

Cours de Bases de données

PL/SQL (Part 2)

Master : Traitement intelligent des systèmes

Préparé par: Mme. Khawla Elansari

Année Universitaire: 2021/ 2022


4. Interactions avec la BDD

Extraction

Pour extraire les données à partir d'un programme PL/SQL on utilise l'instruction:

SELECT liste **INTO** { *nomVariablePLSQL*[,*nomVariablePLSQL*]... / *nomRECORD* } **FROM** nomTable...;

```
DECLARE
    empl Emp%ROWTYPE;
    empl_sal Emp.SAL%TYPE;
BEGIN
    -- Extraction d'un enregistrement dans la variable empl
    SELECT * INTO empl FROM Emp WHERE EMPNO=7499;
    -- Extraction de la valeur de la colonne sal dans la variable empl_sal
    SELECT SAL INTO empl_sal FROM Emp WHERE EMPNO=7499;
    dbms_output.put_line('Nom : ' || empl.ENAME);
    dbms_output.put_line('Sal : ' || empl_sal);
END;
```



Nom : ALLEN
Sal : 1600

Extraction

- Une requête **SELECT ... INTO** doit renvoyer un seul enregistrement .
- Une requete qui renvoie plusieurs enregistrements, ou qui n'en renvoie aucune, génère une erreur PL/SQL:

```
DECLARE
    empl Emp%ROWTYPE;
    -- record basé sur la structure d'une ligne de la table Emp;
BEGIN
    SELECT * INTO empl FROM Emp;

    dbms_output.put_line('Nom : ' || empl.ENAME);
END;
```



```
Error report -
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 5
01422. 00000 - "exact fetch returns more than requested number of rows"
*Cause:      The number specified in exact fetch is less than the rows returned.
*Action:     Rewrite the query or change number of rows requested
```

```
DECLARE
    empl Emp%ROWTYPE;
    -- record basé sur la structure d'une ligne de la table Emp;
BEGIN
    SELECT * INTO empl FROM Emp WHERE EMPNO=749;

    dbms_output.put_line('Nom : ' || empl.ENAME);
END;
```




```
Error report -
ORA-01403: no data found
ORA-06512: at line 5
01403. 00000 - "no data found"
*Cause:      No data was found from the objects.
*Action:     There was no data from the objects which may be due to end of fetch.
```

Extraction

- On peut utiliser des fonctions SQL à l'intérieur de PL/SQL:

```
DECLARE
    empl_ename Emp.ename%TYPE;
    max_sal Emp.sal%TYPE;
BEGIN
    SELECT upper(ename) INTO empl_ename FROM Emp WHERE EMPNO=7499;
    SELECT max(sal) INTO max_sal FROM Emp;
    dbms_output.put_line('Upper Nom : ' || empl_ename);
    dbms_output.put_line('maxSal : ' || max_sal);
END;
```




Upper Nom : ALLEN
maxSal : 5000

Insertion

Pour insérer des données dans une table PL/SQL on utilise l'instruction:

Insert INTO nomTable **Values** (*nomVariablePLSQL*[,*nomVariablePLSQL*]... / *nomRECORD*);

```
DECLARE
    --empl Emp%ROWTYPE;
    empl_name emp.ename%TYPE;
    empl_sal Emp.SAL%TYPE;
BEGIN
    insert INTO emp values (1111,'Sebastien','CLERK',7902,'17-DEC-90',800,0,20);
    empl_name := 'Francis';
    empl_sal := 4500;
    insert INTO emp values (2222,empl_name,'CLERK',7902,'1-DEC-99',empl_sal,0,20);
    COMMIT;
END;
```



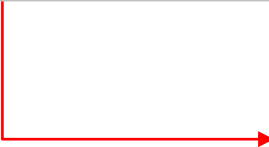
	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	1111	Sebastien	CLERK	7902	17-DEC-90	800	0	20
2	2222	Francis	CLERK	7902	01-DEC-99	4500	0	20
3	7369	SMITH	CLERK	7902	17-DEC-80	800	(null)	20
4	7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30

Modification

Pour mettre à jour une table PL/SQL on utilise l'instruction:

Update nomTable **SET** nomColonne1 = {nomVariablePLSQL / Expression / nomColonne} [,nomColonne2 =..] ..[**WHERE** ...];

```
DECLARE
    empl_sal number := 1000;
BEGIN
    Update emp set sal = empl_sal where empno=2222;
    COMMIT;
END;
```



