Bases de données avancées

Jean-Yves Antoine
LI - Université François Rabelais de Tours
Jean-Yves.Antoine@univ-tours.fr

UFR Sciences et Techniques
IUP SIR Blois – Master 1

Bases de données avancées

Programmation avancée en BD : PL/SQL

UFR Sciences et Techniques
IUP SIR Blois – Master 1

PL/SQL : présentation

Intérêt

- **Opérationnalisation des SGBD-transactionnels**: encapsulation de transactions dans des blocs PL/SQL
- **Programmes complexes travaillant sur les données de la base**: les structures de contrôle classiques en programmation impérative (SQL se limite au LDD / LMD)
- **Modularité** : possibilité de conserver/utiliser des procédures ou fonctions cataloguées dans des paquetages PL/SQL

PL/SQL et SQL

- Extension de SQL: ajout de mécanismes pour parcourir les résultats (**curseurs**), traiter les **exceptions** et réagir à l'état de la base (**déclencheurs – *triggers***)
- Langage propre à Oracle: à comparer à la sous-norme SQL optionnelle *ISO/IEC 9075-5:1996 Flow-control statements*

PL/SQL : présentation

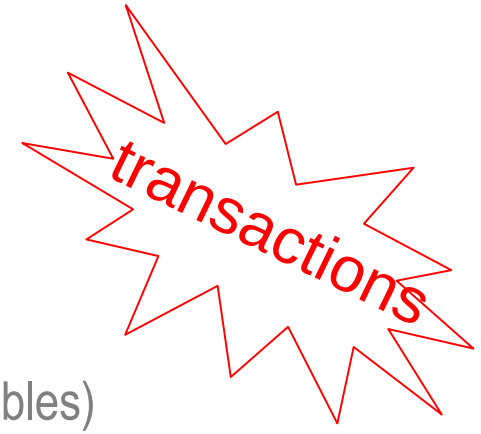
Plan

- **Syntaxe de base**: écriture des instructions de base des blocs PL/SQL
- **Programmes PL/SQL** :
  - procédures et fonctions cataloguées
  - paquetages
  - exceptions
- **Curseurs**: manipuler et effectuer des traitements réels sur la base de données
- **Déclencheurs** : réagir automatiquement à un événement ou un état de la base
- **SQL dynamique**

PL/SQL : syntaxe

Bloc PL/SQL

- **Déclarations** (variables, constantes, types, curseurs, etc...)
- **Code** PL/SQL à proprement parler
- Gestion des **exceptions** (erreurs)
- Encapsulation possible de sous-blocs (**Attention**: portée variables)




```
DECLARE
-- declarations
BEGIN

-- code

EXCEPTION
-- code gestion erreurs
END;
/
```

```
DECLARE
-- declarations
BEGIN
-- code
EXCEPTION
-- code gestion erreurs
END;
```

PL/SQL : syntaxe

Variables : types

- **scalaires SQL** : DATE, VARCHAR(n), NUMBER...
- **PL/SQL simples** : sous-types SQL (norme ISO) : INTEGER, INT, POSITIVE...
- **PL/SQL composites** : %TYPE, %ROWTYPE, RECORD

- **Paramètres** d'invocation en SQL*Plus

Variables : declaration

```
Id_var type [NOT NULL] [ := | DEFAULT expression init];
```

Constantes : declaration

```
Id_cte CONSTANT type [ := | DEFAULT expression init];
```

Exemples

```
DECLARE
        c_euro CONSTANT NUMBER := 6.569;
        prix INT;                    /* prix initialisé à NULL */
        date_courante DATE DEFAULT SYSDATE;
```

PL/SQL : syntaxe

Type %TYPE

- Rattachement du type à celle d'un attribut de table ou vue

```
id_variable id_table.id_attribut %TYPE
```

- Rattachement du type de la variable à une autre variable déjà déclarée

```
id_new_var id_definitevar %TYPE
```

Type enregistrement : %ROWTYPE

Description d'un enregistrement unique complet

déclaration `id_variable id_table %ROWTYPE`

utilisation `id.variable.un_des_champs`

Exemple

```
DECLARE  c_euro      CONSTANT NUMBER := 6.569;
         prix_euro   Table_produits.Prix   %TYPE ;
         prix_fr     prix_euro   %TYPE := prix_euro *  c_euro ;
         produits    Table_produits  %ROWTYPE;
BEGIN    SELECT * INTO produits from Table_Produits WHERE id=1;
         prix_euro := produits.prix;
```

PL/SQL : syntaxe

Type personnalisés

- **RECORD : structure de donnée personnalisée**

  `TYPE nom_record IS RECORD (decl_chp1, decl_chp2 …);`

  où `decl_chp` reprend la syntaxe classique d'une déclaration de variable.

