Principes d'utilisation des systèmes de gestion de bases de données

Cours 3

Ecole informatique d'Orsay - M1
2004/05
Emmanuel Waller, LRI

Résumé des épisodes précédents

- But du module : savoir utiliser un SGBD pour résoudre les problèmes BD rencontrés par une application généraliste
- Architecture, mode interactif, mode programme

Les 12 problèmes de base de données

- Modèle de données
- Conception
- Persistance
- Contraintes d'intégrité
- Confidentialité
- Indépendance des niveaux
- Mise à jour
- Interrogation

Les 12 problèmes de base de données

- Reprise sur panne
- Contrôle de concurrence
- Grandes quantités
- Répartition

Cours 3

- PL/SQL
- Premiers exemples de concurrence

PL/SQL, triggers, procédures stockées

- Principes
- Le langage
- Objets de la base utilisant PL/SQL
  - Triggers
  - Procédures et fonctions stockées
  - Packages
- Utilisation concrète

Qu'est-ce que c'est ? Qui l'utilise ?

- PL/SQL =
  SQL, langage ensembliste déclaratif (quoi) : puissant
  +
  Langage procédural (comment) : souple
- « ProceduraL SQL »
- Variables, tests, boucles, curseurs, exceptions
- Qui : programmeur d'application, DBA

Quand l'utiliser ? A quoi ça sert ?

- Mode interactif : taper le code PL/SQL sous SQL*Plus et l'exécuter
- Dans triggers, procédures stockées
- Mode programme : appel depuis programmes
- But :
  - = du code « BD » stocké dans la base
  - Certains traitements très proches des données
  - Utiles à toutes les applications au-dessus de la base
  - Performances

Exemple

```
SQL>
declare
      x varchar2(20);
begin
       x:= 'Hello World';
       dbms_output.put_line(x);
end;
```

Le bloc PL/SQL

- Séquence d'ordres SQL et instructions PL/SQL
- Unité de base d'exécution (performances) : compilé et exécuté d'un seul coup
- 3 sections
- Commentaires :

```
-- ceci est un commentaire, sur une ligne
/*  et cela
    Aussi, sur plusieurs */
```

Structure du bloc

- Declare

```
. . . /* variables, constantes, exceptions, curseurs */
```

- Begin

```
. . . /* Ordres SQL, instructions PL/SQL, structures de
      contrôle */
```

- Exceptions

```
. . . /* Traitement des exceptions */
```

- End

Quels ordres SQL dans PL/SQL ?

- Select
- Insert, update, delete
- Transactions : commit, rollback, savepoint, etc.
- Fonctions : to_char, to_date, round, etc.
- Syntaxe : identique à SQL (sauf certains cas)

Le langage

- Variables
- Structures de contrôle
- Curseurs
- Gestion des exceptions

Déclaration des variables

- Variable = nom + type
- Nom = lettre, chiffre, $, _, #, 30 caractères, sauf mots réservés
- Type = Oracle, PL/SQL (booléen), « dérivé » (du dictionnaire de données)
- Locales (ou extérieures au bloc)

Exemple

```
Declare
    nom varchar2(10) not null;
    adresse varchar2(20);
    x number := 1;
    pi constant number(7,5) := 3.14159;
    rayon number(15,5) := 1;
    surface number(15,5) := pi * rayon ** 2;
```

Types dérivés

- Référence à une entité existante : colonne, table, curseur, variable
- Exemple :

```
Declare
    vnom client.nom%type;
    x number(10,3);
    y x%type;
    cli client%rowtype;
```

Utilisation des variables

- Affectation à une variable de :
  - Expression
  - Résultat requête : select into, fetch into (curseur)
- Résultat d'une requête (ligne(s)):
  - Sous SQL*Plus : à l'écran
  - Dans PL/SQL : dans une variable (dans un curseur si plusieurs lignes)

Exemple

```
begin
   x := 0;
   vnom := 'Monsieur' || vnom;
   y := (x + 5) * y;
```

