Le langage PL/SQL

Introduction

SQL est un langage non-procédural qui permet de façon simple et aisée de manipuler la base de données. Il est interprété. Des langages comme Pascal ou C sont des langages procéduraux qui sont plus complexes qu'un langage non-procédural, mais qui offrent plus de flexibilité et de puissance.

PL/SQL = « Procedural Language Extensions to SQL ». C'est un langage compilé.

Pour profiter des avantages du procédural et du non-procédural, PL/SQL offre les structures et les routines disponibles dans tout langage de 3ème génération en plus du langage SQL.

Les éléments du langage SQL pris en compte par le PL/SQL sont :

- Les commandes de LMD [INSERT, UPDATE, DELETE, SELECT] et les commande de LCD [COMMIT, ROLLBACK]
- **mais pas** la partie LDD [CREATE, ALTER, DROP], ni la partie sécurité [GRANT, REVOKE] (nb: certains packages permettent d'obtenir ces fonctions)

Les programmes PL/SQL peuvent être utilisés pour du développement du côté client ou du côté serveur. Par exemple :

- un déclencheur (trigger) stocké dans une base Oracle du côté serveur
- de la logique applicative dans une application Oracle Developer (Forms, Reports) du côté client.

Dans le cas du PL/SQL que nous allons utiliser dans le cadre de ce cours, le code sera toujours exécuté par le **serveur**.

Structure d'un programme PL/SQL

Tous les programmes PL/SQL sont structurés à l'aide de blocs.

Structure d'un bloc

Les blocs sont organisés à l'aide des sections suivantes :

```
DECLARE
    déclarations ;  -- optionnel
BEGIN
    instructions ;
EXCEPTION
    gestion des exceptions ; -- optionnel
END ;
```

Chaque instruction PL/SQL se termine par un point virgule (;).

La partie « déclaration » (optionnelle) permet de définir les variables et les constantes utilisées dans le bloc. Elle débute avec le mot clé DECLARE et se termine avec le BEGIN.

La partie « commandes exécutables » (obligatoire) contient le corps du programme. Elle débute après le BEGIN et se termine avec le mot clé END. Elle contient également la partie « exception » (optionnelle) qui contient d'autres instructions qui correspondent aux actions à entreprendre en cas d'erreur. Cette partie de gestion des exceptions débute avec le mot clé EXCEPTION et se situe à la fin du bloc, avant le END.

Des blocs peuvent être **imbriqués** dans d'autres blocs, dans les parties « commandes exécutables » ou « exception ».

Commentaires

- `-- ceci est une ligne de commentaires`
- `IF v_birthdate = SYSDATE THEN -- commente le reste d'une ligne`
- ```
  /* ceci est un
      commentaire qui porte
      sur plusieurs lignes  */
  ```

Les blocs anonymes et nommés

Blocs anonymes

Un bloc est dit anonyme s'il n'est pas stocké dans un schéma de la base de données. Il doit alors être stocké dans un fichier, et doit être compilé à chaque exécution. Un bloc anonyme ne possède pas de nom, il ne peut donc pas être appelé (exécuté) par un autre bloc.

Blocs nommés

Un bloc nommé est stocké en tant qu'objet de schéma dans le SGBD. Il s'agit d'une procédure, d'une fonction ou d'un déclencheur (trigger). Un bloc nommé est compilé au moment de la création de l'objet et est stocké sous cette forme dans la base.

NB: un bloc nommé doit être recompilé dans certaines conditions; notamment si certains objets référencés sont modifiés.

Exécution d'un programme sur un serveur Oracle

Un bloc PL/SQL doit être transmis au moteur PL/SQL d'Oracle pour être exécuté. S'il est anonyme, il doit tout d'abord être compilé, puis seulement, il pourra être exécuté. Dans le cas où le bloc est stocké dans la base de données, il peut directement être exécuté par le moteur PL/SQL.

