Le langage PL/SQL

Généralités, structures de contrôle
Collections, sous-programmes, paquetages
Aspects objet

Généralités

- PL/SQL veut dire « Procedural Language extensions to SQL ».
- PL/SQL n'existe pas comme un langage autonome ; il est utilisé à l'intérieur d'autres produits Oracle.
- PL/SQL ajoute les construits de programmation au SQL de façon bien intégrée : dans PL/SQL on peut exécuter des SELECT, DELETE, INSERT et UPDATE contenant des construits PL/SQL, utiliser des opérateurs, des prédicats et des fonctions prédéfinies du SQL, gérer des transactions, etc.

Généralités

- Types de données prédéfinis :
  - BINARY_INTEGER, NUMBER, PLS_INTEGER
  - CHAR, LONG, VARCHAR2
  - BOOLEAN
  - DATE, TIMESTAMP, INTERVAL
  - REF <nom TDU>
  - BLOB, CLOB, BFILE
- Constructeurs de type
  - RECORD
  - TABLE OF ...
  - TABLE OF ... INDEX BY ...
  - VARRAY(limite) OF ...
  - REF CURSOR

Généralités

- Types de collection :
  - Table imbriquée (TABLE OF <type de données>)
  - Tableau (VARRAY(limite) OF <type de données>)
  - Tableau associatif (TABLE OF ... INDEX BY ...)
- Fonctions prédéfinies
- Paquetages prédéfinis
- Structures de contrôle conditionnelle, itérative, etc.
- Mécanisme de curseur : variable curseur, attribut curseur, boucle curseur, etc.
- Traitement d'exceptions (erreurs)
- Paquetage

Généralités

- Un tableau associatif PL/SQL utilise un index (arbre de recherche) basé sur l'indice des éléments de la table.
- C'est un type de dictionnaire (un ensemble de paires « clé – valeur ») déclaré comme suit :
- TYPE nom_t IS TABLE OF element_t [NOT NULL] INDEX BY {BINARY_INTEGER | PLS_INTEGER | VARCHAR2(limite) } ;
- Un tableau associatif est non dense ; ses indices ne sont pas nécessairement contigus.
- Un tableau associatif non initialisé est vide, mais n'est pas NULL.

Généralités

- Un tableau associatif ne peut pas être stocké comme une valeur de colonne de table dans une base de données.
- Dans PL/SQL, une table imbriquée est représentée par un tableau 1D non borné et non dense (où des trous ont été créés par des DELETE mais gardent encore leur indice).
- Un VARRAY est représenté par un tableau 1D borné et dense (dont les indices sont contigus et inférieurs à un maximum déclaré).
- Il existe plusieurs méthodes prédéfinies pour gérer les tableaux associatifs, les tables imbriquées et les VARRAY en tant que collections. Elles seront décrites plus tard.

Généralités

- PL/SQL est un langage à structure de blocs.
- Un bloc PL/SQL est une unité de traitement pouvant servir à construire des procédures et des fonctions.
- Le bloc PL/SQL fournit une portée (a scope) ou un contexte pour des éléments (tels que variables, énoncés, etc.) logiquement connexes.
- L'imbrication des blocs se fait sans limite de profondeur.
- Les blocs PL/SQL peuvent être anonymes ou nommés.

Généralités

- Un bloc PL/SQL anonyme contenant plusieurs énoncés SQL donne lieu à un seul envoi sur le réseau vers le serveur ; les résultats intermédiaires sont traités sur place et un résultat unique est retourné au client.
- Un bloc nommé est un sous-programme c'est-à-dire ou bien une procédure ou bien une fonction.
- Un paquetage permet de grouper ensemble plusieurs types, collections, enregistrements, constantes, variables, curseurs, exceptions, fonctions et procédures logiquement connexes.

Généralités

- Un bloc PL/SQL peut avoir jusqu'à quatre (4) sections : l'en-tête, la section de déclarations, la section d'exécution et la section d'exceptions.
- L'en-tête fait seulement partie d'un bloc nommé (un sous-programme) et détermine la façon de l'appeler (invoquer).
- La section de déclarations permet de déclarer des variables, des curseurs et des sous blocs référencés dans les sections d'exécution et d'exceptions.
- La section d'exécution contient les énoncés exécutables.
- La section d'exceptions prend soins des conditions d'erreur et d'avertissement.