Select into

```
declare
   vref varchar2(10);
   vprix articles.prixht%type;
   cli client%rowtype;
begin
   select refart, prixht
      into vref, vprix
      from articles
      where nom = 'cravate';
   select *
      into cli
      from client
      where nom = 'Toto';
```

Select into (2)

- Résultat de la requête :
  - Exactement une ligne
  - Zéro ou plusieurs : impossible

Exemple

```
declare
   vnocli client.nocli%type := 10;
   vnom client.nom%type;
begin
   vnom := 'Dupont';
   update client
      set nom = vnom
      where nocli = vnocli;
   commit;
end;
```

Portée des variables

- Leur bloc PL/SQL
- (Il existe blocs imbriqués)

Le langage

- Variables
  - Déclaration
  - Types
  - Utilisation
  - Portée
- Structures de contrôle
  - Séquencement
  - Tests
  - Boucles
- Curseurs
- Gestion des exceptions

Structures de contrôle

- Opérateurs :
  - comme SQL : =, <, >, !, >=, <=, is (not) null, between, like, and, or, etc.
- Tests :

```
if vnocli = 10
then
   update . . . ;
   commit;
else rollback;
end if;
. . . ( il existe aussi elsif)
```

Boucles

- Exemple : créer des clients

```
for n in 100 . . 110 loop
    insert into client(nocli) values (n);
end loop;
n := 100;
while n <= 110 loop
    insert into client(nocli) values (n);
    n := n + 1;
end loop;
```

Curseurs : le problème

- Select a,b into x,y from t where clé = 123
  - Récupère l'unique ligne du résultat de la requête
  - La place dans le couple de variables (x,y)
- But : récupérer un résultat de requête faisant plusieurs lignes

Curseur

- = zone mémoire
- Nommée
- À laquelle est associée une requête
- Peut contenir 0, 1 ou plusieurs lignes
- Taille réglée à l'exécution
- Sert à contenir l'ensemble des lignes résultat de cette requête

Fonctionnement

1. Déclaration du curseur
2. Remplissage en une seule fois par exécution de la requête
3. Récupération des lignes une par une
   (parcours séquentiel du curseur par un pointeur logique)
4. Libération de la zone (facultatif ?) : elle devient inaccessible

Exemple 1

- Table article(refart, nom, prixht)
- Afficher à l'écran la référence du premier article par ordre alphabétique
- On suppose qu'il y en a au moins un
- Select into impossible car il peut y en avoir plusieurs

```
Declare
    cursor c1 is
        select refart from article order by nom;
    art article.refart%type;
begin
    open c1;
    fetch c1 into art; -- affectation de variable (fetch into)
    dbms_output.putline(art);
    -- éventuels autres non parcourus
    close c1;
end;
```

Exemple 2

- Afficher à l'écran la référence du premier article par ordre alphabétique, et du deuxième s'il existe
- On suppose qu'il y en a au moins un
- Attributs : indicateurs sur l'état du curseur
  - Ex : c2 curseur
    - c2%found : booléen, à true ssi le dernier fetch a ramené une ligne
    - c2%notfound
    - C2%isopen

```
Declare
    cursor c2 is
        select refart from article order by nom;
    art article.refart%type;
begin
    open c2; fetch c2 into art; dbms_output.putline(art);
    fetch c2 into art;
    if c2%found
        then dbms_output.putline(art);
    close c2;
end;
```

Parcours du curseur

- Analogue à parcours de fichier séquentiel
- Notion de pointeur logique
- Après le open :
  - pointeur logique positionné avant la 1ère ligne
  - c%found est à vrai
- Fetch c into a :
  1. Avance le pointeur
  2. Lit la ligne pointée par le pointeur
  3. Si pas de ligne pointée (l'avant-dernier fetch avait lu la dernière) : c%found devient faux

