IUT d'Aix-en-Provence
Département Informatique

Mémento PL/SQL

Structure d'un bloc PL/SQL

```
[<specification_en_tete>]
[DECLARE | IS]
   [<liste_déclarations>]
 [BEGIN]
     <liste_instructions>
 [EXCEPTION
     <gestion_exceptions>]
 [END] ;
```

- La spécification d'un en-tête est inutile pour un bloc anonyme. Elle n'est utilisée que pour les procédures et les fonctions.
- La partie déclarative `<liste_déclarations>` n'est obligatoire que s'il y a des variables, constantes, curseurs, exceptions, types ou autres traitements (i.e. procédures ou fonctions) locaux au bloc.
- Le mot clef **`DECLARE`** ne doit être utilisé que pour introduire des déclarations dans les blocs anonymes. Le mot clef **`IS`** est obligatoire pour les procédures et les fonctions (qu'elles aient ou pas des déclarations locales).
- Dans un bloc anonyme, **`BEGIN`** et **`END`** ne sont obligatoires que s'il y a une partie déclarative. Dans un bloc nommé, **`BEGIN`** et **`END`** sont obligatoires.
- La partie `<liste_instructions>` est obligatoire et peut éventuellement être réduite à l'instruction " **`NULL ;`** " qui ne fait rien.
- La partie `<gestion_exceptions>` est optionnelle. Elle est utilisée pour traiter les anomalies détectées lors de l'exécution du bloc.

Partie déclarative d'un bloc PL/SQL

Variables scalaires

Types : un des types prédéfinis d'ORACLE, i.e. ceux utilisés pour définir les attributs (e.g. **`VARCHAR2`**`(n)`, **`NUMBER`**`(x,y)`, **`DATE`** …) ou le type **`BOOLEAN`** (**`TRUE`**, **`FALSE`** et … **`NULL`**).

```
      <nom_variable> <nom_type> [NOT NULL] [DEFAULT <valeur_defaut>] ;
ou    <nom_variable> <nom_type> [NOT NULL] [:=<valeur_defaut>] ;
```

```
     <nom_variable> <nom_reference>%TYPE [NOT NULL] [DEFAULT <valeur_defaut>];
ou   <nom_variable> <nom_reference>%TYPE [NOT NULL] [:=<valeur_defaut>] ;
```

`<nom_reference>` correspond au nom d'une autre variable ou d'un attribut de la base préfixé par le nom de la relation à laquelle il appartient : `<nom_relation>.<nom_attribut>`.

Constantes

```
<nom_constante>   CONSTANT   <nom_type>  := valeur ;
```

Variables composées

```
TYPE <nom_type_enreg> IS RECORD
     (<nom_champ1> <nom_type1> [NOT NULL] [:= <valeur_par_defaut>],
     [<nom_champ2> <nom_type2> [NOT NULL] [:= <valeur_par_defaut>],… ]) ;

<nom_variable_enreg> <nom_type_enreg> ;
<nom_variable_enreg> <nom_table|nom_curseur>.%ROWTYPE ;


TYPE <type_enreg1> IS RECORD
     (<nom_champ1> <type_scal_1> [NOT NULL] [:= <valeur_par_defaut>],
     [<nom_champ2> <type_scal_2> [NOT NULL] [:= <valeur_par_defaut>],…]);

TYPE <type_enreg2> IS RECORD
     (<nom_champ3> <type_enreg1>,
     [<nom_champ4> <type_scal_1> [NOT NULL] [:= <valeur_par_defaut>],…]);
```

Si la variable `<nom_variable_enreg>` est déclarée du type `<type_enreg2>`, ses champs sont désignés par :

```
<nom_variable_enreg>.<nom_champ4>
<nom_variable_enreg>.<nom_champ3>
<nom_variable_enreg>.<nom_champ3>.<nom_champ1>
<nom_variable_enreg>.<nom_champ3>.<nom_champ2>
```

```
TYPE <nom_type_table> IS TABLE OF <type_element> [NOT NULL]
INDEX BY BINARY_INTEGER ;

<nom_variable_table> <nom_type_table> ;
```

Primitives de manipulation des variables tableaux : `<nom_variable_tab>.EXISTS(n)` ; `<nom_variable_tab>.COUNT` ; `<nom_variable_tab>.FIRST` et `<nom_variable_tab>.LAST` ; `<nom_variable_tab>.PRIOR(n)` et `<nom_variable_tab>.NEXT(n)`.