- **TABLE : tableaux dynamiques** (sans dimension prédéfinie)

  `TYPE nom_array IS TABLE OF {`
  `type_sple | var%TYPE | table.att%TYPE | table.%ROWTYPE }`

```
DECLARE     TYPE Type_sondage IS TABLE OF FLOAT;
            TYPE Type_vol is RECORD (num VARCHAR(10), depart DATE, arrivee DATE);
            sondage_ifop            Type_sondages;
            vol_particulier         Type_vol;
BEGIN
            sondage_ifop(0) := 0.5;
            vol_particulier.num := 'AF4563';
            vol_particulier.depart := to_date('12/01/2007');
            vol_particulier.arrivee := to_date('13/01/2007');
END;
```

PL/SQL : syntaxe

Variables de substitution

Passage de paramètres en entrée d'un bloc PL/SQL

- déjà rencontré en L3 lors de la création de requêtes paramétrées en SQL « Oracle »
- Identification du paramètre par le symbole **&** préfixé au nom de la variable de substitution
- Pas de déclaration de la variable à faire
- Directive SQL*Plus **ACCEPT** si on veut afficher un message d'invite (sinon, SQL*Plus demandera simplement la valeur du paramètre identifiée par son nom)

```
ACCEPT s_number PROMPT 'Entrez le numero du vol : '

DECLARE  TYPE Type_vol is RECORD (num VARCHAR(10), depart DATE, arrivee DATE);
        vol_particulier       Type_vol;
BEGIN
        vol_particulier.num := '&s_number' ;
END;
```

Variables de sessions

PL/SQL : syntaxe

Structures de contrôle

On retrouve les structures de contrôle de tous les langages impératifs

- *IF / THEN / ELSE | ELSEIF / END IF*
- SUIVANT (*CASE … WHEN / THEN … END CASE*)
- TANT QUE (*WHILE … LOOP … ENDLOOP*)
- REPETER (*LOOP … EXIT … END LOOP*)
- POUR (*FOR .. IN … END LOOP*)

Branchements

```
IF condition
THEN
   instr_if;
[ELSE
   instr_alt;]
END IF;
```

```
IF condition
THEN
   instr_if;
ELSEIF cond_else
   THEN instr_alt;
  [ELSE | ELSEIF …]
END IF;
```

PL/SQL : syntaxe

Branchement CASE

```
CASE variable
WHEN expression1 THEN
    instruction_cas1;
WHEN …
[ELSE instr_defaut]
END CASE;
```

```
CASE
WHEN condition1 THEN
    instruction_cas1;
WHEN …
[ELSE instr_defaut]
END CASE;
```

Itération

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

```
LOOP
    instructions
    EXIT WHEN condition
END LOOP;
```

```
FOR compteur IN [REVERSE] val_min .. val_max LOOP
    instructions
END LOOP;
```

PL/SQL : entrées / sorties

Paquetage DBMS_OUTPUT

- Activation du paquetage sous SQL*PLUS `SET SERVEROUTPUT ON`
- Activation du paquetage sous PL/SQL `DBMS_OUTPUT.ENABLE;`
- Utilisation d'une procédure du paquetage `DBMS_OUTPUT.Nom_Procedure;`

```
ENABLE / DISABLE
PUT_LINE(texte)          /*  affichage  VARCHAR,  DATE,NUMBER
*/
PUT(texte)
GET_LINE(var,statut)    /* statut = 1 si error */
```

```
DBMS_OUTPUT.PUT_LINE('Date courante : ' || SYSDATE);
```

Autres paquetages

- Générateur aléatoire `DBMS_RANDOM`
- Gestion explicite des verrous `DBMS_LOCK`
- Gestion des ROWID `DBMS_ROWID`

Programmes PL/SQL

Fonctions et procédures cataloguées : introduction

- **A la base**, blocs PL/SQL nommés : changement simplement de déclarations
- **Catalogage** : blocs compilés et stockés dans la base : objets visibles dans le dictionnaire sur qui ont peut définir une politique d'accès :

```
GRANT EXECUTE ON nom_procedure TO utilisateur ;
```