Exemple 3

- Afficher à l'écran la référence de tous les articles
- On ne suppose rien

```
Declare
   cursor c3 is select refart from article order by nom;
   art article.refart%type;
begin
   open c3;
   fetch c3 into art;
   while c3%found loop
      dbms_output.putline(art);
      fetch c3 into art;
   end loop;
   close c3;
end;
```

Exemple 4

- Afficher la référence et le prix de tous les articles

```
Declare
   cursor c4 is
      select refart, prixht
      from article;
   art c4%rowtype; -- type dérivé (curseur)
begin
   open c4;
   fetch c4 into art;
   while c4%found loop
      dbms_output.putline(c1.refart || ' ' || c1.prixht);
      fetch c4 into art;
   end loop;
   close c4;
end;
```

Curseur avec paramètre

- Afficher les références des articles dont le prix est passé en paramètre

```
Declare
   cursor c5 (p article.prixht%type) is
      select refart
      from article
      where prixht = p;
   art article.refart%type;
begin
   open c5(7.77);
   fetch c5 into art;
   dbms_output.putline(art);
   . . .
end;
```

Le langage

- Variables
- Structures de contrôle
- Curseurs
- Gestion des exceptions

Gestion des exceptions

- But : associer un traitement à des « exceptions » survenues lors de l'exécution d'un bloc PL/SQL
  - Exceptions définies par l'utilisateur
  - Erreurs Oracle (ordres SQL non exécutés)
- Même principe qu'en Java, mais moins général
- Exceptions utilisateur :
  - Exemple :
    - Afficher tous les articles
    - Si l'un dépasse 10 000 euros, le dire et arrêter

```
declare
   cursor c1 is select . . .; art c1%rowtype;
   depassement exception;
begin
   open c1; fetch c1 into art;
   while c1%found loop
      dbms_output.putline(c1.refart || ' ' || c1.prixht);
      if c1.prixht > 10 000
         then raise depassement;
      fetch c1 into art;
   end loop; close c1;
exception
   when depassement then dbms_output.putline('c'est trop');
end;
```

Erreurs Oracle

- On dit qu'il y a erreur quand un ordre SQL ne réussit pas à s'exécuter
- Exemple : nom de table incorrect dans un select
- Toutes les erreurs SQL (et PL/SQL) ont un code
- Principe :
  - Associer un nom à ce code
  - Traiter l'erreur correspondante comme une exception utilisateur

Erreurs Oracle (2)

- En cas d'erreur (ex : table inexistante) :
  - Sous SQL*Plus :
    - Annonce par message à l'écran
    - Induit une réaction du programmeur en mode interactif
  - Dans PL/SQL : gestion par une exception :
    - Détection de l'erreur par le gestionnaire d'exception
    - Induit une réaction du programme définie dans la section de traitement des exceptions

```
Declare
   cursor c1 is select . . .; art c1%rowtype;
   pasdetable exception;
   pragma exception_init(pasdetable, -942);
begin -- dès erreur passage à la section exception
   open c1; fetch c1 into art;
   while c1%found loop
      dbms_output.putline(c1.refart || ' ' || c1.prixht);
      fetch c1 into art;
   end loop; close c1;
exception
   when pasdetable
      then dbms_output.putline('table n'existe pas !');
   -- puis sortie du bloc après traitement de l'exception
   when others then dbms_output.putline('autre erreur !');
end;
```

Le langage

- Variables
- Structures de contrôle
- Curseurs
- Gestion des exceptions

PL/SQL, triggers, procédures stockées

- Principes
- Le langage
- Objets de la base utilisant PL/SQL
  - Triggers
  - Procédures et fonctions stockées
  - Packages

Triggers

- « déclencheur » = bloc PL/SQL
  - Associé à une table
  - s 'exécute lors d'un ordre DML (mise à jour d'instance) sur cette table

Exemple 1

- Créer un trigger qui prévienne d'une suppression :

```
create trigger avant_supp_cli
   before delete
   on client
   begin
      dbms.output_putline('je supprime un client');
   end;
```