Exceptions

```
nom_exception   EXCEPTION ;
```

Instructions PL/SQL

Affectation

```
<nom_variable_scalaire> := <expression> ;
<nom_variable_enreg>.<nom_attribut> := <expression> ;
<nom_variable_enreg>.<nom_champ> := <expression> ;
<nom_variable_enreg>.<nom_champ_comp>.<nom_champ_scal> := <expression> ;
<nom_variable_table>(valeur_index) := <expression> ;
```

Affectation par requête

```
SELECT <nom_attribut | liste_attributs | *>
INTO <variable_receptrice>
FROM ... WHERE ...
```

Instruction Conditionnelle

```
IF  <condition1>
THEN  [BEGIN]  <liste1_instructions> [END ;]
[ELSIF <condition2>
THEN  [BEGIN]  <liste2_instructions> [END ;] ]
[ELSE  [BEGIN]  <liste3_instructions> [END ;]]
END IF ;
```

Itérations

```
[<<etiquette_boucle>>]
LOOP
    <liste_instructions>
END  LOOP [etiquette_boucle] ;
```

```
[<<etiquette_boucle>>]
FOR <compteur>  IN [REVERSE]  <borne_inf> .. <borne_sup>
LOOP
    <liste_instructions>
END  LOOP [etiquette_boucle] ;
```

```
[<<etiquette_boucle>>]
WHILE <condition>
LOOP
    <liste_instructions>
END  LOOP [etiquette_boucle]  ;
```

Déclenchement d'exceptions utilisateur nommées

```
IF  <condition>  THEN  RAISE <nom_exception> ; END  IF  ;
```

Exceptions système

CURSOR_ALREADY_OPEN, DUP_VAL_ON_INDEX, INVALID_CURSOR, INVALID_NUMBER, NO_DATA_FOUND, STORAGE_ERROR, TIMEOUT_ON_RESSOURCE, TOO_MANY_ROWS, VALUE_ERROR, ZERO_DIVIDE, OTHERS.

Déclenchement d'exceptions utilisateur anonymes

```
RAISE_APPLICATION_ERROR(<code_excep_ut>, <message_erreur>)
```

où `<code_excep_ut>` est une valeur entière négative comprise entre -20000 et -20999 ;

Traitements des exceptions

```
WHEN <nom_exception>  THEN     [BEGIN]  <liste_instructions> [END]  ;
```

Ordres SQL valides en PL/SQL

- Toute requête **SELECT** … **INTO** … **FROM** … dans la liste des instructions et **SELECT** … **FROM** … dans la partie déclarative pour définir un curseur.
- Les opérations de mise à jour des données : **INSERT, UPDATE, DELETE**.
- Les ordres de gestion de transaction : **COMMIT, ROLLBACK, SAVEPOINT**.
  - **COMMIT** et **ROLLBACK** permettent de définir une transaction. En cas d'incident, les ordres de mise à jour des données pourront alors être " défaits " par **ROLLBACK** via une exception.
  - **SAVEPOINT** <nom_point_sauve> et **ROLLBACK TO** <nom_point_sauve> peuvent être utilisés pour ne défaire qu'une partie de la transaction.

