Le Langage PL/SQL

Présentation

- Programming Language with SQL
- Langage de programmation procédural d'Oracle non sensible à la casse
- Etend SQL qui est un langage ensembliste
- Adapté à la manipulation de bases de données relationnelles : types, requêtes, curseurs, traitement des erreurs
- Instructions SQL intégrées dans PL/SQL
- Instructions spécifiques à PL/SQL

SQL versus PL/SQL

- SQL:
  - Langage déclaratif non procédural
  - N'intègre pas les structures de contrôle permettant d'éxécuter une boucle itérative
- PL/SQL
  - Extension de SQL en y ajoutant la touche programmation
  - Transmission au SGBD d'un bloc de programmation en lieu et place d'une requête SQL
  - Réduction du nombre d'échanges à travers le réseau
  - Optimisation des performances des applications
  - Appel à des procédures externes écrites dans un autre langage (par exemple c)

Instructions de SQL intégrées dans PL/SQL

- Instructions du LMD
  - SELECT INSERT, UPDATE, DELETE
- Instructions du langage de contrôle des transactions(LCT)
  - COMMIT, ROLLBACK,,...
- fonctions
  - TO_CHAR, TO_DATE, UPPER, SUBSTR

Instructions spécifiques à PL/SQL

- Définition de variables
- Traitements conditionnels
- Traitement répétitifs (ou boucles)
- Traitement des curseurs
- Traitement des erreurs

Structure d'un bloc PL/SQL

- Bloc: Unité de base
- PL/SQL n'interprète pas une commande mais un ensemble de commandes contenues dans un bloc PL/SQL
- DECLARE
  - /* déclaration de variables, constantes, exceptions, curseurs */
- BEGIN [<<nom_bloc>>]
  - /* instructions SQL et PL/SQL */
- [EXCEPTION]
  - /* traitement des exceptions (des erreurs) */
- END [nom_bloc] ;

Déclaration des variables et des constante

- Les variables peuvent être d'un type:
  - scalaire, recevant une seule valeur de type SQL (INTEGER, NUMBER, CHAR, DATE, VARCHAR2, . . .)
  - d'un type composé (RECORD, TABLE, VARRAY,NESTED TABLE)
  - d'un type référence (REF) ou d'un type LOB (adresse vers une donnée externe de très grande taille)
- La syntaxe est
- **nom_var [CONSTANT] type_var [[NOT NULL] := exp] ;**

Exemple de déclaration

- numClient NUMBER(4) ;
- Note NUMBER NOT NULL := 3.5 ;
- Test BOOLEAN := false ;
- TVA CONSTANT REAL := 0.18;
- La contrainte NOT NULL doit être suivie d'une clause d'initialisation
- Les déclarations multiples ne sont pas permises
- On ne peut donc pas écrire :
- v1 , v2 NUMBER ;

Expression et opérateurs

- Les opérateurs de SQL sont valides en PL/SQL.
- Un opérande est une variable, une constante, un littéral, ou un appel de fonction.

Expression et opérateurs

- Opérateurs arithmétiques:
  - + - * /
- Opérateurs sur les dates
  - +, -
- Opérateurs sur les chaines de caractères
  - || (concaténation de chaines de caractères)
- Opérateurs de comparaison (chaines, dates , nombre)
  - <; >; = ;<=; >=; <>; !=; IS NULL; LIKE; BETWEEN; IN
- Opérateurs logiques
  - NOT AND OR
- Autres opérateurs
  - BETWEEN, IN, NOT IN, LIKE, NOT LIKE, IS NULL, IS NOT NULL, ANY, ALL, EXISTS

Saisie et d'affichage en PL/SQL

- Saisie
  - &mot: Saisie d'un mot au clavier: un mot est un nombre ou une suite de chaîne de caractères alphanumériques (éventuellement entre apostrophes)
  - Oracle substituera la suite de caractère saisis à &mot
  - On utilise '&mot' s'il s'agit d'une chaîne de caractères, d'une date ou d'une année pour éviter les erreurs
- Affichage (utiliser SET SERVEROUTPUT ON)
  - DBMS_OUTPUT.PUT_LINE(chaîne)
  - DBMS_OUTPUT.PUT(chaîne)

Collections de Type Record

- Une variable de type record peut représenter une ligne d'une table relationnelle.
- TYPE type_var IS RECORD (
  - nom_champ1 type_champ1 [[NOT NULL] := exp],
  - nom_champn type_champn [[NOT NULL] := exp]
  -);
- **nom_var type_var ;**
- **Exemple**:
- Type PERSONNE IS RECORD(
  - nom VARCHAR2(15),
  - prenom VARCHAR2(20)
  -);
- p PERSONNE ; -- champs p.nom, p.prenom