Exemple 2

- Agir en cas de baisse du prix :

```
create trigger post_maj_prix
   after update
   on article
   for each row
   when (old.prixht > new.prixht)
   begin
      dbms.output_putline('ancien prix' || :old.prixht);
   end;
```

Syntaxe

```
Create [or replace] trigger nom
   before | after
      insert | update [of col] | delete | insert or delete
         | ... or ...
   on table
   [ for each row ]
   [ when ( . . .) ]
   bloc PL/SQL
```

Procédures et fonctions stockées

- = code PL/SQL
  - Stocké dans la base
  - Pourra être appelé par les couches supérieures de différentes applications, depuis :
    - Bloc PL/SQL
    - SQL*Plus

Exemple

- Déclaration :

```
create procedure supprime (num in char)
   begin
      delete . . . where . . . num . . . ;
      . . .
   end;
```

- Appel :
  - SQL> execute suppr('W'); -- sous SQL*Plus
  - Declare x char; -- depuis bloc PL/SQL
    begin . . . suppr(x); . . . end;

Syntaxe

```
Create [or replace] procedure p
   (x1 in | out t1, x2 . . . , . . .) as
   . . . -- déclarations
   begin
      . . . -- corps
      [ exception . . . ] -- gestion des exceptions
   end
```

Fonction stockée

```
create function fact (n  in integer)
   return integer as
   begin
      if n = 0
         then return 1;
         else return n * fact(n – 1);
      endif;
   end;
```

Procédures et fonctions non stockées

- Dans la section declare d'un bloc
- Pas mot-clef create
- Is au lieu de as (?)

Packages

- = regrouper des procédures et fonctions
- Éléments cachés : permet encapsulation (style objet, modules)
- Deux parties :
  - En-tête
  - Corps

```
Create package EWPCK as
   procedure p (... paramètres ...);
   procedure q (...);
   function f (...) type;
end;
Create package body EWPCK as
   procedure p (...) begin ... end;
   procedure q (...) begin ... end;
   function f (...) begin ... end;
end;
```

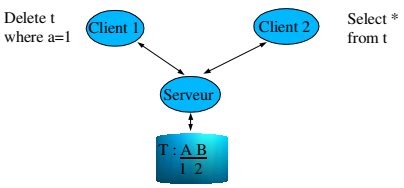
Utilisation concrète

```
SQL> set serveroutput on
SQL> declare
        . . .
        end;
        /
```

PL/SQL, triggers, procédures stockées

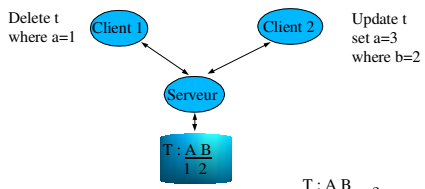
- Principes
- Le langage
- Objets de la base utilisant PL/SQL
  - Triggers
  - Procédures et fonctions stockées
  - Packages
- Utilisation concrète

Premiers exemples de concurrence



- Rappel : seules les modifications d'une transaction validée sont visibles par autres clients (même utilisateur ou non)

Premiers exemples de concurrence (2)



- Quel état final ?
  - T : A B / 3 2 ?
  - T : A B / vide ?

Exemple 1

1. p1 : delete t where b = 1
2. p2 : update t set b = b - 1
3. p1 : commit
4. p2 : commit

Exemple 2

1. p2 : update t set b = b - 1
2. p1 : delete t where b = 1
3. p1 : commit
4. p2 : commit

- Exemple 1 : notion de verrou bloquant
- Exemple 2 : (1,1) verrouillé : p1 ne commence pas son exécution (même pas de sélection)

Résumé du cours 3

- PL/SQL
  - Principe : code BD, manipulations instance, transactions
  - Le langage : variables, contrôle, curseurs, exceptions
  - Triggers, procédures et fonctions stockées, packages
- Premiers exemples de concurrence
  - Verrous bloquant

Projet partie 1 bis

- Commerce inter-entreprise :
  - Tables et vues nécessaires
  - Protocole
  - Incluant banques et CCI
- Une feuille recto-verso tapée
- A rendre secrétariat mercredi 6 octobre minuit