Les curseurs

Définition de curseurs

```
CURSOR <nom_curseur>  IS  <requete_SQL> ;

CURSOR <nom_curseur> (<parametre1> <type1>[, <parametre2> <type2> …])
IS  <requete_SQL> ;
```

Gestion de curseurs

- **OPEN** <nom_curseur> ;
  OPEN <nom_curseur>(<valeur1>[, <valeur2> …]) ;
- **CLOSE** <nom_curseur> ;
- **FETCH** <nom_curseur> **INTO** <variable-receptrice> ;

ou

```
FOR <nom_variable_parcours> IN <nom_curseur>
LOOP
          <liste_instructions>
END  LOOP ;

FOR <variable_parcours> IN <Requete>
LOOP
          <liste_instructions>
END  LOOP ;

FOR <variable_parcours> IN <nom_curseur>(<valeur1>[, <valeur2> …]) …
LOOP
          <liste_instructions>
END  LOOP ;
```

Propriété des curseurs :

<nom_curseur>**%FOUND**, <nom_curseur>**%NOTFOUND**, <nom_curseur>**%ISOPEN**,
<nom_curseur>**%ROWCOUNT**

Procédures, Fonctions, Packages

Spécifications de procédures

```
PROCEDURE <nom_procedure>[(<specification_parametre>)]
IS
        [<liste_declarations>]
BEGIN
        <liste_instructions>
[EXCEPTION
        <gestion_exception> ]
END  [<nom_procedure>] ;
```

```
<specification_parametre> :=
    <nom_parametre> <mode> <nom_type> [:= <valeur_defaut>]
```

`<mode>` peut prendre les valeurs **`IN`**, **`OUT`** ou **`IN OUT`**.

Procédures stockées

```
CREATE [OR REPLACE] PROCEDURE <nom_procedure> <specification_procedure> ;
```

Spécifications de fonctions

```
FUNCTION <nom_fonction>[(<specification_parametre>)]
RETURN <type_donnee>
IS
        [<liste_declarations>]
BEGIN
        <liste_instructions>
[EXCEPTION
        <gestion_exception> ]
END  [<nom_fonction>] ;
```

`<type_donnee>` est un des types scalaires ou composés valides en PL/SQL et ne peut être ni un curseur ni une exception.

Fonctions stockées

```
CREATE [OR REPLACE] FUNCTION <nom_fonction> <specification_fonction> ;
```

Affichage de messages

```
DBMS_OUTPUT.PUT_LINE('texte du message') ;
```

Spécifications de packages

```
PACKAGE <nom_package>
```

```
IS
      [<liste_declarations>]
      [<specification_curseurs>]
      [<specification_procedures_fonctions>]
END [<nom_package>] ;


PACKAGE BODY <nom_package>
IS
      [<liste_declarations_privees>]
      [<definition_curseurs>]
      [<definition_procedures_fonctions>]
BEGIN
      [<liste_instructions_initialisation>] ;
[EXCEPTION
     <gestion_exceptions>] ;

END [<nom_package>] ;


CREATE [OR REPLACE PACKAGE] <nom_package> <specification_package> ;
CREATE [OR REPLACE PACKAGE BODY] <nom_package> <package_body>;
```

Suppression ou modification de programmes

```
DROP PROCEDURE | FUNCTION | PACKAGE | PACKAGE BODY  <nom_programme> ;

ALTER PROCEDURE | FUNCTION | PACKAGE | PACKAGE BODY  <nom_programme>
COMPILE;
```

SQL dynamique

```
EXECUTE IMMEDIATE <ordre_dynamique>
[INTO <variable [, variable2 …] | variable_enreg>
[USING <mode> <argument_lie> [,<mode> <argument_lie> …] ]
[RETURNING INTO <argument_lie> [,<argument_lie> …] ] ;
```

où :