Utilisation des types implicites

- attribut %type : signifie du même type que
  - v1 Employe.nom%type ;
  - -- v1 du même type que la colonne nom
  - -- de la table Employe
  - v2 v1%type ; -- v2 du même type que v1
- RefProd CHAR(5) ;
- RefClient RefProd%type ; -- RefClient est du même type que RefProd

Utilisation des types implicites

- attribut %rowtype : signifie du même type structuré (record) que
- emp_rec Employe%rowtype ;
- -- emp_rec est de type record
- -- ses attributs suivent le schéma de Employe

Collection de Type Tableaux (TABLE)

- Un tableau PL/SQL
  - collection ordonnée d'éléments du même type dont les indices ne sont pas forcément contigus
  - accessibles uniquement en PL/SQL
  - peuvent grandir dynamiquement (on ne connait pas à l'avance la taille du tableau)

TYPE Type_nom_tableau IS TABLE OF datatype [NOT NULL] INDEX BY BINARY_INTEGER;

nom_tableau: nom du tableauDatatype: type des éléments du tableau (curseur, record, type scalaire, ...)

Declaration : mon_tableau nom_tableau;

Collection de Type Tableaux (TABLE)

- Type TYPE_Tab_Number
  IS TABLE OF Number INDEX BY
  BINARY_INTEGER;--- tableau d'entiers (nombres)

```
Type PERSONNE IS RECORD(
    nom VARCHAR2(15),
    prenom VARCHAR2(20)
    );
```

- Type Type_Tab_Personne IS TABLE OF Personne
  INDEX BY BINARY_INTEGER;----tableau de
  personnes
  T1 Type_Tab_Personne;
  T1(1).nom:='Diaw';
  T1(1).prenom:='Moussa';
-

Collections de Type Table

- VARRAY: Tableau dont on connaît le nombre d'éléments
- TYPE nom_type IS VARRAY (taille) OF type_colonne | type_record
- Ex: TYPE Type_tab_emp IS VARRAY (50) OF Employe%ROWTYPE;
- tab1 Type_tab_emp; -- déclaration de la variable

Description des attributs d'un Tableau

Fonction	Description
T.count	Le nombre d'éléments de T
T.delete	Supprime tout le tableau
T.delete(n)	Supprime l'élément à la position N
T.Delete (n,m)	supprime les éléments de la position N à la position M
T.Exists(n)	retourne true si l'élement à la position N existe
T.first	Indice de la première position
T.last	Indice de la dernière position
T.Next (N)	Position suivant l'indice N
T.Prior(N)	Position avant l'indice N

```
SET SERVEROUTPUT ON
Declare
Type  Type_Tab_Number is table of  Number index by binary_integer;
T1 Type_Tab_Number;   i number;
BEGIN
i :=1;
Loop
      T1(i):=i;  i := i + 1,
      exit when i>10 ;
      end loop;
dbms_output.put_line ('Premier élément : '||T1(T1.FIRST) );
dbms_output.put_line ('Dernier élément : '||T1(T1.LAST) );
dbms_output.put_line  ('Nombre d''élément de T1 avant suppression:
'||T1.COUNT);
dbms_output.put_line  ('Suppresion des éléments '||T1.next(T1.First)||' à
'||T1.prior(T1.Last));
T1.delete(T1.next(T1.First), T1.prior(T1.Last));
dbms_output.put_line  ('nombre d''éléments deT1 aprés supression  :
'||T1.COUNT);
END;
/
```

```
DECLARE
i number(2); Type Type_tab_Number is  varray(12) of number;
tab  Type_tab_Number;
 BEGIN
 for i in 1..12 loop  tab(i):=(2*i);  end loop;
 dbms_output.put_line('Nombre d''Èlements du tableau avant suppression:  '||tab.count);
tab.delete(tab.first);
dbms_output.put_line('Nombre d''Èlements du tableau  apres suppression:  '||tab.count);
 for i in tab.first . .tab.last  loop
 if (tab.exists(i)) then
 dbms_output.put_line(i||' '||tab(i));
 end if;  end loop; end; /
```