Généralités

- Un bloc PL/SQL anonyme complet se présente donc comme suit :
- DECLARE
  - Section de déclarations
- BEGIN
  - Section d'exécution
- EXCEPTION
  - Section d'exception
- END;

Généralités

- EXCEPTION
  - WHEN NO_DATA_FOUND
  - THEN
  - sequence of statements
  - WHEN DUP_VAL_ON_INDEX
  - THEN
  - sequence of statements
  - WHEN OTHERS
  - THEN
  - sequence of statements
- END

Structures de contrôle

- IF condition
- THEN
- -- sequence of statements
- [ELSE IF condition
- THEN
- -- sequence of statements]…
- [ELSE
- -- sequence of statements]
- END IF

Structures de contrôle

- FOR counter IN [REVERSE] lower_bound .. higher_bound
- LOOP
- -- sequence of statements
- END LOOP

- FOR record_index IN cursor_name_or_variable
- LOOP
- -- sequence of statements
- END LOOP

Structures de contrôle

- Pour extraire une ligne de résultat d'une requête à la fois, on répète l'opération FETCH sur un curseur ouvert jusqu'à l'épuisement des résultats.
- Il faut donc déclarer le curseur (DECLARE cs CURSOR) avec des paramètres (si nécessaire), l'ouvrir (OPEN cs) avec des arguments (si nécessaire), répéter les extractions de ligne (avec une boucle contenant un FETCH cs) et le fermer (CLOSE cs).
- Une boucle FOR basée sur un curseur permet de réaliser facilement ce genre de traitement.
- L'indice de boucle est une variable RECORD non déclarée et typé implicitement par le type des lignes de résultat associées au curseur.

Structures de contrôle

- Le curseur peut se présenter sous forme d'un nom de curseur, une variable curseur ou une sous – requête.
- Le contrôle de la boucle se fait de façon interne : l'utilisateur n'a pas à se soucier de l'ouverture ni de la fermeture du curseur ni la fin des répétitions.
- Au lieu d'une boucle FOR basée sur un curseur, on peut aussi contrôler la répétition du traitement des lignes de résultat d'un curseur par une boucle conventionnelle LOOP – END LOOP dont la sortie est basée sur l'attribut nommé %NOTFOUND du curseur.
- Les attributs d'un curseur sont : %FOUND, %ISOPEN, %NOTFOUND et %ROWCOUNT.

Structures de contrôle

- LOOP
- -- sequence of statements
- -- exit loop at any point with EXIT [WHEN conditions];
- END LOOP

- WHILE condition
- LOOP
- -- sequence of statements
- END LOOP

Utilisation de collections

- Un élément non initialisé est NULL de façon atomique.
- Pour initialiser un élément, il est possible d'utiliser le constructeur implicite de son type, d'effectuer une affectation (copier un autre élément) ou de faire une extraction de la base de données (par un FETCH).
- Dans un bloc PL/SQL, un VARRAY (ou une table imbriquée) déclaré(e) demeure NULL jusqu'à ce qu'il (elle) soit initialisé(e).
- Il est impossible de faire quoi que ce soit avec une collection NULL. L'exception COLLECTION_IS_NULL survient alors.

Utilisation de collections

- Avant de le (la) manipuler, il est prudent d'effectuer le test IF nom_collection IS NULL sur un VARRAY (ou une table imbriquée).
- Il n'est pas permis de comparer deux collections pour une égalité ou une inégalité. Par conséquent, une collection ne peut pas apparaître dans une clause DISTINCT, GROUP BY ou ORDER BY.
- Dans PL/SQL, pour référer à un élément d'une collection, on utilise le construit ou la forme nom_collection (indice) ; noter que l'exception VALUE_ERROR surviendra si indice est NULL ou non convertible en INTEGER, sauf le cas des tableaux associatifs dont l'indice est de type VARCHAR2.

Utilisation de collections

- Dans SQL, pour manipuler (SELECT, INSERT, UPDATE, DELETE) les éléments individuels d'une collection, il faut utiliser l'opérateur TABLE.
- Il n'est pas possible de référer à un élément de VARRAY dans un INSERT, UPDATE ou DELETE, présentement.
- Il faut donc utiliser un énoncé impératif de PL/SQL sur sa copie locale et employer cette dernière pour effectuer un UPDATE dans la base de données.
- La forme TABLE(CAST(coll_locale AS SQL_coll_type)) permet de manipuler en SQL une collection locale comme si elle était une table de la base de données.