	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	1111	Sebastien	CLERK	7902	17-DEC-90	800	0	20
2	2222	Francis	CLERK	7902	01-DEC-99	1000	0	20
3	7369	SMITH	CLERK	7902	17-DEC-80	800	(null)	20
4	7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30



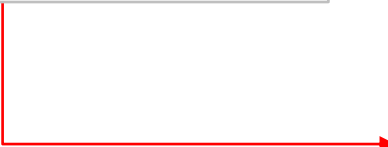
Contrairement à l'instruction Select, si aucun enregistrement n'est mis à jour par l'instruction Update, aucune erreur ne se produit

Suppression

Pour mettre à jour une table PL/SQL on utilise l'instruction:

DELETE FROM nomTable **WHERE** nomColonne1 = {nomVariablePLSQL | Expression | nomColonne} [,nomColonne2 =..].;

```
DECLARE
    empl_NO number := 1111;
BEGIN
    DELETE FROM emp where empno = empl_no;
    COMMIT;
END;
```



	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	1111	Sebastien	CLERK	7902	17-DEC-90	800	0	20
2	2222	Francis	CLERK	7902	01-DEC-99	1000	0	20
3	7369	SMITH	CLERK	7902	17-DEC-80	800	(null)	20
4	7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30



Contrairement à l'instruction Select, si aucun enregistrement n'est supprimé par l'instruction DELETE , aucune erreur ne se produit

5. Transactions

Problèmes de cohérence et transaction

- Un SGBD doit pouvoir supporter :
 - plusieurs utilisateurs l'utilisant en parallèle
 - effectuant des opérations d'écriture et de lecture
 - tout en garantissant la cohérence des données
- Une transaction est un ensemble d'ordres (SQL) indivisibles, faisant passer la base de données d'un état cohérent à un autre en une seule étape.

Opérations Commit / Rollback / Savepoint

- **ROLLBACK** annule entièrement une transaction : toutes les modifications depuis le début de la transaction sont alors défaites.
- **COMMIT** valide entièrement une transaction : les modifications deviennent définitives et visibles à tous les utilisateurs.
- **SAVEPOINT** point de contrôle, état de la base où l'on pourra revenir plus tard.



- En cours de transaction, seul l'utilisateur ayant effectué les modifications les voit.
- En cas de fin anormale d'une tâche utilisateur il y a automatiquement ROLLBACK des transactions non terminées.
- Une transaction commence (implicitement) à la première opération SQL rencontrée et dès qu'une transaction est terminée.
- Les commandes de définition de données sont automatiquement commitées (auto-commit) et valident donc les ordres précédent.
- Un mécanisme de verrouillage permet de gérer les conflits d'accès parallèle.

Opérations Commit / Rollback / Savepoint


```
INSERT INTO transactions VALUES ( 1 , 'Pas de savepoint' ) ;  
-- 1 ligne creee .  
INSERT INTO transactions VALUES ( 2 , 'savepoint A' ) ;  
-- 1 ligne creee .  
SAVEPOINT A;  
-- Savepoint .  
INSERT INTO transactions VALUES ( 3 , 'Savepoint B' ) ;  
-- 1 ligne creee .  
SAVEPOINT B;  
-- Savepoint cree .  
INSERT INTO transactions VALUES ( 4 , 'Pas de savepoint' ) ;  
-- 1 ligne creee .  
  
select * from transactions;
```



ID_TRANS	DESC_TRANS
1	1 Pas de savepoint
2	2 savepoint A
3	3 Savepoint B
4	4 Pas de savepoint

Opérations Commit / Rollback / Savepoint

```
ROLLBACK TO SAVEPOINT B;  
-- rollback effectuee .  
SELECT * FROM transactions ;  
  
ROLLBACK TO SAVEPOINT A;  
-- rollback effectuee .  
SELECT * FROM transactions ;  
COMMIT;  
-- Validation Effectuee  
SELECT * FROM transactions ;
```



ID_TRANS	DESC_TRANS
1	1 Pas de savepoint
2	2 savepoint A
3	3 Savepoint B



ID_TRANS	DESC_TRANS
1	1 Pas de savepoint
2	2 savepoint A



ID_TRANS	DESC_TRANS
1	1 Pas de savepoint
2	2 savepoint A

6. Curseurs

Curseurs PL/SQL

- Un curseur est une zone de mémoire privée, temporairement allouée dans la zone globale de la session utilisateur, qui est utilisée pour traiter les instructions SQL.
- La mémoire privée stocke l'ensemble des résultats récupérés lors de l'exécution du SQL et les attributs du curseur.
- Les curseurs peuvent être classés en curseurs **implicites** et **explicites**.
 - Oracle crée un curseur implicite pour toutes les instructions SQL incluses dans la section exécutable d'un bloc PL/SQL. Dans ce cas, le cycle de vie du curseur est géré par la base de données Oracle.
 - Pour les curseurs explicites, le cycle d'exécution peut être contrôlé par l'utilisateur. Les développeurs de bases de données peuvent déclarer explicitement un curseur dans la section **DECLARE**.

