Cours 2 : PL/SQL
Procedural Language/SQL

Blocs, variables, instructions, structures de contrôle, curseurs, gestion des erreurs, procédures/fonctions stockées, packages, triggers

PL/SQL

- Chapitre 3 de la norme SQL3 sous le nom SQL/PSM (Persistent Stored Modules)
- Langage procédural plus portable
- Un script SQL Developeur peut contenir des blocs de sous-programmes en PL/SQL
- Traitement de transactions

PL/SQL (2)

- Construction de procédures ou fonctions stockées qui améliorent le mode client-serveur par stockage des procédures ou fonctions souvent utilisées au niveau serveur
- Gestion des erreurs (à la ADA)
- Construction de triggers (ou déclencheurs)

Structure d’un bloc

- Un programme ou une procédure PL/SQL est un ensemble de un ou plusieurs blocs. Chaque bloc comporte trois sections :

1. Section déclaration
2. Section corps du bloc
3. Section traitement des erreurs

1. Section déclaration

- Contient la description des structures et des variables utilisées dans le bloc
- Section facultative
- Commence par le mot clé **DECLARE**

2. Section corps du bloc

- Contient les instructions du programme et éventuellement, à la fin, la section de traitement des erreurs
- Obligatoire
- Introduite par le mot clé **BEGIN**
- Se termine par le mot clé **END**

3. Section traitement des erreurs

- Facultative
- Introduite par le mot clé **EXCEPTION**

Syntaxe

DECLARE
déclaration
BEGIN
corps-du-bloc
EXCEPTION
traitement-des-erreurs
END;
/ ← A ajouter obligatoirement
dans l'exécution d'un script

Exemple

```
SET SERVEROUTPUT ON
DECLARE
 x VARCHAR2(10);
BEGIN
 x := 'Bonjour';
 DBMS_OUTPUT.PUT_LINE(x);
END;
/
```

Exemple (2)

```
DECLARE
 erreurEx EXCEPTION;
 num exemplaire.numExemplaire%TYPE;
 film exemplaire.numFilm%TYPE;
 pb exemplaire.probleme%TYPE;
BEGIN
 ...
```

Exemple (2 – suite)

```
...
BEGIN
 SELECT numExemplaire, numFilm, probleme
 INTO num, film, pb
 FROM exemplaire WHERE numExemplaire = 1;
 IF probleme IS NOT NULL
   THEN RAISE erreurEx; END IF;
 DBMS_OUTPUT.PUT_LINE (num || ' OK');
EXCEPTION
...
```

Exemple (2 – suite et fin)

```
...
EXCEPTION
 WHEN NO_DATA_FOUND THEN
  DBMS_OUTPUT.PUT_LINE('numéro inconnu');
 WHEN erreurEx THEN
  DBMS_OUTPUT.PUT_LINE(num || ' problème');
END;
/
```

Types de variables

- Variables scalaires
- Types composés
  - Enregistrement (record)
  - Table

Variables scalaires

- Types issus de SQL : **CHAR**, **NUMBER**, **DATE**, **VARCHAR2**
- Types PL/SQL : **BOOLEAN**, **SMALLINT**, **BINARY_INTEGER**, **DECIMAL**, **FLOAT**, **INTEGER**, **REAL**, **ROWID**
- Les variables hôtes sont préfixées par « **:** »

Déclaration des variables scalaires

- ***nom-variable nom-du-type*;**
  - **Exemple :**
    x VARCHAR2(10);

- ***nom-variable nom-table.nom-attribut*%TYPE;**
  - **Exemple :**
    film exemplaire.numFilm%TYPE;

Déclaration pour un enregistrement

- Soit par référence à une structure de table ou de curseur en utilisant **ROWTYPE :**

  ***nom-variable nom-table*%ROWTYPE;**

  ***nom-variable nom-curseur*%ROWTYPE;**

Exemple

```
DECLARE
 rec film%ROWTYPE;
BEGIN
 SELECT * INTO rec
 FROM film WHERE numFilm = 2210;
 DBMS_OUTPUT.PUT_LINE (rec.numfilm || ' ' ||
        rec.titre  || ' ' || rec.realisateur);
END;
/
```

Déclaration pour un enregistrement (2)

- Soit par énumération des rubriques qui la composent. Cela se fait en deux étapes :
  - Déclaration du type enregistrement

    TYPE *nom-du-type-record* IS RECORD (
    ***nom-attribut*$_1$ *type-attribut*$_1$,**
    ***nom-attribut*$_2$ *type-attribut*$_2$, ...);**
  - Déclaration de la variable de type enregistrement

    ***nom-variable nom-du-type-record*;**

Exemple

```
DECLARE
 TYPE type_film IS RECORD (
 num INTEGER, titre film.titre%TYPE, real INTEGER);
 rec type_film;
BEGIN
 SELECT * INTO rec
 FROM film WHERE numFilm = 2210;
 DBMS_OUTPUT.PUT_LINE (rec.num || ' ' || rec.titre
          || ' ' || rec.real);
END;
/
```