Dans le cas où le bloc contient des commandes SQL, celles-ci sont extraites et envoyées au moteur SQL pour être traitées spécialement, comme toute requête SQL.
  ```

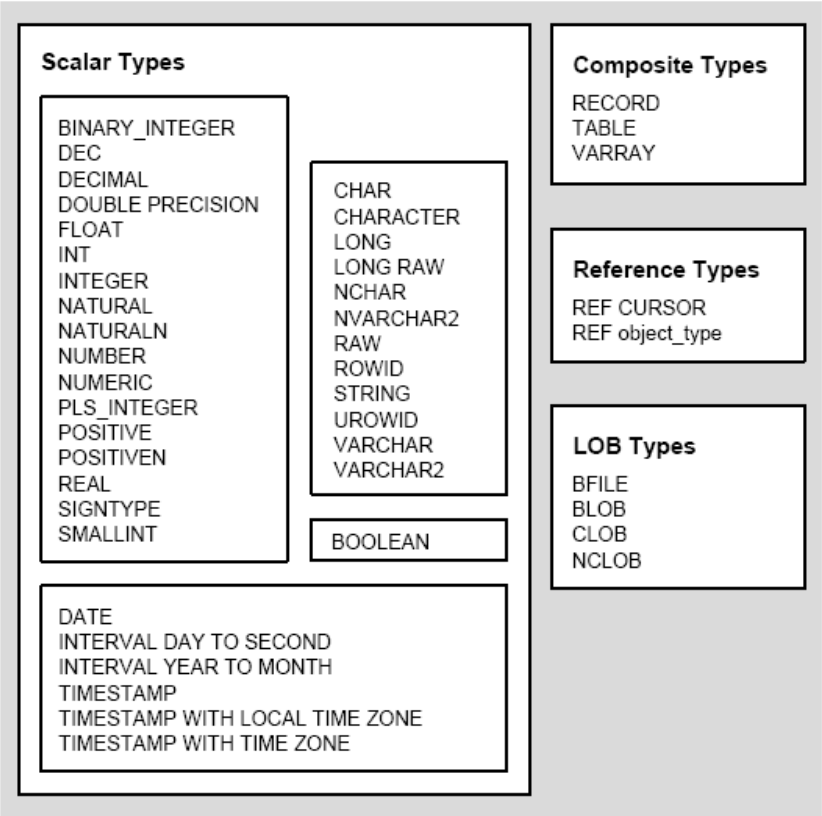
Partie déclaration

Déclaration de variables ou de constantes

PL/SQL est un langage à typage fort, et par conséquent, tous les éléments de programmation doivent être déclarés. Oracle permet la déclaration de variables ou de constantes (et l'attribution d'une valeur initiale éventuelle) grâce à la syntaxe suivante :

```
nom_variable [CONSTANT] {type ¦ table.attribut%TYPE ¦ table%ROWTYPE}
                  [NOT NULL] [{ := ¦ DEFAULT} expression_PL/SQL} ;
```

Types de données prédéfinis :



Types basés sur la structure d'une table

Il est également possible et **très utile** d'utiliser des types basés sur :

- un attribut d'une table : `v_emp_no exe_employe.emp_no%TYPE ;`
- un enregistrement d'une table : `v_employe exe_employe%ROWTYPE ;`

Types composés

On peut également utiliser des types composés, par exemple un record :

```
TYPE ty_emp_dept IS RECORD
        (emp_nom  exe_employe.emp_nom%TYPE,
         dep_nom  exe_dept.dep_nom%TYPE);
v_emp_dept  ty_emp_dept;
```

Types propres au développeur

On peut encore définir ses propres types basés sur des types existants :

```
SUBTYPE BirthDate IS DATE NOT NULL;   -- basé sur le type DATE
TYPE NameList IS TABLE OF VARCHAR2(10);
SUBTYPE Collegues IS NameList;        -- basé sur la TABLE NameList
TYPE TimeRec IS RECORD (minutes INTEGER, hours INTEGER);
SUBTYPE FinishTime IS TimeRec;        -- basé sur le RECORD
```

Visibilité des variables

Une variable est connue dans le bloc dans laquelle elle a été déclarée, et dans tous les blocs internes à ce bloc.

Si une variable est redéclarée dans un sous-bloc avec le même nom, mais avec un type différent, c'est une nouvelle variable qui prime sur la précédente à l'intérieur de ce sous-bloc.

Exemples de déclarations

Exemple de déclaration dans un bloc anonyme :

```
DECLARE          -- NB: dans ce cas le mot clef DECLARE est obligatoire.
   c_rabais     CONSTANT NUMBER(3,2) := 0.1 ;
   v_emp_nom    exe_employe.emp_nom%TYPE;
   v_employe    exe_employe%ROWTYPE;
BEGIN
   …
END ;
```

Exemple de déclaration dans un bloc nommé (procédure/fonction) :