Création et utilisation

- Le serveur Oracle utilise des zone de travail appelées **Zone SQL Privées** pour exécuter les instructions SQL et pour stocker les informations en cours de traitement.
- Un curseur est un pointeur vers une zone SQL privée qui stocke des informations sur le traitement d'une instruction SELECT ou LMD comme INSERT, UPDATE, DELETE ou MERGE.
- Le curseur est un mécanisme qui vous permet d'attribuer un nom à une instruction SELECT et de manipuler les informations contenues dans cette instruction SQL.
- La fonction principale d'un curseur est de récupérer des données, une ligne à la fois, à partir d'un jeu de résultats, contrairement aux commandes SQL qui agissent sur toutes les lignes du jeu de résultats à la fois.
- Les curseurs sont utilisés lorsque l'utilisateur a besoin de mettre à jour les enregistrements de manière singleton ou ligne par ligne, dans une table de base de données.
- Les données stockées dans le curseur sont appelées l'ensemble de données actif.

Création et utilisation

Curseurs Implicites

- Chaque fois qu'une instruction LMD (INSERT, UPDATE et DELETE) est émise, un curseur implicite est associé à cette instruction. Pour les opérations INSERT, le curseur contient les données à insérer. Pour les opérations UPDATE et DELETE, le curseur identifie les lignes qui seraient affectées.



Le nom du curseur implicite est 'sql'.

- Dans PL/SQL, vous pouvez faire référence au curseur implicite le plus récent en tant que curseur SQL, qui a toujours des attributs tels que %FOUND,%ISOPEN,%NOTFOUND et %ROWCOUNT.

Création et utilisation

Curseurs Implicites


- Le tableau suivant fournit la description des attributs d'un curseur:

Attribut	Description
%FOUND	Sa valeur de retour est TRUE si les instructions DML telles que INSERT, DELETE et UPDATE affectent au moins une ou plusieurs lignes ou si une instruction SELECT INTO a renvoyé une ou plusieurs lignes. Sinon, il renvoie FALSE.
%NOTFOUND	Sa valeur de retour est TRUE si les instructions DML telles que INSERT, DELETE et UPDATE n'affectent aucune ligne ou si une instruction SELECT INTO ne renvoie aucune ligne. Sinon, il renvoie FALSE.
%ROWCOUNT	Il renvoie le nombre de lignes affectées par les instructions DML telles que INSERT, DELETE et UPDATE, pour SELECT INTO, il renvoie le nombre de lignes traités par le curseur.
%ISOPEN	Il renvoie toujours FALSE pour les curseurs implicites, car le curseur SQL est automatiquement fermé après l'exécution de ses instructions SQL associées.

Création et utilisation

Curseurs Implicites

```
DECLARE
    nombre_ligne number(2);
BEGIN
    UPDATE Emp
    SET Sal = Sal + 300;
    IF sql%notfound THEN
        dbms_output.put_line('aucun employé sélectionné');
    ELSIF sql%found THEN
        nombre_ligne := sql%rowcount;
        dbms_output.put_line( nombre_ligne || ' employés sélectionnés ');
    END IF;
END;
```




14 employés sélectionnés

PL/SQL procedure successfully completed.

Création et utilisation

Curseurs Implicites

```
DECLARE
    nombre_ligne number(2);
BEGIN
    UPDATE Emp
    SET Sal = Sal + 300 where ename = 'Jack';
    IF sql%notfound THEN
        dbms_output.put_line('aucun employé sélectionné');
    ELSIF sql%found THEN
        nombre_ligne := sql%rowcount;
        dbms_output.put_line( nombre_ligne || ' employés sélectionnés ');
    END IF;
END;
```



aucun employé sélectionné

PL/SQL procedure successfully completed.

Création et utilisation

Curseurs Explicites

- L'utilisation d'un curseur explicite comprend les étapes suivantes :
 - **Déclaration** du curseur pour l'initialisation de la mémoire.
 - **Ouverture** du curseur pour allouer la mémoire.
 - Lorsque le curseur est ouvert, les lignes peuvent être extraites du curseur une par une ou dans un bloc pour effectuer la **manipulation** des données(**FETCH**).
 - **Fermer** le curseur pour libérer la mémoire allouée.