Tables

- Structure composée d'éléments d'un même type ***scalaire***
- L'accès à un élément de la table s'effectue grâce à un indice, ou clé primaire
- Cet index est déclaré de type **BINARY_INTEGER** (valeurs entières signées)

Déclaration pour une table

- Deux étapes :
  - Déclaration du type de l'élément de la table
  - Déclaration de la variable de type table

Déclaration pour une table (2)

- Déclaration du type de l'élément de la table :

  TYPE *nom-du-type-table*
  IS TABLE OF *type-argument*
  INDEX BY BINARY_INTEGER;

- Déclaration de la variable de type table :

  ***nom-variable nom-du-type-table*;**

Exemple

```
DECLARE
 TYPE tabNom IS TABLE OF VARCHAR2(20)
 INDEX BY BINARY_INTEGER;
 tableNom tabNom;
 i BINARY_INTEGER;
BEGIN
 tableNom(5) := 'Dupont';
 i := 10;
 tableNom(i) := 'Dupond';
END;
/
```

Variables (scalaires ou composées)

- Valeur initiale :

  nom-variable nom-type := valeur;

- Constante :

  ***nom-variable nom-type* DEFAULT *valeur*;**

  ou

  ***nom-variable* CONSTANT*nom-type* := *valeur*;**

Variables (scalaires ou composées) (2)

- Visibilité : une variable est utilisable dans le bloc où elle a été définie ainsi que dans les blocs imbriqués dans le bloc de définition, sauf si elle est redéfinie dans un bloc interne

Conversion de type

- Explicite avec

  TO_CHAR, TO_DATE, TO_NUMBER, RAWTOHEX, HEXTORAW

- Implicites, par conversion automatique

Instructions

- Affectations
- Instructions du langage SQL : CLOSE, COMMIT, DELETE, FETCH, INSERT, LOCK, OPEN, ROLLBACK, SAVEPOINT, SELECT, SET TRANSACTION, UPDATE

Instructions (2)

- Instructions de contrôle itératif ou répétitif
- Instructions de gestion de curseurs
- Instructions de gestion des erreurs

Affectation

- Opérateur d’affectation **:=**
- Option **INTO** dans un ordre **SELECT**
- Instruction **FETCH** avec un curseur

Exemple

```
DECLARE
 TYPE recFilm IS RECORD (
       titre film.titre%TYPE,
       realisateur INTEGER);
 monFilm recFilm;
 convers NUMBER(6,5);
BEGIN
 convers := 6.55957;
 monFilm.titre := 'The Pillow Book';
 monFilm.realisateur := 2013;
END;
/
```

Exemple (2)

```
DECLARE
 v_titre film.titre%TYPE;
 v_real film.realisateur%TYPE;
BEGIN
 SELECT titre, realisateur INTO v_titre, v_real
 FROM film WHERE numFilm = 11300;
 DBMS_OUTPUT.PUT_LINE (v_titre ||' '|| v_real);
END;
/
```

Exemple (3)

```
DECLARE
TYPE recFilm IS RECORD (
  r_titre film.titre%TYPE,
  r_real film.realisateur%TYPE);
  r_film recFilm;
BEGIN
 SELECT titre, realisateur INTO r_film
 FROM film WHERE numFilm = 11300;
 DBMS_OUTPUT.PUT_LINE (r_film.r_titre ||' '||
       r_film.r_real);
END;
/
```

Exemple (4)

```
DECLARE
 r_film film%ROWTYPE;
BEGIN
 SELECT * INTO r_film
 FROM film WHERE numFilm = 11300;
 DBMS_OUTPUT.PUT_LINE (r_film.titre ||' '||
      r_film.realisateur);
END;
/
```

Structures de contrôle

- Structure alternative
- Structure répétitives

Structures alternatives

IF *condition* THEN *instructions*;
END IF;

IF *condition* THEN *instructions*;
ELSE *instructions*; END IF;

IF *condition* THEN *instructions*;
ELSIF *condition* THEN *instructions*;
ELSE *instructions*; END IF;

Structures répétitives

LOOP *instructions*; END LOOP;

LOOP *instructions*; ...
EXIT WHEN *condition;* ...
END LOOP;

LOOP ...
IF *condition* THEN EXIT; END IF;
... END LOOP;

Structures répétitives (2)

FOR ***variable-indice*** **IN [REVERSE]**
val-début **..** ***val-fin***
LOOP ***instructions*****; END LOOP;**

- ***variable-indice*** est une variable locale (locale à la boucle) non déclarée
- ***val-début*** et ***val-fin*** sont des variables locales déclarées et initialisées ou alors des constantes
- le pas est -1 si **REVERSE** est présent, sinon il est égal à +1

Structures répétitives (3)

WHILE ***condition***
LOOP
instructions**;**
END LOOP;

