

Cours 1 : Développement d'applications de bases de données

1. Applications de bases de données

Application : un programme d'ordinateur permettant d'effectuer une tâche quelconque.

Application de base de données : un programme qui utilise des données stockées dans une base de données qui est gérée par un Système de Gestion de Bases de Données (SGBD).

2. SGBD

- est un logiciel qui permet d'interagir avec une BD et l'intermédiaire entre les utilisateurs et les fichiers physiques.
- Un SGBD facilite :
 - La gestion de données, avec une représentation intuitive simple sous forme de tables.
 - La manipulation de données. On peut insérer, modifier les données et les structures sans modifier les programmes qui manipulent la base de données.
- La plupart des applications des bases de données doivent accéder à une base de données pour : Récupérer les données et les afficher, saisir de nouvelles données et mettre à jour les données

Objectifs d'un SGBD

- Persistance et Efficacité de l'accès aux données : Possibilité de réponses rapides(index).
- Description des données, Manipulation des données (Insérer, Supprimer, Modifier, Interroger)
- Contrôle
 - Partage de données : Gestion des accès concurrents aux données
 - Intégrité des données : Définition de contraintes sur les données.
 - Confidentialité des données : Accès aux données autorisés que pour les personnes habilitées.
 - sécurité physique (sauvegarde), reprise en cas de panne.

Outils de développement d'applications

Il existe différents outils de développement d'applications de BD :

- Langages de programmation : (exemple, COBOL, C, C++ (ODBC), JAVA(JDBC), ...) avec des drivers.
- Générateurs d'applications : (exemple Developer)
 - Outils permettant de développer des applications, principalement, à travers des spécifications déclaratives à partir d'une interface graphique. – Possibilité d'utiliser des procédures
- Utiliser un langage de programmation d'un SGBD comme PL/SQL

3. Conception d'un schéma relationnel

3.1. Sémantique des attributs

Les schémas de relations doivent avoir une sémantique claire

Ex) EMP-DEP(nss, nom1, dateN, adresse, Depnum, nom2, numhabitant) Ce schéma mélange des informations concernant des objets différents du monde réel.

3.2. Réduire (supprimer) les valeurs nulles

Dans la mesure du possible, éviter d'avoir dans une relation des attributs qui peuvent avoir des valeurs nulles. Sinon, assurez-vous que les valeurs nulles sont des cas exceptionnels.

3.3. Réduire les valeurs redondantes dans les tuples (c'est la plus importante)

La redondance est la source de beaucoup de problèmes : Stockage redondant, Anomalies de mise à jour : risque d'incohérence des données, Perte d'information

Décomposition d'un schéma de relation

Considérons un schéma de relation $R(A_1 \dots A_n)$. Une décomposition de R consiste à remplacer R par deux ou plusieurs schémas de relation.

Problèmes engendrés par la décomposition : – Certaines requêtes peuvent être plus coûteuses. – Étant donné une instance d'une relation décomposée, on n'est pas capable de retrouver l'instance originale.

Propriété de la décomposition :

- Décomposition sans perte d'information : La décomposition doit être réversible pour permettre de retrouver l'instance de la relation originale. La relation initiale est obtenue par jointure des composants
- Une autre formulation du problème de conception : Trouver un compromis entre les problèmes engendrés par la décomposition des schémas de relations et la redondance – Comment déterminer si un schéma relationnel a besoin d'être décomposé ?
- Pour des applications complexes, le « bon sens » n'est pas suffisant.
- Théorie de la normalisation : Représentation formelle et précise, Basée sur la notion de dépendances fonctionnelles.

la normalisation

1^{er} forme normale 1FN : Est en première forme normale, une relation (ayant par définition une clé) dont les attributs possèdent tous une valeur sémantiquement atomique ;

2^{ème} forme normale 2FN : Un attribut non clé ne dépend pas d'une partie de la clé mais de toute la clé.

3^{ème} forme normale 3FN : Tous les attributs non clé doivent dépendre directement de la clé.

Cours 2 : Définition d'une base de données avec Oracle

1. Les éléments fonctionnels

- **Schéma**: Ensemble d'objets (tables, vues, séquence, index, procédure,) à un utilisateur et qui porte son nom. Create table [shcema].table_name. Si vous ne précisez pas le schéma vous créer pas défaut la table dans votre schéma.
- **Table** : schéma de relation + relation
- **Vue** : table non matérialisée définie par une requête SQL sur d'autres tables et/ou vues. Les vues ne sont pas stockées. Par exemple, on définit une vue sur une jointure et l'utiliser plusieurs fois.
- **Snapshot** : table matérialisée(stockée) définie par une requête SQL sur d'autres tables et/ou vues. il est plus rapide que la vue

Create snapshot s-emp As select ename, sal from emp where id>100;

```
create view v-emp As select ename, sal from emp where id >100;
```

```
insert into emp values (2000, 'toto',..); select * from v-emp; select * from s-emp;
```

Dans la vue, employé est ajouté par contre non dans le snapshot. La vue est exécuté à chaque fois elle est nécessaire.

Pour rafraichir le contenu d'une snapshot on utilise refresh. La requete suivante permet de créer une snapshot qui sera rafraichit une fois par semaine : Create snapshot s-emp refresh fast with sysdate next sysdate +7 As select ename, sal from emp where id >100;

- **Snapshot log** : Table associée à la table maître d'un snapshot dans laquelle Oracle sauvegarde les mises à jour effectuées sur la table maître afin de rafraîchir le snapshot.
- **Index** : Structure d'accès pour améliorer l'efficacité des requêtes
- **Séquence** : Permet de définir des entiers tous différents
- **Synonym** : Identification secondaire d'une table ou vue.

Create synonym emp for scott.emp@db distant

Au lieu d'écrire `select * from scott.emp@db` distant, On peut écrire `select * from emp`

- **Cluster** : Regroupement de tables ayant des colonnes communes
- **Fonction** : Ensemble nommé de commandes PL/SQL renvoyant une valeur à l'appelant.
- **Procédure** : Ensemble nommé de commandes PL/SQL.
- **Trigger** : Procédure associée à un événement de MAJ.
- **Package** : Collection de fonctions, procédures et objets stockés dans une même base.
- **Profil** : Ensemble de limitations de ressources affecté à un utilisateur pour le restreindre à ses limites. Exemple : nombre de sessions simultanées, nombre d'essai de saisi de mot de passe, le temps pour rester inactif
- **Rôle** : Ensemble de privilèges attribuées à un utilisateur ou à d'autres rôles.

```
Grant select on emp to scott;
```

- **Tablespace** : Structure logique de stockage. Allocation d'espace disque

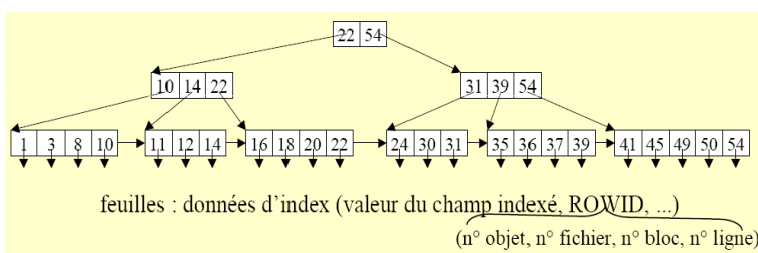
2. L'index B-tree

Index par défaut sous ORACLE et la plupart des SGBD relationnels.

```
CREATE [UNIQUE] INDEX index ON table (attributs);
```

Index intéressants pour les attributs servant fréquemment d'attribut de jointure ou de critère de recherche.

- Avantages :
 - procurent de bonnes performances pour une large gamme de requêtes avec des restrictions par égalité ou par intervalle.
 - les performances ne se dégradent pas trop lorsque la taille de la table augmente.
- Inconvénients :
 - Occupation d'espace et le temps de mise à jour.(dans le cas d'une modification et de suppression)



voir record 24min

3. Cluster

Définition : Le cluster est une organisation physique des données qui consiste à regrouper physiquement (dans un même bloc disque) les lignes d'une ou plusieurs tables ayant une caractéristique commune (une même valeur dans une ou plusieurs colonnes) constituant la clé du cluster.

La mise en cluster a trois objectifs :

- Accélérer la jointure selon la clé de cluster des tables mises en cluster,
- Accélérer la sélection des lignes d'une table ayant même valeur de clé, par le fait que ces lignes sont regroupées physiquement,
- Économiser de la place, du fait que chaque valeur de la clé du cluster ne sera stockée qu'une seule fois.

Par exemple on pourrait mettre en cluster les tables emp et dept selon n_dept. Ces tables seraient réorganisées de la façon suivante : un bloc de cluster serait créé pour chaque numéro de département, ce bloc contenant à la fois les lignes de la table emp et de la table dept correspondant à ce numéro de département. La jointure entre les tables emp et dept selon n_dept deviendrait alors beaucoup plus rapide, puisqu'elle serait déjà réalisée dans l'organisation physique des tables.

Mise en cluster d'une table

En principe c'est dès sa création qu'il faut spécifier si une table sera implantée dans un cluster.

• Exemple :

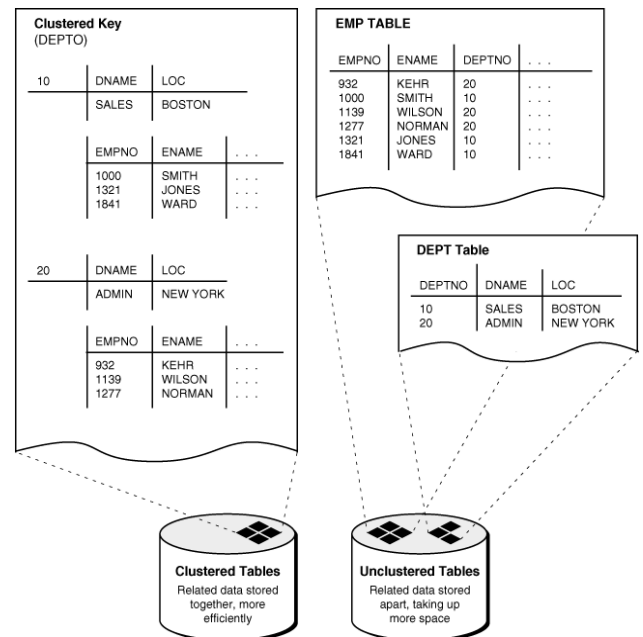
– Déclaration du cluster et de la clé de cluster

```
create cluster DeptEmp (deptno Number) index;
```

– Déclaration des tables et de leur appartenance au cluster avec attribut de cluster

```
create table Dept (deptno Number Primary Key, ...) cluster DeptEmp (deptno);
```

```
create table Emp (... , deptno Number references Dept(deptno),...) cluster DeptEmp (deptno);
```



4. Séquence

Définir une séquence équivaut à définir une suite de nombres entiers. L'évolution de cette suite est régie par un certain nombre de paramètres.

L'utilisation d'une séquence permet donc d'avoir à disposition une suite de valeurs. Ceci peut permettre de :

- générer des clés uniques dans des tables
- avoir un compteur à titre informatif, que l'on incrémente quand on veut
- etc...

La séquence sera conservée dans le dictionnaire comme tout autre objet.

Vues du dictionnaire : USER_SEQUENCES pour les séquences utilisateurs et ALL_SEQUENCES pour les séquences publiques.

Syntaxe:

```
CREATE SEQUENCE nom_séquence [INCREMENTED BY val1] [START WITH val2] [MINVALUE val3 | NOMINVALUE] [MAXVALUE val4 | NOMAXVALUE] [CYCLE | NOCYCLE]
```

val2: Valeur initiale (1 par défaut)

val1: Valeur d'incrément (1 par défaut)

val3 et val4: Valeurs minimale et maximale que peut prendre la séquence (1 et 10e27 par défaut)

CYCLE: Reprise de la valeur initiale dès que la valeur maximale est atteinte.