- `<ordre_dynamique>` est une chaîne stockant un ordre SQL ou un appel de bloc PL/SQL. `<ordre_dynamique>` peut contenir des variables préfixées par le caractère " : ". Il ne faut pas déclarer ces variables.
- La clause **INTO** n'est utilisée que pour les requêtes **SELECT** rendant ***au plus un tuple***. Elle spécifie une ou plusieurs variables réceptrice pour ce tuple. Plus précisément, on peut indiquer une liste de variables scalaires respectant l'ordre des attributs dans le **SELECT** ou une variable enregistrement dont le type est soit défini par le développeur (**RECORD**) soit référencé via **%ROWTYPE**.
- La clause **USING** permet, au moment de l'exécution, de spécifier des arguments liés aux variables éventuellement données dans `<ordre_dynamique>` avec leur mode d'utilisation (**IN**, **IN OUT**, **OUT**). Par défaut, ce mode est **IN**. Ces arguments doivent être indiqués dans l'ordre d'apparition des variables associées.
- La clause **RETURNING** indique des arguments liés en sortie. Dans ce cas, les ordres **INSERT**,

UPDATE et **DELETE** stockés dans <ordre_dynamique> doivent inclure une clause :
RETURNING <nom_attribut1[, nom_attribut2…]> **INTO** :variable1[,:variable2…]
Les arguments sont indiqués en respectant l'ordre d'apparition des variables en sortie.
Si cette clause est utilisée, alors la clause **USING** ne doit contenir que des arguments de mode **IN**.

SQL dynamique accepte seulement des variables et arguments dont les types sont valides en SQL. Ainsi ils ne peuvent pas être des booléens ou des tableaux (types spécifiques de PL/SQL). La seule exception est la variable enregistrement indiquée dans la clause **INTO**.
Une autre restriction concerne les objets du schéma : variables et arguments liés ne peuvent pas être utilisés pour passer le nom de tels composants à l'ordre dynamique. Il faut utiliser les paramètres des procédures et fonctions.

Curseur implicite : SQL avec les mêmes propriétés que les autres curseurs.

Curseurs en SQL dynamique

```
OPEN <nom_curseur> FOR <ordre_dynamique>
[USING <argument_lie> [, <argument_lie> …] ] ;

FETCH <nom_curseur> INTO <variable_receptrice> ;

CLOSE <nom_curseur> ;

TYPE <type_curseur> IS REF CURSOR ;
<nom_curseur> <type_curseur> ;
```

Les triggers sur les relations

```
CREATE [OR REPLACE] TRIGGER <nom_trigger>
<chronologie>
<specification_evenements_declencheurs>
[FOR EACH ROW]
[WHEN ( <condition> )]
[DECLARE
     <declarations_locales> ]
BEGIN
     <liste_instructions>
[EXCEPTION
     <gestion_exceptions> ]
END ;
/
```

- <chronologie> := **BEFORE** | **AFTER**
- <specification_evenements_declencheurs> := <evenement1> [**OR** <evenement2> [**OR** <evenement3>]] **ON** <nom_relation>

où <evenementi> := **INSERT** | **DELETE** | **UPDATE** [**OF** <attribut1[, attribut2…]>] |
ALTER | **CREATE** | **DROP** | **RENAME** | **GRANT** | **REVOKE**

Paramètres des événements pour les triggers ***FOR EACH ROW***

new.<nom_attribut> ou old.<nom_attribut> dans la clause **WHEN**
:new.<nom_attribut> ou :old.<nom_attribut> dans <liste_instructions>

Prédicats conditionnels

INSERTING, DELETING, UPDATING peuvent être utilisés dans la condition d'une instruction conditionnelle **IF**

Exceptions

```
IF <condition> THEN
RAISE_APPLICATION_ERROR(<numero_exception>, 'message erreur') ;
```

ou, dans la partie **EXCEPTION** :

```
WHEN <exception>
THEN RAISE_APPLICATION_ERROR(<numero_exception>, 'message erreur') ;
```

Gestion de triggers

```
ALTER TRIGGER <nom_trigger> COMPILE ;

DROP TRIGGER <nom_trigger> ;

ALTER TRIGGER <nom_trigger> ENABLE | DISABLE ;

ALTER TABLE <nom_relation> ENABLE ALL TRIGGER ;
ALTER TABLE <nom_relation> DISABLE ALL TRIGGER ;
```