Notion de sous-type

- Chaque type de données PL/SQL prédéfini a ses caractéristiques (domaine, opérateurs, ...)
- Un sous-type d'un type de base permet d'en restreindre certaines de ses caractéristiques.
- sous-types prédéinis en PL/SQL servant à rendre les applications compatibles avec la norme SQL (CHARACTER, INTEGER, POSITIVE, . . .).
- L'utilisateur peut définir ses propres sous-types.
  - SUBTYPE nomSousType IS TypeDeBase [NOT NULL];
  - subtype DATE_NAISS_TYPE is DATE NOT NULL ;
  - subtype NOMBRE_TYPE is NUMBER(1,0) -- entre -9 et +9
  - subtype MARQUE_TYPE is Voiture.Marque%type
  - -- type de la colonne Maque de la table Voiture

Conversion de types

- Comme en SQL, Oracle autorise des conversions implicites ou explicites.
- une_date DATE := ’27-11-2007’;
- une_chaine VARCHAR2(20) := sysdate ;
- date1 DATE:= to_date(’10/11/12’,’DD/MM/YY’);

Les structures de contrôle
L'instruction Si alors

- **if** condition ***then***
  - instr
  - [elsif condition then
  - instr]
  - [else
  - instr]
- **end if ;**

Les structures de contrôle

La boucle Pour

- **For** indice in [**reverse**] debut .. Fin **loop**
  - Instruction;
- **end loop**;

NB: Dans la boucle for l'indice est incrémenté par le système

- sans REVERSE : indice varie de debut à Fin, pas de 1
- Avec REVERSE : indice varie de Fin à debut, pas de -1

Exemple Factorielle de 9: Boucle FOR

- **SET SERVEROUTPUT ON**
- **DECLARE**
  - fact number := 1 ;
  - i number(2);
- **BEGIN**
  - FOR i IN 1..9 LOOP
    - fact := fact * i ;
    - END LOOP;
  - DBMS_OUTPUT.PUT_LINE('Factorielle de 9 est : ' ||fact) ;
- **END;**
- /

Les structures de contrôle

La boucle Tant que

- **While** condition **loop**
  - Instruction;
- **end loop;**

Exemple Reste de la division de 10 par 3: While

- **SET SERVEROUTPUT ON**
- **DECLARE**
  - reste number := 10 ;
- **BEGIN**
  - WHILE reste >= 3 LOOP
    - reste := reste - 3 ;
  - END LOOP;
  - DBMS_OUTPUT.PUT_LINE(‘Le reste de la division de 10 par 3 est: ‘||reste);
- **END**;
- /

Les structures de contrôle

La boucle Répéter

- **Loop**
  - Instruction;
  - Exit when condition;
- **end loop;**

Exemple: Insertion dans la table Resulat : Loop

- **CREATE TABLE Resultat (nombre number(2));**
- **DECLARE**
  - nbre number(2) := 1 ;
- **BEGIN**
  - LOOP
    - insert into resultat values (nbre);
    - nbre := nbre +1 ;
    - EXIT WHEN nbre > 10 ;
  - END LOOP ;
- **END ;**
- /
- Select * FROM Resultat;
- Truncate Table Resultat;

Select retournant une seule ligne

- Syntaxe
- SELECT col1, col2 INTO var1, var2 FROM table;
- Règle
  - clause INTO : obligatoire
  - le SELECT doit obligatoirement ramener une seule ligne et une seule, sinon erreur
  - Utiliser un curseur si la requête doit retourner plusieurs lignes
  - **Exemple:**
  - **Declare**
  - **moy Real;**
  - **BEGIN**
  - **Select avg(prix) into moy from Article;**
  - **Dbms_output.put_line(‘La moyenne est :’** ||moy);
  - **END; /**

Curseurs

- Zone de mémoire de taille fixe, utilisée par le noyau d'Oracle pour analyser et interpréter tout ordre SQL
- curseur implicite :
  - créé et géré par Oracle à chaque ordre SQL
- curseur explicite :
  - créé et géré par l'utilisateur afin de pouvoir traiter un SELECT qui retourne plusieurs lignes

Déclaration d'un curseur

- CURSOR nom_curseur IS une_requête ;
  On peut définir des paramétres en entrée dans un curseur
- CURSOR nom_curseur(p1, ..., pn) IS une_requête ;
- Si c est un curseur, le type d'une ligne est c%rowtype.
- Cursor cli_curseur(dnumcli NUMBER) is select Prenom, Nom from Client where numcli= dnumcli ;

