/* Creation d'un trigger */
CREATE OR REPLACE TRIGGER bf_insert_employe
BEFORE INSERT ON Employes
FOR EACH ROW
DECLARE
existe INTEGER;
BEGIN
    --> verifie si le service existe sinon il est créé
SELECT COUNT(*) INTO existe FROM Departements WHERE codeDepartement= :NEW.codeDepartement;
  IF existe = 0 THEN
    INSERT INTO DepartementsVALUES(:NEW.codeDepartement, 'Service '||:NEW.codeDepartement);
END IF;
--> verification de la clé primaire
SELECT COUNT(*) INTO existe FROM Employes WHERE CodeEmp= :NEW.codeEmp;
  IF existe != 0 OR :NEW.codeEmp IS NULL THEN
    SELECT MAX(codeEmp)+1 INTO :NEW.codeEmp FROM Employes;
  END IF;
END;
```

(Des)activer un trigger:

```
ALTER TRIGGER bf_insert_employe DISABLE;
ALTER TRIGGER bf_insert_employe ENABLE;
```