

4



## PL / SQL

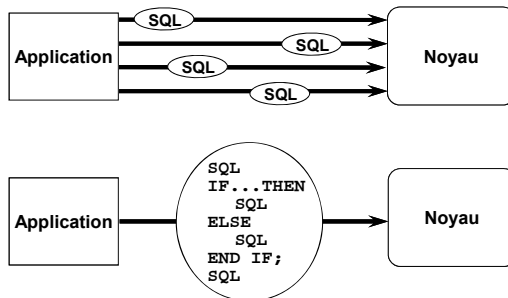


## Généralités - PL/SQL

- PL/SQL est une extension procédurale du langage SQL. Langage propriétaire créé par Oracle
- Possibilité d'inclure des requêtes et des ordres de manipulation des données à l'intérieur d'une structure algorithmique

## Intérêts du PL/SQL

### Amélioration des Performances



## Intérêts du PL/SQL

### Développement MODULAIRE

DECLARE

BEGIN

EXCEPTION

END;

## Intérêts du PL/SQL

- Portabilité ( /serveur Oracle)
- Utilisation de variables
- Structures de contrôle
- Gestion des erreurs

97

Bases de données - © Christine Bonnet



## PL/SQL Structure de BLOC

- DECLARE – Facultatif
  - Variables, curseurs, exceptions utilisateur
- BEGIN – Obligatoire
  - Ordres SQL
  - Instructions PL/SQL
- EXCEPTION – Facultatif
  - Actions à exécuter en cas d'erreur
- END; – Obligatoire

```
DECLARE
...
BEGIN
...
EXCEPTION
...
END;
```

98

Bases de données - © Christine Bonnet

## Structure BLOC PL/SQL

```
DECLARE
  v_variable VARCHAR2(5);
BEGIN
  SELECT      nomColonne
    INTO      v_variable
  FROM        nomTable;
EXCEPTION
  WHEN exception_nom_erreur THEN
  ...
END;
```

```
DECLARE
...
BEGIN
...
EXCEPTION
...
END;
```

99

Bases de données - © Christine Bonnet

## Types de BLOC

### Anonyme

```
[DECLARE]

BEGIN
  --statements

[EXCEPTION]

END;
```

### Procédure

```
PROCEDURE name
IS
BEGIN
  --statements

[EXCEPTION]

END;
```

### Fonction

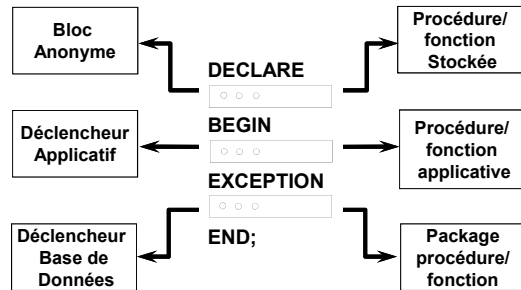
```
FUNCTION name
RETURN datatype
IS
BEGIN
  --statements
  RETURN value;
[EXCEPTION]

END;
```

100

Bases de données - © Christine Bonnet

## Utilisation d'un BLOC



101

Bases de données - © Christine Bonnet



## Variables

102

Bases de données - © Christine Bonnet

## Utilisation des variables en PL/SQL

- **Déclaration dans la section DECLARE**
- **Affectation de valeurs à la déclaration ou dans la section exécution**
- **Passage de valeurs pour les procédures et fonctions**

103

Bases de données - © Christine Bonnet



## Types de Variables

### • Variables PL/SQL :

- **Scalaire**
- **Structurée**
- **Référence**
- **LOB (Large Object)**

Convention de nommage : variable préfixe v\_  
constante préfixe c\_

Identificateur : 30 caractères maximum

### • Variables de liens (Non PL/SQL)

104

Bases de données - © Christine Bonnet

## Déclaration des Variables PL/SQL

### Syntaxe

```
Nom_variable [CONSTANT] type_donnée [NOT NULL]
[{::= | DEFAULT} expression];
```

### Exemples

```
Declare
  v_hiredate    DATE;
  v_deptno      NUMBER(2) NOT NULL := 10;
  v_location    VARCHAR2(13) := 'Atlanta';
  c_comm        CONSTANT NUMBER(4) := 1400;
```

105

Bases de données - © Christine Bonnet

## Affectations de valeurs

### Syntaxe

```
Nom_variable := expr;
```

### Exemples :

#### Affecter une date d'embauche :

```
v_hiredate := to_date('03-01-2017','DD-MM-YYYY');
```

#### Affecter un nom d'employé :

```
v_ename := 'Maduro';
```

106

Bases de données - © Christine Bonnet



## Initialisation d'une variable

### Dans la section DECLARE

- Opérateur d'affectation (:=)
- DEFAULT valeur
- NOT NULL

### Exemples :

```
v_mgr NUMBER(4) DEFAULT 7839
v_loc VARCHAR2(50) NOT NULL := 'PARIS'
```

107

Bases de données - © Christine Bonnet

## Type Scalaire

- VARCHAR2 (*longueur-maximale*)
- NUMBER [(*précision, décimales*)]
- DATE
- CHAR [(*longueur-maximale*)]
- BOOLEAN
- PLS\_INTEGER | BINARY\_INTEGER
- BINARY\_FLOAT
- BINARY\_DOUBLE
- ...

108

Bases de données - © Christine Bonnet

## Déclarations de variables de type scalaire

### Exemples

```
v_job      VARCHAR2(9);  
v_count    BINARY_INTEGER := 0;  
v_total_sal NUMBER(9,2) := 0;  
v_orderdate DATE := SYSDATE + 7;  
c_tax_rate CONSTANT NUMBER(3,2) := 8.25;  
v_valid    BOOLEAN NOT NULL := TRUE;
```

109

Bases de données - © Christine Bonnet

## Déclaration de type par référence

- Déclarer une variable par référence à :
  - une colonne de table
  - une autre variable déclarée
- Utilisation du suffixe %TYPE après :
  - nom\_table.nom\_colonne
  - nom\_variable

110

Bases de données - © Christine Bonnet

## Déclaration de type par référence

### Exemples

```
...  
v_ename      emp.ename%TYPE;  
v_balance    NUMBER(7,2);  
v_min_balance v_balance%TYPE := 10;  
...
```

111

Bases de données - © Christine Bonnet

## Déclaration de type Booléen

- Valeurs TRUE, FALSE ou NULL
- Opérateurs AND, OR, et NOT
- Possibilité d'obtenir une valeur booléenne à partir d'une expression  
`v_comm_sal BOOLEAN := (v_sal < v_comm);`

112

Bases de données - © Christine Bonnet

## Type Booléen -Table logique

AND	TRUE	FALSE	NULL	OR	TRUE	FALSE	NULL	NOT	
TRUE	TRUE	FALSE	NULL	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	NULL	FALSE	TRUE
NULL	NULL	FALSE	NULL	NULL	TRUE	NULL	NULL	NULL	NULL

113

Bases de données - © Christine Bonnet

## Type Varchar2 - Délimiteur de chaîne de caractères

q' suivi du délimiteur

DECLARE

v\_evenement1 VARCHAR2(15);

v\_evenement2 VARCHAR2(15);

v\_evenement3 VARCHAR2(15);

BEGIN

v\_evenement1 := q'! début d'année!';

v\_evenement2 := q'[fin d'année]';

v\_evenement3 := 'milieu d'année';

...