Utilisation de collections

- Dans PL/SQL on peut manipuler les collections locales à l'aide des méthodes suivantes :
  - EXISTS(n IN INT) retourne TRUE si le n^e élément existe.
  - COUNT retourne le nombre d'éléments.
  - LIMIT retourne le nombre maximal d'éléments d'un VARRAY ou NULL (s'il s'agit d'une table imbriquée).
  - FIRST et LAST retournent les premier et dernier indices ou NULL.
  - PRIOR(n IN INT) et NEXT(n IN INT) retournent les indices précédant et succédant l'indice n ou NULL.
  - EXTEND et EXTEND(n IN INT) ajoutent à la fin de la collection, un ou plusieurs éléments tous NULL.

Utilisation de collections

  - EXTEND(n IN INT, i IN INT) ajoute à la fin de la collection, n copies de l'élément i.
  - TRIM et TRIM(n IN INT) suppriment un ou plusieurs éléments à la fin de la collection.
  - DELETE supprime tous les éléments d'une table.
  - DELETE(n IN INT) supprime l'élément n d'une table.
  - DELETE(m IN INT, n IN INT) supprime les éléments de m à n.
- La manipulation d'une collection en PL/SQL peut causer les erreurs suivantes :
  - COLLECTION_IS_NULL, VALUE_ERROR, SUBSCRIPT_OUTSIDE_LIMIT, SUBSCRIPT_BEYOND_COUNT, NO_DATA_FOUND.

Utilisation de collections

- DELETE ne s'applique pas à un VARRAY.
- L'interaction entre DELETE et TRIM est complexe.
- Il est préférable de traiter les tables imbriquées comme les tableaux de taille fixe et d'utiliser seulement DELETE (sans TRIM !).
- Il est possible de gérer une collection comme une pile en utilisant les méthodes EXTEND (pour faire un PUSH) et TRIM (pour faire un POP).
- Un constructeur de collection acceptent les éléments d'une collection comme arguments.
- Une collection vide s'obtient par la forme constructeur( ).

Fonctions et procédures

- Les fonctions et procédures sont des sous-programmes considérées comme des blocs PL/SQL nommés.
- Les fonctions et procédures cataloguées sont des sous-programmes PL/SQL autonomes qui sont compilés et stockées dans le dictionnaire de données à l'aide d'un CREATE OR REPLACE FUNCTION / PROCEDURE souvent exécuté en mode dialogue à travers SQL*PLUS.
- Une fonction ou procédure non autonome peut être déclarée dans un bloc PL/SQL, un sous-programme ou un paquetage sans utiliser la clause CREATE OR REPLACE.

Fonctions et procédures

- Une déclaration de fonction ou procédure doit alors se trouver à la fin d'une section de déclaration.
- La définition ou déclaration d'une fonction ou procédure comprend 2 parties : le corps et la spécification.
- Une fonction et procédure doit être définie ou déclarée avant d'être appelée dans une autre fonction ou procédure.
- Dans le cas des fonctions ou procédures mutuellement récursive, il est nécessaire de faire appel au mécanisme d'annonce (a forward declaration) en ne donnant que la spécification d'une fonction ou procédure.

Fonctions et procédures

- Pour pouvoir être appelée en SQL, une fonction cataloguée doit obéir aux règles de « pureté » ci-dessous, qui servent à contrôler les effets de bord (side effects).
- Si elle est appelée par un SELECT ou un INSERT, UPDATE ou DELETE rendu parallèle, la fonction ne peut modifier aucune table de la BD.
- Si elle est appelée dans un INSERT, UPDATE ou DELETE, la fonction ne peut interroger ni modifier aucune table modifiée par cet énoncé.