Modification et suppression d'une séquence:

```
ALTER SEQUENCE nom_séquence tous les paramètres sauf START WITH.
```

```
DROP SEQUENCE nom_séquence
```

Exemple

```
Create sequence es_pilote Start with 100 Increment by 10 Nomaxvalue Nocycle ;
```

Utilisation :

Insert into Pilote (no_pilote) VALUES(es_pilote.NEXTVAL) ;
es_pilote.curval : donne la valeur actuelle de la sequence

5. Table sous Oracle

Quelques types supportés par Oracle

- char(n) : Un type chaîne de caractères à longueur fixe (n caractères).
- varchar2(n) : Un type chaîne de caractères à longueur variable mais ne dépassant pas n caractères.
- number(n,d) : Un type numérique à n chiffres et à d décimales
- date : Un type permettant de représenter des dates. Attention la syntaxe diffère selon le pays ('jj/mm/aa' ou 'jj-mmm-aa').
- long : Un type texte de taille maximum 2 Go.
- raw : Un type binaire (256 octets maximum).

Quelques fonctions prédéfinies :

Les fonctions numériques :

- ABS(n) : retourne la valeur absolue de n.
- CEIL(n) : retourne l'entier par défaut.
- MOD(m,n) : calcule le reste de la division entière de m par n.
- POWER(m,n) : calcule la valeur de m élevé à la puissance n
- SIGN(n) : retourne le signe de n (-1, 0 ou 1).
- FLOOR(n) : idem mais avec l'entier par excès
- SQRT(n) : retourne la racine carrée de n.

Les fonctions sur chaînes de caractères :

- LENGTH(ch) : retourne la longueur de la chaîne.
- UPPER(ch) : met la chaîne de caractères en majuscule.
- LOWER(ch) : met la chaîne de caractères en minuscule.
- INITCAP(ch) : met la première lettre de la chaîne en majuscule, le reste en minuscule.
- LPAD(ch,l,sch) : complémente la chaîne à gauche ex) LPAD('ESSAI',10,'#@') = '#@#@#ESSAI'
- RPAD(ch,l,sch) : complémente la chaîne à droite ex) RPAD('ESSAI',10,'_') = 'ESSAI_ '
- SUBSTR(ch,d,l) : renvoie la sous chaîne spécifiée ex) SUBSTR('Dominique',2,4) = 'omin'

Les fonctions sur les dates :

- NEXT_DAY(date,'jour') : retourne la date du prochain jour après date, où 'jour' est un jour de la semaine : 'Lundi', 'Mardi', etc.
- LAST_DAY(date) : retourne le dernier jour dans le mois.
- MONTHS_BETWEEN(d1,d2) : calcul le nombre de mois entre deux dates.
- ADD_MONTHS(date,n) : ajoute n mois à la date.

6. Gestion des vues

Création des vues

- Create or Replace view AS ...
- Create Force view as ... (création de vues avec erreurs)

Suppression des vues : DROP VIEW

Exemple :

2 relations : Etudiant (N° étudiant, nom, n°projet) et Projet (n°projet, nom_prof_responsable)

Create View Affectation As Select n°étudiant, nom, nom_prof_responsable

From Etudiant, Projet Where étudiant.n° projet = projet.n° projet;

Interrogation à travers une vue : Select * from Affectation ;

Tout se passe comme s'il existait une table Affectation. En réalité, cette table est recomposée à chaque appel de la vue.

Mis à jour :

Il est possible d'effectuer des modifications de données par INSERT, DELETE et UPDATE à travers une vue, en tenant compte des restrictions suivante :

- La vue doit être construite sur une seule table
- L'ordre SELECT utilisé pour définir la vue ne doit comporter ni jointure, ni clause GROUP BY, CONNECT BY ou START WITH.
- Les colonnes résultats de l'ordre SELECT doivent être des colonnes réelles d'une table de base et non des expressions. Ex) AVG, MAX, MIN, etc ...
- La vue contient toutes les colonnes ayant l'option NOT NULL de la table de base.

EX) Create View vue_pilote As Select * From pilote Where adresse = 'Paris';

UPDATE vue_pilote SET sal = sal * 1.5 ;

=> Toute les lignes de la table de base pilote ayant Paris sont modifiées.

Informations sur les colonnes modifiables

- USER_UPDATABLE_COLUMNS : Shows all columns in all tables and views in the user's schema that are modifiable
- DBA_UPDATABLE_COLUMNS : Shows all columns in all tables and views in the DBA schema that are modifiable
- ALL_UPDATABLE_VIEWS : Shows all columns in all tables and views that are modifiable

Contrôle d'intégrité

Une vue peut être utilisée pour contrôler l'intégrité des données, grâce à la clause WITH CHECK OPTION qui interdit :

- D'insérer à travers la vue des lignes qui ne seraient pas affichées par la vue.
- De modifier une ligne de telle sorte qu'avec les nouvelles valeurs, elle ne soit plus sélectionnée par la requête de définition de la vue.

Ex) Lors de la modification ou de l'insertion d'un pilote dans la table pilote, on veut s'assurer qu'un pilote qui habite Paris a toujours une commission et qu'un pilote qui n'habite pas à Paris n'en a jamais.

Create View cr_pilote As Select * From pilote Where (adresse= 'Paris' And comm IS NOT NULL) OR (adresse != 'Paris' And comm IS NULL) WITH CHECK OPTION ;

7. Gestion des contraintes d'intégrité

Nommage des contraintes

Intérêts de nommer les contraintes

- Activer ou désactiver certaines contraintes
- Ajout/Modification/Suppression de contraintes existantes

Ajouter une contrainte à une colonne existante ALTER TABLE NomTable ADD définition_contrainte ;

ex) ALTER TABLE NomTable ADD UNIQUE Colonne

Modification et suppression : Il est possible d'activer, de désactiver ou de supprimer une contrainte par :

ALTER TABLE NomTable [DROP{ Primary Key | Unique (colonne [, colonnne]) |

CONSTRAINT NomContrainte }]

[ENABLE NomContrainte | DISABLE NomContrainte];

LES CONTRAINTES

Contrainte de colonne

Format:

[CONSTRAINT nom_contrainte] [NOT] NULL UNIQUE | PRIMARY KEY REFERENCES [schéma.]table [(colonne)]

où schéma indique le nom du créateur de la table. [ON DELETE CASCADE] | CHECK (condition]

[USING INDEX [PCTFREE entier] [INITRANS entier] [MAXTRANS entier] [TABLESPACE nomts]

[STORAGE clause]] [EXCEPTIONS INTO [schéma.]table DISABLE UNIQUE (colonne [, colonne]...) PRIMARY KEY

CONSTRAINT contrainte [CASCADE]

ALL TRIGGERS

Le paramètre **STORAGE** caractérise l'allocation d'espace disque.

Si omis: Celui de la Tablespace par défaut.

Le mot-clé **CONSTRAINT** est facultatif et sert à donner un nom à la contrainte, nom qui sera mémorisé au dictionnaire. Par défaut, ORACLE attribue un nom de la forme SYS_Cn avec n: Numéro unique dans la base.

EXCEPTIONS INTO: Table d'exceptions pour copie des lignes violant les contraintes. Cette table doit être définie auparavant.

ON DELETE CASCADE: La suppression d'une ligne parent entraîne automatiquement la suppression des lignes dépendantes. Par défaut, la ligne parent ne peut être supprimée avant toutes les lignes dépendantes.

Exemple:

La suppression du département informatique dans la table TDEPT entraîne la suppression des employés affectés à ce département dans la table TEMPL.

CHECK: Condition pour insertion ou mise à jour.

Exemples:

- Création d'une table sans contrainte particulière:

CREATE TABLE client (idcli NUMBER, nom CHAR(20), adresse CHAR(80), ville CHAR(10), codepost NUMBER(5), adhésion DATE) ;

- Création d'une table avec idcli toujours renseigné et contrôle de domaine pour le code postal:

CREATE TABLE client (idcli NUMBER NOT NULL, nom CHAR(20), adresse CHAR(80), ville CHAR(10), codepost NUMBER(5), CHECK codepost BETWEEN 10000 AND 99999, adhésion DATE);

- Création d'une table avec clé primaire et contrainte NULL:

CREATE TABLE client (idcli NUMBER CONSTRAINT u_id PRIMARY KEY, nom CHAR(20) CONSTRAINT n_nom NOT NULL, adresse CHAR(80), ville CHAR(10), codepost NUMBER(5), adhésion DATE);

- Création d'une contrainte de référence:

CREATE TABLE commande (numcom NUMBER, idclient NUMBER CONSTRAINT identif_contr REFERENCES client(idcli));

La clause Check

CREATE TABLE Climbers (Cid INTEGER, CName CHAR(20), Skill CHAR(4), Age INTEGER, PRIMARY KEY (Cid), UNIQUE (CName,Age), CHECK (age>=10 AND age<=100));

Dictionnaire pour les contraintes

Vue USER_CONSTRAINTS : Ex) desc USER_CONSTRAINTS

Pour connaître le nom de la contrainte :

Ex) Select owner, constraint_name from USER_CONSTRAINTS Where table_name='EMP' and Constraint_type = 'P' ;
(primary key)

Une fois que l'on connaît le nom, on peut examiner les colonnes associées via la vue USER_CONS_COLUMNS.

Ex) Select Column_name From USER_CONS_COLUMNS Where Constraint_name = 'Sys_c0071' ;

Exception de contraintes : EXCEPTIONS

Lorsque on active une contrainte sur des tables qui contiennent déjà des données, on peut rencontrer des violations de contraintes.

Oracle permet d'obtenir des info. Sur les lignes qui provoquent l'échec de création de contraintes. => Option

EXCEPTIONS INTO nom_table_recup

EX) Alter table emp add constraint pk_emp primary key(num) exceptions into nom_table_recup;

Nom_table_recup

- Row_id : le rowid de la ligne qui a violé la contrainte
- Owner : le propriétaire de la contrainte
- Table_name
- Constraint : le nom de la contrainte violée par la ligne

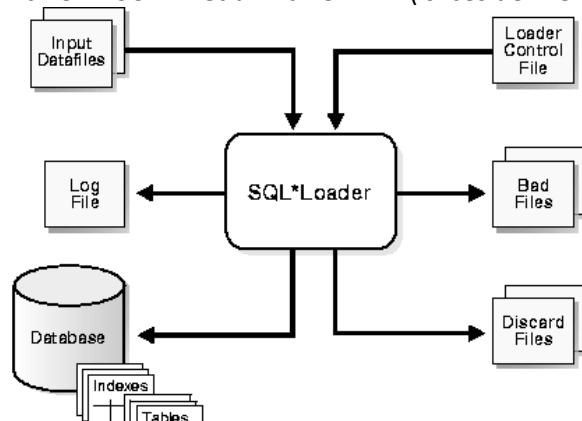
Déterminer les lignes de la table correspond à ces exceptions

ex) Select * from emp where ROWID in (select ROW_ID from nom_table_recup) ;

8. SQL*LOADER

Il est très utile (par exemple, remplissage initial d'une table) de charger des données à partir d'un fichier texte préalablement saisi. L'outil SQL*Loader (SQL LOADER) permet cela. (*)

Comme nous le voyons nous allons fournir un fichier de données, et un fichier de contrôle qui nous permette de contrôler le chargement des données. Nous allons avoir outre le chargement des données un fichier de LOG, un fichier DISCARD et un fichier BAD (si ces derniers sont paramétrés).



Le fichier de contrôle

Le fichier de contrôle est un fichier qui est écrit dans le « langage SQL*Loader ». Il va nous permettre de :

- Décrire les actions que Sql*Loader doit effectuer.
- Trouver les données à charger
- Effectuer une analyse syntaxique et interpréter les données.
- Insérer les données.