- **Compilation** : recompilation automatique lors d'une invocation si un objet cité dans le code (table, vue, champ…) a été modifié (≠ interprétation)
- **Invocation** : dans tout utilitaire Oracle (SQL*Plus, SQL*Forms…) ou dans un programme externe (C, JAVA) reconnu par la base
- **Imbrication** : sous-fonctions et sous-procédures.
- **PL/SQL** mais on peut définir des fonctions ou procédures C ou JAVA utilisables dans des programmes PL/SQL

Paquetages

Notion classique de package : regroupement d'objets PL/SQL (procédures, fonctions, exceptions…) formant un ensemble de services cohérent.

Code: structure générale

- **Directive** de définition `CREATE FUNCTION / PROCEDURE`
- **Déclaration** des variables **puis** déclaration et définition des sous-programmes
- Corps du programme
- Gestion des **exceptions** (erreurs)

```
CREATE PROCEDURE nom_proc …
IS
-- declarations variables
-- declarations ss_prog
BEGIN

-- code

[EXCEPTION
-- code gestion erreurs]
END nom_proc;
```

```
PROCEDURE ss_prog
IS
-- declarations
BEGIN
-- code
EXCEPTION
-- code gestion erreurs
END ss_prog;
```

Programmes PL/SQL

Procédure PL/SQL

```
CREATE [OR REPLACE] PROCEDURE [schema].nom_proc
[( param1 [IN | OUT | IN OUT] [NOCOPY] typeSQL
   [, param2 …] )]
IS
   -- declarations variables / ss_prog
BEGIN
   -- code
END nom_proc;
```

Fonction PL/SQL

```
CREATE [OR REPLACE] FUNCTION [schema].nom_fct
[( param1 [IN | OUT | IN OUT] [NOCOPY] typeSQL
   [, param2 …] )]
RETURN typeSQL IS
   -- declarations variables / ss_prog
BEGIN
   -- code
   RETURN expression;
END nom_fct;
```

Programmes PL/SQL

Récursivité

possible en PL/SQL

Sous-programmes

- Toute procédure ou fonction déclarée et compilée peut-être utilisée dans un programme … si on a les droits pour l'exécuter
- On peut également définir un sous-programme imbriqué (*nested subprogram*) dans un programme donné.
- Dans ce cas, sous-programme local n'ayant d'existence que lors de l'exécution du programme.
- A déclarer (spécification et corps) impérativement après les variables du programme principal. Même syntaxe que pour un programme, le « CREATE » en moins.

Programmes PL/SQL

Compilation

- Directive de compilation / à placer sous SQL*PLUS après le lancement de l'ordre CREATE PROCEDURE ou CREATE FUNCTION (i.e. à la ligne suivant le END)
- Objet Oracle : les **erreurs de compilation** sont détaillées dans le dictionnaire de données, dans la vue USER_ERRORS.
- `SHOW ERRORS;` directive SQL simplifiée pour consulter ces erreurs

Exécution

- Directement sous SQL*PLUS EXECUTE
- Dans un bloc ou un programme PL/SQL

Programmes PL/SQL

Paquetages

- Regroupe n'importe quels types d'éléments définis pour cet usage : types de données (RECORD…), variables, procédures, fonctions, curseurs, exceptions…

- **Spécification** : ensemble des signatures des éléments paquetés

```
CREATE [OR REPLACE] PACKAGE nom_package
AS | IS
   -- declarations données / types / ss_prog
END [nom_package];
```

- **Implémentation**: totalité des définitions des éléments du paquetage

```
CREATE [OR REPLACE] PACKAGE BODY nom_package
AS | IS
   -- définitions données / types /
   -- corps des procedure / fonctions
END [nom_package];
```

PL/SQL : manipulation de la BD

Manipulation de tuples

Les directives INSERT INTO, UPDATE et DELETE FROM peuvent être utilisées sans restriction avec des variables PL/SQL (scalaires, %ROWTYPE)

Lecture de tuples

Chargement d'un variable à partir de la lecture d'un **unique** enregistrement dans la base (exception si 0 ou plusieurs enregistrements en réponse)

```
DECLARE
    heure_depart Vols.depart%TYPE;
BEGIN
    SELECT Vols.depart
    INTO heure_depart
    FROM Vols
    WHERE Vols.id = 'AF3517' ;
END;
```