```
prc_nom_procédure(param1 [,paramn,…]) IS
   v_emp_nom    exe_employe.emp_nom%TYPE;
BEGIN          -- NB: dans ce cas le mot clef DECLARE NE doit PAS être utilisé.
   …
END ;
```

Visibilité des variables – exemple avec imbrication

```
DECLARE
   v_variable VARCHAR2(20) := 'coucou';
BEGIN
   dbms_output.put_line(v_variable);     -- affiche coucou

   DECLARE
      v_variable NUMBER := 5;
   BEGIN
      dbms_output.put_line(v_variable);  -- affiche 5
   END;

END;
```

Commandes exécutables

Les commandes exécutables comprennent des commandes SQL, des structures conditionnelles (IF, …), des structures de boucle (LOOP, FOR, WHILE), l'instruction NULL, …

SELECT INTO

Les commandes exécutables sont combinées aux déclarations de variables, et à l'assignation de données résultant des requêtes SQL dans des variables. Cette affectation dépendra du résultat de la requête.

```
SELECT  liste_selection  INTO {liste_variables | nom_enregistrement}
   FROM liste_de_tables  [WHERE condition] ;
```

- Si la requête retourne **un ou plusieurs attributs d'UN SEUL enregistrement**, alors ce résultat pourra être stocké dans une ou plusieurs variables de mêmes types que les attributs
  (exemple : `v_emp_nom  exe_employe%TYPE`)

  ```
  SELECT emp_nom, emp_prenom, dep_nom
     INTO v_emp_nom, v_emp_prenom, v_dep_nom
     FROM exe_employe JOIN exe_dept ON dep_no = emp_dep_no
     WHERE emp_no = i_emp_no;
  ```

- Si la requête retourne **tous les attributs d'UN SEUL enregistrement**, alors ce résultat pourra être stocké dans une variable de type `%ROWTYPE`
  (exemple : `v_employe  exe_employe%ROWTYPE`).

  ```
  SELECT * INTO v_employe FROM exe_employe;
  ```

- Si la requête retourne **plus d'un enregistrement**, cette instruction SELECT INTO va générer une exception (`too_many_rows`). Il faudra donc passer par un CURSEUR – nous verrons ce cas plus tard.

Structures de contrôle : alternatives et boucles

Les principales structures de contrôle sont les suivantes :

Alternative IF-THEN-ELSIF-ELSE-END

```
IF condition1 THEN
   sequence_of_statements1;
ELSIF condition2 THEN
   sequence_of_statements2;
ELSE
   sequence_of_statements3;
END IF;
```

Boucle WHILE-LOOP

```
WHILE condition LOOP
   instructions;
END LOOP;
```

La condition est évaluée avant chaque itération. Si elle retourne FALSE ou NULL, alors la boucle est interrompue.

Boucle FOR-LOOP

```
FOR v_compteur IN [REVERSE] val_inférieure..val_supérieure LOOP
   instructions;
END LOOP;
```

Cette boucle s'exécute un nombre définit de fois.

Exemples :

- ```
  FOR i IN 1..3 LOOP  -- donne les valeurs 1,2,3 à la variable i
     instructions;    -- s'exécute 3 fois
  END LOOP;
  ```

- ```
  SELECT COUNT(emp_no) INTO v_nb_emp FROM exe_employe;
  FOR i IN 1..v_nb_emp LOOP
     ...
  END LOOP;
  ```

NB: un compteur de boucle FOR n'a pas besoin d'être déclaré, car il l'est implicitement comme étant une variable de type INTEGER.

Les blocs nommés : procédures et fonctions

Les procédures et les fonctions sont des blocs PL/SQL nommés qui sont stockés dans la base de données.

Tout comme les blocs anonymes, ils comportent une section déclarative, un bloc de commandes et une section optionnelle de gestion des erreurs.

La section déclarative se trouve après la définition de la procédure. Le mot clé DECLARE n'est donc pas utilisé contrairement aux blocs anonymes.

Procédure

```
CREATE PROCEDURE prc_affi_employe (i_emp_no IN exe_employe.emp_no%TYPE) IS
   v_emp_nom     exe_employe.emp_nom%TYPE;
   v_emp_prenom  exe_employe.emp_prenom%TYPE;
   v_dep_nom     exe_dept.dep_nom%TYPE;