Événement 1 : début d'année  
Événement 2 : fin d'année  
Événement 3 : milieu d'année

114

Bases de données - © Christine Bonnet



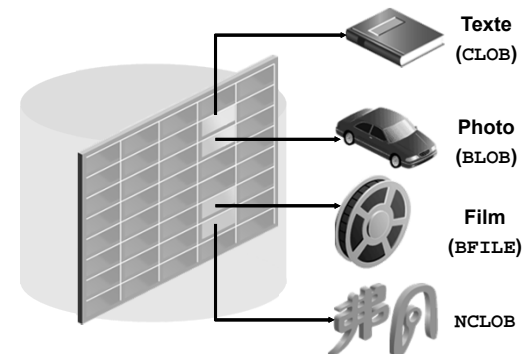
## Types Structurés

- PL/SQL Table
- PL/SQL Enregistrement (RECORD)
- Table imbriquée, VARRAY (non étudiés ici)

115

Bases de données - © Christine Bonnet

## Type LOB

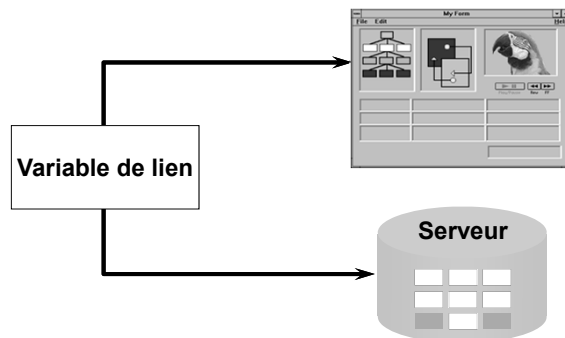


116

Bases de données - © Christine Bonnet



## Variables de lien (variable hôte)



117

Bases de données - © Christine Bonnet

## Déclaration d'une variable de lien dans l'environnement SQL\*Plus

### Commande SQL\*Plus

**VARIABLE nomVariable Type**

```
VARIABLE salaire_mensuel NUMBER  
VARIABLE nom_emp VARCHAR2(15)
```

118

Bases de données - © Christine Bonnet

## Référencer une variable de lien dans un bloc PL/SQL

Préfixer le nom de variable par ":"

### Exemple

Ranger le salaire annuel dans la variable de lien  
salaire\_mensuel

```
BEGIN  
...  
:salaire_mensuel := v_sal / 12;  
...  
End;
```

119

Bases de données - © Christine Bonnet

## Affichage d'une variable de lien dans l'environnement SQL\*Plus

### Commande SQL\*Plus

**Print nomVariable**

### Exemple

```
BEGIN  
...  
:salaire_mensuel := v_sal / 12;  
...  
End;  
/  
Print salaire_mensuel
```

120

Bases de données - © Christine Bonnet

## Variable de substitution

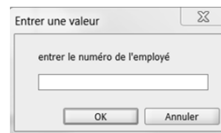
- Sont utilisées pour obtenir des valeurs entrées par l'utilisateur lors de l'exécution d'un bloc PL/SQL

Commande ACCEPT

```
ACCEPT p_empno
```

## Affichage d'un message pour l'utilisateur

```
ACCEPT p_empno PROMPT "Entrez le numéro de l'employé"
```



121

Bases de données - © Christine Bonnet

- Sont référencées dans un bloc PL/SQL avec le préfixe & devant le nom de la variable

```
ACCEPT p_empno PROMPT "Entrez le numéro de l'employé"
DECLARE
v_nom_emp emp.ename%type;
BEGIN
SELECT ename INTO v_nom_emp FROM emp
WHERE empno = &p_empno;
...
END;
```

- Permettent d'éviter de coder des valeurs en dur

122

Bases de données - © Christine Bonnet

## SORTIE ECRAN EN PL/SQL DBMS\_OUTPUT.PUT\_LINE(chaine);

- Les sorties écran doivent préalablement être activées pour la session. Commande SQL\*PLUS :

SET SERVEROUTPUT ON

- DBMS\_OUTPUT : package fourni par Oracle
- Procédure PUT\_LINE de ce package : affichage écran

```
DBMS_OUTPUT.PUT_LINE('Salaire mensuel : ' || v_sal);
```

123

Bases de données - © Christine Bonnet



## Instructions

124

Bases de données - © Christine Bonnet



## BLOC PL/SQL Syntaxe

- Une instruction peut être écrite sur plusieurs lignes
- Chaque instruction est terminée par ";"
- Identificateur :
  - permet de référencer un élément PL/SQL
  - doit commencer par une lettre
  - maximum 30 caractères

125

Bases de données - © Christine Bonnet

## Lignes de Commentaire

- Une seule ligne : deux tirets (--) en début de ligne
- Plusieurs lignes : entre les symboles /\* et \*/

### Exemple

```
...  
v_sal NUMBER(9,2);  
BEGIN  
/* calcul du salaire annuel à partir de données  
fournies par l'utilisateur */  
v_sal := :p_monthly_sal * 12;  
END; -- fin du bloc
```

126

Bases de données - © Christine Bonnet



## Fonctions SQL en PL/SQL

- Utilisables directement (sans ordre SELECT)
  - fonction-ligne numérique (round, trunc, ...)
  - fonction-ligne alphanumérique (lower, upper, length, ...)
  - conversion de type (to\_char, to\_date, ...)
  - date (months\_between, last\_day, ...)
- Non utilisables (directement)
  - decode
  - fonctions de groupe

127

Bases de données - © Christine Bonnet

## Fonctions SQL en PL/SQL

### Exemples

- formater l'adresse d'une entreprise

```
v_mailing_address := v_name || CHR(10) ||  
v_address || CHR(10) || v_state ||  
CHR(10) || v_zip;
```

- Mettre le nom d'un employé en lettres minuscules

```
v_ename := LOWER(v_ename);
```

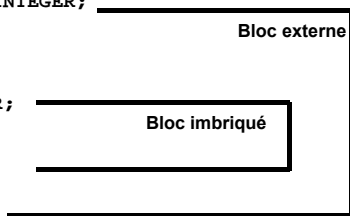
128

Bases de données - © Christine Bonnet

## Blocs Imbriqués (Nested Blocks)

### Exemple

```
...  
  v_x BINARY_INTEGER;  
BEGIN  
  ...  
  DECLARE  
    v_y NUMBER;  
  BEGIN  
    ...  
  END;  
  ...  
END;
```



129

Bases de données - © Christine Bonnet

## Blocs Imbriqués

- Un bloc peut être inséré en lieu et place d'une instruction
- Un bloc imbriqué correspond à une instruction
- La section EXCEPTION peut contenir des blocs imbriqués

130

Bases de données - © Christine Bonnet

## Blocs Imbriqués Visibilité des variables

Un identificateur (variable, curseur,...) est visible dans tous les blocs imbriqués par rapport à celui où il est défini

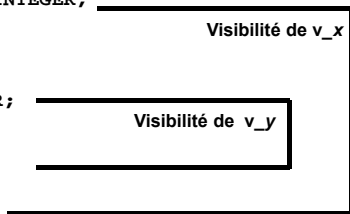
131

Bases de données - © Christine Bonnet

## Blocs Imbriqués Visibilité des variables

### Exemple

```
...  
  v_x BINARY_INTEGER;  
BEGIN  
  ...  
  DECLARE  
    v_y NUMBER;  
  BEGIN  
    ...  
  END;  
  ...  
END;
```



132

Bases de données - © Christine Bonnet

## Blocs Imbriqués Visibilité des variables

```
...  
DECLARE  
V_SAL          NUMBER(7,2) := 60000;  
V_COMM         NUMBER(7,2) := V_SAL * .20;  
V_MESSAGE      VARCHAR2(255) := ' éligible pour une commission';  
BEGIN ...  