Plusieurs enregistrements de travail : utilisation de **curseurs**

PL/SQL : manipulation de la BD

Curseurs : généralités

- Zone mémoire permettant de travailler individuellement sur des enregistrements (ou simplement lignes) retournés par une requête de sélection SELECT
- **Curseur paramétrable** : paramètres utilisés dans la définition du SELECT
- Permet également de **mettre à jour** la table via le curseur en gérant facilement les accès concurrents (FOR UPDATE)

Curseurs : déclaration

```
CURSOR nom_curseur[(param IN type_param)]
IS
   SELECT { * | liste_attributs_projection }
   FROM   table
   WHERE  expression_avec_paramètres
[FOR UPDATE [OF liste_attributs_concernés] [NOWAIT]]
END;
```

PL/SQL : manipulation de la BD

Parcours systématique d'un curseur : FOR … LOOP

```
FOR cptr IN nom_curseur LOOP
      instructions
ENDLOOP;
```

```
DECLARE
 CURSOR  riches(min IN integer) IS SELECT id, nom FROM Paie WHERE Paie.paie_m >
    min;

BEGIN
 FOR cptr IN riches(4000) LOOP
    DBMS_OUTPUT.PUT_LINE(cptr.nom);
 ENDLOOP;
END;
```

PL/SQL : manipulation de la BD

Parcours contrôlé d'un curseur

- ouverture du curseur : OPEN
- avancement d'une ligne et chargement dans une variable : FETCH … INTO
- fermeture du curseur : CLOSE

```
DECLARE
        CURSOR   riches(min IN integer) IS SELECT id, nom FROM Paie WHERE
        Paie.paie_m > min;
        tuple_crt   riches%ROWTYPE;
BEGIN
        OPEN riches(4000);
        WHILE (riches%FOUND) LOOP              % TRUE tq pas à la fin
        FETCH riches INTO tuple_crt;
        DBMS_OUTPUT.PUT_LINE(tuple_crt.nom);
        ENDLOOP;
        CLOSE riches
END;
```

PL/SQL : manipulation de la BD

Mise à jour par curseur : WHERE CURRENT OF

```
DECLARE
    CURSOR  riches(min) IS SELECT id, nom FROM Paie WHERE Paie.paie_m > min
    FOR UPDATE OF paie_m NOWAIT;
BEGIN
    FOR cptr IN riches(4000) LOOP
        UPDATE Paie SET Paie_m := Paie_m – 1000 WHERE CURRENT OF riches;
    ENDLOOP;
END;
```

De même **DELETE FROM** … **WHERE CURRENT OF** … ;

PL/SQL : manipulation de la BD

Curseurs dynamiques

- **Souplesse** : ne pas lier le curseur à une clause SELECT fixe
- **Déclaration** d'un type

```
DECLARE
    TYPE nomTypeCurseurDyn IS REF CURSOR;
    nomCurseurDyn  nomTypeCurseurDyn ;
```

- **Utilisation** : ouverture en associant dynamiquement le curseur

```
OPEN nomCurseurDyn FOR SELECT liste_attr
    FROM NomTable [WHERE expression;]
```

PL/SQL : exceptions

Erreurs internes

- **Erreur interne** levée par Oracle

  Récupération d'un nom d'exception, d'un message d'erreur (SQLERRM) et d'un code d'erreur (SQLCODE) utilisables sous PL/SQL, Java, C etc…

Nom exception	Numéro	Message
NO_DATA_FOUND	ORA-01403	SELECT ne retournant rien
TOO_MANY_ROW	ORA-01422	Requête retournant plusieurs lignes
ZERO_DIVIDE	ORA-01476	Division par zérom

SQLCODE exemple : ORA-01403 SQLCODE = **- 1403**

Nom exception identificateur utilisé pour la gestion de l'exception

- **Erreur interne non documentée** (par de nom d'exception PL/SQL)

  Association d'un nom au numéro de l'erreur dans la zone des déclarations

PRAGMA_EXCEPTION_INIT (*Nom*, *SQLCODE*);

PL/SQL : exceptions

Exceptions utilisateurs

- **Erreur définie par l'utilisateur**

  Essentiellement pour traiter une erreur applicative (mauvaise saisie de l'utilisateur …) en la traitant comme une erreur interne.