BEGIN
   SELECT emp_nom, emp_prenom, dep_nom INTO v_emp_nom, v_emp_prenom, v_dep_nom
      FROM exe_employe  JOIN exe_dept ON dep_no = emp_dep_no
      WHERE emp_no = i_emp_no;

   dbms_output.put_line('Employé :');
   dbms_output.put_line('*********');
   dbms_output.put_line(' Nom prénom  : ' || v_emp_nom || ' ' || v_emp_prenom);
   dbms_output.put_line(' Département : ' || v_dep_nom);
END;
```

L'appel d'une procédure se fait de la façon suivante depuis un bloc PL/SQL (anonyme ou nommé) :

```
BEGIN
   prc_affi_employe(3);
END;
```

Fonction

```
CREATE OR REPLACE FUNCTION fct_nb_employes_departement
                  (i_dep_no IN exe_employe.emp_dep_no%TYPE)  RETURN INTEGER  IS
   v_nb_employes INTEGER;

BEGIN
   SELECT count(emp_no) INTO v_nb_employes
      FROM exe_employe WHERE emp_dep_no = i_dep_no;

   RETURN v_nb_employes;
END;
```

L'appel d'une fonction se fait de la façon suivante depuis un bloc PL/SQL (anonyme ou nommé) :

```
BEGIN
   dbms_output.put_line('Résultat: ' || fct_nb_employes_departement(3));
END;
```

Ou de cette manière, en utilisant des variables (pour conserver le résultat et non pas l'afficher uniquement) :

```
DECLARE
   v_dep_no      exe_dept.dep_no%TYPE;
   v_nb_employes INTEGER;

BEGIN
   v_dep_no := 30;
   v_nb_employes := fct_nb_employes_departement(v_dep_no);
   dbms_output.put_line('Nb employes du département ' || v_dep_no || ' = ' ||
                        v_nb_employes);
END;
```

Gestion des exceptions

La partie « exception » d'un bloc permet de gérer les exceptions ou erreurs qui surviennent lors de l'exécution.
Ces exceptions peuvent être dues à des problèmes ou peuvent être générées dans le bloc lui-même.

Exceptions internes

Les exceptions internes correspondent à des erreurs ou exceptions générées par le moteur PL/SQL. Si une telle exception survient, il est possible de l'intercepter et de la traiter spécifiquement dans la partie « exception ».

Par exemple, si une requête SELECT ne retourne aucun enregistrement, l'exception NO_DATA_FOUND survient, et l'exécution se poursuit dans la partie « exception ».

```
…
EXCEPTION
   WHEN no_data_found  THEN
      dbms_output.put_line('Il n''y a pas de département numéro: ' ||
                           v_dep_no || ' dans la table exe_dept');
```

Liste des exceptions internes :

Exception	Raised when ...
ACCESS_INTO_NULL	Your program attempts to assign values to the attributes of an uninitialized (atomically null) object.
CASE_NOT_FOUND	None of the choices in the WHEN clauses of a CASE statement is selected, and there is no ELSE clause.
COLLECTION_IS_NULL	Your program attempts to apply collection methods other than EXISTS to an uninitialized (atomically null) nested table or varray, or the program attempts to assign values to the elements of an uninitialized nested table or varray.
CURSOR_ALREADY_OPEN	Your program attempts to open an already open cursor. A cursor must be closed before it can be reopened. A cursor FOR loop automatically opens the cursor to which it refers. So, your program cannot open that cursor inside the loop.
DUP_VAL_ON_INDEX	Your program attempts to store duplicate values in a database column that is constrained by a unique index.
INVALID_CURSOR	Your program attempts an illegal cursor operation such as closing an unopened cursor.
INVALID_NUMBER	In a SQL statement, the conversion of a character string into a number fails because the string does not represent a valid number. (In procedural statements, VALUE_ERROR is raised.) This exception is also raised when the LIMIT-clause expression in a bulk FETCH statement does not evaluate to a positive number.
LOGIN_DENIED	Your program attempts to log on to Oracle with an invalid username and/or password.
NO_DATA_FOUND	A SELECT INTO statement returns no rows, or your program references a deleted element in a nested table or an uninitialized element in an index-by table. SQL aggregate functions such as AVG, SUM and COUNT always return a value or a null. So, a SELECT INTO statement that calls an aggregate function never raises NO_DATA_FOUND. The FETCH statement is expected to return no rows eventually, so when that happens, no exception is raised.
NOT_LOGGED_ON	Your program issues a database call without being connected to Oracle.
PROGRAM_ERROR	PL/SQL has an internal problem.
ROWTYPE_MISMATCH	The host cursor variable and PL/SQL cursor variable involved in an assignment have incompatible return types. For example, when an open host cursor variable is passed to a stored subprogram, the return types of the actual and formal parameters must be compatible.
SELF_IS_NULL	Your program attempts to call a MEMBER method on a null instance. That is, the built-in parameter SELF (which is always the first parameter passed to a MEMBER method) is null.
STORAGE_ERROR	PL/SQL runs out of memory or memory has been corrupted.
SUBSCRIPT_BEYOND_COUNT	Your program references a nested table or varray element using an index number larger than the number of elements in the collection.

Exception	Raised when ...
SUBSCRIPT_OUTSIDE_LIMIT	Your program references a nested table or varray element using an index number (-1 for example) that is outside the legal range.
SYS_INVALID_ROWID	The conversion of a character string into a universal rowid fails because the character string does not represent a valid rowid.
TIMEOUT_ON_RESOURCE	A time-out occurs while Oracle is waiting for a resource.
TOO_MANY_ROWS	A SELECT INTO statement returns more than one row.
VALUE_ERROR	An arithmetic, conversion, truncation, or size-constraint error occurs. For example, when your program selects a column value into a character variable, if the value is longer than the declared length of the variable, PL/SQL aborts the assignment and raises VALUE_ERROR. In procedural statements, VALUE_ERROR is raised if the conversion of a character string into a number fails. (In SQL statements, INVALID_NUMBER is raised.)
ZERO_DIVIDE	Your program attempts to divide a number by zero.

Association d'un nom à une exception

Quand survient une erreur interne à laquelle ne correspond pas de nom, il est possible d'utiliser l'instruction PRAGMA EXCEPTION_INIT (nom_exception, n°_erreur) pour lui associer un nom qui pourra être traité dans la partie « exception » :