```
DECLARE  
V_SAL          NUMBER(7,2) := 50000;  
V_COMM         NUMBER(7,2) := 0;  
V_TOTAL_COMP   NUMBER(7,2) := V_SAL + V_COMM;  
BEGIN ...  
V_MESSAGE := 'CLERK non' || V_MESSAGE;  
END;
```



```
V_MESSAGE := 'SALESMAN' || V_MESSAGE;  
END;
```


```

133

Bases de données - © Christine Bonnet



## Opérateurs en PL/SQL

- Logique
- Arithmétique
- Concaténation (||)
- Opérateur exponentiel (\*\*)

(précédence des opérateurs → parenthèses possibles)

134

Bases de données - © Christine Bonnet

## Opérateurs en PL/SQL

### Exemples

- Incrément de l'indice d'une boucle

```
v_count := v_count + 1;
```

- Initialisation de valeur pour un indicateur booléen

```
v_equal := (v_n1 = v_n2);
```

- Test de la valeur d'un numéro d'employé

```
v_valid := (v_empno IS NOT NULL);
```

135

Bases de données - © Christine Bonnet



## Accès aux données

136

Bases de données - © Christine Bonnet

## Ordres SQL en PL/SQL

- Consultation par SELECT : une seule ligne peut être retournée
- Modification des données par les ordres de manipulation INSERT, UPDATE DELETE
- Contrôle des transactions par COMMIT, ROLLBACK, ou SAVEPOINT
- Curseur implicite

137

Bases de données - © Christine Bonnet

## Ordre SELECT en PL/SQL

### Utilisation de la clause INTO

#### Syntaxe

```
SELECT liste_de_projections
INTO   {nom variable[, nom variable]...
       | nom enregistrement}
FROM   table
WHERE  condition;
```

138

Bases de données - © Christine Bonnet

### Exemple 1

```
DECLARE
  v_deptno dept.deptno%type;
  v_loc    dept.loc%type;
BEGIN
  SELECT deptno, loc
  INTO   v_deptno, v_loc
  FROM   dept
  WHERE  dname = 'SALES';
  ...
END;
```

139

Bases de données - © Christine Bonnet

### Exemple 2

#### Montant total des salaires des employés d'un département

```
DECLARE
  v_sum_sal emp.sal%TYPE;
  v_deptno  dept.deptno%type NOT NULL := 10;
BEGIN
  SELECT SUM(sal) -- fonction de groupe
  INTO   v_sum_sal
  FROM   emp
  WHERE  deptno = v_deptno;
END;
```

140

Bases de données - © Christine Bonnet



## Exercices Question 1

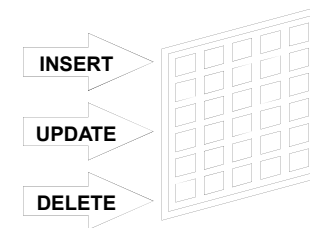
141

Bases de données - © Christine Bonnet

## Mise à jour des données

Utilisation des ordres :

- INSERT
- UPDATE
- DELETE



142

Bases de données - © Christine Bonnet

## Ajout de données

### Exemple

Ajout d'un nouvel employé dans la table EMP

```
BEGIN
  INSERT INTO emp(empno, ename, job, deptno)
  VALUES(empno_sequence.NEXTVAL, 'HARDING',
  'CLERK', 10);
  COMMIT;
END;
```

143

Bases de données - © Christine Bonnet

## Modification de données

### Exemple

Modification de la valeur du salaire des employés 'ANALYST'

```
DECLARE
  v_sal_increase  emp.sal%TYPE := 2000;
BEGIN
  UPDATE    emp
  SET       sal = sal + v_sal_increase
  WHERE     job = 'ANALYST';
  COMMIT;
END;
```

144

Bases de données - © Christine Bonnet

## Suppression de données

### Exemple

Supprimer les employés d'un département

```
DECLARE
  v_deptno emp.deptno%TYPE := 10;
BEGIN
  DELETE FROM emp
  WHERE deptno = v_deptno;
  COMMIT;
END;
```

## Ordres COMMIT et ROLLBACK

- Le premier ordre LMD modifiant des données débute une transaction
- Fin de transaction explicite : **COMMIT**, **ROLLBACK**, ou **ROLLBACK TO SAVEPOINT nomSavePoint**



## Structures de contrôle



## Structures de contrôle

Deux structures :

- alternative
- répétitive

## Structures de contrôle

### STRUCTURE ALTERNATIVE

#### - Instruction IF

Trois formes :

- IF THEN END IF
- IF THEN ELSE END IF
- IF THEN ELSIF END IF

#### - Instruction CASE

149

Bases de données - © Christine Bonnet

## Instruction IF

### Syntaxe

```
IF condition THEN
  instructions;
[ELSIF condition THEN
  instructions;]
[ELSE
  instructions;]
END IF;
```

### Exemple

```
IF v_ename = 'OSBORNE' THEN
  v_mgr := 22;
END IF;
```

150

Bases de données - © Christine Bonnet

## Instruction IF THEN ELSE

### Exemple

```
...
IF v_shipdate - v_orderdate < 5 THEN
  v_ship_flag := 'Acceptable';
ELSE
  v_ship_flag := 'Unacceptable';
END IF;
...
```

151

Bases de données - © Christine Bonnet

## Instruction IF THEN ELSIF

### Exemple

```
...
IF v_start > 100 THEN
  v_start := 2 * v_start;
ELSIF v_start >= 50 THEN
  v_start := .5 * v_start;
ELSE
  v_start := .1 * v_start;
END IF;
...
```

152

Bases de données - © Christine Bonnet

## Instruction CASE

### Syntaxe

```
CASE sélecteur
  WHEN expression1 THEN résultat1
  WHEN expression2 THEN résultat2
  ...
  WHEN expressionN THEN résultatN
  [ELSE autreRésultat]
END CASE;
```

### Exemple

```
CASE v_niveau
  WHEN 'A' THEN 'Excellent'
  WHEN 'B' THEN 'Très bon'
  WHEN 'C' THEN 'Bon'
  ELSE 'Ce niveau n''existe pas'
END CASE;
```

153

Bases de données - © Christine Bonnet

## Structure répétitive



- Une boucle répète une *instruction* ou une *séquence d'instructions* plusieurs fois
- Trois possibilités :
  - instruction LOOP
  - Instruction FOR
  - instruction WHILE



154

Bases de données - © Christine Bonnet

## Instruction Loop

### Syntaxe

```
LOOP                                -- début de boucle
  instruction(s);                  -- instructions
  . . .
  EXIT [WHEN condition];          -- EXIT instruction
END LOOP;                          -- fin de boucle
```