Les curseurs

- Il y a création d'un curseur dès qu'on exécute une instruction SQL. C'est une zone de travail de l'environnement utilisateur qui contient les informations relatives à l'instruction SQL :
  - Le texte source de l'ordre SQL
  - Le texte «compilé» de l'ordre SQL
  - Un tampon pour une ligne du résultat
  - Le statut (*cursor status*)
  - Des informations de travail et de contrôle

Curseurs implicites

- Gérés automatiquement par le noyau dans les cas suivants :
  - Une instruction **SELECT** exécutée sous SQL Developer
  - Une instruction **SELECT** donnant une seule ligne de résultat sous PL/SQL
  - Les instructions **UPDATE**, **INSERT** et **DELETE**
  - …

Curseurs explicites

- Obligatoires pour un **SELECT** susceptible de produire plusieurs lignes résultat
- Quatre étapes :
  1) Déclaration du curseur
  2) Ouverture du curseur
  3) Traitement des lignes du résultat
  4) Fermeture du curseur

1) Déclaration du curseur

- Association d'un nom de curseur à une requête **SELECT**
- Se fait dans la section **DECLARE** d'un bloc PL/SQL

CURSOR *nom-curseur* IS *requête*;

- Un curseur peut être paramétré :

CURSOR *nom-curseur* (*nom-p*$_1$ *type-p*$_1$ [:= *val-défaut*], ...) IS *requête*;

Exemple

```
DECLARE
 CURSOR C1 IS SELECT numIndividu, nomIndividu
 FROM individu WHERE nomindividu LIKE 'A% ';
 CURSOR C2 (p NUMBER(4),q NUMBER(4)) IS
 SELECT titre FROM film
 WHERE realisateur >= p AND realisateur <= q;
BEGIN
 ...
```

2) Ouverture d'un curseur

- Alloue un espace mémoire au curseur et positionne les éventuels verrous

  OPEN ***nom-curseur*****;**

  ou

  OPEN ***nom-curseur*****(*****liste-par-effectifs*****);**

- Pour les paramètres, association par position ou par nom sous la forme

  paramètre-formel **=>** ***paramètre-réel***

Exemple

- **OPEN C1;**
- **OPEN C2 (1600, 1800);**
- **OPEN C2 (q => 1800, p => 1600);**

3) Traitement des lignes

- Autant d'instructions **FETCH** que de lignes résultats :

FETCH ***nom-curseur*** **INTO** ***liste-variables*****;**

ou

FETCH ***nom-curseur*** **INTO** ***nom-enregistrement*****;**

- Au moins quatre formes possibles

Première forme : exemple

```
DECLARE
 CURSOR C1 IS SELECT numIndividu, nomindividu
 FROM individu WHERE nomindividu LIKE 'A% ';
 v_num individu.numIndividu%TYPE;
 v_nom individu.nomIndividu%TYPE;
BEGIN
 OPEN C1; LOOP FETCH C1 INTO v_num, v_nom;
 EXIT WHEN C1%NOTFOUND;
 dbms_output.put_line(v_num|| ' ' ||v_nom) ;
 END LOOP; CLOSE C1;
END;
```

Deuxième forme : exemple

```
DECLARE
 CURSOR C1 IS SELECT numIndividu, nomindividu
 FROM individu WHERE nomindividu LIKE 'A% ';
 TYPE recINDIVIDU IS RECORD(
 v_num individu.numIndividu%TYPE,
 v_nom individu.nomIndividu%TYPE);
 r_ind recINDIVIDU;
BEGIN
 OPEN C1; LOOP FETCH C1 INTO r_ind;
 EXIT WHEN C1%NOTFOUND;
 dbms_output.put_line(r_ind.v_num|| ' ' ||r_ind.v_nom) ;
 END LOOP; CLOSE C1;
END;
```

Troisième forme : exemple

```
DECLARE
 CURSOR C1 IS SELECT numIndividu, nomindividu
 FROM individu WHERE nomindividu LIKE 'A% ';
 r_ind C1%ROWTYPE;
BEGIN
 OPEN C1; LOOP FETCH C1 INTO r_ind;
 EXIT WHEN C1%NOTFOUND;
 dbms_output.put_line(r_ind.numIndividu || ' ' ||
      r_ind.nomIndividu) ;
 END LOOP; CLOSE C1;
END;
```

Quatrième forme : exemple

```
DECLARE
 CURSOR C1 IS SELECT numIndividu, nomindividu
 FROM individu WHERE nomindividu LIKE 'A% ';
BEGIN
 FOR r_ind IN C1 LOOP
 dbms_output.put_line(r_ind.numindividu || ' ' ||
     r_ind.nomindividu) ;
 END LOOP;
END;
```

Statut d’un curseur

Attribut	Valeur
%FOUND	Vrai si exécution correcte de l’ordre SQL
%NOTFOUND	Vrai si exécution incorrecte de l’ordre SQL
%ISOPEN	Vrai si curseur ouvert
%ROWCOUNT	Nombre de lignes traitées par l’ordre SQL, évolue à chaque ligne traitée par un **FETCH** (zéro au départ)