```
DECLARE
   not_null_constraint_violated EXCEPTION;
   PRAGMA EXCEPTION_INIT(not_null_constraint_violated, -1400);

   check_constraint_violated EXCEPTION;
   PRAGMA EXCEPTION_INIT(check_constraint_violated, -2290);

   integrity_constraint_violated EXCEPTION;
   PRAGMA EXCEPTION_INIT(integrity_constraint_violated, -2291);
BEGIN
   ...
EXCEPTION
   WHEN not_null_constraint_violated THEN
      ...  -- handle the error
END;
```

Exceptions définies par l'utilisateur *(le programmeur)* à l'aide de RAISE

Le programmeur a la possibilité de définir ses propres exceptions, s'il désire réagir lorsque survient une condition particulière dans un programme. Il faut alors déclarer une exception particulière et activer cette exception lorsque survient la condition :

```
DECLARE
   out_of_stock EXCEPTION;
   nombre_en_stock NUMBER(4);
BEGIN
   ...
   IF nombre_en_stock < 1 THEN
      RAISE out_of_stock;
   END IF;
   ...
EXCEPTION
   WHEN out_of_stock THEN
      ...  -- traiter l'erreur
END;
```

Exceptions définies par l'utilisateur *(le programmeur)* à l'aide de RAISE_APPLICATION_ERROR

Le programmeur a également la possibilité de déclencher une erreur sans la gérer dans la partie « exception » à l'aide de la commande `RAISE_APPLICATION_ERROR(codeErreur, 'Message')` :

Attention, les codes d'erreurs attribuables vont de -20000 à -20999