155

Bases de données - © Christine Bonnet

### Exemple

```
DECLARE
  v_ordid   item.ordid%TYPE := 601;
  v_counter NUMBER(2) := 1;
BEGIN
  LOOP
    INSERT INTO item(ordid, itemid)
      VALUES(v_ordid, v_counter);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 10;
  END LOOP;
  COMMIT;
END;
```

156

Bases de données - © Christine Bonnet



## Instruction FOR

### Syntaxe

```
FOR indice in [REVERSE]
  borne_inférieure..borne_supérieure LOOP
  instruction 1;
  instruction 2;
  . . .
END LOOP;
```

- Le nombre de répétitions est contrôlé par l'indice
- Ne pas déclarer l'indice; sa déclaration est implicite

## Instruction FOR

### Règles :

- L'indice n'est utilisable qu'à l'intérieur de la boucle
- Il est interdit d'affecter une valeur à l'indice

### Exemple

Création de 10 lignes pour la commande de n° 601

```
DECLARE
  v_ordid  item.ordid%TYPE := 601;
BEGIN
  FOR i IN 1..10 LOOP
    INSERT INTO item(ordid, itemid)
      VALUES(v_ordid, i);
  END LOOP;
  COMMIT;
END;
```

## Instruction WHILE

### Syntaxe

```
WHILE condition LOOP ← La condition
  instruction 1;          est évaluée
  instruction2;           en début
  . . .                  de boucle
END LOOP;
```

Les instructions de la boucle sont répétées tant que la condition est vraie

## Exemple

Variable de substitution

```
ACCEPT p_new_order PROMPT 'Enter the order number: '
ACCEPT p_items -
  PROMPT 'Enter the number of items in this order: '
DECLARE
v_count      NUMBER(2) := 1;
BEGIN
  WHILE v_count <= &p_items LOOP
    INSERT INTO item (ordid, itemid)
      VALUES (&p_new_order, v_count);
    v_count := v_count + 1;
  END LOOP;
  COMMIT;
END;
/
```

161

Bases de données - © Christine Bonnet

## Structures imbriquées et étiquettes

- Plusieurs niveaux d'imbrication possibles
- Utiliser des étiquettes pour différencier BLOC et structures imbriquées
- Possibilité de sortir d'une boucle interne par l'ordre EXIT

162

Bases de données - © Christine Bonnet

## Structures imbriquées et étiquettes

```
...
BEGIN
  <<Boucle-externe>>
  LOOP
    v_counter := v_counter+1;
    EXIT WHEN v_counter>10;
    <<Boucle-interne>>
    LOOP
      ...
      EXIT Boucle-externe WHEN prédicat;
      -- Sortie des deux boucles
      EXIT WHEN prédicat;
      -- sortie de la boucle interne uniquement
      ...
    END LOOP boucle-interne;
    ...
  END LOOP Boucle-externe;
END;
```

163

Bases de données - © Christine Bonnet



## Exercices Questions 2, 3, 4

164

Bases de données - © Christine Bonnet



## Utilisation des types structurés :

- enregistrement
- tableau

165

Bases de données - © Christine Bonnet

## Types Structurés

- Types étudiés :
  - enregistrement (RECORD)
  - tableau (TABLE PL/SQL)
- Contiennent des composants internes
- Sont réutilisables

166

Bases de données - © Christine Bonnet



## Enregistrement PL/SQL

- Peut contenir un ou plusieurs composants de type: scalaire, RECORD ou TABLE PL/SQL
- Identique à la structure d'enregistrement en L3G
- Différent de la notion de ligne de table relationnelle
- Considère un ensemble de champs comme une unité logique
- Peut être utilisé pour recevoir une ligne d'une table

167

Bases de données - © Christine Bonnet

## Déclaration d'un type Enregistrement

### Syntaxe

- 1 `TYPE nom_type IS RECORD -- déclaration de type  
(déclaration de champ[, déclaration de champ]...);`
- 2 `-- déclaration d'une variable de ce type  
nom_variable nom_type; -- déclaration de variable`

### Avec déclaration de champ :

```
Nom_champ {type_champ | variable%TYPE  
| table.colonne%TYPE | table%ROWTYPE}  
[[NOT NULL] {:= | DEFAULT} expression]
```

168

Bases de données - © Christine Bonnet

### Exemple

**Déclarations d'un type enregistrement pour stocker nom, emploi, et salaire d'un employé et d'une variable de ce type**

```
...  
TYPE emp_record_type IS RECORD  
  (ename      VARCHAR2(10),  
   job        VARCHAR2(9),  
   sal        NUMBER(7,2));  
emp_record    emp_record_type;  
...
```

169

Bases de données - © Christine Bonnet

### Utilisation de %ROWTYPE

- Permet de déclarer une variable de même structure qu'une ligne d'une table ou d'une vue
- Syntaxe : **Nom\_table%ROWTYPE**
- Les champs de l'enregistrement ont même noms et même types que ceux des colonnes de la table ou de la vue

170

Bases de données - © Christine Bonnet

### Exemples

**Déclarer une variable pour stocker la même information que celle définie dans la table DEPT**

```
dept_record    dept%ROWTYPE;
```

**Déclarer une variable pour stocker la même information que celle définie dans la table EMP**

```
emp_record    emp%ROWTYPE;
```

171

Bases de données - © Christine Bonnet

### Avantage de %ROWTYPE

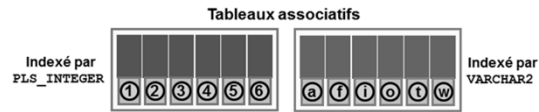
- Il n'est pas nécessaire de connaître les caractéristiques des colonnes de la ligne de référence
- Mise à jour automatique en cas de modification de la structure de la ligne de référence
- Utilisable avec **SELECT** pour recueillir les données d'une ligne

172

Bases de données - © Christine Bonnet



## Tables PL/SQL (index by table)



- Ensemble de paires clé-valeur. Clé unique, valeur de type :
  - scalaire
  - enregistrement
- Référence à un poste par la clé de type entier (BINARY\_INTEGER ou PLS\_INTEGER) ou chaîne (VARCHAR2(taille))

173

Bases de données - © Christine Bonnet

## Déclaration d'un type Table

### Syntaxe – poste de type scalaire

- 1 

```
TYPE nom_type IS TABLE OF
  {type_colonne | variable%TYPE
  | table.colonne%TYPE} [NOT NULL]
  INDEX BY BINARY_INTEGER | PLS_INTEGER |
  VARCHAR2(taille);
```
- 2 

```
-- déclaration d'une variable de ce type
nom_variable    nom_type;
```

174

Bases de données - © Christine Bonnet

### Exemple

#### Déclarer une table de noms et une variable de type table de noms

```
...
TYPE nom_table_type IS TABLE OF emp.ename%TYPE
  INDEX BY BINARY_INTEGER;
table_nom    nom_table_type;
...
```