Le fichier de données

C'est un fichier plat (csv, txt ...) qui stocke les données et les séparateurs.

Le fichier : Bad Files

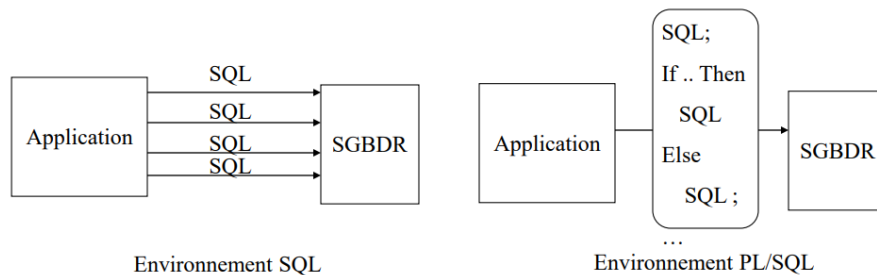
Il contient les enregistrements qui ont été rejeté soit par SQL*LOADER soit par Oracle. En effet le processus de chargement se déroule ainsi : Sql*Loader lit les lignes une à une puis les envoie à Oracle. Si une ligne venait à ne pas être conforme pour SQL*LOADER celle-ci serait rejetée et un fichier bad file serait créé ou alimenté s'il est déjà créé. Lorsque la ligne parvient à Oracle, celui-ci va voir si son insertion est possible ou pas en fonction de clé primaire, contrainte, définition de la colonne ? S'il apparaît que la ligne n'est pas en conformité avec cela alors Oracle va rejeter la ligne en suivant la même procédure que Sql*Loader.

Cours 3 : Le langage PL/SQL

1. Environnement SQL et PL/SQL

Dans l'environnement SQL, les ordres sont transmis au moteur SQL et exécutés les uns après les autres.

Dans l'environnement PL/SQL, les ordres SQL et PL/SQL sont regroupés en blocs. Un bloc ne demande qu'un seul transfert vers le moteur, qui interprète en une seule fois l'ensemble des commandes contenues dans le bloc.



2. Structure de bloc

```
[DECLARE
-- Déclarations des variables locales au bloc, constantes, ...]
BEGIN -- Instructions PL/SQL et SQL. Possibilité de blocs imbriqués
[EXCEPTION -- Traitement des erreurs]
END ;
/
```

- : commentaire sur une ligne
- /*...*/ : commentaire sur plusieurs lignes
- Déclarations multiples non autorisées .ex), i,j,k number ; -- Illicite
- Prédicats utilisés dans WHERE (and, or, not) et les opérateurs de comparaison between, in, is null, like

3. Type de données

- a. **Types scalaires** : Outre les types CHAR, NUMBER, DATE, VARCHAR2 (type natif d'Oracle), le langage PL/SQL offre les types supplémentaires suivant : BOOLEAN, INTEGER, REAL, ROWID, etc.
- b. **Types composés** :

• Enregistrement (record)

Un enregistrement, ou un RECORD en PL/SQL, permet de regrouper dans un même type un ensemble d'informations caractéristiques d'un objet déterminé. Il permet de combiner différents types de données et est défini par l'utilisateur. Les éléments d'un record ou d'un article sont appelés les champs de l'enregistrement, et peuvent être à leur tour des structures (tableaux, enregistrements...).

La déclaration d'une variable de record se fait : – Soit par référence à une structure de table ou de curseur, en utilisation – Soit par une définition d'utilisateur

```
TYPE DeptRecType IS RECORD
(nodept NUMBER(2) , nomdept dept.dnom%TYPE, ...) ;
DeptRec DeptRecType;
```

• Table

```
TYPE EnomTabType IS TABLE OF CHAR(10) NOT NULL
INDEX BY BINARY_INTEGER;
NomsTab EnomTabType; -- déclare la table PL/SQL
» Identification de 5em élément du tableau : NomsTab(5)
```

Le type admis dans une table est OBLIGATOIREMENT un type scalaire.

Tableau de tableau et tableau de record non permis (cf. record imbriqués permis donc, record de record)

4. Variable

Affectation de variables

- Le mot-clé DEFAULT initialise des variables sans utiliser l'opérateur d'affectation. Ex) taxe Number DEFAULT 10.50;
 - Constantes. Ex) tax_rate Constant Number := 0.03;
 - Affectation de variables :
- par l'opérateur :=
- par le mot réservé INTO : affecter à une variable le résultat d'une requête.

```
DECLARE TYPE t_emprec IS RECORD (r_nom pilote.nom%type, r_sal pilote.sal%type) ;
Emprec t_emprec;
BEGIN SELECT nom, sal INTO emprec FROM PILOTE WHERE NOPILOTE = '7922' ; END;
```

Instructions de contrôle

```
IF condition THEN séquence_de_commandes-1;
ELSIF condition2 THEN séquence_de_commandes-2;
ELSE séquence_de_commandes-3;
END IF;
```

<pre>LOOP ... ; IF condition THEN EXIT; END IF; END LOOP;</pre>	<pre>LOOP ... ; EXIT WHEN condition ; .. ; END LOOP;</pre>
---	--

```
WHILE condition LOOP
    séquence_de_commandes;
END LOOP;
```

```
FOR compteur IN [REVERSE] valeur_début..valeur_fin LOOP
    séquence_de_commandes;
END LOOP;
```

Visibilité d'une variable

Une variable est utilisable dans le bloc où elle est définie ainsi que les blocs imbriqués dans le bloc de définition, sauf si elle est renommée dans un bloc interne.

Note) Il ne faut pas faire :

```
DECLARE ename CHAR(10):= 'KING ' BEGIN delete from emp where ename =ename; END;
```

Puisque les noms d'attribut sont toujours prioritaires

Conversion de types

- Conversions explicites, par utilisation de fonctions SQL telles que to_date , to_char...
- Conversions implicites faites par le compilateur (à éviter)

5. Curseur

Il existe deux types de curseurs:

- Curseur explicite : c'est un curseur qui est défini par l'utilisateur. Il permet de traiter les requêtes SQL dont le résultat est constitué de plusieurs lignes.
- Curseur implicite : géré automatiquement par PL/SQL lorsqu'un curseur explicite n'a pas été déclaré.

Curseur Implicite

Les curseurs implicites sont gérées automatiquement aux cas suivants :

- Les ordres SELECT exécutés sous SQL*PLS
- Les ordres SELECT donnant une seule ligne résultat avec les autres produits (PL/SQL, SQL*FORMS, PROC, etc...)
- Ordres UPDATE, INSERT et DELETE avec tous les produits.

PL/SQL permet d'accéder à des informations même dans le cas d'un curseur implicite

- La récupération des informations se fait par l'intermédiaire des attributs du curseur implicite
- Le nom du dernier curseur implicite est SQL%

Curseur Explicite

<pre>-- FETCH nom_curseur INTO liste_variables ; DECLARE Cursor c IS SELECT nom, sal FROM pilote ; v_nom pilote.nom%type; v_sal pilote.sal%type; BEGIN OPEN C; LOOP FETCH c INTO v_nom, v_sal;</pre>	<pre>FETCH nom_curseur INTO enregistrement; DECLARE TYPE t_emp IS RECORD (v_nom pilote.nom%type, v_sal pilote.sal%type) ; r_emp t_emp ; Cursor c IS SELECT nom, sal FROM pilote ; BEGIN OPEN c;</pre>
--	---

EXIT WHEN (c%NOTFOUND) ; ... END LOOP ; CLOSE c; END;	LOOP FETCH c INTO r_emp; EXIT WHEN (c%NOTFOUND) ; ... END LOOP ; CLOSE c; END ;
---	---

```

FETCH nom_curseur INTO no_enregistrement;
DECLARE
Cursor c IS SELECT * FROM pilote ;
r_emp c%ROWTYPE ;
BEGIN
OPEN c;
LOOP FETCH c INTO r_emp;
EXIT WHEN (c%NOTFOUND) ;
...
END LOOP ;
CLOSE c;
END;

```

Curseur For...Loop

Le curseur For loop est une simplification d'écriture d'un curseur explicite. Il permet à la fois d'ouvrir le curseur, d'exécuter des Fetch pour ramener chaque ligne retournée par l'ordre SQL associé et enfin de fermer le curseur quand toutes les lignes ont été traitées.

```

DECLARE
Cursor c IS SELECT nom, sal FROM pilote ;
V_nom pilote.nom%type;
v_sal pilote.sal%type;
BEGIN
for rec_c in C loop
v_nom := rec_c.nom ;
v_sal := rec_c.sal;
end loop ;
END;

```

Curseur Paramétré

```

DECLARE
CURSOR c_empp(s1 number,s2 number) IS SELECT * FROM emp WHERE salaire between s1 and s2;
BEGIN
FOR eng IN c_empp(10000,21000) LOOP
DBMS_OUTPUT.PUT_LINE(eng.nom||' '||eng.salaire);
END LOOP;
END;

```

Curseur pour la mise à jour

Un curseur peut être utilisé pour la mise à jour d'une relation.

```

DECLARE
CURSOR cur1 IS SELECT nom, sal, comm FROM pilote WHERE nopilot BETWEEN 1280 AND 1999 FOR UPDATE;
v_nom pilote.nom%type; v_sal pilote.sal%type ; v_comm pilote.comm%type;
BEGIN
OPEN cur1;
LOOP
FETCH cur1 INTO v_nom, v_sal, v_comm;
EXIT WHEN cur1%NOTFOUND;
IF v_comm IS NULL THEN
DELETE FROM pilote WHERE CURRENT OF cur1;

```

```

ELSEIF v_comm > v_sal THEN
    UPDATE pilote SET sal = v_sal + v_comm, comm=0 WHERE CURRENT OF cur1;
END IF;
END LOOP;
CLOSE cur1;
END;

```

SQL%ROWCOUNT : retourne le nombre de lignes, de l'ensemble des lignes actives dans le curseur, ramenées par le fetch.

6. Gestion des erreurs

Gestion des erreurs standard

Déclenchée automatiquement par le système : NO_DATA_FOUND, VALUE_ERROR, LOGIN_DENIED, ZERO_DIVIDE,...

```

BEGIN
    ..
EXCEPTION
    WHEN no_data_found THEN
        .. -- traitement de l'erreur
END;

```

Gestion des erreurs (anomalies) utilisateurs

Déclenchée explicitement par l'utilisateur par la commande RAISE

```

DECLARE
    xnum NUMBER(3,1);
    yvar NUMBER(3,1);
    test1 exception; . . .
BEGIN
    . . .
    If xnum > ynum Then
        RAISE test1;
    xnum:= 15/yvar;
EXCEPTION
    When test1 Then . . .;
    When ZERO-DIVIDE Then . . .;
    When others Then . . .;
END;

```

Porté des Exceptions

Si une Exception est déclarée dans un bloc elle est : – locale au BLOC – globale pour les SOUS-BLOCS.

Propagation des Exceptions

Si une exception est déclenchée et que son traitement n'est pas défini dans le BLOC courant, alors l'exécution du BLOC courant est arrêtée et l'exception est propagée au BLOC supérieur qui devient le BLOC courant.

Si le traitement d'une exception n'est défini dans aucun BLOC, l'exception est propagée jusqu'au dernier BLOC puis un message d'erreur « unhanled exception » est renvoyé à l'environnement hôte.