Projet partie 2

- Listing de l'application complète en mode interactif (partie solo seulement)
- Gestion de tous les problèmes BD, commentée dans le listing
- A rendre lundi 18 octobre minuit

Prochains cours

- Partie II
  - Cours 4 : contrôle de concurrence
  - Cours 8 : grandes quantités
- Partie III : mode programme (C, Java, PHP)

Exercices

Principes d'utilisation des systèmes de gestion de bases de données

Cours 3

Ecole informatique d'Orsay - M1
2004/05
Emmanuel Waller, LRI

Résumé des épisodes précédents

- But du module : savoir utiliser un SGBD pour résoudre les problèmes BD rencontrés par une application généraliste
- Architecture, mode interactif, mode programme

Les 12 problèmes de base de données

- Reprise sur panne
- Contrôle de concurrence
- Grandes quantités
- Répartition

Les 12 problèmes de base de données

- Modèle de données
- Conception
- Persistance
- Contraintes d'intégrité
- Confidentialité
- Indépendance des niveaux
- Mise à jour
- Interrogation

Cours 3

- PL/SQL
- Premiers exemples de concurrence

PL/SQL, triggers, procédures stockées

- Principes
- Le langage
- Objets de la base utilisant PL/SQL
  - Triggers
  - Procédures et fonctions stockées
  - Packages
- Utilisation concrète

Qu'est-ce que c'est ? Qui l'utilise ?

- PL/SQL =

  SQL, langage ensembliste déclaratif (quoi) : puissant

  +

  Langage procédural (comment) : souple
- « ProceduraL SQL »
- Variables, tests, boucles, curseurs, exceptions
- Qui : programmeur d'application, DBA

Quand l'utiliser ? A quoi ça sert ?

- Mode interactif : taper le code PL/SQL sous SQL*Plus et l'exécuter
- Dans triggers, procédures stockées
- Mode programme : appel depuis programmes
- But :
  - = du code « BD » stocké dans la base
  - Certains traitements très proches des données
  - Utiles à toutes les applications au-dessus de la base
  - Performances

Exercices

Le bloc PL/SQL

- Séquence d'ordres SQL et instructions PL/SQL
- Unité de base d'exécution (performances) : compilé et exécuté d'un seul coup
- 3 sections
- Commentaires :

```
-- ceci est un commentaire, sur une ligne
/*  et cela
    Aussi, sur plusieurs */
```

Exemple

```
SQL>
declare
      x varchar2(20);
begin
       x:= 'Hello World';
       dbms_output.put_line(x);
end;
```

Prochains cours

- Partie II
  - Cours 4 : contrôle de concurrence
  - Cours 8 : grandes quantités
- Partie III : mode programme (C, Java, PHP)

Projet partie 2

- Listing de l'application complète en mode interactif (partie solo seulement)
- Gestion de tous les problèmes BD, commentée dans le listing
- A rendre lundi 18 octobre minuit

Quels ordres SQL dans PL/SQL ?

- Select
- Insert, update, delete
- Transactions : commit, rollback, savepoint, etc.
- Fonctions : to_char, to_date, round, etc.
- Syntaxe : identique à SQL (sauf certains cas)

Structure du bloc

- Declare

  . . . /* variables, constantes, exceptions, curseurs */
- Begin

  . . . /* Ordres SQL, instructions PL/SQL, structures de contrôle */
- Exceptions

  . . . /* Traitement des exceptions */
- End

Projet partie 1 bis

- Commerce inter-entreprise :
  - Tables et vues nécessaires
  - Protocole
  - Incluant banques et CCI
- Une feuille recto-verso tapée
- A rendre secrétariat mercredi 6 octobre minuit

Résumé du cours 3

- PL/SQL
  - Principe : code BD, manipulations instance, transactions
  - Le langage : variables, contrôle, curseurs, exceptions
  - Triggers, procédures et fonctions stockées, packages
- Premiers exemples de concurrence
  - Verrous bloquant