175

Bases de données - © Christine Bonnet

## Structure d'une table PL/SQL

Clé primaire	Colonne
1	Jones
2	Smith
3	Maduro
...	...
BINARY_INTEGER	Scalaire

176

Bases de données - © Christine Bonnet

### Exemple : Déclaration et utilisation

```
DECLARE
  TYPE ename_table_type IS TABLE OF emp.ename%TYPE
    INDEX BY BINARY_INTEGER;
  TYPE hiredate_table_type IS TABLE OF DATE
    INDEX BY BINARY_INTEGER;
  ename_table      ename_table_type;
  hiredate_table   hiredate_table_type;
BEGIN
  ename_table(1) := 'CAMERON';
  hiredate_table(8) := SYSDATE + 7;
  IF ename_table.EXISTS(1) THEN
    INSERT INTO ...
  ...
END;
```

177

Bases de données - © Christine Bonnet

### Méthodes associées aux index by tables

#### Méthodes fournies en standard :

- |               |          |
|---------------|----------|
| • EXISTS      | • NEXT   |
| • COUNT       | • EXTEND |
| • FIRST, LAST | • TRIM   |
| • PRIOR       | • DELETE |

178

Bases de données - © Christine Bonnet

- Exists(n) : vrai si le n-ième élément de la table existe
- Count : nombre d'éléments de la table
- First/last : 1 et dernière valeur d'index, null si la table est vide
- Prior(n) : valeur de l'index précédant l'index n
- Next(n) : valeur de l'index suivant l'index n
- Extend : ajoute un élément null à la table
- Extend(n) : ajoute n éléments à la table
- Extend(n,i) : ajoute n copies du i-ème élément
- Trim : supprime 1 élément à partir de la fin de la table
- Trim(n) : supprime n éléments à partir de la fin de la table
- Delete : supprime tous les éléments

179

Bases de données - © Christine Bonnet

### index by Table d'enregistrements

#### Syntaxe - poste de type enregistrement

- Utilisation de %ROWTYPE

#### Exemple

**Déclarer un type table pour recevoir les lignes de la table DEPT et une variable de ce type**

```
DECLARE
  TYPE dept_table_type IS TABLE OF dept%ROWTYPE
    INDEX BY BINARY_INTEGER;
  dept_table dept_table_type;
```

180

Bases de données - © Christine Bonnet



## Exercices Question 5

181

Bases de données - © Christine Bonnet



## Curseur

- Un curseur est un pointeur vers la zone mémoire privée allouée par le serveur Oracle
- Tout ordre SQL utilise un curseur pour s'exécuter :
  - curseur implicite (curseur nommé SQL)
    - tout ordre LMD
    - SELECT ... INTO ... sous PL/SQL
  - curseur explicite
    - utilisé dans un bloc PL/SQL pour effectuer des select multilignes

182

Bases de données - © Christine Bonnet

## Structure (simplifiée) du curseur

- Zone de travail privée

Lignes sélectionnées			Ligne Courante
7369	SMITH	CLERK	
7566	JONES	MANAGER	
7788	SCOTT	ANALYST	
7876	ADAMS	CLERK	
7902	FORD	ANALYST	

Curseur

183

Bases de données - © Christine Bonnet

## Code statut d'un curseur implicite : 4 attributs

Positionné à la fin de l'exécution d'un ordre SQL  
Informe sur la façon dont il s'est déroulé

SQL%ISOPEN	positionné à VRAI si le curseur est ouvert
SQL%ROWCOUNT	Nombre de lignes traitées (entier)
SQL%FOUND	positionné à VRAI si l'ordre a traité une ou plusieurs lignes
SQL%NOTFOUND	positionné à VRAI si l'ordre n'a pas traité de ligne

184

Bases de données - © Christine Bonnet

## Exemple

### Affichage du nombre de lignes supprimées par un ordre Delete

```
VARIABLE rows_deleted VARCHAR2(30)
DECLARE
  v_ordid  NUMBER(4) := 605;
BEGIN
  DELETE FROM item
  WHERE ordid = v_ordid;
  :rows_deleted := (SQL%ROWCOUNT ||
                   ' rows deleted. ');
  COMMIT;
END;
/
PRINT rows_deleted
```

185

Bases de données - © Christine Bonnet

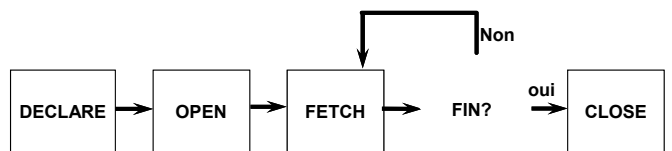


## Accès multilignes CURSEUR EXPLICITE

186

Bases de données - © Christine Bonnet

## Mise en œuvre d'un curseur explicite

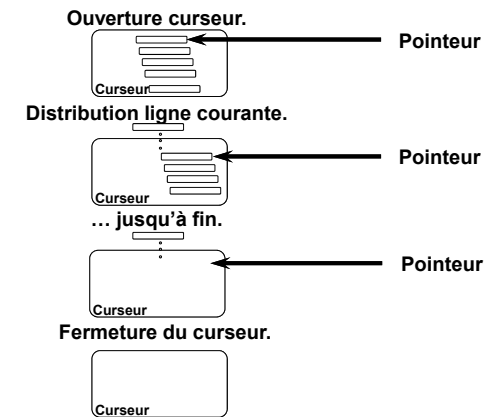


- Déclaration requête SQL (select multilignes)
- Ouverture et exécution
- Distribution ligne courante
- Teste existence ligne
- Libération du curseur

187

Bases de données - © Christine Bonnet

## Mise en œuvre curseur explicite



188

Bases de données - © Christine Bonnet



## Déclaration du curseur

### Syntaxe

```
CURSOR nom_curseur IS  
    requête;
```

- Requête sans clause INTO
- Possibilité de clause ORDER BY

## Exemple

```
DECLARE  
    CURSOR emp_cursor IS  
        SELECT empno, ename  
        FROM   emp;  
  
BEGIN  
    ...
```



## Ouverture du curseur

### Syntaxe

```
OPEN nom_curseur;
```

- Exécution de la requête et génération des lignes résultats au niveau du serveur
- Pas d'erreur si la requête ne sélectionne pas de ligne
- Possibilité de tester le statut du curseur après exécution de l'ordre FETCH



## Distribution des lignes

### Syntaxe

```
FETCH nom_curseur INTO [variable1, variable2,  
    ...]  
/ [nom_enregistrement];
```

- Distribue les valeurs des colonnes de la ligne courante dans les variables de réception (ou dans un enregistrement)
- Effectue une correspondance par position
- Renvoie un code statut

## Mise en œuvre de l'ordre FETCH

- Inclure l'ordre FETCH dans une structure répétitive
- Une ligne est distribuée à chaque itération
- Utiliser %NOTFOUND ou %FOUND pour contrôler la sortie de la boucle

193

Bases de données - © Christine Bonnet

## Exemples

```
FETCH emp_cursor INTO v_empno, v_ename;
```

```
...  
OPEN nom_curseur;  
LOOP  
    FETCH nom_curseur INTO variables...;  
    EXIT WHEN nom_curseur%NOTFOUND OR ...;  
    ...  
    -- utilisation des valeurs distribuées à  
    -- chaque itération  
    ...  
END LOOP;  
CLOSE nom_curseur;
```

194

Bases de données - © Christine Bonnet



## Fermeture du curseur

### Syntaxe

```
CLOSE nom_curseur;
```

- Ferme le curseur et libère les ressources
- Possibilité de ré-ouvrir le même curseur (la requête s'exécute à nouveau)