Cours 4 : Procédures et fonctions stockées (sous-programmes)

Procédure et Fonction :

- Procédure : effectue une action (en général)
- Fonction : renvoie une valeur (calcul une valeur)

1. Sous-programmes

```
Create [ or replace] procedure [schéma.]nom_procédure
[(paramètre [, paramètre] ...)]
[IS|AS]
[déclarations locales]
Begin
...
[Exception manipulation des exceptions]
End [nom_procédure] ;
```

Syntaxe d'un paramètre :

- Var-name [IN | OUT | IN OUT] type_données [{:= | DEFAULT} value]
- IN : paramètre formel en entrée, OUT : en sortie, IN OUT : entrée-sortie. Le mode par défaut : IN
- Type_données : tous les types de PL/SQL sont utilisables, mais sans spécification de traille pour les types explicites

```
// Création d'un nouveau pilote
Create or replace procedure nv_pilote (x_nopilot IN pilote.nopilote%type, x_nom IN pilote.nom%type, x_sal IN
pilote.sal%type, x_comm IN pilote.comm%type)
IS
Begin
    Insert Into Pilote Values (x_nopilot, x_nom, x_sal, x_comm) ;
    Commit;
End nv_pilote;

// Suppression d'un pilote à partir de son numéro
Create or replace procedure del_pilote (x_nopilot IN pilote.nopilote%type)
IS
Begin
    Delete from Pilote where nopilot = x_nopilot ;
End del_pilote;
```

Les blocs PL/SQL de sous-programmes peuvent inclure des instructions LMD, mais pas LDD (comme create , alter, ...)

2. Fonction

```
Create [ or replace] function [schéma.]nom_fonction [(paramètre [, paramètre] ...)]
RETURN type_données
[IS|AS]
[déclarations locales] ... ;
-- bloc PL/SQL , le corps de la fonction
```

Si pas de RETURN, l'exception prédéfinie PROGRAMME-ERROR est levée.

```
Create or replace function moy_h_vol (x_codetype IN avion.type%type)
RETURN number
IS
nbhvol_avg NUMBER(8,2) := 0 /* sert à transmettre la valeur résultat */
Begin
    Select Avg(nbhvol) Into nbhvol_avg From Avion Where type = x_codetype;
    Return (nbhvol_avg);
End moy_h_vol ;
```

les paramètres

Trois modes applicables à tout sous-programme : IN (par défaut), OUT, IN OUT

Fortement conseillé de ne pas les utiliser dans les fonctions. => le mode des paramètres est en général en entrée.

- IN : Dans le sous-programme, un paramètre marqué IN se comporte comme une constante. Impossible d'avoir un paramètre IN en partie gauche d'une affectation.
- OUT : Permet à une procédure de retourner une valeur au programme appelant.
- IN OUT : combine les deux

Compilation et dictionnaire de données

Diagnostics d'erreur correspondant en utilisant les vues suivantes du dictionnaire de données :

- USER_ERROR : pour les objets procéduraux appartenant à l'utilisateur
- ALL_ERRORS : pour les objets procéduraux appartenant à l'utilisateur ou auxquels il peut accéder
- DBA_ERROR : pour les objets procéduraux de la base.
- Visualiser les erreurs associées à des objets procéduraux qui ont été créés ;

```
select line , /* n° de ligne de l'erreur */  
position, /* n° de colonne de l'erreur */  
text /* texte de message d'erreur */  
From USER_ERRORS Where name = 'del_pilote' and type = 'PROCEDURE' Order by 1
```

Note) type : PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY

- **Suppression d'un sous-programme**

Pour supprimer une procédure : Drop procedure nom_procedure ;

Pour supprimer une fonction : Drop function nom_fonction ;

Utilisation d'un sous-programme stocké

- **Transmission des paramètres**

La transmission de valeurs des paramètres en entrée (IN) peut se faire de trois façons différentes :

- Transmission par positionnement : SQL> EXECUTE nv_pilote ('3751', 'MARTIN', 25000.00, 6000.00) ;
- Transmission par nom : SQL> EXECUTE nv_pilote (x_nopilote => '3751', x_np=> 'MARTIN', x_comm => 6000.00, x_sal => 25000.00,) ;
- Transmission mixte : SQL> EXECUTE nv_pilote ('3751', 'MARTIN', x_comm => 6000.00 , x_sal => 25000.00,) ;

- **Les paramètres par défaut**

create or replace procedure create_dep (new_dep CHAR DEFAULT 'TEMP', new_loc CHAR DEFAULT 'TEMP') IS ..

SQL> Execute Create_dep ;

SQL> Execute Create_dep('VOIRIE');

SQL> Execute Create_dep('VOIRIE', 'NEW-YORK');

- **Appel aux procédures et aux fonctions**

- En mode interactif

- Appel à la procédure nv_pilote à partir d'un script SQL*PLUS ;

EXECUTE nv_pilote (10, 'toto', 2000) ;

- Appel à la fonction moy_h_vol ;

VARIABLE moyenne NUMBER

EXECUTE :moyenne := moy_h_vol('AB3');

PRINT moyenne ;

- A partir de Developer (Oracle)

- Appel à la procédure del_pilote depuis une application SQL*Forms

```
Begin  
...  
del_pilote (numero);  
...  
End ;
```

- A partir d'un programme hôte

suppression d'un pilote par une procédure hôte écrite en C faisant appel à la procédure stockée del_pilote

```
Void sup_emp() {  
    EXEC SQL BEGIN DECLARE SECTION;  
    Char numero [4];  
    EXEC SQL END DECLARE SECTION;  
    Printf("\n Entrez le numéro d'employé : ") ;  
    Scanf ("%d", &numero);  
    EXEC SQL del_pilote (:numero) ;  
}
```

- A partir d'un bloc PL/SQL Ex) appel à la procédure del_pilote à partir d'un bloc PL/SQL

```
DECLARE  
    Numero pilote.nopilot%type;  
BEGIN  
    del_pilote(numero);  
END;
```

- A partir d'un autre schéma

appel à la procédure del_pilote, créée par martin depuis le schéma durand

SQL > EXECUTE martin.del_pilote('3761');

Pour autoriser un autre utilisateur à exécuter un objet procédural vous appartenant, octroyez-lui le privilège EXECUTE sur cet objet : Grant Execute on My_PROCEDURE to durand ;

- A partir d'une base distante

appel de la procédure del_pilote, créée par martin, sur la base distante airbase, depuis durand ;

SQL> Execute martin.del_pilote@airbase('3761');

- En tant que fonction utilisateur dans les expressions SQL

Les types de données de ses arguments sont limitées à CHAR, DATE et NUMBER

Au moment de l'appel, les paramètres doivent être transmis par position et non par nom

La fonction ne doit pas comporter d'ordres INSERT, UPDATE, DELETE . Elle ne doit pas mettre de données à jour dans la base

Elle ne peut pas être utilisé dans des contraintes CHECK

Select type, moy_h_vol (type) from avion ;

Gestion des erreurs :

- L'exécution de la procédure (fonction) est considérée comme réussie, par l'environnement appelant
- La procédure (fonction) génère un diagnostic d'erreur et communique les erreurs à l'environnement.

L'environnement appelant gère l'erreur.

- Numéro_erreur : numéro fourni pour l'erreur utilisateur. Il doit être compris entre -20 000 et -20 999.

Create or replace Procedure verifi_nom (x_nopilot pilote.nopilot%type)

IS

 dif pilote.sal%type := 0 ;

Begin

 Select sal – NVL(comm, 0) Into dif From Pilote Where nopilot = x_nopilot;

 If dif < 0 Then

 RAISE_APPLICATION_ERROR(-20001, 'commission supérieure au salaire') ;

End If;

End;

Nommage de procédures et de fonctions

- Les procédures (fonctions) devraient être nommées en accord avec la fonction de gestions qu'elles exécutent. – Ex) ADD_WORKER : elle ajoute (ADD) un tuple dans la table WORKER.
- Le nom de la procédure devrait inclure celui de la table(ou des tables) qu'elle affecte.

Cours 5 : Groupement de procédure et packages

1. Structure d'un package

Deux parties dans un package :

- La partie spécification : déclaration des types, variables, constants, exceptions, curseurs et sous-programmes de type publiques.
- La partie corps (body) : Implémentation de la spécification, la définition des curseurs et sous-programmes déclarés dans la partie spécification.

2. Développement d'un package

a. Partie spécification : contient les déclarations publiques

```
Create [ or replace] Package [schéma.]nom_package
[IS|AS]
{ [déclarations de variables; ] | [déclarations de curseur ; ]
  | [déclarations d'exception; ] | [déclarations de procédure; ]
  | [déclarations de fonction; ] ... }
End [nom_package] ; /
```

b. Partie corps : contient

```
Create [ or replace] Package Body [schéma.]nom_package
[IS|AS]
{[déclaration des variables, exception, ...]
  | [définition de procédure; ] | [définition de fonction; ] ... }
[ Begin
  sequence_of_statements ]
End [nom_package] ;
```

```
Create Package pilote_work AS
  Procedure del_pilote(x_nopilot pilote.nopilot%type);
  Function moy_h_vol (xcodetype avion.type%type) Return number;
  Procedure valid_pilote (x_pilote pilote.nopilot%type) ; err_comm Exception ;
End pilote_work ;
/

Create Package Body pilote_work AS
  Procedure del_pilote(x_nopilot pilote.nopilot%type) Is
  Begin
    Delete from Pilote where nopilot = x_nopilot ;
  End del_pilote;
  Function moy_h_vol (x_codetype avion.type%type) RETURN number IS
  nbhvol_avg NUMBER(8,2) := 0 /* sert à transmettre la valeur résultat */
  Begin
    Select Avg(nbhvol) Into nbhvol_avg From Avion Where type = x_codetype;
    Return (nbhvol_avg);
  End moy_h_vol ;
  Procedure valid_pilote (x_pilote pilote.nopilot%type) Is
  dif pilote.sal%type := 0 ;
  Begin
    Select sal – NVL(comm, 0) Into dif From Pilote Where nopilot = x_nopilot;
    If dif < 0 Then RAISE err_comm End If;
    Exception When err_comm Then
      RAISE_APPLICATION_ERROR(-20001, 'commission supérieure au salaire') ;
    End valid_pilote;
  End pilote_work ;
/
```

3. Packages de curseurs

```
CREATE PACKAGE emp_actions AS
CURSOR c1 RETURN emp%ROWTYPE; -- declare cursor specification
...
```

```

END emp_actions;
/
CREATE PACKAGE BODY emp_actions AS
CURSOR c1 RETURN emp%ROWTYPE IS -- define cursor body
SELECT * FROM emp WHERE sal > 3000;
...
END emp_actions;
/

```

Initialisation d'un package

Surcharge d'une procédure ou d'une fonction

Dans un package, il est possible de définir plusieurs procédures ou fonctions avec le même nom mais avec des paramètres de type différentes.

Un package sans corps

Seuls les sous-programmes et curseurs ont une implémentation dans le corps du package. Donc, un package sans corps est possible.

```

CREATE PACKAGE trans_data AS -- bodiless package
    TYPE TimeRec IS RECORD (minutes SMALLINT, hours SMALLINT);
    TYPE TransRec IS RECORD ( category VARCHAR2, account INT, amount REAL, time_of TimeRec);
    minimum_balance CONSTANT REAL := 10.00;
    number_processed INT;
    insufficient_funds EXCEPTION;
END trans_data;

```

Ce genre de packages permet de définir des variables globales qui persistent durant une session.

Utilisation d'un package

Référencer un curseur de package dans un bloc PL/SQL :

```

OPEN emp_actions.c1;
emp_actions.fonction(param) ;

```

4. Triggers

Caractéristiques d'un déclencher

– Un programme PL/SQL stocké – Associé à une et une seule table – ORACLE exécute le déclencheur automatiquement quand une opération SQL affecte la table. – Actif ou inactif

Description d'un déclencheur

Modèle : Événement-Condition- Action

Événement est une action de mise à jour sur la table, INSERT, UPDATE ou DELETE (LMD sur une table).

Condition permet de déclencher l'action du trigger dans le cas où elle est vérifiée.