Fonctions et procédures

- Si elle est appelée dans un SELECT, INSERT, UPDATE ou DELETE, la fonction ne peut pas exécuter d'énoncés de contrôle de transaction, de séance de travail ou de système. Elle ne peut pas non plus exécuter d'énoncés du LDD.
- Pour vérifier les violations de ces règles, il est possible d'utiliser la directive au compilateur (the) RESTRICT_ REFERENCES (pragma) pour assurer que la fonction ne lit aucune table de BD (RNDS) ni aucune variable de paquetage (RNPS), n'écrit dans aucune table de la BD (WNDS) ni aucune variable de paquetage (WNPS).

Fonctions et procédures

- Mode IN de paramètres/arguments
  - Choix implicite : disponible
  - Passage de valeur : de l'appelant à l'appelé
  - Sémantique : une constante
  - Affectation de valeur : défendue
  - Argument : une constante, une variable initialisée, un littéral ou une expression
  - Passage d 'argument : par référence
  - Particularité : possibilité de valeur initialisée implicitement

Fonctions et procédures

- Mode OUT de paramètres/arguments
  - Choix implicite : non disponible
  - Passage de valeur : de l'appelé à l'appelant
  - Sémantique : une variable locale
  - Affectation de valeur : obligatoire
  - Argument : une variable
  - Passage d'argument : par valeur sauf quand NOCOPY est spécifié (une demande au compilateur d'utiliser le passage par référence)
  - Particularités : argument initialisé à NULL, sans valeur affectée dans le cas d'une exception non traitée

Fonctions et procédures

- Mode IN OUT de paramètres/arguments
  - Choix implicite : non disponible
  - Passage de valeur : de l'appelant à l'appelé d'abord, et ensuite de l'appelé à l'appelant
  - Sémantique : une variable initialisée
  - Affectation de valeur : attendue
  - Argument : une variable
  - Passage d'argument : par valeur sauf quand NOCOPY est spécifié (une demande au compilateur d'utiliser le passage par référence)
  - Particularité : sans valeur affectée dans le cas d'une exception non traitée

Fonctions et procédures

- Quand une variable globale est fournie comme argument dans un appel à un sous-programme et ensuite référée dans le sous-programme, le résultat dépend de la méthode de passage d'arguments choisie par le compilateur.
- La même difficulté peut survenir avec les multiples (>= 2) utilisations d'un même argument (variable, curseur) dans un appel à un sous-programme.
- Il est permis de surcharger les sous-programmes (locaux ou mis en paquetage) et les méthodes (des TDU).
- Une manière de résoudre les appels s'impose.

Fonctions et procédures

- Une fonction tabulaire est une fonction qui produit une collection de rangées (une table imbriquée ou un tableau).
- Une fonction tabulaire peut prendre une collection de rangées (sous forme de REF CURSOR, VARRAY ou TABLE) comme argument d'entrée.
- L'exécution d'une fonction tabulaire peut être rendue parallèle sur un argument REF CURSOR à partitions selon les critères indiqués dans la clause PARTITION BY.
- Les rangées d'une collection retournée par une fonction tabulaire peut être mises en pipeline.

Fonctions et procédures

- Une fonction tabulaire se présente comme une table dans la clause FROM d'un SELECT à l'aide de la forme TABLE (<un appel à une fonction tabulaire>).
- Un sous-programme autonome ou une méthode SQL peut s'exécuter avec les privilèges de son réalisateur (definer) ou de son utilisateur (invoker).
- Un sous-programme peut être (mutuellement) récursif.
- Il est possible d'appeler un sous-programme externe (écrit dans un autre langage) à partir d'un programme PL/SQL.
- Il faut déclarer un sous-programme externe au moyen d'un CREATE FUNCTION/PROCEDURE du SQL.

Paquetage

- Les paquetages permettent de rassembler des structures de données et des programmes au sein d'un module unique.
- Un paquetage est constitué de types, variables, constantes, curseurs, procédures, fonctions et exceptions à utiliser.
- Un paquetage se compose de deux parties : la partie spécification (l'interface) et la partie corps (la réalisation).
- L'interface contient la déclaration des composantes publiques et se définit à l'aide d'un CREATE PACKAGE.
- Le corps contient les composantes publiques et privées et se définit à l'aide d'un CREATE PACKAGE BODY.