- Déclaration du curseur:

`CURSOR nom_curseur IS requete_sql;`

- Ouverture du curseur:

`OPEN nom_curseur ;`

N.B: Aucune exception n'est levée si la requête ne ramène aucune ligne

Création et utilisation

Curseurs Explicites

- Extraction des lignes:

FETCH nom_curseur **INTO** liste_variables;

Positionnement sur la ligne suivante et chargement de l'enregistrement courant dans une ou plusieurs variables.

liste_variables - représentent une liste de variables pour stocker la valeur de chaque colonne. Généralement nous utilisons record

- Fermer le curseur

CLOSE nom_curseur ; -- Libération de l'espace mémoire.

Si vous déclarez un curseur dans un bloc anonyme, une procédure ou une fonction, le curseur sera automatiquement fermé à la fin de l'exécution de ces objets.



Il existe un moyen de manipuler le curseur et de laisser Oracle faire le reste (OPEN, CLOSE, FETCH ...), en utilisant la boucle for :

```
FOR nom_record IN nom_curseur LOOP
```

```
-- traitements
```

```
END LOOP;
```

Création et utilisation

Curseurs Explicites

```
SELECT * FROM Emp;  
  
DECLARE  
    CURSOR list_emp IS SELECT * FROM Emp;  
    record_emp Emp%rowtype;  
BEGIN  
    open list_emp;  
    fetch list_emp into record_emp;  
    -- manipuler record_emp  
    -- récupérer uniquement la première ligne  
    dbms_output.put_line( '1st employee ' || record_emp.ename);  
END;
```

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	17-DEC-80	1100	(null)	20
2	7499	ALLEN	SALESMAN	7698	20-FEB-81	1900	300	30
3	7521	WARD	SALESMAN	7698	22-FEB-81	1550	500	30
4	7566	JONES	MANAGER	7839	02-APR-81	3275	(null)	20
5	7654	MARTIN	SALESMAN	7698	28-SEP-81	1550	1400	30
6	7698	BLAKE	MANAGER	7839	01-MAY-81	3150	(null)	30
7	7782	CLARK	MANAGER	7839	09-JUN-81	2750	(null)	10
8	7788	SCOTT	ANALYST	7566	19-APR-87	3300	(null)	20
9	7839	KING	PRESIDENT	(null)	17-NOV-81	5300	(null)	10
10	7844	TURNER	SALESMAN	7698	08-SEP-81	1800	0	30
11	7876	ADAMS	CLERK	7788	23-MAY-87	1400	(null)	20
12	7900	JAMES	CLERK	7698	03-DEC-81	1250	(null)	30
13	7902	FORD	ANALYST	7566	03-DEC-81	3300	(null)	20
14	7934	MILLER	CLERK	7782	23-JAN-82	1600	(null)	10

1st employee SMITH

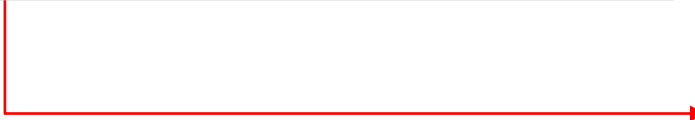
PL/SQL procedure successfully completed.

Création et utilisation

Curseurs Explicites

```
DECLARE
  CURSOR liste_emp IS SELECT * FROM Emp;
  record_emp Emp%rowtype;
BEGIN
  OPEN liste_emp;
  LOOP
    FETCH liste_emp INTO record_emp;
    EXIT WHEN liste_emp%notfound; -- sortir si le curseur ne pointe sur aucune ligne
    -- Traitements sur la ligne courante
    dbms_output.put_line('Nom : ' || record_emp.ename || ' - Professeion : ' || record_emp.job);
  END LOOP;

  CLOSE liste_emp;
END;
```