Action permet de réaliser une ou plusieurs opérations sur la base. Un bloc PL/SQL qui peut faire appel à des sous-programmes stockés en PL/SQL ou en Java.

```

Create [or replace] trigger [utilisateur.]nom_déclencheur
{ before | after | instead of }
{ delete | insert | update [of colonne [, colonne] ... ] }
[ or { delete | insert | update [of colonne [, colonne] ... ] ... }
On [utilisateur].{ Table | Vue }
[ Referencing { [OLD [AS] ancien | NEW [AS] nouveau ] } ] For each row [ when (condition) ]

```

Un seul déclencheur sur plusieurs événements

```

CREATE TRIGGER ..
BEFORE INSERT OR UPDATE Of Salaire OR DELETE
ON employé
BEGIN
    IF INSERTING THEN .. END IF;
    IF DELETING THEN .. END IF;
    IF UPDATING (Salaire) THEN .. END IF;

```

```

Create or replace trigger emp_sal

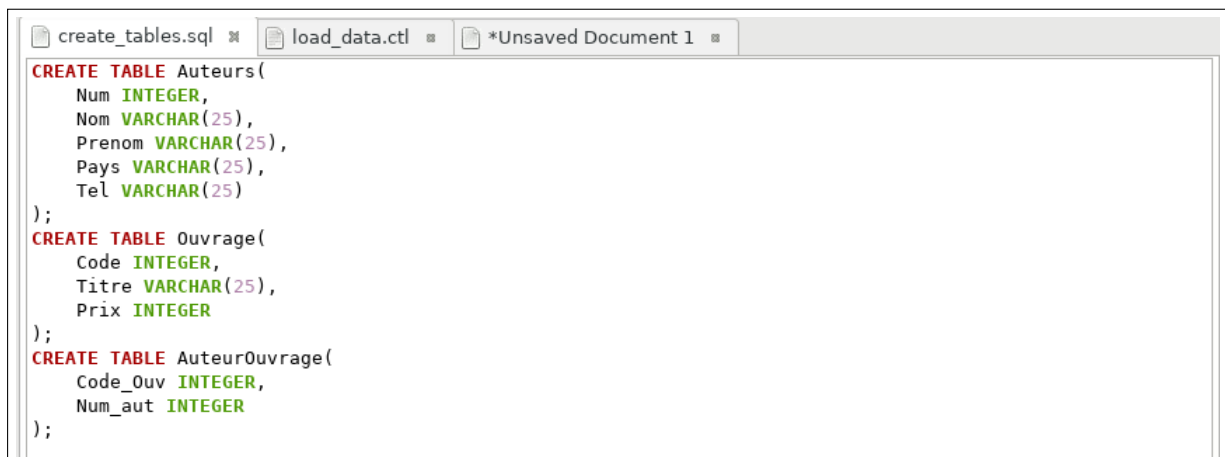
```



```
before update on sal on emp
For each row
Begin
    Dbms_output.put_lien( :old.no || 'ancien salaire : ' || :old.sal || 'nouveau salaire : ' || :new.sal);
End; /
```

```
CREATE TRIGGER vérification_salaire
Before Insert Or Update of Sal, job On emp
For each row When (new.job != 'PRESIDENT') -- attention !! Non :new dans WHEN
Declare
Sal_min Number; Sal_max Number;
Begin
    Select lsal, hsal into sal_min, sal_max from SalIntervalles where Job = :new.job ;
    if (:new.sal < sal_min OR :new.sal > sal_max) Then
        Raise_application_error (-20230, 'salaire hors norme');
    Elseif (:new.sal < :old.sal) Then
        Raise_application_error (-20231, 'salaire négative');
    Elseif (:new.sal > :old.sal * 1.1 ) Then
        Raise_application_error (-20232, 'augmentation supérieur à 10 %');
    End if;
    Exception
        WHEN no_data_found THEN Raise_application_error (-20233, 'Invalide Job ');
End ;
```

1.2 La création des tables



```
create_tables.sql  load_data.ctl  *Unsaved Document 1
CREATE TABLE Auteurs(
  Num INTEGER,
  Nom VARCHAR(25),
  Prenom VARCHAR(25),
  Pays VARCHAR(25),
  Tel VARCHAR(25)
);
CREATE TABLE Ouvrage(
  Code INTEGER,
  Titre VARCHAR(25),
  Prix INTEGER
);
CREATE TABLE AuteurOuvrage(
  Code_Ouv INTEGER,
  Num_aut INTEGER
);
```

FIGURE 3 – Création des tables



```
SQL> @/home/user10/Desktop/create_tables.sql
Table creee.

Table creee.

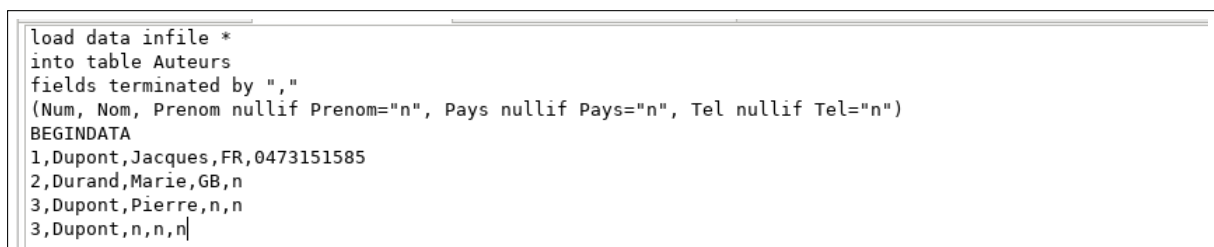
Table creee.
```

FIGURE 4 – Création des tables - Résultat

1.3 Insertion des données

1.3.1 Avec SQL Loader

J'ai crée le fichier contrôle.



```
load data infile *
into table Auteurs
fields terminated by ","
(Num, Nom, Prenom nullif Prenom="n", Pays nullif Pays="n", Tel nullif Tel="n")
BEGINDATA
1,Dupont,Jacques,FR,0473151585
2,Durand,Marie,GB,n
3,Dupont,Pierre,n,n
3,Dupont,n,n,n|
```

FIGURE 5 – fichier contrôle

Et avec la commande sqlldr, j'ai pu insérer les données écrites dans le fichier contrôle.

```

[user10@oracle-18c Desktop]$ /opt/oracle/product/18c/dbhome_1/bin/sqlldr yasaiddi
/Nada_1999 load_data.ctl

SQL*Loader: Release 18.0.0.0.0 - Production on Fri Feb 19 16:35:24 2021
Version 18.3.0.0.0

Copyright (c) 1982, 2018, Oracle and/or its affiliates. All rights reserved.

Path used:          Conventional
Commit point reached - logical record count 4

Table AUTEURS:
  4 Rows successfully loaded.

Check the log file:
  load_data.log
for more information about the load.

```

FIGURE 6 – Execution de la commande sqlldr

1.3.2 Avec la requête insert

```

create_tables.sql  load_data.ctl  insert.sql  contraintes.sql
insert into Ouvrage values(1,'Intro aux BD',260);
insert into Ouvrage values(2,'Journal de Bolivie',NULL);
insert into Ouvrage values(3,'L'homme aux sandales',NULL);
insert into AuteurOuvrage values(1,1);
insert into AuteurOuvrage values(2,2);
insert into AuteurOuvrage values(3,2);

```

FIGURE 7 – Insertion des données avec la commande insert

1.4 Ajout des contraintes

1.4.1 Les clés primaires

J'ai créé une table pour stocker les informations sur les lignes qui provoquent des problèmes lorsqu'on ajoute des contraintes.

```

create_tables.sql  load_data.ctl  insert.sql  contraintes.sql
create table pk_violation (
  row_id rowid,
  owner varchar2(120),
  table_name varchar2(120),
  constraint varchar2(120)
);
alter table Auteurs add constraint pk_auteur primary key (Num) exceptions into pk_violation;
alter table Ouvrage add constraint pk_ouvrage primary key (Code) exceptions into pk_violation;
alter table AuteurOuvrage add constraint pk_auteurOuvrage primary key (Code_Ouv,Num_aut) exceptions
pk_violation;

```

FIGURE 8 – L'ajout des contraintes

Donc la table pk_violation va contenir les lignes qui ne respectent pas les contraintes ajoutées. On va modifier la ligne erronée. On va donner un autre Num à l'enregistrement qui pose un problème.

```
SQL> select * from Auteurs where rowid in (select row_id from pk_violation);
```

NUM	NOM	PRENOM
3	Dupont	Pierre
3	Dupont	

FIGURE 9 – Le contenu de la table pk_violation

```
SQL> update Auteurs set Num=4 where Prenom='Pierre';
1 ligne mise a jour.
```

FIGURE 10 – Update de la table

1.4.2 Les noms des auteurs

J'ai modifié les lignes qui ne respectent pas la contrainte.

```
SQL> update Auteurs set Nom='DUPONT' where Num=1;
1 ligne mise a jour.
SQL> update Auteurs set Nom='DUPONT' where Num=3;
1 ligne mise a jour.
SQL> update Auteurs set Nom='DUPONT' where Num=4;
1 ligne mise a jour.
SQL> update Auteurs set Nom='DURAND' where Num=2;
1 ligne mise a jour.
```

FIGURE 11 – Update de la table - 2

```
SQL> alter table Auteurs add constraint ck_upper_check (upper(Nom) = Nom) exceptions into pk_violation;
Table modifiée.
```

FIGURE 12 – La contrainte Majuscule

1.4.3 Supprimer la contrainte clé primaire

```
SQL> alter table Auteurs drop Constraint pk_auteur;
Table modifiée.
```

FIGURE 13 – Supprimer la contrainte de la clé primaire

2 TP 1 : PL/SQL

2.1 Afficher à l'écran le nom , salaire, la commission de l'employé 'MILLER' ainsi que le nom du département dans lequel il travail.