Manipulation d'un curseur

- OPEN : initialise le curseur
- FETCH : extrait la ligne courante et passe à la suivante (pas d'exception si plus de ligne)
- CLOSE : ferme et invalide le curseur
- Si on veut parcourir toutes les lignes d'un curseur on utilise une structure répétitive (For, loop ou while)

Attributs Curseur

- %found vrai si le dernier fetch a ramené une ligne (il y a au moins une ligne dans le curseur)
- %notfound vrai si le fetch n'a pas ramené de ligne
- %isopen vrai si le curseur est ouvert
- %rowcount – le nombre de lignes déjà ramenées

Requêtes SQL : exemple avec curseur (1)

- **DECLARE**
  - **CURSOR** c_client IS SELECT prenom, nom from client ORDER BY nom ;
  - client_rec c_client%rowtype ;
- **BEGIN**
  - **OPEN** c_client ;
  - **LOOP** -- parcours des lignes du curseur
  - **FETCH** c_client into client_rec ; ---la ligne courante du curseur est rangée dans la variable client_rec ...
  - **EXIT WHEN** c_client%NOTFOUND ;
  - DBMS_OUTPUT.PUT_LINE('Prenom = '||client_rec.prenom'||' Nom= '||client_rec.nom);
  - **END LOOP ;**
  - **CLOSE** c_client ;
- **END** ;
- /

```
Accept ville  prompt 'Veuillez Saisir une ville: ';
DECLARE
CURSOR c_client (dville Client.ville%type)  IS
select prenom, nom from client where ville=dville;
client_rec c_client%rowtype ; vil  Client.ville%type;
BEGIN
vil:='&ville';
OPEN  c_client (vil) ;
LOOP
FETCH c_client into client_rec ;  -
EXIT WHEN c_client%notfound ;
DBMS_OUTPUT.PUT_LINE('Prenom = '||client_rec.prenom||'Nom=
'||client_rec.nom);
END LOOP ;
IF c_client%rowcount=0 then
DBMS_OUTPUT.PUT_LINE('Aucun client n''habite la ville '||vil);
End if;
CLOSE c_client ;
END ;
/
```

Curseurs et Tableaux

```
-- on va lire les salaires de employés  et les ranger dans  un curseur
Declare
Type  T_TabSalaire is table of employe.salaire%type index by
binary_integer;
tsal T_TabSalaire;
i number;
cursor c_salaire is select salaire from employe;
c_sal_rec  c_salaire%rowtype;
begin
open c_salaire;
i :=1;
loop
fetch c_salaire into c_sal_rec ;
exit when c_salaire%notfound;
Tsal(i):=c_sal_rec.salaire;
i := i + 1;
end loop;
For j  in Tsal.FIRST..Tsal.Last Loop
Dbms_output.put_line (Tsal(j));
End loop;
close c_salaire;
end;
```

Curseur implicite

- Avec un curseur implicite, on peut obtenir des informations sur la requête réalisées, grâce aux attributs :
- SQL%attribut applique l'attribut sur la dernière instruction SQL exécutée
- SQL%rowcount : nombre de lignes affectées par la dernière instruction.
- SQL%found : TRUE si la dernière instruction affecte au moins 1 ligne.
- SQL%notfound : TRUE si la dernière instruction n'affecte aucune ligne.
- delete from Employee where ...
  - if SQL%rowcount > 10 then
  - -- on a supprimé plus de 10 lignes
  - end if ;

Les Exceptions

- En PL/SQL, la gestion des erreurs se fait grâce aux exceptions.
- il existe de nombreuses exceptions prédéfinies (dans le package STANDARD)
- Le programmeur peut aussi définir de nouvelles exceptions
- Le programmeur peut explicitement déclencher une exception
- Le programmeur peut traiter des exceptions.

Déclaration d'une exception

- DECLARE
  - ...nom_erreur EXCEPTION ;
- ... BEGIN
  - ...IF (anomalie) THEN RAISE nom_erreur ;
  - ... EXCEPTION
  - WHEN nom_erreur THEN (traitement) ;

NOTE: Sortie du bloc après exécution du traitement.