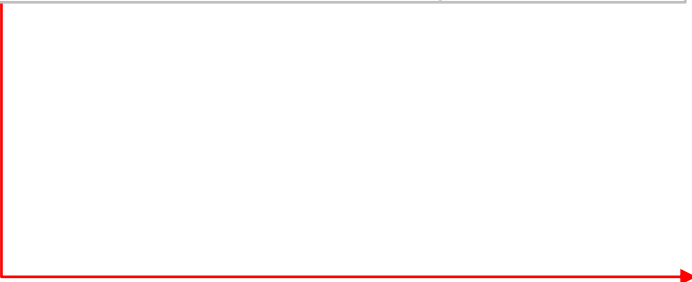


```
Nom :SMITH - Professeion : CLERK
Nom :ALLEN - Professeion : SALESMAN
Nom :WARD - Professeion : SALESMAN
Nom :JONES - Professeion : MANAGER
Nom :MARTIN - Professeion : SALESMAN
Nom :BLAKE - Professeion : MANAGER
Nom :CLARK - Professeion : MANAGER
Nom :SCOTT - Professeion : ANALYST
Nom :KING - Professeion : PRESIDENT
Nom :TURNER - Professeion : SALESMAN
Nom :ADAMS - Professeion : CLERK
Nom :JAMES - Professeion : CLERK
Nom :FORD - Professeion : ANALYST
Nom :MILLER - Professeion : CLERK
```


Création et utilisation

Curseurs Explicites

```
DECLARE
  CURSOR liste_emp IS SELECT * FROM Emp;
  record_emp Emp%rowtype;
BEGIN
  FOR record_emp IN liste_emp LOOP
    -- Traitements sur la ligne courante
    dbms_output.put_line('Nom : ' || record_emp.ename || ' - Professeion : ' || record_emp.job);
  END LOOP;
END;
```



```
Nom : SMITH - Professeion : CLERK
Nom : ALLEN - Professeion : SALESMAN
Nom : WARD - Professeion : SALESMAN
Nom : JONES - Professeion : MANAGER
Nom : MARTIN - Professeion : SALESMAN
Nom : BLAKE - Professeion : MANAGER
Nom : CLARK - Professeion : MANAGER
Nom : SCOTT - Professeion : ANALYST
Nom : KING - Professeion : PRESIDENT
Nom : TURNER - Professeion : SALESMAN
Nom : ADAMS - Professeion : CLERK
Nom : JAMES - Professeion : CLERK
Nom : FORD - Professeion : ANALYST
Nom : MILLER - Professeion : CLERK
```

Création et utilisation

Curseurs Explicites paramétrés

```
DECLARE
  CURSOR liste_emp(Salaire Number) IS SELECT * FROM Emp where Sal >= Salaire;
  record_emp liste_emp%rowtype;
BEGIN
  FOR record_emp IN liste_emp(4000) LOOP
    -- Traitements sur la ligne courante
    dbms_output.put_line('Nom : ' || record_emp.ename || ' - Professeion : ' || record_emp.job || ' - Salaire : ' || record_emp.sal);
  END LOOP;
END;
```

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	17-DEC-80	1100	(null)	20
2	7499	ALLEN	SALESMAN	7698	20-FEB-81	1900	300	30
3	7521	WARD	SALESMAN	7698	22-FEB-81	1550	500	30
4	7566	JONES	MANAGER	7839	02-APR-81	3275	(null)	20
5	7654	MARTIN	SALESMAN	7698	28-SEP-81	1550	1400	30
6	7698	BLAKE	MANAGER	7839	01-MAY-81	3150	(null)	30
7	7782	CLARK	MANAGER	7839	09-JUN-81	2750	(null)	10
8	7788	SCOTT	ANALYST	7566	19-APR-87	3300	(null)	20
9	7839	KING	PRESIDENT	(null)	17-NOV-81	5300	(null)	10
10	7844	TURNER	SALESMAN	7698	08-SEP-81	1800	0	30
11	7876	ADAMS	CLERK	7788	23-MAY-87	1400	(null)	20
12	7900	JAMES	CLERK	7698	03-DEC-81	1250	(null)	30
13	7902	FORD	ANALYST	7566	03-DEC-81	3300	(null)	20
14	7934	MILLER	CLERK	7782	23-JAN-82	1600	(null)	10

Nom :KING - Professeion : PRESIDENT - Salaire : 5300

PL/SQL procedure successfully completed.

7. Gestion des exceptions

Gestion des Exceptions

- Une exception est une erreur qui survient durant une exécution

2 types d'exception :
 - **Interne** (exception oracle prédéfinie, exception oracle non prédéfinie)
 - **Externe** (exception définie par l'utilisateur)
- Les **exceptions internes** sont générées par le moteur du système (division par zéro, connexion non établie, table inexistante, privilèges insuffisants, mémoire saturée, espace disque insuffisant, ...).
 - Une erreur interne est produite quand un bloc PL/SQL viole une règle d'Oracle ou dépasse une limite dépendant du système d'exploitation.
 - Chaque erreur ORACLE correspond à un code SQL (SQLCODE)
- Les **exceptions externes** sont générées par l'utilisateur (stock à zéro, ...).