Paquetage

- Les sous-programmes d'un paquetage peuvent s'exécuter tous avec les privilèges du réalisateur du paquetage ou ceux de son utilisateur.
- Les variables et curseurs d'un paquetage persistent pour toute la durée d'une séance de travail (a session) ; des données peuvent être maintenues, eu égard aux frontières des transactions, sans avoir à les stocker dans la BD.
- Seuls les sous-programmes et curseurs ont une implantation sous-jacente dans le corps du paquetage.
- La notation pointée est utilisée pour référer à des composantes d'un paquetage.

Paquetage

- Le paquetage STANDARD définit l'environnement PL/SQL.
- Des paquetages spécifiques sont fournis par Oracle et ses outils pour construire des applications en PL/SQL.
- On peut en citer :
  - DBMS_ALERT pour alerter une application quand des valeurs spécifiques de BD changent,
  - DBMS_OUTPUT pour afficher des résultats,
  - DBMS_PIPE pour la communication entre des séances à travers un tube nommé,
  - UTL_FILE pour lire et écrire des fichiers texte,
  - UTL_HTTP pour faire des appels HTTP.

Paquetage

- Paquetage DBMS_OUTPUT :
  - PROCEDURE DBMS_OUTPUT.ENABLE (buffer_size IN INTEGER DEFAULT 2000);
  - PROCEDURE DBMS_OUTPUT.NEW_LINE;
  - PROCEDURE DBMS_OUTPUT.PUT (A VARCHAR2);
  - PROCEDURE DBMS_OUTPUT.PUT (A NUMBER);
  - PROCEDURE DBMS_OUTPUT.PUT (A DATE);
  - PROCEDURE DBMS_OUTPUT.PUT_LINE (A VARCHAR2);
  - PROCEDURE DBMS_OUTPUT.PUT_LINE (A NUMBER);
  - PROCEDURE DBMS_OUTPUT.PUT_LINE (A DATE);
  - PROCEDURE DBMS_OUTPUT.DISABLE;

Paquetage

- Paquetage DBMS_OUTPUT (continuation) :
  - PROCEDURE DBMS_OUTPUT.GET_LINE (line OUT VARCHAR2, status OUT INTEGER);
  - PROCEDURE DBMS_OUTPUT.GET_LINES (lines OUT CHARARR, numlines IN OUT INTEGER);
- Le paquetage DBMS_OUTPUT permet d'afficher de l'information au terminal de session à travers un tampon lors de l'exécution d'un programme PL/SQL.
- C'est un seul moyen facilement accessible pour déboguer un programme PL/SQL.
- SET SERVEROUTPUT ON

Paquetage

- Les avantages des programmes catalogués et des paquetages couvrent plusieurs aspects d'un SGBD : la sécurité, l'intégrité, la performance, la productivité et la maintenance.
- L'appel aux sous-programmes (autonomes catalogués ou mis en paquetage) peut se faire sous l'interface interactive SQL*PLUS, à partir d'un code source PL/SQL, dans un programme C avec l'interface Pro*C, etc.
- Écrire des paquetages aussi général que possible.
- Garder une spécification de paquetage aussi petite que possible.

Réalisation de déclencheurs

- Les déclencheurs (triggers) introduisent un aspect dynamique dans la base de données car ils lancent l'exécution d'un module de programme automatiquement quand quelque chose survient.
- Un déclencheur est donc une procédure stockée dans la base de données et exécutée (ou déclenchée) en réponse à des activités particulières comme un INSERT, UPDATE ou DELETE à effectuer sur une table.
- Le mécanisme de déclencheur supporte les activités associées à la mise à jour d'une table, à celle d'une vue, au support de la base de données et à l'usage d'un schéma.
- L'activité causant l'exécution d'un déclencheur est associée à un énoncé (SQL) dit déclencheur.

Réalisation de déclencheurs

- Un déclencheur peut être créé et remplacé (avec CREATE OR REPLACE TRIGGER), supprimé (avec DROP TRIGGER), activé ou désactivé (avec ALTER TRIGGER et les directives ENABLE ou DISABLE) habituellement dans le contexte d'un outil interactif (SQL*Plus, par exemple).
- Dans la définition d'un déclencheur, il faut préciser l'énoncé déclencheur, le moment d'exécution (BEFORE, AFTER, INSTEAD OF), le mode d'exécution (FOR EACH ROW), la condition d'exécution (WHEN), etc. ainsi que le traitement à effectuer (constituant le corps sous forme d'un bloc PL/SQL).