Exemple 1

- Create Table Resultat (valeur number (3,5));
- ----inserer des donneés dans Achat avec Qte=0
- SET SERVEROUTPUT ON
- **DECLARE**
  - div NUMBER(3,5);
- BEGIN
  - SELECT 1/Sum(Quantite) INTO div FROM Achat;
    -- 1/prix peut causer une division par zéro
  - INSERT INTO resultat VALUES (div);
  - **EXCEPTION** – Traitement des exception démarre
  - WHEN ZERO_DIVIDE THEN
  - Dbms_output.put_line(‘ Attention division par zéro’);
  - **END**;
  - /
  - Select * from Resultat;
  - Truncate Table Resulat;

Exemple 2

- **DECLARE**
  - Somme NUMBER (4);
  - Div Real;
  - DivisionParZero Exception;
- **BEGIN**
  - SELECT Sum(Quantite) INTO somme FROM Achat
  - If (Somme is null) then Raise DivisionParZero; end if;
  - Div:=1/Somme;
  - INSERT INTO resultat VALUES (div);
  - **EXCEPTION** –-- Traitement des exception démarre
  - WHEN DivisionParZero THEN
  - Dbms_output.put_line('Attention Division par Zéro');
  - **END**;
  - /
  - Select * from resultat;
  - Truncate table resultat;

Déclaration d'une exception

- Les Exceptions et les codes d'erreur
- PL/SQL a défini des exceptions pour certaines erreurs Oracle.
- Par exemple
- Exception TOO_MANY_ROWS
  - erreur Oracle ORA-01422 (code erreur -1422)
- Pour lier une exception à un code d'erreur SQL :
- PRAGMA EXCEPTION_INIT(nom_exception, code_erreur);
- code_erreur est un entier négatif compris entre -20999 et -20000 (codes réservés aux erreurs de l'utilisateur

Déclenchement d'une exception

- Une erreur à l'exécution déclenche une exception
  - Par exemple une division par 0 déclenche l'exception ZERO_DIVIDE
- Dans un programme, on déclenche explicitement une exception grâce à l'instruction **raise nom_exception**
- Pour déclencher une erreur qui n'est pas liée à une exception, utiliser l'instruction :
- raise_application_error(num_erreur, message);
  - num_erreur est un entier négatif compris entre -20999 et -20000 (codes réservés aux erreurs de l'utilisateur)
  - message est une chaine de caractères.

Traitement d'un exception

- Dès que l'erreur Oracle est rencontrée alors passage automatique à la section EXCEPTION pour réaliser le traitement approprié
- **EXCEPTION**
  - WHEN nom_erreur1 THEN (traitement) ;
  - WHEN nom_erreur2 THEN (traitement) ;
  - [WHEN OTHERS THEN (traitement) ;]
    ;-----optionnel

Exemple

```
DECLARE
Type T_TabArticle IS TABLE OF Article.refart%Type INDEX BY BINARY_INTEGER;
Tab  T_TabArticle;
 ArticleInexistant EXCEPTION;
CURSOR C_Article IS SELECT refart FROM Article WHERE
prix <=10000 AND prix>=3000; i Number(2);
BEGIN
Open C_Article;
i:=1;
loop
FETCH C_Article INTO Tab(i);
 i:=i+1;
EXIT WHEN C_Article%NOTFOUND;
end loop;
 if (C_Article%rowcount=0) THEN RAISE ArticleInexistant;
 END IF;
close C_Article;
EXCEPTION
WHEN ArticleInexistant THEN
dbms_output.put_line('Pas d''article dont le prix est compris entre 3 000 et 10 000');
end;
/
```

Modules stockés

- Un module stocké est un programme rangé dans la base de données
- Ces programmes peuvent être appelées par les programmes clients, et sont exécutées par le serveur.
- Avec Oracle, l'utilisateur peut définir des procédures, des fonctions ou des paquetages stockés.

Syntaxe Procédures

- CREATE or replace Procedure
- **NomProcedure (** [<les_parametres>]) is
- <déclarations des variables>
- begin
- <corps Procédure>
- end [NomProcedure];

Les Paramètres

- Pour déclarer un paramétre, la syntaxe est
- <param> [mode] <Type> [:= <valeur_defaut>]
- Pour une procédure, il y a trois modes de passage de paramétre
- Le mode IN : en entréee (par défaut)
- Le mode OUT : en sortie
- Le mode INOUT : en entrée et sortie
- Tous les paramètres d'une fonction sont en mode IN (pas obligatoire d' écrire le mode).