Gestion des Exceptions

- Le mot clé **EXCEPTION** débute la section de la gestion des exceptions
 - Plusieurs exceptions sont permises
 - Une seule exception est exécutée avant de sortir d'un bloc
 - **WHEN OTHERS** est la dernière clause
 - *Intercepte toutes les exceptions non gérées dans la même section d'exception*
 - *Utilisez le gestionnaire d'erreurs **OTHERS** et placez le en dernier lieu après tous les autres gestionnaires d'erreurs, sinon il interceptera toutes les exceptions mêmes celle qui sont prédéfinies.*

EXCEPTION

```
WHEN exception1 [OR exception2 ...] THEN  
  instructions;
```

```
[WHEN exception3 [OR exception4 ...] THEN  
  instructions; ]
```

```
[WHEN OTHERS THEN  
  instructions; ]
```

Gestion des Exceptions

- Lorsqu'une erreur se produit, une exception est levée => l'exécution normale s'arrête et contrôle les transferts vers la partie de gestion des exceptions de votre bloc ou sous-programme PL/SQL.
- Les exceptions internes sont levées implicitement (automatiquement) par le système d'exécution.
- Les exceptions définies par l'utilisateur doivent être déclenchées explicitement par les instructions **RAISE**, qui peuvent également déclencher des exceptions prédéfinies.

```
BEGIN
...
IF (...) THEN RAISE PILOTE TROP JEUNE ;
...
SELECT ... INTO ... FROM ...;
...
EXCEPTION
  WHEN NO DATA FOUND THEN
    Instructions - A
  WHEN ZERO DIVIDE THEN
    Instructions - B
  WHEN PILOTE TROP JEUNE THEN
    Instructions - C
  WHEN OTHERS THEN
    Instructions - D
END;
```

The diagram illustrates the flow of exception handling in a PL/SQL block. It shows a sequence of instructions: a BEGIN statement, followed by an IF statement with a RAISE instruction (highlighted in yellow), then a SELECT statement, and finally an EXCEPTION block. The EXCEPTION block contains four WHEN clauses, each followed by a box representing the instructions to be executed: 'WHEN NO DATA FOUND THEN' leads to 'Instructions - A'; 'WHEN ZERO DIVIDE THEN' leads to 'Instructions - B'; 'WHEN PILOTE TROP JEUNE THEN' leads to 'Instructions - C'; and 'WHEN OTHERS THEN' leads to 'Instructions - D'. The block ends with an END statement. Three arrows indicate the flow of control: a grey arrow from 'NOT FOUND' (a predefined exception) points to 'Instructions - A'; a blue arrow from the 'RAISE' instruction points to 'Instructions - B'; and a red arrow from the 'PILOTE TROP JEUNE' (a user-defined exception) points to 'Instructions - C'.

Gestion des Exceptions

Exceptions Internes

- Une exception interne est déclenchée implicitement chaque fois que votre programme PL/SQL viole une règle Oracle ou dépasse une limite dépendant du système.
- Chaque erreur Oracle a un numéro, mais les exceptions doivent être gérées par leur nom. Ainsi, **PL/SQL prédéfinit certaines erreurs Oracle courantes en tant qu'exceptions**. Par exemple, PL/SQL lève l'exception prédéfinie NO_DATA_FOUND si une instruction SELECT INTO ne renvoie aucune ligne.

Gestion des Exceptions

Exceptions prédéfinies

ACCESS_INTO_NULL	ORA-06530	Affectation d'une valeur à un objet non initialisé.
CASE_NOT_FOUND	ORA-06592	Aucun des choix de la structure CASE sans ELSE n'est effectué.
COLLECTION_IS_NULL	ORA-06531	Utilisation d'une méthode autre que EXISTS sur une collection (<i>nested table</i> ou <i>varray</i>) non initialisée.
CURSOR_ALREADY_OPEN	ORA-06511	Ouverture d'un curseur déjà ouvert.
DUP_VAL_ON_INDEX	ORA-00001	Insertion d'une ligne en doublon (clé primaire).
INVALID_CURSOR	ORA-01001	Ouverture interdite sur un curseur.
INVALID_NUMBER	ORA-01722	Échec d'une conversion d'une chaîne de caractères en NUMBER.
LOGIN_DENIED	ORA-01017	Connexion incorrecte.
NO_DATA_FOUND	ORA-01403	Requête ne retournant aucun résultat.
NOT_LOGGED_ON	ORA-01012	Connexion inexistante.