Réalisation de déclencheurs

- Le corps peut être exécuté avant (BEFORE), après (AFTER) ou à la place de (INSTEAD OF) l'énoncé déclencheur.
- Il peut être exécuté une fois pour chaque ligne (avec FOR EACH ROW) de table affectée par l'énoncé déclencheur ou une seule fois pour toutes les lignes affectées par ce dernier (sans FOR EACH ROW).
- On distingue donc les déclencheurs de type ligne (avec FOR EACH ROW) des déclencheurs de type énoncé (sans FOR EACH ROW).
- Le traitement associé à une ligne affectée peut nécessiter les versions ancienne et nouvelle (désignées par OLD et NEW) de celle-ci.

Réalisation de déclencheurs

- Le traitement associé à toutes les lignes affectées peut nécessiter les versions ancienne et nouvelle (désignées par OLD_TABLE et NEW_TABLE) de celles-ci.
- :NEW et :OLD doivent être utilisés à l'intérieur du corps du déclencheur.
- NEW et OLD doivent être utilisés dans la condition d'une clause WHEN et l'option REFERENCING NEW AS nom_1 OLD AS nom_2 (pour résoudre un conflit de nom).
- Les mêmes règles s'appliquent aussi aux deux symboles OLD_TABLE et NEW_TABLE.
- Il est possible de regrouper la programmation de différents déclencheurs associés à la même table.

Réalisation de déclencheurs

- Les prédicats INSERTING, UPDATING et DELETING permettent de dissocier les traitements en fonction de l'événement qui réveille le mécanisme.
- Le corps d'un déclencheur peut contenir les énoncés du LMD (DELETE, INSERT, SELECT et UPDATE), mais il ne peut contenir ni les énoncés du LDD (ALTER, CREATE et DROP) ni les énoncés du contrôle de transactions (COMMIT, ROLLBACK et SAVEPOINT).
- Le déclencheur INSTEAD OF permet l'insertion, la modification et la suppression de lignes d'une vue objet relationnelle basée sur plusieurs tables en les programmant avec des opérations sur des tables et vues sous-jacentes.

Réalisation de déclencheurs

- L'exécution du corps d'un déclencheur INSTEAD OF remplace donc celle de l'énoncé déclencheur.
- Une table mutante est une table
  - qui est actuellement modifiée par un énoncé DELETE (INSERT, UPDATE) ou
  - qui pourrait être modifiée par les effets d'une contrainte DELETE CASCADE.
- Une séance de travail qui comprend l'énoncé déclencheur ne peut ni interroger ni modifier une table mutante.
- Les vues en train d'être modifiée par des déclencheurs INSTEAD OF ne sont pas considérées comme mutantes.

Réalisation de déclencheurs

- L'option INSTEAD OF ne peut être utilisée que dans les déclencheurs créés sur une vue.
- Les options BEFORE et AFTER ne peuvent pas être utilisées dans les déclencheurs créés sur une vue.
- Le corps d'un déclencheur INSTEAD OF doit assurer le renforcement de la contrainte de l'option CHECK déclarée dans le CREATE VIEW associé.
- Applications de déclencheurs : audit évolué, mise en œuvre de règles d'affaires, autorisation de sécurité, intégrité référentielle, validation de données et de transactions, calcul de valeurs dérivées, mise à jour de vues complexes, suivi et journaux d'opérations, etc.

Aspects objet

- Le terme de TDU objet sera utilisé pour désigner un TDU qui sert à typer une table d'objets dans la BD.
- Un objet persistant (stocké dans la BD) est une ligne d'une table d'objets et une instance d'un TDU objet.
- Une instance d'un TDU objet n'est pas nécessairement un objet persistant et peut être tenue dans une variable PL/SQL locale adéquatement typée.
- Elle n'a alors aucun OID associé et ne peut pas être pointée par un REF.
- Par contre, elle peut très bien contenir un REF qui pointe vers un objet persistant (une ligne d'une table d'objets).