Exemple de Procédure

- **Table Client(Numcli, Nom,Prenom)**
- **Create** or Replace Procedure **AfficherClient**(numclient Client.numcli%type) IS
- Dnom Client.nom%type;
- Dprenom Client.prenom%type;
- Begin
- Select prenom,nom INTO Dprenom,Dnom From Client where numcli=numclient;
- Dbms_output.put_line('Prenom= '||Dprenom||' Nom= '||Dnom);
- End; /

Appel de Procédure

- Accept valeur prompt ‘Saisir le numéro d’un client ‘;
- Declare
- Ncli Client.numcli%type;
- Begin
- Ncli:=&valeur;
- AfficherClient(Ncli);---appel procédure
- End;
- /

Syntaxe Fonctions

- CREATE or replace function
- **NomFonction** [<les_parametres>]
- return <TypeRetour> is
- <déclarations des variables >
- begin
- <corps Fonction>
- end [NomFonction];

Exemple de Fonction

- **Create** or Replace Function
- **PrenomClient**(numclient Client.numcli%type) Return varchar IS
- Dprenom Client.prenom%type;
- Begin
- Select prenom INTO Dprenom From Client where numcli=numclient;
- Return Dprenom;
- End; /

Appel Fonction

- Declare
- Ncli Client.numcli%type;
- Pren varchar(20);
- Begin
- Ncli:=&valeur;
- Pren:=PrenomClient(Ncli);---appel fonction
- Dbms_output.put_line('Prenom du client '||Ncli|| 'est :'||Pren);
- End;

Tansformation Fonction en Procédure

- **Create** or Replace Procedure
- **PrenomClient**(numclient IN Client.numcli%type, Prenomcli OUT Client.prenom%type) IS
- Begin
- Select prenom INTO Prenomcli From Client where numcli=numclient;
- End;

Appel de Procédure

- Declare
- Ncli Client.numcli%type;
- Pren Client.prenom%type;
- Begin
- Ncli:=&valeur;
- PrenomClient(Ncli,Pren);---appel procédure
- Dbms_output.put_line('Prenom du client '||Ncli|| 'est :'||Pren);
- End;

Utilisation Variable globale

- Sous SQLPlus, on peut aussi utiliser des variables globales, et l'instruction execute :
  - **VARIABLE** le_prenom VARCHAR2(20);
  - **EXECUTE** PrenomClient(1,:le_prenom); ---appel d'une fonction à l'extérieur d'un bloc PL/SQL;
  - print le_prenom ;--affichage en dehors du bloc PL/SQL
  - **VARIABLE** le_prenom VARCHAR2(20);
  - **VARIABLE** dnumcli NUMBER(3) ;
  - ----Declaration avant le bloc PL/SQL
  - Begin
  - :dnumcli := 3 ;
  - PrenomClient(:dnumcli,:le_prenom);
    - ---appel à l'intérieur d'un bloc PL/SQL
  - end ;
  - /
  - **print le_prenom ;----Affichage à l'extérieur d'un bloc PL/SQL**

Erreur de compilation

- Si le programme ne compile pas, show errors permet de voir les erreurs sous SQLPlus
- Certaines vues du dictionnaire donnent des informations sur les modules stockés
- la vue USER_OBJECTS donne tous les objets et en particulier les programmes stockés.
- la vue USER_SOURCE donne accés aux sources des programmes de la vue
- USER_ERRORS donne accés aux erreurs de compilation.

Les paquetages

- Un paquetage est constitué d'une spécification (partie visible) et d'une implémentation (partie cachée)
- La spécification du paquetage contient des éléments que l'on rend accessibles à tous les utilisateurs du paquetage.
- Le corps du paquetage contient l'implémentation et ce que l'on veut cacher.
- Au niveau des droits, on donne le droit d' éxécuter un paquetage : on a alors accès à toute la spécification, pas au corps.

Paquetage

- La spécification contient :
  - des signatures de procédures et fonctions,
  - des constantes et des variables
  - des définitions d'exceptions
  - des définitions de curseurs
- Le corps
  - Les corps des procédures et fonctions de la spécification (obligatoire)
  - D'autres procédures et fonctions (cachées)
  - Des déclarations que l'on veut rendre privées
  - Un bloc d'initialisation du paquetage si nécessaire.