Gestion des Exceptions

Exceptions prédéfinies

PROGRAM_ERROR	ORA-06501	Problème PL/SQL interne (invitation au contact du support...).
ROWTYPE_MISMATCH	ORA-06504	Incompatibilité de types entre une variable externe et une variable PL/SQL.
SELF_IS_NULL	ORA-30625	Appel d'une méthode d'un type sur un objet NULL (extension objet).
STORAGE_ERROR	ORA-06500	Dépassement de capacité mémoire.
SUBSCRIPT_BEYOND_COUNT	ORA-06533	Référence à un indice incorrect d'une collection (<i>nested table</i> ou <i>varray</i>) ou variables de type TABLE.
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	
SYS_INVALID_ROWID	ORA-01410	Échec d'une conversion d'une chaîne de caractères en ROWID.
TIMEOUT_ON_RESOURCE	ORA-00051	Dépassement du délai alloué à une ressource.
TOO_MANY_ROWS	ORA-01422	Requête retournant plusieurs lignes.
VALUE_ERROR	ORA-06502	Erreur arithmétique (conversion, troncature, taille) d'un NUMBER.
ZERO_DIVIDE	ORA-01476	Division par zéro.

Gestion des Exceptions

Exceptions prédéfinies

```
DECLARE
    v_sal emp.sal%type;
BEGIN
    SELECT sal INTO v_sal from emp;
EXCEPTION
    WHEN TOO_MANY_ROWS then
        dbms_output.put_line('Too many rows can't be selected into a single record');
        -- gérer erreur trop de lignes
    WHEN NO_DATA_FOUND then
        dbms_output.put_line('No data found in EMP Table');
        -- gérer erreur pas de ligne
    WHEN OTHERS then
        dbms_output.put_line('Other Exception raised');
        -- gérer toutes les autres erreurs
END ;
```

Too many rows can't be selected into a single record

PL/SQL procedure successfully completed.

Gestion des Exceptions

Exceptions non prédéfinies

- Pour capturer une erreur Oracle non prédéfinie, il est nécessaire soit de travailler avec la clause **WHEN OTHERS** du bloc de traitement des exceptions, soit d'associer un nom au numéro de l'erreur Oracle que l'on souhaite capturer. Cette association d'un nom d'exception et d'un numéro d'erreur Oracle est possible grâce à la directive **PRAGMA EXCEPTION_INIT**.

nom_exception **EXCEPTION**;

PRAGMA EXCEPTION_INIT (nom_exception, code_erreur);


où :

- **nom_exception** est défini par l'utilisateur préalablement.
- **code_erreur** désigne le code d'erreur associé à l'exception.

Gestion des Exceptions

Exceptions Non prédéfinies

```
DECLARE
    v_sal emp.sal%type;
    EMP_Too_Many_Rows Exception;
    PRAGMA EXCEPTION_INIT (EMP_Too_Many_Rows, -1422);
BEGIN
    SELECT sal INTO v_sal from emp;
EXCEPTION
    WHEN EMP_Too_Many_Rows then
        dbms_output.put_line('Too many rows can't be selected into a single record');
        -- gérer erreur trop de lignes
    WHEN NO_DATA_FOUND then
        dbms_output.put_line('No data found in EMP Table');
        -- gérer erreur pas de ligne
    WHEN OTHERS then
        dbms_output.put_line('Other Exception raised');
        -- gérer toutes les autres erreurs
END ;
```



Too many rows can't be selected into a single record

PL/SQL procedure successfully completed.

Gestion des Exceptions

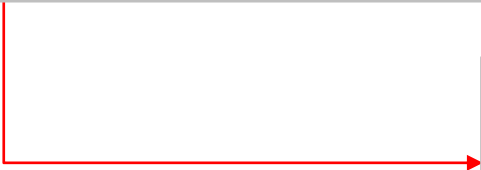
Exceptions Externes

- PL/SQL offre au programmeur la possibilité de définir ses propres exceptions de façon très souple et puissante.
- Le programmeur doit lui-même **déclarer** et **lever** ses exceptions, cela pour bénéficier des blocs de traitements d'erreurs et aborder des erreurs applicatives (définies par le programmeur), pour améliorer et faciliter la maintenance et l'évolution des programmes.
- Pour capturer une exception définie par l'utilisateur, le programmeur doit suivre les étapes suivantes :
 1. Déclarer l'exception en la nommant dans la section déclarative.
`nom_exception EXCEPTION;`
 2. Déclencher explicitement l'exception dans la section exécutable en utilisant l'instruction **RAISE**.