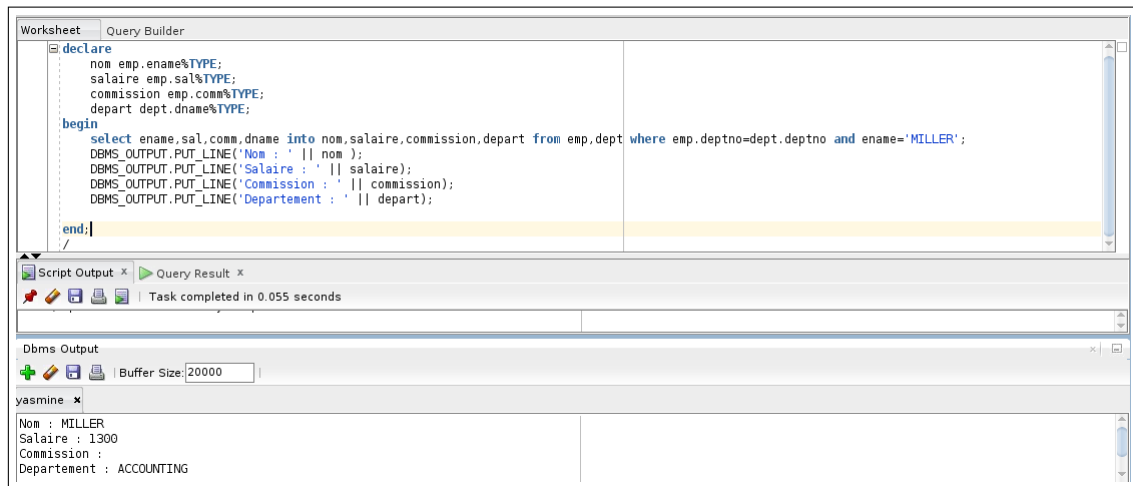


FIGURE 14 – Affichage des informations sur un employé

2.2 Ecrire un programme PL/SQL permettant d'insérer dans la table Temp dix tuples.

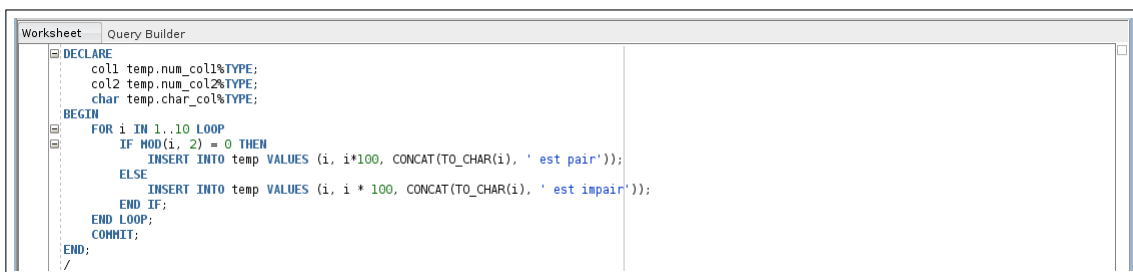


FIGURE 15 – Insertion des tuples

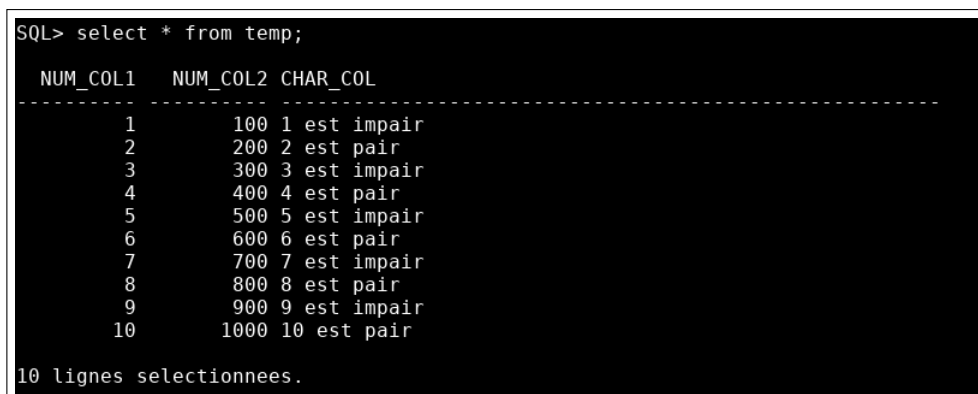


FIGURE 16 – Insertion des tuples

2.3 Insérer dans la table Temp (sal, empno, ename) les 5 employés les mieux payés.

```
Worksheet Query Builder
DECLARE
CURSOR c is select sal,empno,ename from emp order by sal;
v_sal emp.sal%TYPE;
v_empno emp.empno%TYPE;
v_enam emp.ename%TYPE;
BEGIN
open c;
for i in 1..5 loop
fetch c into v_sal, v_empno, v_enam;
insert into temp values(v_sal,v_empno,v_enam);
end loop;
close c;
END;
```

FIGURE 17 – Insertion des tuples avec les curseurs - 1 -

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
7902	FORD	ANALYST	7566	03/12/81	3000	
7934	MILLER	CLERK	7782	23/01/82	1300	
7000	SAIDI	SALESMAN	7566	17/12/80	2200	

15 lignes selectionnees.

FIGURE 18 – Résultat d'insertion avec les curseurs - 1 -

2.4 Récupérer dans une table Temp tous les employés dont les revenus mensuels (salaire + commission) sont supérieurs à 2000.

```
Worksheet Query Builder
DECLARE
cursor c is select sal, NVL(comm, 0), empno, ename from emp where sal + NVL(comm, 0) > 2000;
v_sal emp.sal%TYPE;
v_empno emp.empno%TYPE;
v_enam emp.ename%TYPE;
v_comm emp.comm%TYPE;
BEGIN
OPEN c;
LOOP
FETCH c INTO v_sal, v_comm, v_empno, v_enam;
EXIT WHEN (c%notfound);
INSERT INTO temp VALUES (v_sal + v_comm, v_empno, v_enam);
END LOOP;
commit;
END;
```

FIGURE 19 – Insertion des tuples avec les curseurs - 2 -

```
SQL> select * from temp;
```

NUM_COL1	NUM_COL2	CHAR_COL
1	100	1 est impair
2	200	2 est pair
3	300	3 est impair
4	400	4 est pair
5	500	5 est impair
6	600	6 est pair
7	700	7 est impair
8	800	8 est pair
9	900	9 est impair
10	1000	10 est pair
2975	7566	JONES
2650	7654	MARTIN
2850	7698	BLAKE
2450	7782	CLARK
3000	7788	SCOTT
5000	7839	KING
3000	7902	FORD
2200	7000	SAIDI
2975	7566	JONES
2650	7654	MARTIN
2850	7698	BLAKE
2450	7782	CLARK
3000	7788	SCOTT
5000	7839	KING
3000	7902	FORD
2200	7000	SAIDI

26 lignes selectionnees.

FIGURE 20 – Résultat d'insertion avec les curseurs - 2 -

2.5 Insérer dans une table Temp(sal, ename) le premier employé qui a un salaire supérieur à 4000 et qui est plus haut dans la chaîne de la commande que l'employé 7902.

```
SQL Worksheet History
Worksheet Query Builder

DECLARE
  cursor c is select unique sal, ename, mgr, empno from emp where sal > 4000;
  v_sal emp.sal%TYPE;
  v_enam emp.ename%TYPE;
  v_empno emp.empno%TYPE;
  v_mgr emp.mgr%TYPE;
  v_mgr_7902 emp.mgr%TYPE;
BEGIN
  OPEN c;
  LOOP
    FETCH c INTO v_sal, v_enam, v_mgr, v_empno;
    EXIT WHEN (c%notfound);
    SELECT mgr INTO v_mgr_7902 FROM emp WHERE empno=7902;
    LOOP
      EXIT WHEN (v_mgr_7902 IS NULL OR v_mgr_7902=v_mgr);
      SELECT mgr INTO v_mgr_7902 FROM emp where empno=v_mgr_7902;
      if v_mgr_7902 IS NULL OR v_mgr_7902=v_mgr then
        INSERT INTO temp VALUES (v_sal, NULL, v_enam);
      end if;
    END LOOP;
  END LOOP;
  commit;
END;
```

FIGURE 21 – Insertion des tuples avec les curseurs - 3 -


```
SQL> select * from temp;
```

NUM_COL1	NUM_COL2	CHAR_COL
----------	----------	----------

1	100	1 est impair
2	200	2 est pair
3	300	3 est impair
4	400	4 est pair
5	500	5 est impair
6	600	6 est pair
7	700	7 est impair
8	800	8 est pair
9	900	9 est impair
10	1000	10 est pair
2975	7566	JONES

NUM_COL1	NUM_COL2	CHAR_COL
----------	----------	----------

2650	7654	MARTIN
2850	7698	BLAKE
2450	7782	CLARK
3000	7788	SCOTT
5000	7839	KING
3000	7902	FORD
2200	7000	SAIDI
2975	7566	JONES
2650	7654	MARTIN
2850	7698	BLAKE
2450	7782	CLARK

NUM_COL1	NUM_COL2	CHAR_COL
----------	----------	----------

3000	7788	SCOTT
5000	7839	KING
3000	7902	FORD
2200	7000	SAIDI
5000		KING

27 lignes selectionnees.

FIGURE 22 – Résultat d'insertion avec les curseurs - 3 -

3 TP 2 : Programmation en PL/SQL

3.1 Procédures et fonctions stockées

3.1.1 Procédure createdept_votrenom (numéro_dept, dept_name, localisation).

```
CREATE OR REPLACE PROCEDURE createdept_saidi(numero_dept IN NUMBER, dept_name IN VARCHAR2, localisation IN VARCHAR2) IS
    d NUMBER;
    DUPLICATE EXCEPTION;
    CURSOR c IS SELECT deptno FROM dept;
    flag integer:=0;
    v_deptno dept.deptno%type;
BEGIN
    OPEN c;
    LOOP
        FETCH c INTO v_deptno;
        EXIT WHEN (c%notfound);
        if v_deptno=numero_dept then
            flag:=1;
        end if;
    END LOOP;
    if flag=1 then
        raise DUPLICATE;
    end if;
    DBMS_OUTPUT.PUT_LINE('Le departement a été bien ajouté');
    INSERT INTO dept VALUES(numero_dept, dept_name, localisation);
EXCEPTION
    WHEN DUPLICATE THEN
        DBMS_OUTPUT.PUT_LINE('Ce numero de departement existe deja');
END;
/
```

FIGURE 23 – Procedure - 1 -

Cette procédure nous permet d'avoir des enregistrements qui ont un deptno different.

```
SQL> SET SERVEROUTPUT ON
SQL> exec createdept_saidi(13,'yasmine','maroc');
Ce numero de departement existe deja

Procedure PL/SQL terminee avec succes.

SQL> SET SERVEROUTPUT ON
SQL> exec createdept_saidi(13,'nada','maroc');
Ce numero de departement existe deja

Procedure PL/SQL terminee avec succes.
```

FIGURE 24 – Résultat de procedure - 1 -

3.1.2 Fonction salok_votrenom (job, salaire) Return Number.

```
CREATE OR REPLACE FUNCTION salok_saidi(job IN VARCHAR2, salaire IN NUMBER) RETURN NUMBER IS
    v_lsai NUMBER;
    v_hsal NUMBER;
BEGIN
    SELECT lsai, hsal INTO v_lsai, v_hsal FROM SalIntervalle_F2 WHERE SalIntervalle_F2.job = job;
    IF salaire >= v_lsai AND salaire <= v_hsal THEN
        RETURN 1;
    ELSE
        RETURN 0;
    END IF;
EXCEPTION WHEN NO_DATA_FOUND THEN RETURN 0;
END;
/
```

FIGURE 25 – Fonction - 1 -

Pour les paramètres job='PRESEDENT' et salaire=1000, nous avons obtenu la valeur 0 :

```
Connecting to the database yas_2.  
v_Return = 0  
Process exited.  
Disconnecting from the database yas_2.
```

FIGURE 26 – Resultat de fonction - 1 -

3.1.3 Procédure raisesalary_votrenom (emp_id , $amount$).

```
CREATE OR REPLACE PROCEDURE raisesalary_saidi(emp_id IN NUMBER, amount IN NUMBER) IS  
    v_empno NUMBER;  
    v_job VARCHAR(9);  
    v_sal NUMBER;  
    v_return NUMBER;  
    e EXCEPTION;  
BEGIN  
    SELECT empno, job, sal INTO v_empno, v_job, v_sal FROM emp WHERE empno = emp_id;  
    v_return := salok_saidi(v_job, v_sal + amount);  
    IF v_return = 1 THEN  
        DBMS_OUTPUT.PUT_LINE('Le salaire de cette employé est bien modifié');  
        UPDATE emp SET sal = v_sal + amount WHERE empno = v_empno;  
    ELSE  
        raise e;  
    END IF;  
EXCEPTION  
    when e then  
        DBMS_OUTPUT.PUT_LINE('Le salaire de cette employé a déjà atteint le maximum');  
END;  
/
```

FIGURE 27 – Procedure - 2 -

Cette procédure nous permet de modifier le salaire des employés sous condition de respecter le salaire maximum. Si on essaye 1000 au salaire de l'employé numéro 7655, on aura un message d'erreur.

```
Connecting to the database yas_3.  
Le salaire de cette employé a déjà atteint le maximum  
Process exited.  
Disconnecting from the database yas_3.
```

FIGURE 28 – Résultat de procedure - 2 -

3.2 Bloc PL/SQL

```
DECLARE
  Cursor c1 IS SELECT table_name FROM user_tables WHERE table_name LIKE '%_OLD';
  Cursor c2 IS SELECT table_name FROM user_tables;
  v_table_name user_tables.table_name%TYPE;
BEGIN
  commit;
  OPEN c1;
  LOOP
    FETCH c1 INTO v_table_name;
    EXIT WHEN (c1%NOTFOUND);
    EXECUTE IMMEDIATE 'DROP TABLE ' || v_table_name;
  END LOOP;
  close c1;
  commit;
  OPEN c2;
  LOOP
    FETCH c2 INTO v_table_name;
    EXIT WHEN (c2%NOTFOUND);
    EXECUTE IMMEDIATE 'CREATE TABLE ' || v_table_name || '_old AS SELECT * FROM ' || v_table_name;
  END LOOP;
  close c2;
END;
/
```