Schéma de BD

- Gestion des achats des clients
  - Client (**Numcli**,Nom,Prenom,Ville,Telephone)
  - Article(**RefArt**,Designation,Categorie,Prix)
  - Achat(**Numclient**,**Refaticle**,DateAchat,Quantite)

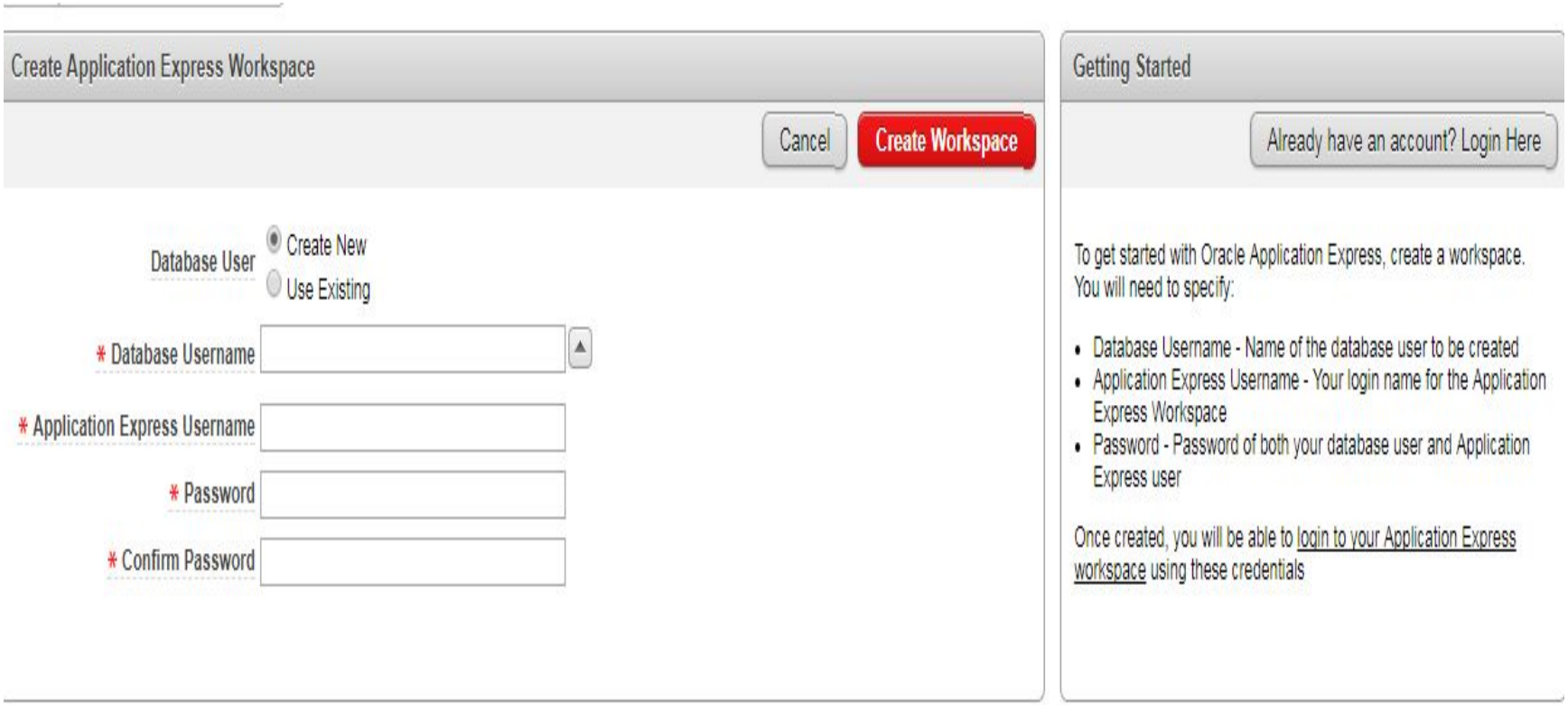
Exemple: Spécification

- **CREATE** or replace package Gestion_Achats AS
- **Procedure** ajout_Article(rec_article Article%Rowtype);
- **Procedure** ajout_Client(rec_client Client%rowtype);
- **Procedure** ajout_Achat(rec_achat Achat%rowtype);
- **ARTICLE_INCONNU** EXCEPTION ;
- PRAGMA **exception_init(ARTICLE_INCONNU**,-20100);
- **ARTICLE_UTILISE** EXCEPTION ;
- PRAGMA **exception_init(ARTICLE_UTILISE**,-20101);
- end Gestion_Achats;