Statut d’un curseur (2)

Curseur implicite	Curseur explicite
SQL%FOUND	*nom-curseur*%FOUND
SQL%NOTFOUND	*nom-curseur*%NOTFOUND
SQL%ISOPEN	*nom-curseur*%ISOPEN
SQL%ROWCOUNT	*nom-curseur*%ROWCOUNT

Modification des données

- Se fait habituellement avec **INSERT**, **UPDATE** ou **DELETE**
- Possibilité d'utiliser la clause **FOR UPDATE** dans la déclaration du curseur. Cela permet d'utiliser la clause

  CURRENT OF *nom-curseur*

  dans la clause **WHERE** des instructions **UPDATE** et **DELETE**. Cela permet de modifier la ligne du curseur traitée par le dernier **FETCH**, et donc d'accélérer l'accès à cette ligne

Exemple

```
DECLARE
 CURSOR C IS SELECT * FROM location
 WHERE dateEnvoi IS NULL FOR UPDATE OF dateEnvoi;
 newDate DATE;
BEGIN
 FOR rec IN C LOOP
 IF rec.dateLocation < '01-02-2020' THEN newDate := SYSDATE;
 ELSIF rec.dateLocation < '08-02-2020' THEN newDate := SYSDATE+1;
 ELSE newDate := NULL; END IF;
 DBMS_OUTPUT.PUT_LINE('OK '||rec.numExemplaire || ' ' ||rec.dateLocation);
 UPDATE location
 SET dateEnvoi = newDate WHERE CURRENT OF C;
 END LOOP;
END;
/
```

Modification des données (2)

- Dans le cas d'une clause **FOR UPDATE**, la table est verrouillée en mode row share (RS). Les lignes concernées par le verrou sont les lignes du **SELECT** de la définition du curseur
- En général, un **COMMIT** à l'emplacement de la flèche ferme le curseur. Mais ça n'est pas vrai sous Oracle en PL/SQL

Gestion des erreurs (erreurs standard)

Code d'erreur SQLCODE	Erreur
100	NO_DATA_FOUND
-1	DUP_VAL_ON_INDEX
-6502	VALUE_ERROR
-1001	INVALID CURSOR
-1722	INVALID NUMBER
-6501	PROGRAM ERROR
-1017	LOGIN DENIED
-1422	TOO_MANY_ROWS
-1476	ZERO_DIVIDE

Gestion des erreurs (erreurs standard) (2)

- La nature d'une erreur peut être connue par appel au fonctions **SQLCODE** et **SQLERRM**
- **SQLCODE** renvoie le statut d'erreur de la dernière instruction SQL exécutée (0 si n'y a pas d'erreur)
- **SQLERRM** renvoie le message d'erreur correspondant à **SQLCODE**

Erreurs utilisateur

```
DECLARE
 nom-anomalie EXCEPTION;
BEGIN
 ...
 IF ... THEN RAISE nom-anomalie;
 ...
EXCEPTION
 WHEN nom-anomalie THEN traitement;
END;
```

Erreurs anonymes

- Pour les codes d’erreur n’ayant pas de nom associé, il est possible de définir un nom d’erreur (code entre -20000 et -20999)

Exemple

```
DECLARE
 e EXCEPTION;
 PRAGMA EXCEPTION_INIT(e, -20091);
 ...
BEGIN
 ...
IF ... THEN RAISE e;
EXCEPTION
 WHEN e THEN ...
END;
```

Exemple (2)

```
DECLARE
 e exception;
BEGIN
 ...
 IF ... THEN RAISE e; END IF;
 ...
EXCEPTION
 WHEN e THEN RAISE_APPLICATION_ERROR(
 -20099, 'nom inéxistant');
END;
```

Exemple (2 bis)

```
DECLARE
 ...
BEGIN
 ...
 IF ... THEN RAISE_APPLICATION_ERROR(
 -20099, 'nom inéxistant');
   END IF;
END;
```

Description du traitement de l'erreur (syntaxe)

BEGIN

...

EXCEPTION

WHEN ***nom-erreur*$_1$ THEN** ***traitement-erreur*$_1$;**

...

WHEN ***nom-erreur*$_n$ THEN** ***traitement-erreur*$_n$;**

WHEN OTHERS THEN ***traitement-autres-erreurs*;**

END;

Description du traitement de l'erreur (syntaxe) (2)

- Possibilité d'écrire :

WHEN ***nom-erreur*** $_1$ **OR** ***nom-erreur*** $_2$
THEN ... ;

Exemple

```
...
EXCEPTION
   WHEN NO_DATA_FOUND THEN ...;
   WHEN OTHERS THEN
   DBMS_OUTPUT.PUT_LINE(
   'SQLCODE = '||TO_CHAR(SQLCODE));
   DBMS_OUTPUT.PUT_LINE(
   'SQLERRM : '||TO_CHAR(SQLERRM));
END;
```