  Déclaration du nom dans la zone des déclarations PL/SQL

Nom_exception **EXCEPTION**;

  Déclenchement explicite de l'exception dans le corps PL/SQL : **RAISE**

IF age < 65 then **RAISE** exception_non_carte_vermeil; END IF;

- **Associer un code d'erreur à une exception utilisateur**

RAISE_APPLICATION_ERROR (*numéro_erreur*, *message*);

  - Numéro compris entre -20 000 et -20999
  - A la place du RAISE simple
  - Numéro d'erreur associé à un nom par un PRAGMA EXCEPTION UNIT

PL/SQL : exceptions

Traitement de l'exception

Dans la partie exception après le corps du bloc PL/SQL

```
WHEN nom_exception_1 THEN
    bloc_instructions_1;
WHEN nom_exception_2 THEN
    bloc_instructions2 ;
…
[ WHEN OTHERS THEN
    bloc_instructions_exceptions_defaut;
]
```

Pas de différence de traitement entre erreurs internes et exceptions posées par l'utilisateur

PL/SQL : exceptions

Curseur implicite

Pseudo-curseur de nom *SQL* remis à jour après chaque opération LMD (`INSERT, UPDATE, DELETE`) et donnant des informations sur l'exécution de l'instruction.

Attribut du curseur	Information
SQL%ROWCOUNT	Nombre de lignes affectées par la commande
SQL%FOUND	Vrai si l'instruction a affecté au moins un tuple
SQL%NOTFOUND	Vrai si l'instruction n'a affecté aucun tuple

Curseur implicite et exception

L'exception NO_DATA_FOUND ne concerne que le SELECT: utilisation du pseudo-curseur pour les autres ordres LMD.

```
DELETE FROM Resultats WHERE Resultat.journee = 5;
IF SQL%NOTFOUND Then RAISE err_aucune_modif ;
ENDIF;
```

PL/SQL : déclencheurs

Triggers / Déclencheurs

- Programme se déclenchant automatiquement suite à un événement ou au passage à un état donné de la base de données.
- Règles de gestion non modélisables par des contraintes d'intégrité
- Porte sur des tables ou des vues
- Trois parties : a) description de l'évènement traqué, b) conditions éventuelles de déclenchement sur l'événement, c) action à réaliser

Différents types de déclencheurs

- sur ordre LMD (INSERT, UPDATE, DELETE).
- sur ordre LDD (CREATE, ALTER, DROP, GRANT, COMMENT…)
- sur événement Oracle (démarrage, arrêt..) : déclencheur d'instance

Déclencheur LMD

- sur l'état global de l'objet considéré : déclencheur d'état
- sur chaque tuple modifié : déclencheur de ligne

PL/SQL : déclencheurs

Déclencheur sur mise à jour de table ou vue (LMD)

CREATE [OR REPLACE] TRIGGER [*schéma*.]*nom_trigger*

{ BEFORE | AFTER | INSTEAD OF}

{ DELETE | INSERT | UPDATE [OF *col1* [,*col2* …]]}

ON {[*schéma*.]*nom_table* | *nom_vue* }

[FOR EACH ROW]

[WHEN (condition)]

```
DECLARE …

BEGIN ….

END;
```

vue multitable

Attention
Tables mutantes

Bloc PL/SQL

PL/SQL : déclencheurs

Déclencheur d'instance LMD : directives :NEW & :OLD

```
CREATE OR REPLACE TRIGGER Trigger_Arlette
BEFORE UPDATE ON Table_salaires
FOR EACH ROW
WHEN (NEW.Position = 'PDG')
BEGIN
  IF New.Salaire > Old.Salaire THEN
        RAISE_APPLICATION_ERROR(-20100, 'Le patron ' || New.Nom || 'vous exploite');
  ENDIF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN RAISE_APPLICATION_ERROR(-20101,'No modif');
  WHEN OTHER THEN RAISE;
END;
```

Directives :NEW et :OLD
uniquement pour
déclencheurs de ligne

	:OLD.*col*	:NEW.*col*
INSERT	NULL	*nouvelle valeur*
UPDATE	*ancienne valeur*	
DELETE		NULL

PL/SQL : déclencheurs

Déclencheur d'instance LMD : directives :NEW & :OLD

```
CREATE OR REPLACE TRIGGER Trigger_Arlette_Justiciere
BEFORE UPDATE ON Paie_direction
FOR EACH ROW
DECLARE
  Juste_compensation CONSTANT INT := 500 ;
BEGIN
  IF New.Salaire > Old.Salaire THEN
        UPDATE Paie_Ouvriers
        SET Paie.Ouvriers.Salaire = Paie_Ouvriers.Salaire + Juste Compensation;
  ENDIF;
END;
```