195

Bases de données - © Christine Bonnet

## Code statut d'un curseur explicite : 4 attributs

Informe sur la façon dont s'est déroulé  
l'ordre SQL

Code mnémonique	Type	Description
%ISOPEN	Booléen	VRAI si le curseur est ouvert
%NOTFOUND	Booléen	VRAI si le dernier ordre fetch exécuté n'a pas distribué de ligne
%FOUND	Booléen	VRAI si le dernier ordre fetch exécuté a distribué une ligne - complément de %NOTFOUND
%ROWCOUNT	Nombre	Nombre de lignes distribuées

196

Bases de données - © Christine Bonnet

## %ISOPEN

- La distribution de ligne ne s'effectue que pour un curseur ouvert
- Permet de savoir si un curseur est ouvert avant d'exécuter un ordre fetch

### Exemple

```
IF NOT emp_cursor%ISOPEN THEN
  OPEN emp_cursor;
END IF;
LOOP
  FETCH emp_cursor...
```

197

Bases de données - © Christine Bonnet

## NOTFOUND, ROWCOUNT et FOUND

- %ROWCOUNT donne, après chaque exécution de l'ordre fetch, le nombre de lignes distribuées
- %NOTFOUND indique la fin de distribution des lignes d'un curseur
- %FOUND, testé après exécution du premier ordre fetch, indique si la requête a sélectionné au moins une ligne

198

Bases de données - © Christine Bonnet

## Curseur et Enregistrement

### Distribution des données de la ligne dans une structure RECORD

### Exemple

```
DECLARE
  CURSOR emp_cursor IS
    SELECT empno, ename
    FROM emp;
  emp_record emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO emp_record;
    ...
```

199

Bases de données - © Christine Bonnet

## Curseur explicite FOR LOOP

### Syntaxe

```
FOR nom_enregistrement IN nom_curseur LOOP
  instruction1; instruction2;
  . . .
END LOOP;
```

- Raccourci pour gérer la distribution des lignes
- Exécute toutes les étapes (open, fetch, close)
- Déclaration implicite de l'enregistrement

200

Bases de données - © Christine Bonnet

## Curseur explicite FOR LOOP

### Exemple

```
DECLARE
  CURSOR emp_cursor IS
    SELECT ename, deptno
    FROM emp;
BEGIN
  FOR emp_record IN emp_cursor LOOP
    -- open et fetch implicites
    IF emp_record.deptno = 30 THEN
      ...
    END LOOP; -- close implicite
  END;
END;
```

201

Bases de données - © Christine Bonnet

## Curseur implicite FOR LOOP sans déclaration de curseur

### Exemple 1

```
BEGIN
  FOR emp_record IN (SELECT ename, deptno FROM
    emp) LOOP
    -- open et fetch implicites
    IF emp_record.deptno = 30 THEN
      ...
    END LOOP; -- close implicite
  END;
END;
```

202

Bases de données - © Christine Bonnet

### Exemple 2

```
DECLARE
  TYPE typ_film IS TABLE OF film%ROWTYPE
  INDEX BY film.titre%TYPE; -- titre de type Varchar2(50)
  v_by_film typ_film;
  i VARCHAR2(60);
BEGIN
  FOR rec IN (SELECT * FROM film WHERE titre IS NOT NULL) LOOP
    v_by_film(rec.titre) := rec;
  END LOOP;
  i:= v_by_film.FIRST;
  WHILE i IS NOT NULL LOOP
    dbms_output.put_line ( 'Film '|| v_by_film(i).titre || ' --Directeur-- ' ||
      v_by_film(i).directeur || ' --Acteur principal-- ' ||
      v_by_film(i).acteurprincipal);
    i := v_by_film.NEXT(i);
  END LOOP;
END;
```

203

Bases de données - © Christine Bonnet

```
Film 7 ans au Tibet --Directeur-- Jean-jacques Annaud --Acteur principal--
Brad Pitt
Film Alien --Directeur-- Ridley Scott --Acteur principal-- Sigourney Weaver
Film Amadeus --Directeur-- Milos Forman --Acteur principal-- Tom Hulce
Film Basic instinct --Directeur-- Paul Verhoeven --Acteur principal-- Sharon
Stone
Film Braveheart --Directeur-- Mel Gibson --Acteur principal-- Mel Gibson
Film Cyrano de Bergerac --Directeur-- Jean-paul Rappeneau --Acteur
principal-- Gerard Depardieu
Film Highlander --Directeur-- Russel Mulcahy --Acteur principal--
Christophe Lambert
Film J.F.K. --Directeur-- Oliver Stone --Acteur principal-- Kevin Kostner
Film L'empire contre attaque --Directeur-- Gorges Lucas --Acteur principal--
Harrison Ford
Film La guerre des etoiles --Directeur-- Gorges Lucas --Acteur principal--
Harrison Ford
```

204

Bases de données - © Christine Bonnet



## Curseur paramétré

205

Bases de données - © Christine Bonnet

## Curseur avec paramètre(s)

### Syntaxe

```
CURSOR nom_curseur  
[(nom_paramètre type, ...)]  
IS  
  requête;
```

- Affectation des valeurs des paramètres lors de l'ouverture du curseur
- Le même curseur peut être ouvert plusieurs fois avec des valeurs de paramètres différentes

206

Bases de données - © Christine Bonnet

### Exemple

Donner le n° de département et l'emploi sous forme de paramètres, utilisés dans la clause WHERE d'une requête :

```
DECLARE  
  CURSOR emp_cursor  
  (v_deptno NUMBER, v_job VARCHAR2) IS  
  SELECT empno, ename  
  FROM emp  
  WHERE deptno = v_deptno  
  AND job = v_job;  
BEGIN  
  OPEN emp_cursor(10, 'CLERK');  
  ...
```

N.B.: pas de taille

207

Bases de données - © Christine Bonnet

### Exemple avec boucle FOR

```
DECLARE  
  CURSOR emp_cursor  
  (v_deptno NUMBER, v_job VARCHAR2) IS  
  SELECT empno, ename  
  FROM emp  
  WHERE deptno = v_deptno  
  AND job = v_job;  
BEGIN  
  FOR emp_enrg IN emp_cursor(10, 'CLERK') LOOP  
    ...
```

208

Bases de données - © Christine Bonnet



## Mise à jour avec utilisation d'un curseur

209

Bases de données - © Christine Bonnet

## Clause FOR UPDATE

### Syntaxe

```
SELECT ...  
FROM      ...  
FOR UPDATE [OF nom_colonne][NOWAIT];
```

**Verrouille les lignes sélectionnées pour la durée de la transaction**

210

Bases de données - © Christine Bonnet

### Exemple

**Sélectionner les employés du département 30 en vue de les modifier ou de les supprimer**

```
DECLARE  
CURSOR emp_cursor IS  
  SELECT empno, ename, sal  
  FROM   emp  
  WHERE  deptno = 30  
  FOR UPDATE;
```

211

Bases de données - © Christine Bonnet

## Clause WHERE CURRENT OF

### Syntaxe

```
WHERE CURRENT OF nom_curseur;
```

- Curseur en vue de modifier ou supprimer les lignes sélectionnées
- Utiliser la clause FOR UPDATE dans l'expression du curseur
- Utiliser la clause WHERE CURRENT OF pour faire référence à la dernière ligne distribuée par le curseur