Procédures stockées

CREATE [OR REPLACE] PROCEDURE ***nom-procédure***
[(*argument* [*mode*] *type*, ...)]
[IS | AS]
***bloc-procédure*;**

- ***argument*** : nom d'un paramètre formel
- ***mode*** : définit si le paramètre formel est en entrée (**IN**), en sortie (**OUT**) ou en entrée-sortie (**IN OUT**). Par défaut : **IN**
- ***type*** : le type du paramètre formel
- ***bloc-procédure*** : le corps PL/SQL de la procédure

Exemple

```
CREATE OR REPLACE PROCEDURE
modifierTitre (num IN film.numFilm%TYPE) IS
BEGIN
    BEGIN
    UPDATE film SET titre = UPPER(titre)
    WHERE numFilm = num AND realisateur < 1000;
    END;
    BEGIN
    UPDATE film SET titre = LOWER(titre)
    WHERE numFilm = num AND realisateur >= 1500;
    END;
END modifierTitre;
```

Exemple (bis)

```
CREATE OR REPLACE PROCEDURE
modifierTitre (num IN film.numFilm%TYPE) IS
BEGIN
    BEGIN
    UPDATE film SET titre = UPPER(titre)
    WHERE numFilm = num AND realisateur < 1000;
    END;
END modifierTitre;
```

Exemple (ter)

```
CREATE OR REPLACE PROCEDURE
modifierTitre (num IN film.numFilm%TYPE) IS
X date;
BEGIN
    UPDATE film SET titre = UPPER(titre)
    WHERE numFilm = num AND realisateur < 1000;
END modifierTitre;
```

Fonctions stockées

CREATE [OR REPLACE] FUNCTION ***nom-fonction***
[(*argument* **[IN]** *type*, ...)]
RETURN ***type-retour***
[IS | AS]
***bloc-fonction*;**

- Les paramètres sont forcément en entrée (**IN**)
- Dans le ***bloc-fonction*** :

RETURN ***nom-variable*;**

Exemple

```
CREATE OR REPLACE FUNCTION
moyenneLocation (dd IN DATE ) RETURN NUMBER IS
moy NUMBER(8) := 0;
e EXCEPTION;
BEGIN
SELECT AVG(COUNT(*)) INTO moy FROM location
WHERE dateLocation <= dd GROUP BY login;
IF moy IS NULL THEN RAISE e; END IF;
RETURN moy;
EXCEPTION
WHEN e THEN RETURN 0;
END moyenneLocation;
```

Informations à propos des procédures/fonctions

- Erreurs
  - **USER_ERRORS**
  - **ALL_ERRORS**
  - **DBA_ERRORS**
- Infos sur les procédures/fonctions :
  - **USER_OBJECTS**
  - **ALL_OBJECTS**
  - **DBA_OBJECTS**
- Infos sur les textes source :
  - **USER_SOURCE**
  - **ALL_SOURCE**
  - **DBA_SOURCE**

Supression d'une procédure/fonction stockée

DROP PROCEDURE ***nom-procedure*****;**

DROP FUNCTION ***nom-fonction*****;**

Appel d'une procédure/fonction stockée dans un bloc PL/SQL

***nom-procédure* (*liste-paramètres-effectifs*);**

***nom-variable* := *nom-fonction*(*liste-paramètres-effectifs*);**

Appel d'une procédure/fonction stockée dans un script SQL Developer

EXECUTE ***nom-procédure*** **(*liste-paramètres-effectifs*);**

EXECUTE :*nom-variable* := *nom-fonction* (*liste-paramètres-effectifs*);

EXECUTE DBMS_OUTPUT.PUT_LINE(*nom-fonction* (*liste-paramètres-effectifs*));

Exemple
1. Création procédure

```
CREATE OR REPLACE PROCEDURE
nvFilm(num film.numFilm%TYPE,
        titre film.titre%TYPE)
IS
BEGIN
   INSERT INTO film (numFilm, titre)
   VALUES (num, titre);
END nvFilm;
```

Exemple
2. Exécution procédure

```
EXECUTE nvFilm (6000, 'El Camino');
```

Exemple
2^{bis}. Exécution procédure

EXECUTE nvFilm (&num, &titre);

Enter Substitution Variable
NUM: 6000

Enter Substitution Variable
TITRE: 'El Camino'

Exemple
2*ter*. Exécution procédure

```
PROMPT Entrez les infos sur un
       nouvel film
ACCEPT num PROMPT Numéro
ACCEPT titre PROMPT Titre
```

Numéro : 6000

Titre : 'El Camino'

```
EXECUTE nvFilm (&num, &titre);
```

Exemple
1. Création fonction

```
CREATE OR REPLACE FUNCTION
nbEx(nom film.titre%TYPE)
RETURN NUMBER IS nb NUMBER(8);
BEGIN
SELECT COUNT(*) INTO nb FROM exemplaire
WHERE numFilm IN (SELECT numFilm
FROM film WHERE titre = nom);
RETURN nb;
END;
```

Exemple
2. Exécution fonction

```
EXECUTE DBMS_OUTPUT.PUT_LINE (
   nbEx('The Irishman'));
```