FIGURE 29 – Backup

```
PK_VIOLATION
AUTEURS
AUTEUROUVRAGE
OUVRAGE
DEPT
EMP
VOTRENOM
TEMP
SALINTERVALLE_F2_OLD
PK_VIOLATION_OLD

TABLE_NAME
-----
AUTEURS_OLD
AUTEUROUVRAGE_OLD
OUVRAGE_OLD
DEPT_OLD
EMP_OLD
VOTRENOM_OLD
TEMP_OLD

18 lignes selectionnees.
```

FIGURE 30 – Backup - Résultat

4 TP3 : Dev. BD

4.1 Package

4.1.1 Partie spécification

```
CREATE OR REPLACE PACKAGE saidi AS
    TYPE EmpType IS RECORD (emp_no NUMBER, ename VARCHAR2(100));
    CURSOR emp_par_dep_saidi(dept_no NUMBER) RETURN EmpType;
    FUNCTION salok_saidi(v_job SalIntervalle_F2.job%type, salaire NUMBER) RETURN NUMBER;
    PROCEDURE raise_salary_saidi(emp_id NUMBER, amount NUMBER);
    PROCEDURE afficher_emp_saidi(dept_n NUMBER);
END saidi;
```

FIGURE 31 – Package - Partie specification

4.1.2 Partie corps

```
CREATE OR REPLACE PACKAGE BODY saidi IS
    CURSOR emp_par_dep_saidi(dept_no NUMBER) RETURN EmpType IS
        SELECT empno, ename FROM emp WHERE emp.deptno=dept_no;
    FUNCTION salok_saidi(v_job SalIntervalle_F2.job%type, salaire NUMBER) RETURN NUMBER IS
        v_lsal NUMBER;
        v_hsal NUMBER;
    BEGIN
        SELECT lsal, hsal INTO v_lsal, v_hsal FROM SalIntervalle_F2 WHERE SalIntervalle_F2.job=v_job;
        IF salaire <= v_hsal THEN
            RETURN 1;
        ELSE
            RETURN 0;
        END IF;
        EXCEPTION WHEN NO_DATA_FOUND THEN
            RETURN 2;
    END;
```

FIGURE 32 – Package - Partie corps - 1

```
PROCEDURE raise_salary_saidi(emp_id NUMBER, amount NUMBER) IS
    v_empno NUMBER;
    v_sal NUMBER;
    v_job VARCHAR2(100);
    v_return NUMBER;
    e EXCEPTION;
    BEGIN
        SELECT empno, job, sal INTO v_empno, v_job, v_sal FROM emp WHERE empno = emp_id;
        v_return := salok_saidi(v_job, v_sal + amount);
        IF v_return = 1 THEN
            DBMS_OUTPUT.PUT_LINE('Le salaire de cette employé est bien modifié');
            UPDATE emp SET sal = v_sal + amount WHERE empno = v_empno;
        ELSIF v_return = 0 THEN
            raise e;
        ELSE
            DBMS_OUTPUT.PUT_LINE('Données introuvable');
        END IF;
        EXCEPTION
            when e then
                DBMS_OUTPUT.PUT_LINE('Le salaire de cette employé a déjà atteint le maximum');
    END;
```

FIGURE 33 – Package - Partie corps - 2

```

PROCEDURE afficher_emp_saidi(dept_n NUMBER) IS
    v_ename VARCHAR(100);
    v_empno NUMBER;
BEGIN
    OPEN emp_par_dep_saidi(dept_n);
    LOOP
        FETCH emp_par_dep_saidi INTO v_empno, v_ename;
        EXIT WHEN emp_par_dep_saidi%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('empno = ' || v_empno || '   ename = ' || v_ename);
    END LOOP;
    CLOSE emp_par_dep_saidi;
END;
END saidi;

```

FIGURE 34 – Package - Partie corps - 3

```

11 declare
12     v number;
13 begin
14     saidi.afficher_emp_saidi(20);
15 end;
16
Script Output x Query Result x Query Result 1 x
Task completed in 0.067 seconds
PL/SQL procedure successfully completed.

Dbms Output x Compiler - Log x
Buffer Size: 20000

yasaidi x
empno = 7369 ename = SMITH
empno = 7566 ename = JONES
empno = 7788 ename = SCOTT
empno = 7876 ename = ADAMS
empno = 7902 ename = FORD
empno = 7000 ename = SAIDI

```

FIGURE 35 – Appel de la fonction *afficher_{emp}saidi*

```

11 declare
12     v number;
13 begin
14     saidi.raise_salary_saidi(7369,100);
15 end;
16
17
Script Output x Query Result x Query Result 1 x
Task completed in 0.027 seconds

Dbms Output x Compiler - Log x
Buffer Size: 20000

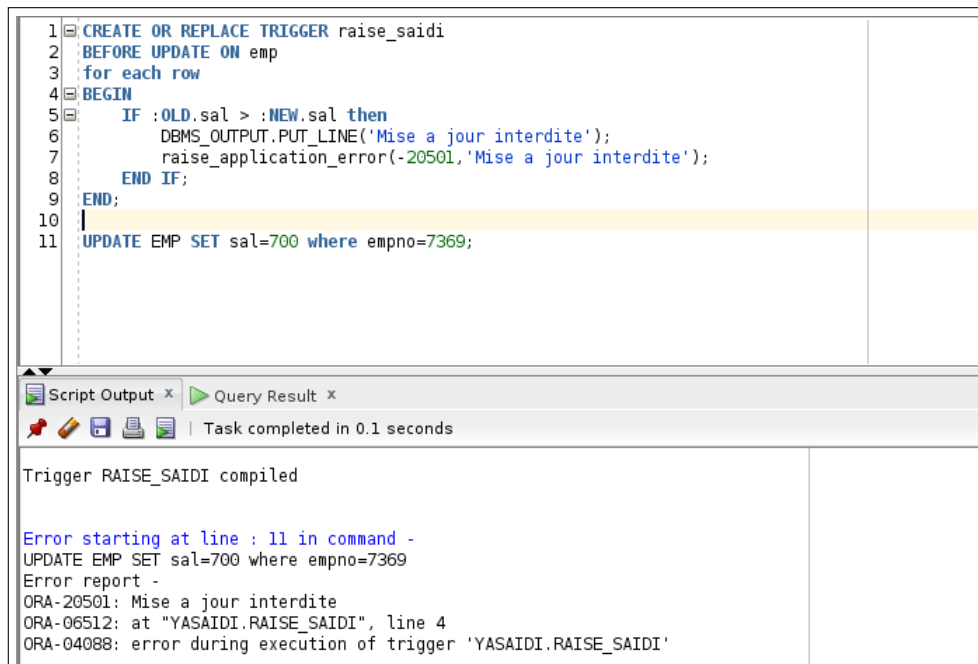
yasaidi x
Le salaire de cette employé est bien modifié

```

FIGURE 36 – Appel de la fonction *raise_salary_saidi*

4.2 Triggers

4.2.1 (Nom du trigger : raise_votrenom) Le salaire d'un employé ne diminue jamais.



```
1 CREATE OR REPLACE TRIGGER raise_saidi
2 BEFORE UPDATE ON emp
3 FOR EACH ROW
4 BEGIN
5     IF :OLD.sal > :NEW.sal then
6         DBMS_OUTPUT.PUT_LINE('Mise a jour interdite');
7         raise_application_error(-20501,'Mise a jour interdite');
8     END IF;
9 END;
10
11 UPDATE EMP SET sal=700 where empno=7369;
```

Script Output x Query Result x

Task completed in 0.1 seconds

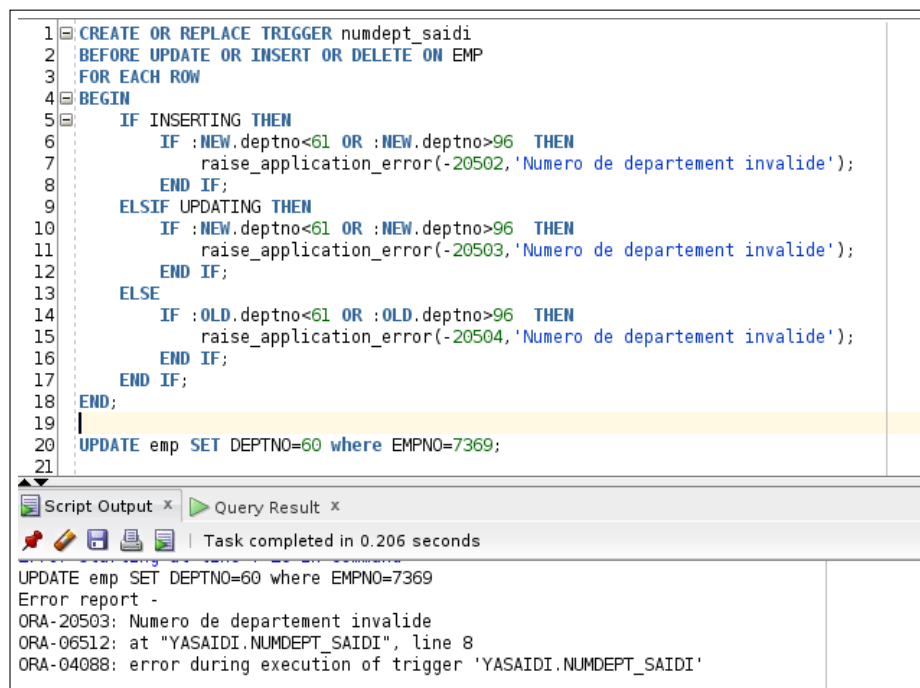
Trigger RAISE_SAIDI compiled

Error starting at line : 11 in command -
UPDATE EMP SET sal=700 where empno=7369
Error report -
ORA-20501: Mise a jour interdite
ORA-06512: at "YASAIID.RAISE_SAIDI", line 4
ORA-04088: error during execution of trigger 'YASAIID.RAISE_SAIDI'

FIGURE 37 – Trigger - raise_saidi

4.2.2 (Nom du trigger : numdept_votrenom)

Le numéro de département doit être entre 61 - 69.



```
1 CREATE OR REPLACE TRIGGER numdept_saidi
2 BEFORE UPDATE OR INSERT OR DELETE ON EMP
3 FOR EACH ROW
4 BEGIN
5     IF INSERTING THEN
6         IF :NEW.deptno<61 OR :NEW.deptno>96 THEN
7             raise_application_error(-20502,'Numero de departement invalide');
8         END IF;
9     ELSIF UPDATING THEN
10        IF :NEW.deptno<61 OR :NEW.deptno>96 THEN
11            raise_application_error(-20503,'Numero de departement invalide');
12        END IF;
13    ELSE
14        IF :OLD.deptno<61 OR :OLD.deptno>96 THEN
15            raise_application_error(-20504,'Numero de departement invalide');
16        END IF;
17    END IF;
18 END;
19
20 UPDATE emp SET DEPTNO=60 where EMPNO=7369;
```