212

Bases de données - © Christine Bonnet

## Exemple

```
DECLARE
CURSOR sal_cursor IS
  SELECT  sal
  FROM    emp
  WHERE   deptno = 30
  FOR UPDATE of sal;
BEGIN
  FOR emp_record IN sal_cursor LOOP
    UPDATE  emp
    SET     sal = emp_record.sal * 1.1
    WHERE CURRENT OF sal_cursor;
  END LOOP;
  COMMIT;
END;
```



## Exercices Questions 6, 7



## Gestion des exceptions

## Gestion des exceptions en PL/SQL

- **Exception ?**  
tout événement qui survient pendant l'exécution d'un ordre
- **Différents cas**
  - erreur diagnostiquée par le SGBDR
  - événement généré par le développeur
- **Gestion**
  - capture dans le module qui l'a détectée
  - propagation à l'environnement

## Gestion des exceptions en PL/SQL

### Capture de l'exception

```
DECLARE
  [ ]
BEGIN
  [ ]
EXCEPTION
  [ ]
END;
```

Création de l'exception

Capture de l'exception

### Propagation de l'exception

```
DECLARE
  [ ]
BEGIN
  [ ]
EXCEPTION
  [ ]
END;
```

Création de l'exception

L'exception n'est pas capturée

Exception propagée à l'environnement

## Types d'Exceptions

- Erreurs émises par le serveur
  - prédéfinies
  - non prédéfinies
  - déclenchées implicitement
- Exceptions générées par l'utilisateur
  - déclenchées explicitement



## Capture des Exceptions

### Syntaxe

```
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
    instruction1;
    instruction2;
    . . .
  [WHEN exception3 [OR exception4 . . .] THEN
    instruction1;
    instruction2;
    . . .]
  [WHEN OTHERS THEN
    instruction1;
    instruction2;
    . . .]
```

## Capture des Exceptions

- WHEN OTHERS est la dernière clause
- Le mot clé EXCEPTION introduit la section de gestion des exceptions
- Plusieurs gestionnaires d'exception peuvent être définis dans un même bloc
- Un seul gestionnaire d'exception est exécutée suite à la détection d'une exception, avant de sortir du bloc



# Exceptions serveur prédéfinies

- Erreur émise par le serveur
- Repérable par un nom d'erreur et un message
- Exemples :
  - NO\_DATA\_FOUND
  - TOO\_MANY\_ROWS
  - INVALID\_CURSOR
  - ZERO\_DIVIDE
  - DUP\_VAL\_ON\_INDEX

221

Bases de données - © Christine Bonnet

- # Exceptions serveur prédéfinies
- Erreur émise par le serveur
  - Repérable par un nom d'erreur et un message
  - Exemples :
    - NO\_DATA\_FOUND
    - TOO\_MANY\_ROWS
    - INVALID\_CURSOR
    - ZERO\_DIVIDE
    - DUP\_VAL\_ON\_INDEX
- 221
- Bases de données - © Christine Bonnet

# Exceptions serveur prédéfinies

- Erreur émise par le serveur
- Repérable par un nom d'erreur et un message
- Exemples :
  - NO\_DATA\_FOUND
  - TOO\_MANY\_ROWS
  - INVALID\_CURSOR
  - ZERO\_DIVIDE
  - DUP\_VAL\_ON\_INDEX

221

Bases de données - © Christine Bonnet

# Utilisation des noms d'erreurs

## Syntaxe

```
BEGIN SELECT .UPDATE.. COMMIT;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    instruction1;
    instruction2;
  WHEN TOO_MANY_ROWS THEN
    instruction1;
  WHEN OTHERS THEN
    instruction1;
    instruction2;
    instruction3;
END;
```

222

Bases de données - © Christine Bonnet

# Utilisation des noms d'erreurs

## Syntaxe

```
BEGIN SELECT .UPDATE.. COMMIT;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    instruction1;
    instruction2;
  WHEN TOO_MANY_ROWS THEN
    instruction1;
  WHEN OTHERS THEN
    instruction1;
    instruction2;
    instruction3;
END;
```

222

Bases de données - © Christine Bonnet

# Utilisation des noms d'erreurs

## Syntaxe

```
BEGIN SELECT .UPDATE.. COMMIT;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    instruction1;
    instruction2;
  WHEN TOO_MANY_ROWS THEN
    instruction1;
  WHEN OTHERS THEN
    instruction1;
    instruction2;
    instruction3;
END;
```

222

Bases de données - © Christine Bonnet

# Utilisation des noms d'erreurs

## Syntaxe

```
BEGIN SELECT .UPDATE.. COMMIT;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    instruction1;
    instruction2;
  WHEN TOO_MANY_ROWS THEN
    instruction1;
  WHEN OTHERS THEN
    instruction1;
    instruction2;
    instruction3;
END;
```

222

Bases de données - © Christine Bonnet

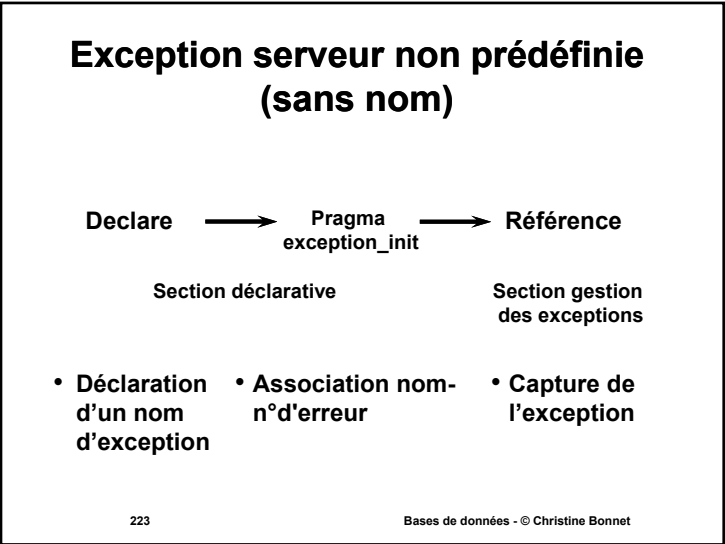
# Exception serveur non prédéfinie (sans nom)

Declare      →      Pragma  
                                exception\_init      →      Référence

Section déclarative                                      Section gestion des exceptions

- Déclaration d'un nom d'exception
- Association nom-n°d'erreur
- Capture de l'exception

223 Bases de données - © Christine Bonnet



- # Exception serveur non prédéfinie (sans nom)
- Declare      →      Pragma  
                                exception\_init      →      Référence
- Section déclarative                                      Section gestion des exceptions
- Déclaration d'un nom d'exception
  - Association nom-n°d'erreur
  - Capture de l'exception
- 223 Bases de données - © Christine Bonnet

# Exemple

## Déclaration d'un nom-erreur pour l'erreur n° -2292 (intégrité référentielle)

```
DECLARE
    e_emps_remaining    EXCEPTION;
PRAGMA EXCEPTION_INIT (
    e_emps_remaining, -2292);
v_deptno    dept.deptno%TYPE := &p_deptno;
BEGIN
    DELETE FROM dept
    WHERE      deptno = v_deptno;
    COMMIT;
EXCEPTION
    WHEN e_emps_remaining THEN
        DBMS_OUTPUT.PUT_LINE ( 'Suppression impossible
        ' || TO_CHAR(v_deptno) || ' '. Existence d'employés.
        ');
        RAISE;
END;
```