Exemple
2^{bis}. Exécution fonction

```
EXECUTE DBMS_OUTPUT.PUT_LINE (
   nbEx(&nom));
```

```
Enter Substitution Variable
NOM: 'The Irishman'
```

Exemple
2*ter*. Exécution fonction

```
ACCEPT nom PROMPT Titre
Titre : 'The Irishman'

EXECUTE DBMS_OUTPUT.PUT_LINE (
   nbEx(&nom));
```

Exemple
2*quater*. Exécution fonction

```
VARIABLE nb NUMBER
EXECUTE :nb := nbEx('The Irishman');
PRINT nb
```

Gestion des erreurs

- Erreur détectée par le SGBD
- Erreur générée par l'utilisateur

Chaque catégorie peut être prise en compte dans la section **EXCEPTION** ou par l'environnement appelant

Gestion par une section EXCEPTION

- Dans ce cas, l'exécution de la procédure ou de la fonction est toujours considérée comme réussie par l'environnement appelant. En général, on stocke les messages d'erreurs dans une table spécifique accessible à l'environnement
- Exemple : en cas de suppression d'un film, vérifier qu'il n'a pas d'exemplaires dans la base

Exemple

```
CREATE PROCEDURE delFilm(
num IN film.numFilm%TYPE) IS
filler CHAR(1); erreur EXCEPTION;
BEGIN
SELECT  'x' INTO filler FROM exemplaire
WHERE numFilm = num; RAISE erreur;
EXCEPTION
WHEN NO_DATA_FOUND THEN DELETE FROM film
WHERE numFilm = num; COMMIT;
WHEN erreur OR TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE ('Le film numéro : '||num||' a des
    exemplaires dans la base');COMMIT;
END delFilm;
```

Gestion des erreurs par l'environnement

- Erreur émise par le SGBD : le code d'erreur, sous la forme **ORA_xxxxx**, et le message associé sont transmis au bloc appelant
- Erreur générée par l'utilisateur : utilisation de la procédure standard

RAISE_APPLICATION_ERROR(*numéro*, *texte*);

  - Le numéro doit être compris entre -20000 et -20999
  - L'exécution de **RAISE_APPLICATION_ERROR** annule la transaction en cours

Exemple (on ne tient pas compte des CIR)

```
CREATE PROCEDURE delFilm (
num IN film.numFilm%TYPE) IS
tit film.titre%TYPE;
BEGIN
SELECT titre INTO tit FROM film
WHERE numFilm = num;
DELETE FROM film WHERE numFilm = num;
DBMS_OUTPUT.PUT_LINE('Film '||tit||' supprimé');
EXCEPTION
WHEN NO_DATA_FOUND THEN
RAISE_APPLICATION_ERROR (-20002,'Le film '||num||' n''existe
    pas');
END delFilm;
```

Packages

- Un package contient deux types de procédures ou fonctions :
  - Publiques
  - Privées

Packages (2)

- Deux parties distinctes dans un package (*chaque partie doit être créée et compilée séparément*) :
  - La partie déclaration ou spécification qui contient la déclaration des procédures, fonctions, variables et traitement d'exceptions de type public (accessibles de l'extérieur du package)
  - La partie corps, ou body, qui contient la définition des procédures ou fonctions de type public déclarées dans la partie spécification ainsi que les déclarations de procédures ou fonctions de type privé

Partie déclaration

CREATE [OR REPLACE] PACKAGE
***nom-package* [IS | AS]**
[*déclaration-de-variable*;]
[*déclaration-de-curseur*;]
[*déclaration-de-procédure*;]
[*déclaration-de-fonction*;]
[*déclaration-d'exception*;]
END *nom-package*;

Déclaration : exemple

```
CREATE OR REPLACE PACKAGE gestionV IS
  dateMax CONSTANT DATE := SYSDATE – 30;
  FUNCTION leReal (monFilm film.titre%TYPE)
    RETURN individu.nomIndividu%TYPE;
  PROCEDURE etablirAgenda(monClient
    client.login%TYPE);
END gestionV;
```

Partie corps

CREATE [OR REPLACE] PACKAGE BODY
nom-package **[IS | AS]**
[*définition-de-variable*;]
[*définition-de-curseur*;]
[*définition-de-procédure*;]
[*définition-de-fonction*;]
[*définition-d'exception*;]
END *nom-package*;

Corps : exemple

```
CREATE OR REPLACE PACKAGE BODY gestionV IS
 FUNCTION leReal (monFilm film.titre%TYPE)
  RETURN individu.nomIndividu%TYPE IS R individu.nomIndividu%TYPE;
   BEGIN SELECT nomIndividu INTO R FROM individu
   WHERE numIndividu = (SELECT realisateur FROM film
                        WHERE titre = monFilm);
 RETURN R; END leReal;
 PROCEDURE etablirAgenda(monClient client.login%TYPE) IS
  CURSOR clientCUR IS SELECT * FROM location
  WHERE login = monClient;
  BEGIN FOR r IN clientCUR
   LOOP IF r.dateLocation <= dateMax
   THEN DBMS_OUTPUT.PUT_LINE (r.dateLocation||r.dateEnvoi);
 END IF; END LOOP; END etablirAgenda;
END gestionV;
```

Référence à un élément d'un package

- En PL/SQL :

  ***nom-package.nom-élément*;**

- En mode interactif :

EXECUTE *nom-package.nom-variable* := ...