Déclaration des variables

- Variable = nom + type
- Nom = lettre, chiffre, $, _, #, 30 caractères, sauf mots réservés
- Type = Oracle, PL/SQL (booléen), « dérivé » (du dictionnaire de données)
- Locales (ou extérieures au bloc)

Le langage

- Variables
- Structures de contrôle
- Curseurs
- Gestion des exceptions

- Exemple 1 : notion de verrou bloquant
- Exemple 2 : (1,1) verrouillé : p1 ne commence pas son exécution (même pas de sélection)

Exemple 2

1. p2 : update t set b = b - 1
2. p1 : delete t where b = 1
3. p1 : commit
4. p2 : commit

Types dérivés

- Référence à une entité existante : colonne, table, curseur, variable
- Exemple :

```
Declare
    vnom client.nom%type;
    x number(10,3);
    y x%type;
    cli client%rowtype;
```

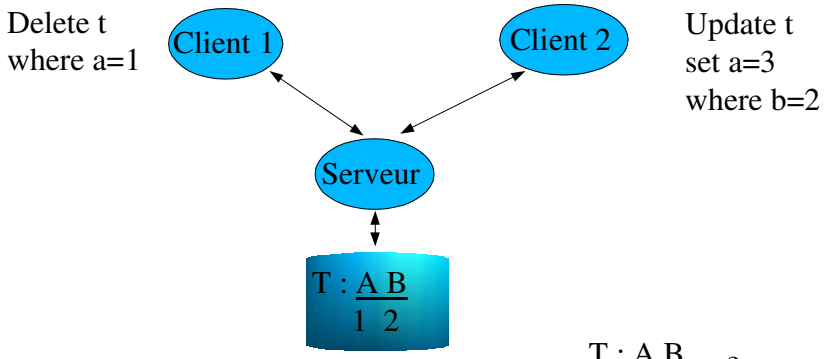
Exemple

```
Declare
    nom varchar2(10) not null;
    adresse varchar2(20);
    x number := 1;
    pi constant number(7,5) := 3.14159;
    rayon number(15,5) := 1;
    surface number(15,5) := pi * rayon ** 2;
```

Exemple 1

1. p1 : delete t where b = 1
2. p2 : update t set b = b - 1
3. p1 : commit
4. p2 : commit

Premiers exemples de concurrence (2)



- Quel état final ?

T : A B ?
3 2

T : A B ?
vide

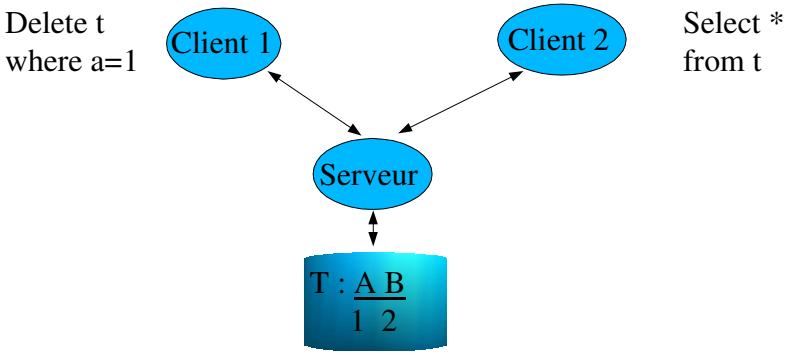
Exemple

```
begin
    x := 0;
    vnom := 'Monsieur' || vnom;
    y := (x + 5) * y;
```

Utilisation des variables

- Affectation à une variable de :
  - Expression
  - Résultat requête : select into, fetch into (curseur)
- Résultat d'une requête (ligne(s)):
  - Sous SQL*Plus : à l'écran
  - Dans PL/SQL : dans une variable (dans un curseur si plusieurs lignes)

Premiers exemples de concurrence



- Rappel : seules les modifications d'une transaction validée sont visibles par autres clients