Gestion des Exceptions

Exceptions Externes

```
DECLARE
    v_sal emp.sal%type;
    Exep_sal_trop_bas EXCEPTION;
BEGIN
    SELECT sal INTO v_sal from emp where empno= 7499;
    dbms_output.put_line(' Le salire est : ' || v_sal);
    if v_sal < 2000 then
        Raise Exep_sal_trop_bas;
    END If;
EXCEPTION
    WHEN TOO_MANY_ROWS then
        dbms_output.put_line('Too many rows can''t be selected into a single record');
        -- gérer erreur trop de lignes
    WHEN Exep_sal_trop_bas then
        dbms_output.put_line('Alerte : Salaire trop bas');
    WHEN OTHERS then
        dbms_output.put_line('Other Exception raised');
        -- gérer toutes les autres erreurs
END ;
```



```
Le salire est : 1900
Alerte : Salaire trop bas

PL/SQL procedure successfully completed.
```

Gestion des Exceptions

[ORA-02292 Error](#)

```
CREATE OR REPLACE PROCEDURE détruitCompagnie  
    (p_comp IN VARCHAR2) IS
```

Déclaration des exceptions.

```
    erreur_ilResteUnPilote EXCEPTION;  
    PRAGMA EXCEPTION_INIT(erreur_ilResteUnPilote , -2292);  
    erreur_compagnieInexistante EXCEPTION;
```

```
BEGIN  
    DELETE FROM Compagnie WHERE comp = p_comp;  
    IF SQL%NOTFOUND THEN  
        RAISE erreur_compagnieInexistante;  
    END IF;  
    COMMIT;  
    DBMS_OUTPUT.PUT_LINE('Compagnie ' || p_comp || ' détruite.');
```

Corps du traitement
(validation).

```
EXCEPTION  
    WHEN erreur_ilResteUnPilote THEN  
        DBMS_OUTPUT.PUT_LINE ('Désolé, il reste encore un  
                                pilote à la compagnie ' || p_comp);  
    WHEN erreur_compagnieInexistante THEN  
        DBMS_OUTPUT.PUT_LINE ('La compagnie ' || p_comp ||  
                                ' n''existe pas dans la base!');  
    WHEN OTHERS THEN  
        DBMS_OUTPUT.PUT_LINE('Erreur d''Oracle ' || SQLERRM ||  
                                '(' || SQLCODE || ')');
```

Gestion des exceptions.

Gestion des autres
exceptions.

Gestion des Exceptions

RAISE_APPLICATION_ERROR

- Un ensemble de procédures et de fonctions est fourni dans le package **DBMS_STANDARD** d'Oracle afin de faciliter le développement. Parmi ces procédures, on trouve **RAISE_APPLICATION_ERROR**.
- La procédure **RAISE_APPLICATION_ERROR** permet de publier ses propres messages et codes des erreurs (exception utilisateur). Cette procédure évite le renvoi des exceptions non traitées, car le numéro d'erreur (inclus dans **RAISE_APPLICATION_ERROR**) sera communiqué à l'environnement appelant.
- Pour invoquer **RAISE_APPLICATION_ERROR**, on utilise cette syntaxe :

RAISE_APPLICATION_ERROR (numero_erreur, message [{TRUE | FALSE}]);

où :

- ***numero_erreur** est le code d'erreur défini par l'utilisateur pour l'exception, il peut être un nombre compris entre -20.999 et -20.000.*
- ***message** est le texte décrivant l'erreur. Il peut contenir jusqu'à 2048 octets.*
- ***TRUE | FALSE** est un paramètre booléen optionnel. TRUE positionne l'erreur dans une pile si plusieurs exceptions doivent être propagées en cascade, ou bien (FALSE) l'erreur doit remplacer toutes les autres erreurs. C'est cette dernière solution qui est adoptée par défaut.*

Gestion des Exceptions

RAISE_APPLICATION_ERROR

```
DECLARE
    v_sal emp.sal%type;
    Exep_sal_trop_bas EXCEPTION;
BEGIN
    SELECT sal INTO v_sal from emp where empno= 7499;
    dbms_output.put_line(' Le salire est : ' || v_sal);
    if v_sal < 2000 then
        Raise Exep_sal_trop_bas;
    END If;
EXCEPTION
    WHEN Exep_sal_trop_bas then
        RAISE_APPLICATION_ERROR (-20222, 'Alerte 22222 : Salaire trop bas');
    WHEN OTHERS then
        dbms_output.put_line('Other Exception raised');
        -- gérer toutes les autres erreurs
END ;
```

Error report -

ORA-20222: Alerte 22222 : Salaire trop bas

ORA-06512: at line 15

Thank you!