```
CREATE PROCEDURE prc_AugmenterSalaire(i_emp_no IN NUMBER, i_montant IN NUMBER) IS
   v_salaire exe_employe.emp_salaire%TYPE
BEGIN
   SELECT emp_salaire INTO v_salaire FROM exe_employe
      WHERE emp_no = i_emp_no;
   IF v_salaire IS NULL THEN
      raise_application_error(-20101, 'Salaire manquant');
   ELSE
      UPDATE exe_employe SET emp_salaire = v_salaire + i_montant
         WHERE emp_no = i_emp_no;
   END IF;
END prc_AugmenterSalaire;
```

Gestion des autres exceptions

Pour les exceptions non gérées spécifiquement dans la partie « exception », on peut ajouter une gestion générale d'exception à l'aide de la clause WHEN OTHERS :

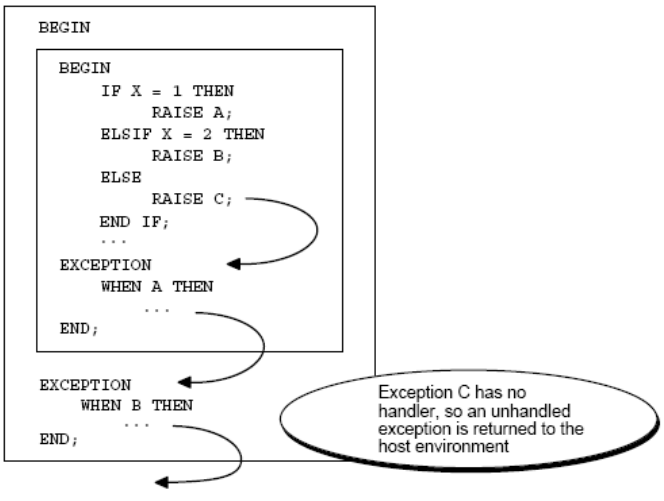
```
EXCEPTION
   WHEN exception_name1 THEN -- handler
      sequence_of_statements1
   WHEN exception_name2 THEN -- another handler
      sequence_of_statements2
   WHEN OTHERS THEN -- optional handler
      sequence_of_statements3
END;
```

Ceci permet de s'assurer que toutes les exceptions seront traitées.

Propagation des exceptions

Si une exception n'est pas gérée dans le bloc où elle survient, celle-ci est propagée dans le bloc de niveau supérieur, jusqu'à ce qu'elle soit traitée ou que le bloc de plus haut niveau ne s'interrompe.

Figure 7–3 Propagation Rules: Example 3



Standards de dénomination (conventions HEG)

Elément	Standard	Exemple
Mots-clés	Comme pour SQL, tous les mots clés du langage PL/SQL seront écrits en **majuscule**, y compris les noms de fonctions utilisées.	**DECLARE** v_emp_nom **VARCHAR2**(20); **BEGIN** … ; **END**;

Objets manipulés (variable, tables, triggers, etc…)	Les objets manipulés seront écrits en **minuscule**, même si Oracle ne fait pas la différence.	DECLARE
		v_emp_nom exe_employe.emp_nom%TYPE;
		BEGIN
		… ;
		END;

Noms de variables internes	• Le nom est préfixé de '**v**_' • Le nom dépend du contenu	**v_emp_nom** exe_employe.emp_nom%TYPE; **v_emp_salaire** exe_employe.emp_salaire%TYPE;
Noms de constantes	• nomConstante	**pi** CONSTANT NUMBER := 3.14 ;
Exception utilisateur	• **e**_nomException	EXCEPTION WHEN **e_out_of_stock** THEN
Noms de variables paramètres	• **i**_nom_paramètre_entrée • **o**_nom paramètre_sortie • **io**_nom_par_entrée_sortie	PROCEDURE prc_affi_employe (**i_param IN** datatype, **o_param OUT** datatype, **io_param IN OUT** datatype) IS
Noms de curseurs	• **c**_nom_curseur	CURSOR **c_employes** IS SELECT …
Noms de ligne de curseur	• **r**_nom_curseur	FETCH c_employes INTO **r_employes**;
Noms de procédures	• **prc**_nom_procédure	PROCEDURE **prc_affi_employe** (i_emp_no IN NUMBER) IS
Noms de fonctions	• **fct**_nom_fonction	FUNCTION **fct_nb_employes_departement (param)** RETURN datatype IS
Noms de trigger (déclencheurs)	• **trg**_nom_trigger	CREATE OR REPLACE TRIGGER **trg_changement_dept**
Noms de packages	• **pkg**_nom_package	PACKAGE **pkg_entreprise** IS

Autres informations

Eléments	Valeurs	
Les opérateurs	Arithmétiques	**** , * , / , + , -**
	Comparaison	**= , <> , != , < , > , <= , >=** , **LIKE , IN , BETWEEN , IS NULL**
	Logiques	**AND , OR , NOT**
	Caractères	**¦¦ , LIKE**
Les types de données les plus couramment utilisés	**CHAR**	Longueur fixe
	VARCHAR2	Longueur variable
	NUMBER	Numérique, entier ou réel
	BOOLEAN	Valeurs possibles : TRUE/FALSE/NULL Uniquement pour des variables, car pas de BOOLEAN pour les colonnes de tables.
	DATE	Date et Heure
Déclaration d'une variable	v_nom_variable [CONSTANT] {type ¦ table.attribut%Type ¦ table%Rowtype} [NOT NULL] [{ := ¦ DEFAULT} expression_PL/SQL} ;	
La commande de « non-action »	**NULL ;** -- Aucune action	