EXECUTE *nom-package.nom-procedure* (*paramètres-effectifs*)

EXECUTE :*nom-variable* := *nom-package.nom-function* (*paramètres-effectifs*);

Exécution : exemple

- En PL/SQL :

  ***nom-package.nom-élément*;**

- En mode interactif :

EXECUTE *nom-package.nom-variable* := ...

EXECUTE *nom-package.nom-procedure* (*paramètres-effectifs*)

EXECUTE :*nom-variable* := *nom-package.nom-function* (*paramètres-effectifs*);

Exécution : exemple

```
EXECUTE gestionV.etablirAgenda ('mo');

EXECUTE DBMS_OUTPUT.PUT_LINE
(gestionV.leReal ('The Departed'));
```

Suppression d’un package

- Pour la totalité du package :

  DROP PACKAGE *nom-package*;

- Pour seulement le corps :

  DROP PACKAGE BODY *nom-package*;

Surcharge d’une procédure ou d’une fonction

- A l’intérieur d’un package, il est possible de *surcharger* une procédure ou une fonction, c’est-à-dire de définir plusieurs procédures ou fonctions avec le même nom mais avec une liste de paramètres différente

Exemple

- On peut prévoir deux fonctions de même nom qui calculent, pour un réalisateur passé en paramètre, le nombre de films mis en scène par cette personne. Une fonction aura le nom comme paramètre, c'est-à-dire une chaîne de caractères, l'autre le numéro d'individu, c'est-à-dire un **NUMBER**

Packages (3)

- Lorsqu'un package est utilisé par plusieurs sessions, chaque session utilise sa propre copie des variables et des curseurs.
- Un utilisateur doit posséder le privilège **CREATE PROCEDURE** pour créer un package qui utilise ses propres objets
- Un utilisateur doit posséder le privilège **CREATE ANY PROCEDURE** pour créer un package qui utilise n'importe quels objets

Les déclencheurs (triggers)

- Un traitement déclenché par un événement
- L'exécution d'un déclencheur est un succès ou un échec
- En cas d'échec, l'exécution du traitement est stoppée, mais la transaction qui l'a appelé peut soit continuer soit être annulée

12 types de déclencheurs

- 3 événements
  - **INSERT**
  - **UPDATE**
  - **DELETE**
- 2 modes
  - Ordre
  - Ligne (**FOR EACH ROW**)
- 2 moments
  - **BEFORE**
  - **AFTER**

Ordre d’exécution

- Il est possible d’associer un et un seul déclencheur de chaque type à chaque table. Lorsque plusieurs déclencheurs sont associés à la même table, l’ordre d’exécution est le suivant :
  1. Déclencheur par ordre **BEFORE**
  2. Pour chaque ligne (**FOR EACH ROW**)
     - Déclencheur par ligne **BEFORE**
     - Déclencheur par ligne **AFTER**
  3. Déclencheur par ordre **AFTER**

Déclencheur par ordre

CREATE [OR REPLACE] TRIGGER
nom-déclencheur
moment
événement **[OR** ***événement*****]**
ON ***nom-table***
bloc-PL/SQL**;**

Exemple

```
CREATE OR REPLACE TRIGGER ajoutFilm
BEFORE
INSERT ON film
BEGIN
   IF USER != 'FFIOREN' THEN
     RAISE_APPLICATION_ERROR (-20001,
     'Utilisateur interdit'); END IF;
END ajoutFilm;
```

Déclencheur par ordre (2)

- Pour l'événement **UPDATE**, on peut spécifier les attributs concernés en mettant

  UPDATE OF *nom-attribut*$_1$, ...

Exemple

```
CREATE OR REPLACE TRIGGER updateFilm
BEFORE
UPDATE OF titre ON film
BEGIN
  IF USER != 'FFIOREN' THEN
    RAISE_APPLICATION_ERROR (-20001,
    'Utilisateur interdit'); END IF;
END updateFilm;
```

Déclencheur par ordre BEFORE

- Un déclencheur par ordre avec l'option **BEFORE** peut servir à soumettre l'exécution d'un traitement de mise à jour d'une table à certaines conditions, avec émission d'un message d'erreur si les conditions ne sont pas vérifiées

Déclencheur par ordre AFTER

- Un déclencheur par ordre avec l'option **AFTER** peut servir à faire des validations a posteriori afin de vérifier que les modifications se sont bien déroulées. Il peut aussi permettre de propager des modifications dans plusieurs tables