Paie_Direction

INSEE	Nom	Salaire
…	..	…

Paie_Ouvriers

INSEE	Nom	Salaire
…	..	…

PL/SQL : déclencheurs

Déclencheur INSTEAD OF

- **Uniquement sur déclencheur d'instance** : FOR EACH ROW
- **Uniquement sur vues** : utile surtout pour passer des ordres LMD sur vues multitables qui sont possibles en théorie mais non autorisés par SQL (cf. TP sur les vues)

Déclencheur d'état LMD : exemple

```
CREATE OR REPLACE TRIGGER Ferie
BEFORE DELETE OR UPDATE OR INSERT ON Table_Prix
BEGIN
  IF To_Char(SYSDATE,'DAY') = 'DIMANCHE' THEN
        RAISE_APPLICATION_ERROR(-20102, 'Désolé, pas de modification de
  dimanche');
  ENDIF;
END;
```

PL/SQL : déclencheurs

Déclencheur LDD: modification de la structure de BD

```
CREATE [OR REPLACE] TRIGGER [schéma.]nom_trigger
{ BEFORE | AFTER } action_LDD
ON {[schéma.]SCHEMA| DATABASE }
….
```

Actions LDD : CREATE, ALTER, DROP, COMMENT, GRANT, REVOKE, etc…

Déclencheur d'instance: évènement Oracle

```
CREATE [OR REPLACE] TRIGGER [schéma.]nom_trigger
{ BEFORE | AFTER } event_Oracle
ON {[schéma.]SCHEMA| DATABASE …
```

Evènements Oracle

- **Serveur Oracle**: STARTUP, SHUTDOWN, LOGON,LOGOFF, SERVERROR …
- **Erreur spécifique SGBD**: NO_DATA_FOUND, DUP_VAL_ON_INDEX…

PL/SQL : déclencheurs

Modification de déclencheurs

- ALTER TRIGGER
- DROP TRIGGER

Voir la documentation Oracle pour plus d'information

PL/SQL : déclencheurs

SQL Dynamique

- En PL/SQL, possibilité de lancer automatiquement des actions LMD, LDD et des actions liées aux sessions
- **Intérêt** : lancement d'ordre LDD **paramétrés**
- **Directive EXECUTE IMMEDIATE** : ordre SQL à exécuter précisé en tant que chaîne de caractères

```
EXECUTE IMMEDIATE chaîne de caractères
[ INTO var1 [, var 2 …] ]
[ USING [ param 1 [, param 2…] ]
[RETURNING INTO param …. ]
```

PL/SQL : déclencheurs

SQL Dynamique : exemple

```
DECLARE
   ordre_SQL_Dyn VARCHAR2(100);
   v_param INT := 4;
BEGIN
   EXECUTE IMMEDIATE 'INSERT INTO Clients
   VALUES('4','JYA','Blois') ';
   Ordre_SQL_Dyn := 'DELETE FROM Clients WHERE Id = :1' ;
   EXECUTE IMMEDIATE ordre_SQL_Dyn USING v_param ;
   v_param := 5;
   EXECUTE IMMEDIATE ordre_SQL_Dyn USING v_param ;
END;
```

PL/SQL : déclencheurs

SQL Dynamique et variable curseur

- Pour paramétrer un ordre SQL travaillant sur plusieurs tuples
- Programmation en pilotage fin du curseur (OPEN, FETCH, CLOSE)
- Action à réaliser à préciser dans la directive OPEN

OPEN nom_curseur
FOR chaîne_caractères_décrivant_ordre_SQL_dynamique
[USING *param 1* [, *param 2*…]]

```
DECLARE
   TYPE t_ref_cursor IS REF CURSOR;
   ex_cursor t_ref_cursor;
   v_p1 INT DEFAULT 100;
BEGIN
   OPEN ex_cursor FOR ‘SELECT Nom FROM Clients
          WHERE Debit > :p1 ’ USING v_p1;
   LOOP …
```

PL/SQL : syntaxe

Ouvrages d'entrée & Oracle 10g

C. SOUTOU (2005) *SQL pour Oracle*, Eyrolles, Paris. *Partie II : PL/SQL* ISBN 2-212-11697-7