Script Output x Query Result x

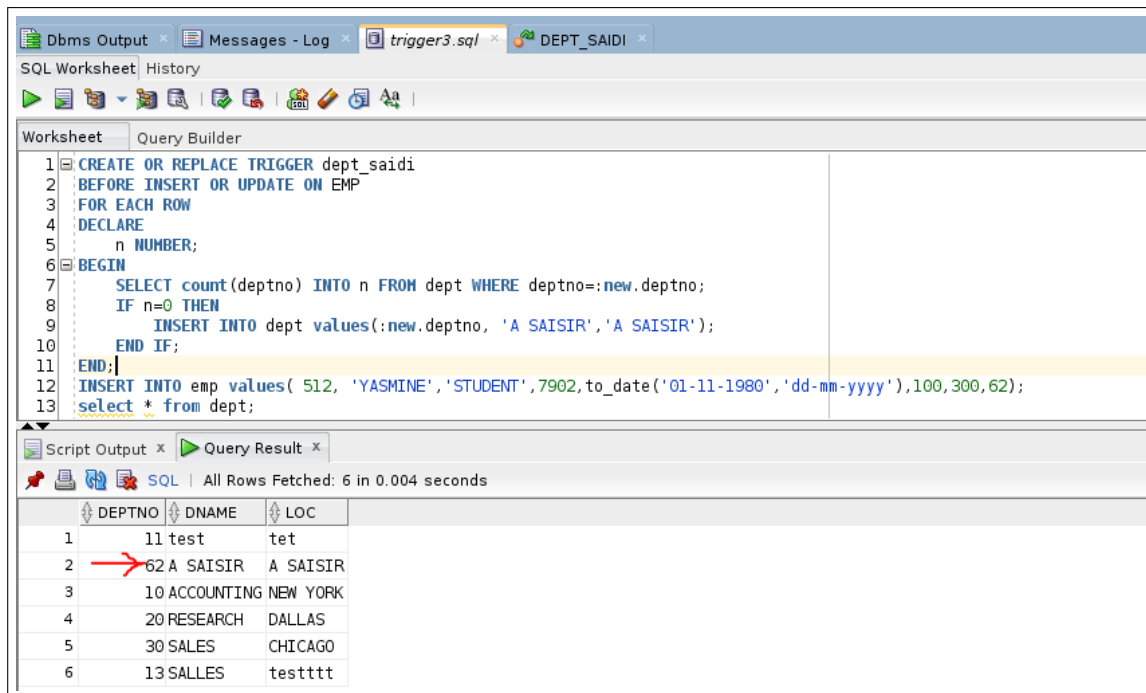
Task completed in 0.206 seconds

UPDATE emp SET DEPTNO=60 where EMPNO=7369
Error report -
ORA-20503: Numero de departement invalide
ORA-06512: at "YASAIID.NUMDEPT_SAIDI", line 8
ORA-04088: error during execution of trigger 'YASAIID.NUMDEPT_SAIDI'

FIGURE 38 – Trigger - numdept_saidi

4.2.3 (Nom du trigger : dept_votrenom)

Si un employé est affecté à un département qui n'existe pas dans la base de données, ce département doit être rajouté avec pour valeur « A SAISIR » pour les attributs Dname et Loc.



```
1 CREATE OR REPLACE TRIGGER dept_saidi
2 BEFORE INSERT OR UPDATE ON EMP
3 FOR EACH ROW
4 DECLARE
5     n NUMBER;
6 BEGIN
7     SELECT count(deptno) INTO n FROM dept WHERE deptno=:new.deptno;
8     IF n=0 THEN
9         INSERT INTO dept values(:new.deptno, 'A SAISIR', 'A SAISIR');
10    END IF;
11 END;
12 INSERT INTO emp values( 512, 'YASMINE', 'STUDENT', 7902, to_date('01-11-1980', 'dd-mm-yyyy'), 100, 300, 62);
13 select * from dept;
```

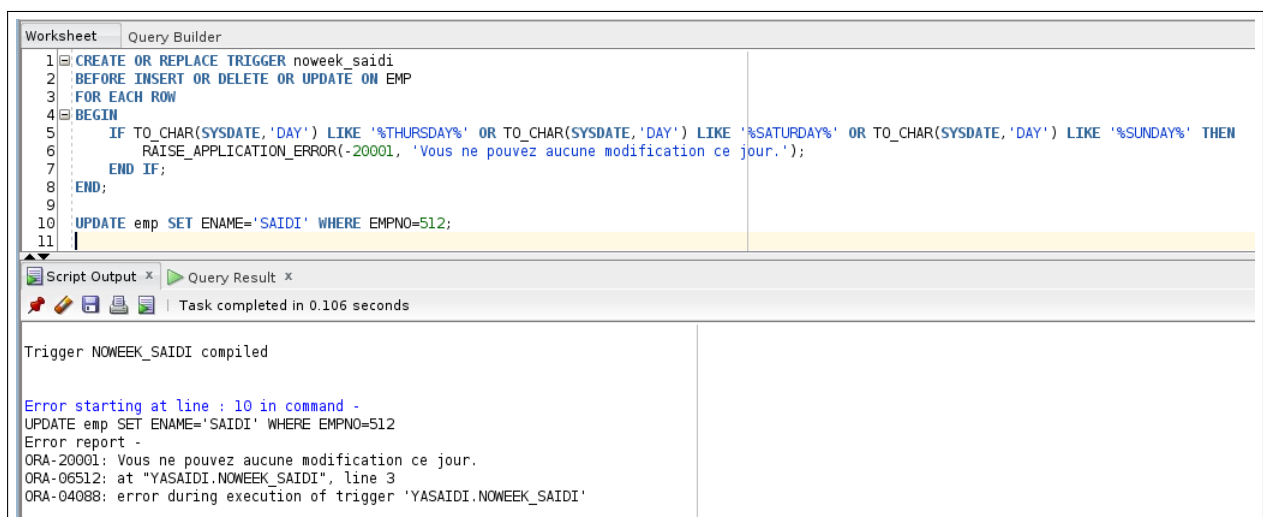
	DEPTNO	DNAME	LOC
1	11	test	tet
2	62	A SAISIR	A SAISIR
3	10	ACCOUNTING	NEW YORK
4	20	RESEARCH	DALLAS
5	30	SALES	CHICAGO
6	13	SALLES	testttt

FIGURE 39 – Trigger - dept_saidi

4.2.4 (Nom du trigger : noweek_votrenom)

Pour des raisons de sécurité, on souhaite interdire toute modification de la relation employé pendant le week-end (samedi et dimanche).

j'ai interdit les modifications pendant jeudi aussi pour faire le test.



```
1 CREATE OR REPLACE TRIGGER noweek_saidi
2 BEFORE INSERT OR DELETE OR UPDATE ON EMP
3 FOR EACH ROW
4 BEGIN
5     IF TO_CHAR(SYSDATE, 'DAY') LIKE '%THURSDAY%' OR TO_CHAR(SYSDATE, 'DAY') LIKE '%SATURDAY%' OR TO_CHAR(SYSDATE, 'DAY') LIKE '%SUNDAY%' THEN
6         RAISE_APPLICATION_ERROR(-20001, 'Vous ne pouvez aucune modification ce jour. ');
7     END IF;
8 END;
9
10 UPDATE emp SET ENAME='SAIDI' WHERE EMPNO=512;
11
```

Trigger NOWEEK_SAIDI compiled

Error starting at line : 10 in command -
UPDATE emp SET ENAME='SAIDI' WHERE EMPNO=512
Error report -
ORA-20001: Vous ne pouvez aucune modification ce jour.
ORA-06512: at "YASAI.DI.NOWEEK_SAIDI", line 3
ORA-04088: error during execution of trigger 'YASAI.DI.NOWEEK_SAIDI'

FIGURE 40 – Trigger - noweek_saidi

4.2.5 Désactiver le trigger nowweek_saidi

Pour désactiver le trigger, j'ai utilisé DISABLE.

4.2.6 Réactiver le trigger nowweek_saidi

Pour réaliser cette question, j'ai utilisé ENABLE.

```
10 ALTER TRIGGER nowweek_saidi DISABLE;  
11 ALTER TRIGGER nowweek_saidi ENABLE;
```

FIGURE 41 – Désactiver et réactiver le trigger nowweek_saidi

4.2.7 (Nom du trigger : stat_votrenom) On souhaite conserver des statistiques concernant les mises à jour sur la table EMP.

```
1 CREATE TABLE STATS_saidi (  
2     TypeMaj VARCHAR2(9),  
3     NbMaj NUMBER,  
4     Date_derniere_Maj DATE  
5 );  
6  
7 INSERT INTO STATS_saidi VALUES ('INSERT',0,NULL);  
8 INSERT INTO STATS_saidi VALUES ('UPDATE',0,NULL);  
9 INSERT INTO STATS_saidi VALUES ('DELETE',0,NULL);
```

FIGURE 42 – Création et remplissage de la table STATS_saidi

```
11 CREATE OR REPLACE TRIGGER stats_saidi  
12 BEFORE UPDATE OR INSERT OR DELETE ON emp  
13 BEGIN  
14     IF INSERTING THEN  
15         UPDATE STATS_SAIDI SET NbMaj = NbMaj +1 WHERE TypeMaj='INSERT';  
16         UPDATE STATS_SAIDI SET Date_derniere_Maj = SYSDATE WHERE TypeMaj='INSERT';  
17     ELSIF UPDATING THEN  
18         UPDATE STATS_SAIDI SET NbMaj = NbMaj +1 WHERE TypeMaj='UPDATE';  
19         UPDATE STATS_SAIDI SET Date_derniere_Maj = SYSDATE WHERE TypeMaj='UPDATE';  
20     ELSE  
21         UPDATE STATS_SAIDI SET NbMaj = NbMaj +1 WHERE TypeMaj='DELETE';  
22         UPDATE STATS_SAIDI SET Date_derniere_Maj = SYSDATE WHERE TypeMaj='DELETE';  
23     END IF;  
24 END;  
25 UPDATE emp SET job='etudiant' where empno=512;  
26 select * from STATS_SAIDI;
```

Script Output x Query Result x Query Result 1 x

SQL | All Rows Fetched: 3 in 0.007 seconds

	TYPEMAJ	NBMAJ	DATE_DERNIERE_MAJ
1	INSERT	0 (null)	
2	UPDATE	6	26-FEB-21
3	DELETE	0 (null)	

FIGURE 43 – Trigger - STATS_saidi

La clause FOR EACH ROW est importante dans le cas d'une requete qui requete qui retourne plusieurs lignes.

```

26 CREATE OR REPLACE TRIGGER foreach_saidi
27 BEFORE UPDATE ON emp
28 BEGIN
29     UPDATE emp SET SAL = SAL*1.5;
30 END;
31 UPDATE emp SET job = 'STUDENT' WHERE empno=512;

```

Script Output x Query Result x Query Result 1 x

Task completed in 0.102 seconds

```

ORA-04088: error during execution of trigger 'YASAI.DI.FOREACH_SAIDI'
ORA-06512: at "YASAI.DI.FOREACH_SAIDI", line 2
ORA-04088: error during execution of trigger 'YASAI.DI.FOREACH_SAIDI'
ORA-06512: at "YASAI.DI.FOREACH_SAIDI", line 2

```

FIGURE 44 – L'importance de la clause FOR EACH ROW

4.2.8 (Nom du trigeur : checksal_votrenom)

```

1 CREATE OR REPLACE TRIGGER checksal_saidi
2 BEFORE UPDATE OF JOB ON EMP
3 FOR EACH ROW
4 DECLARE
5     NSAL NUMBER;
6     V_LSAL NUMBER;
7     V_HSAL NUMBER;
8 BEGIN
9     IF :OLD.job!='PRESIDENT' THEN
10        SELECT LSAL,HSAL INTO V_LSAL,V_HSAL FROM SALINTERVALLE_F2 WHERE SALINTERVALLE_F2.JOB= :OLD.JOB;
11        NSAL := :OLD.SAL+100;
12        IF NSAL>V_LSAL AND NSAL<V_HSAL THEN
13            :NEW.SAL := NSAL;
14        ELSIF NSAL<V_LSAL THEN
15            :NEW.SAL := V_LSAL;
16        ELSE
17            :NEW.SAL := V_HSAL;
18        END IF;
19    END IF;
20 END;

```

FIGURE 45 – Trigger - checksal_saidi