Déclencheur par ligne

CREATE [OR REPLACE] TRIGGER
nom-déclencheur
moment
événement **[OR** ***événement*****]**
ON ***nom-table***
FOR EACH ROW
[WHERE ***condition*****]**
bloc-PL/SQL**;**

Déclencheur par ligne (2)

- On peut faire référence, dans la condition **WHERE** ou dans le bloc PL/SQL associé au déclencheur
  à la valeur d'un attribut avant modification en préfixant le nom de l'attribut par **:OLD**,
  et/ou à la valeur après modification en préfixant le nom de l'attribut par **:NEW**

Déclencheur par ligne (3)

- La valeur prise en compte dépend de l'ordre SQL :

Ordre SQL	:OLD	:NEW
INSERT	NULL	Valeur créée
DELETE	Valeur avant suppression	NULL
UPDATE	Valeur avant modification	Valeur après modification

Exemple

```
CREATE OR REPLACE TRIGGER auditEx
AFTER DELETE OR UPDATE OR INSERT
ON exemplaire FOR EACH ROW
BEGIN
IF DELETING OR UPDATING THEN
INSERT INTO reportExemplaire
VALUES(SYSDATE, :OLD.numExemplaire,
   :OLD.numFilm, :OLD.codeSupport, :OLD.vo,
   :OLD.probleme, :OLD.detailSupport);
END IF;
IF INSERTING THEN NULL; END IF;
END;
```

On suppose ici qu'il existe une table reportExemplaire qui a la meme structure de la table Exemplaire

Déclencheur par ligne (4)

- La clause **WHERE** ne peut pas contenir de requête SQL
- Un déclencheur par ligne avec l'option **BEFORE** peut servir à effectuer des traitements d'initialisation avant l'exécution des modifications sur la table
- Un déclencheur par ligne avec l'option **AFTER** permet de propager les modifications ou de gérer l'historique

Tables système

- **USER_TRIGGERS**
- **ALL_TRIGGERS**
- **DBA_TRIGGERS**

Suppression

DROP TRIGGER *nom-déclencheur*;

Activation/Désactivation

- **ALTER TRIGGER *nom-déclencheur* DISABLE;**
- **ALTER TABLE *nom-table* DISABLE ALL TRIGGERS;**
- **ALTER TRIGGER *nom-déclencheur* ENABLE;**
- **ALTER TABLE *nom-table* ENABLE ALL TRIGGERS;**

Restrictions

- L'exécution d'un déclencheur dont le bloc PL/SQL inclut des ordres **INSERT**, **DELETE** ou **UPDATE** peut entraîner la mise en œuvre d'un autre déclencheur associé à la table modifiée par ces actions
- Dans ce cas, lors de l'exécution d'un déclencheur de type ligne :
  - Aucun ordre SQL ne doit consulter ou modifier une table déjà utilisée en mode modification par un autre utilisateur

Restrictions (suite)

- Un déclencheur ne peut modifier la valeur d'un attribut déclaré avec l'une des contraintes **PRIMARY KEY**, **UNIQUE** ou **FOREIGN KEY**
- Un déclencheur ne peut pas consulter les données d'une table en *mutation* : une *table en mutation* est une table directement ou indirectement concernée par l'événement qui a provoqué la mise en œuvre du déclencheur

Restrictions : exemple

```
CREATE OR REPLACE TRIGGER verifDate
AFTER UPDATE OF dateEnvoi OR INSERT ON
location
FOR  EACH ROW
DECLARE
delaiMax NUMBER(6);
BEGIN
SELECT MAX(dateEnvoi – dateLocation) INTO
     delaiMax
FROM location;
IF :NEW.dateEnvoi – :NEW.dateLocation > delaiMax
THEN RAISE_APPLICATION_ERROR (-20002, 'Envoi '||
     :NEW.numExemplaire||' hors limites'); END IF;
END verifDate;
```

Restrictions : exemple (suite)

- Exécution :

INSERT INTO location VALUES (2908, SYSDATE, 'mo', SYSDATE, null)

Erreur commençant à la ligne: 1 de la commande -

INSERT INTO location VALUES (…)

Rapport d'erreur -

Erreur SQL : ORA-04091: la table FFIOREN.LOCATION est en mutation ; le déclencheur ou la fonction ne peut la voir

Exemple

```
CREATE OR REPLACE TRIGGER verifDate
AFTER UPDATE OF dateEnvoi OR INSERT ON
location
FOR  EACH ROW
DECLARE
delaiMax NUMBER(6) : = 45;

BEGIN
IF :NEW.dateEnvoi – :NEW.dateLocation > delaiMax
THEN RAISE_APPLICATION_ERROR (-20002, 'Envoi '||
     :NEW.numExemplaire||' hors limites'); END IF;
END verifDate;
```

Erreurs de compilation (warnings)

- Sous SQL Developer, pour afficher les erreurs de compilation :
  - Se positionner sur l'objet crée avec des erreurs
  - Avec la touche droite de la souris, sélectionner « Compile for Debug »