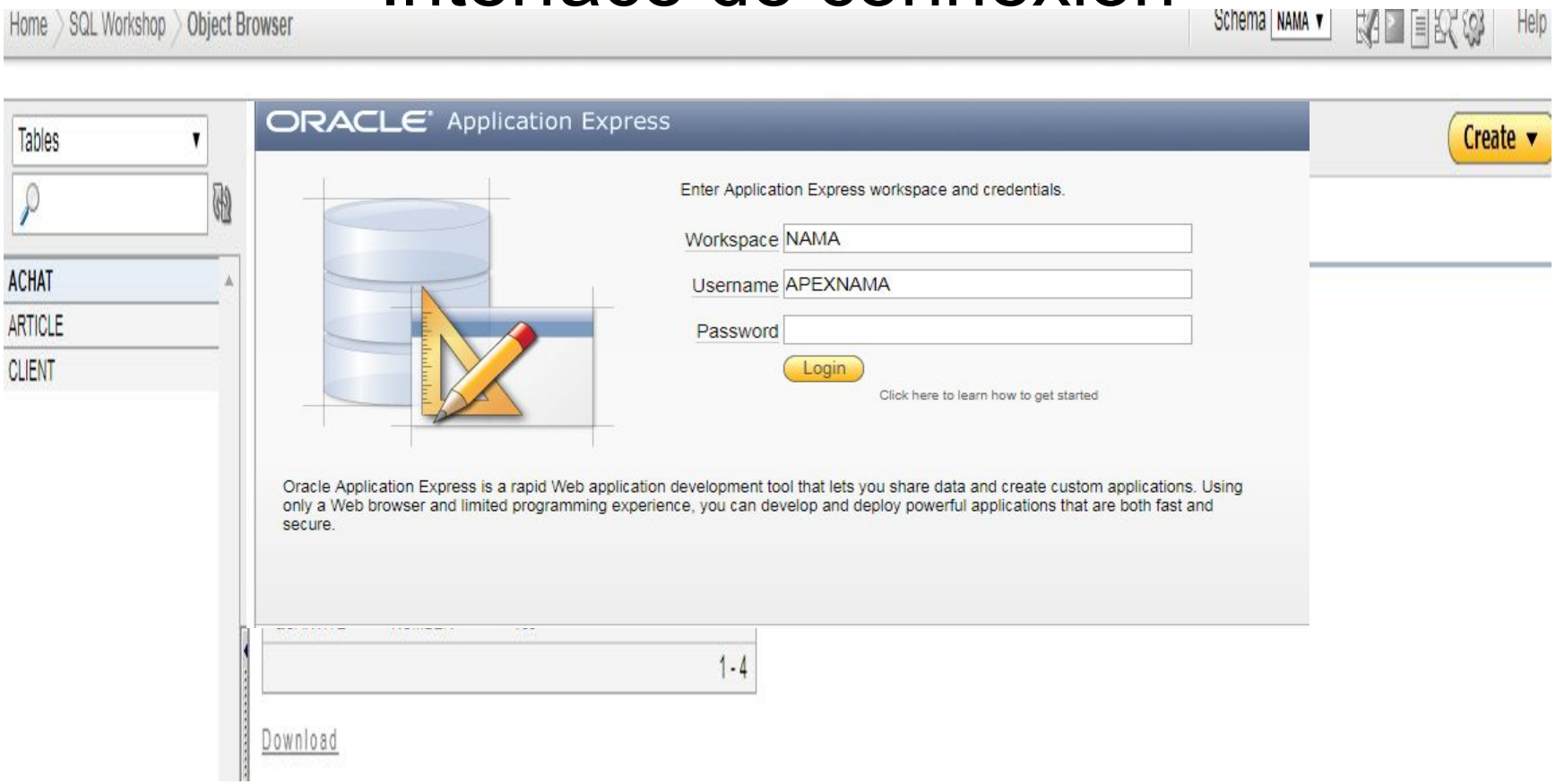
Exemple:Implémentation

- **CREATE or replace package body Gestion_Achats** AS
- **Procedure ajout_Achat** (rec_achat Achat%rowtype) IS
- anc_qte Achat.quantite%type:=0 ;
- begin
- **select** quantite into anc_qte from Achat where numclient=rec_achat.numclient
- and refarticle=rec_achat.refarticle ;
- -- il y a déjà une ligne pour cette commande et ce produit
- If (anc_qte =0)THEN RAISE ARTICLE_INCONNU;
- End if;
- **Update** Achat SET quantite = anc_qte+rec_achat.quantite where
- numclient=rec_achat.numclient and refarticle=rec_achat.refarticle;
- ---qte est la nouvelle quantité et anc_qte est l'ancienne quantité
- Exception
- when ARTICLE_INCONNU then -- creer une nouvelle ligne
- insert into Achat values(rec_achat.numclient,rec_achat.refaricle,rec_achat.quantite);
- **end ajout_Achat;**
- **end Gestion_Achats;**
- /

TRAVAUX PRATIQUE



Interface de connexion



Modules de APEX

- FIN DU COURS