1

2

3

224

Bases de données - © Christine Bonnet

# Exemple

## Déclaration d'un nom-erreur pour l'erreur n° -2292 (intégrité référentielle)

```
DECLARE
    e_emps_remaining    EXCEPTION;
PRAGMA EXCEPTION_INIT (
    e_emps_remaining, -2292);
v_deptno    dept.deptno%TYPE := &p_deptno;
BEGIN
    DELETE FROM dept
    WHERE      deptno = v_deptno;
    COMMIT;
EXCEPTION
    WHEN e_emps_remaining THEN
        DBMS_OUTPUT.PUT_LINE ( 'Suppression impossible
        ' || TO_CHAR(v_deptno) || ' '. Existence d'employés.
        ');
        RAISE;
END;
```

1

2

3

224

Bases de données - © Christine Bonnet

# Exemple

## Déclaration d'un nom-erreur pour l'erreur n° -2292 (intégrité référentielle)

```
DECLARE
    e_emps_remaining    EXCEPTION;
PRAGMA EXCEPTION_INIT (
    e_emps_remaining, -2292);
v_deptno    dept.deptno%TYPE := &p_deptno;
BEGIN
    DELETE FROM dept
    WHERE      deptno = v_deptno;
    COMMIT;
EXCEPTION
    WHEN e_emps_remaining THEN
        DBMS_OUTPUT.PUT_LINE ( 'Suppression impossible
        ' || TO_CHAR(v_deptno) || ' '. Existence d'employés.
        ');
        RAISE;
END;
```

1

2

3

224

Bases de données - © Christine Bonnet

# Exemple

## Déclaration d'un nom-erreur pour l'erreur n° -2292 (intégrité référentielle)

```
DECLARE
    e_emps_remaining    EXCEPTION;
PRAGMA EXCEPTION_INIT (
    e_emps_remaining, -2292);
v_deptno    dept.deptno%TYPE := &p_deptno;
BEGIN
    DELETE FROM dept
    WHERE      deptno = v_deptno;
    COMMIT;
EXCEPTION
    WHEN e_emps_remaining THEN
        DBMS_OUTPUT.PUT_LINE ( 'Suppression impossible
        ' || TO_CHAR(v_deptno) || ' '. Existence d'employés.
        ');
        RAISE;
END;
```

1

2

3

224

Bases de données - © Christine Bonnet

# Exemple

## Déclaration d'un nom-erreur pour l'erreur n° -2292 (intégrité référentielle)

```
DECLARE
    e_emps_remaining    EXCEPTION;
PRAGMA EXCEPTION_INIT (
    e_emps_remaining, -2292);
v_deptno    dept.deptno%TYPE := &p_deptno;
BEGIN
    DELETE FROM dept
    WHERE      deptno = v_deptno;
    COMMIT;
EXCEPTION
    WHEN e_emps_remaining THEN
        DBMS_OUTPUT.PUT_LINE ( 'Suppression impossible
        ' || TO_CHAR(v_deptno) || ' '. Existence d'employés.
        ');
        RAISE;
END;
```

1

2

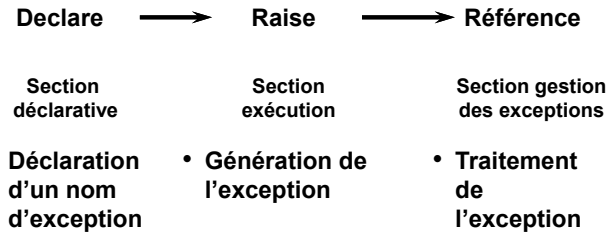
3

224

Bases de données - © Christine Bonnet



## Exception définie par l'utilisateur



225

Bases de données - © Christine Bonnet

## Exception définie par l'utilisateur

### Syntaxe

```
DECLARE
    nom_exception EXCEPTION;
BEGIN
    ...;
    RAISE nom_exception;
    ...;

EXCEPTION
    WHEN nom_exception THEN
        ...;
END;
```

226

Bases de données - © Christine Bonnet

## Exception définie par l'utilisateur

### Exemple

```
DECLARE
    e_invalid_product EXCEPTION;
BEGIN
    UPDATE    product
    SET       descrip = '&product_description'
    WHERE     prodid = &product_number;
    IF SQL%NOTFOUND THEN
        RAISE e_invalid_product;
    END IF;
    COMMIT;
EXCEPTION
    WHEN e_invalid_product THEN
        DBMS_OUTPUT.PUT_LINE(' N° produit inconnu. ');
        RAISE;
END;
```

①

②

③

227

Bases de données - © Christine Bonnet



## Procédure RAISE\_APPLICATION\_ERROR

### Syntaxe

```
raise_application_error (numéro_erreur,
                        message[, {TRUE | FALSE}]);
```

- Permet de définir une erreur (numéro [entre -20000 et -20999] et texte du message) dans un bloc PL/SQL
- Utilisable dans les sections de code d'un bloc PL/SQL

228

Bases de données - © Christine Bonnet

## Procédure RAISE\_APPLICATION\_ERROR

- Utilisable
  - dans la section Exécution
  - dans la section Exception
- La génération de l'erreur est conforme au standard du serveur et est traitable comme telle

229

Bases de données - © Christine Bonnet

## Procédure RAISE\_APPLICATION\_ERROR

### Exemple

```
...  
EXCEPTION  
  WHEN NO_DATA_FOUND THEN  
    RAISE_APPLICATION_ERROR (-20201,  
      ' Ligne NON trouvée ');  
END;
```

230

Bases de données - © Christine Bonnet

## Informations associées à toute erreur

2 fonctions d'interception des exceptions :

- **SQLCODE**

renvoie la valeur numérique de l'erreur



- **SQLERRM**

renvoie le texte du message associé à l'erreur

231

Bases de données - © Christine Bonnet

## Informations associées à une exception serveur

```
DECLARE  
  v_error_code    NUMBER(6);  
  v_error_message VARCHAR2(255);  
BEGIN  
  ...  
EXCEPTION  
  ...  
  WHEN OTHERS THEN  
    ROLLBACK;  
    v_error_code := SQLCODE;   
    v_error_message := SQLERRM;   
    INSERT INTO errors VALUES(v_error_code,  
                              v_error_message);  
    COMMIT;  
END;
```

232

Bases de données - © Christine Bonnet



## Propagation d'une exception

Un bloc peut gérer ses exceptions ou les transmettre au bloc de niveau supérieur

```
DECLARE
. . .
e_no_rows      exception;
e_integrity     exception;
PRAGMA EXCEPTION_INIT (e_integrity, -2292);
BEGIN
FOR c_record IN emp_cursor LOOP
  BEGIN
    SELECT ...
    UPDATE ...
    IF SQL%NOTFOUND THEN
      RAISE e_no_rows;
    END IF;
  EXCEPTION
    WHEN e_integrity THEN ...
    WHEN e_no_rows THEN ...
  END;
END LOOP;
EXCEPTION
  WHEN NO_DATA_FOUND THEN . . .
  WHEN TOO_MANY_ROWS THEN . . .
END;
```



## Exercices