Aspects objet

- Un objet persistant, étant donné son OID, peut être pointé par une référence (un pointeur) déclarée avec le modificateur de type (the type modifier) REF.
- Une association binaire entre deux objets persistants A et B est souvent représentée par une référence à B dans A (attribut toB) et une référence à A dans B (attribut toA).
- La navigation objet se manifeste en notation pointée par les formes « A.toB.attr_B », « B.toA.attr_A », etc. dans un énoncé LMD du SQL.
- Si la variable PL/SQL locale vA contient une copie non persistante de A, la navigation objet ne s'applique pas à toB contenu dans vA : il n'est pas permis d'écrire « vA.toB.attr_B » dans un énoncé de la partie procédurale (PL) du PL/SQL.

Aspects objet

- Afin de manipuler en PL/SQL l'objet pointé par vA.toB, il faut utiliser l'opérateur DEREF dans un énoncé SQL pour obtenir une copie de l'objet pointé dans une variable PL/SQL locale vB typée par le TDU de l'objet A (le type de vA).
  - SELECT DEREF(vA.toB) INTO vB FROM dual;
- Autrement dit, la navigation objet ne s'applique directement pas à un REF contenu dans une instance de TDU objet tenue dans une variable PL/SQL locale.
- De façon plus générale, il est impossible de naviguer dans la base de données à travers les variables REF de PL/SQL.
- Il est à noter que DEREF ne peut pas être utilisé dans un énoncé PL du PL/SQL.

Aspects objet

- Il n'est pas permis de définir des TDU objet dans un bloc anonyme, un sous-programme ou un paquetage PL/SQL.
- La définition d'un TDU se fait en SQL à l'aide de l'énoncé CREATE TYPE, souvent exécuté avec SQL*PLUS.
- La réalisation de toutes les méthodes d'un TDU se fait dans le contexte d'un énoncé SQL CREATE TYPE BODY, souvent exécuté avec SQL*PLUS.
- Une méthode d'un TDU peut être écrite en PL/SQL, Java ou C.
- Pour une méthode écrite en C ou Java, il faut introduire dans le dictionnaire de données les informations requises pour l'appeler sous forme d'une spécification d'appel dans un CREATE TYPE (BODY).

Aspects objet

- PL/SQL ne permet pas de surcharger les opérateurs.
- Un TDU existant peut être utilisé pour typer un élément (une variable, par ex.) PL/SQL dans un bloc, un sous-programme ou un paquetage PL/SQL.
- Le constructeur d'un TDU est utilisé pour initialiser une instance du TDU.
- Dans un appel à un constructeur implicite, il faut fournir une valeur explicite à chacun des attributs.
- Une instance non initialisée d'un TDU est NULL de façon atomique (dans son entier).
- Les attributs d'un NULL s'évaluent tous à NULL.

Aspects objet

- Le prédicat IS NULL permet de vérifier si un élément PL/SQL est NULL.
- Affecter une valeur à un attribut d'un NULL donne naissance à une exception ACCESS_INTO_NULL.
- Un appel à une méthode destiné à un NULL donne naissance à une exception NULL_SELF_DISPATCH.
- La notation pointée est utilisée pour accéder à la valeur d'un attribut d'une instance de TDU.
- Les noms d'attributs peuvent être enchaînés dans la notation pointée pour accéder aux attributs d'une instance de TDU imbriquée.

Aspects objet

- Les méthodes peuvent être appelées à l'aide de la notation pointée.
- Les appels de méthodes en natation pointée peuvent être chaînés
- Dans les énoncés SQL, les appels à une méthode sans paramètre doivent contenir la liste vide ( ).
- La même restriction s'applique aux appels chaînés dans un énoncé PL, sauf le dernier de la chaîne.
- Après un appel de procédure, une chaîne d'appels de méthodes doit se terminer car une procédure ne retourne aucun objet pour effectuer une méthode subséquente.
- Le prédicat IS DANGLING vérifie l'état d'un REF.

Aspects objet

- L'opérateur VALUE peut accepter un alias de table comme argument et chercher à travers le SQL la valeur d'un objet (en faisant une copie de ce dernier) pour affecter à une variable locale typée par le TDU objet associé.
- L'opérateur REF fonctionne comme VALUE mais avec la référence à un objet à la place de sa valeur.
- L'opérateur TREAT retourne seulement les objets qui sont d'un sous-type spécifié.
- Pour obtenir les erreurs de compilation des méthodes d'un TDU, il faut utiliser la commande « show errors » de SQL*PLUS.