PL/SQL, triggers, procédures stockées

- Principes
- Le langage
- Objets de la base utilisant PL/SQL
  - Triggers
  - Procédures et fonctions stockées
  - Packages
- Utilisation concrète

Select into

```
declare
    vref varchar2(10);
    vprix articles.prixht%type;
    cli client%rowtype;
begin
    select refart, prixht
      into vref, vprix
      from articles
      where nom = 'cravate';
    select *
      into cli
      from client
```

Select into (2)

- Résultat de la requête :
  - Exactement une ligne
  - Zéro ou plusieurs : impossible

Utilisation concrète

```
SQL> set serveroutput on
SQL> declare
        . . .
        end;
        /
```

```
Create package EWPCK as
    procedure p (... paramètres ...);
    procedure q (...);
    function f (...) type;
end;
Create package body EWPCK as
    procedure p (...) begin ... end;
    procedure q (...) begin ... end;
    function f (...) begin ... end;
end;
```

Portée des variables

- Leur bloc PL/SQL
- (Il existe blocs imbriqués)

Exemple

```
declare
     vnocli client.nocli%type := 10;
     vnom client.nom%type;
begin
     vnom := 'Dupont';
     update client
        set nom = vnom
        where nocli = vnocli;
     commit;
end;
```

Packages

- = regrouper des procédures et fonctions
- Éléments cachés : permet encapsulation (style objet, modules)
- Deux parties :
  - En-tête
  - Corps

Procédures et fonctions non stockées

- Dans la section declare d'un bloc
- Pas mot-clef create
- Is au lieu de as (?)

Structures de contrôle

- Opérateurs :
  - comme SQL : =, <, >, !, >=, <=, is (not) null, between, like, and, or, etc.
- Tests :

```
if vnocli = 10
then
   update . . . ;
   commit;
else rollback;
end if;
. . . ( il existe aussi elsif)
```

Le langage

- Variables
  - Déclaration
  - Types
  - Utilisation
  - Portée
- Structures de contrôle
  - Séquencement
  - Tests
  - Boucles
- Curseurs
- Gestion des exceptions

Fonction stockée

```
create function fact (n  in integer)
   return integer as
   begin
      if n = 0
         then return 1;
         else return n * fact(n – 1);
      endif;
   end;
```

Syntaxe

```
Create [or replace] procedure p
   (x1 in | out t1, x2 . . . , . . .) as
   . . . -- déclarations
   begin
      . . . -- corps
      [ exception . . . ] -- gestion des exceptions
   end
```

- Out : passage par adresse

Boucles

- Exemple : créer des clients

```
for n in 100 . . 110 loop
   insert into client(nocli) values (n);
end loop;
n := 100;
while n <= 110 loop
   insert into client(nocli) values (n);
   n := n + 1;
end loop;
```

Curseurs : le problème

- Select a,b into x,y from t where clé = 123
  - Récupère l'unique ligne du résultat de la requête
  - La place dans le couple de variables (x,y)
- But : récupérer un résultat de requête faisant plusieurs lignes

Exemple

- Déclaration :

```
create procedure supprime (num in char)
   begin
      delete . . . where . . . num . . . ;
      . . .
   end;
```

- Appel :
  - SQL> execute suppr('W'); -- sous SQL*Plus
  - Declare x char; -- depuis bloc PL/SQL

Procédures et fonctions stockées

- = code PL/SQL
  - Stocké dans la base
  - Pourra être appelé par les couches supérieures de différentes applications, depuis :
    - Bloc PL/SQL
    - SQL*Plus

Fonctionnement

1. Déclaration du curseur
2. Remplissage en une seule fois par exécution de la requête
3. Récupération des lignes une par une

   (parcours séquentiel du curseur par un pointeur logique)
4. Libération de la zone (facultatif ?) : elle devient inaccessible

Curseur

- = zone mémoire
- Nommée
- À laquelle est associée une requête
- Peut contenir 0, 1 ou plusieurs lignes
- Taille réglée à l'exécution
- Sert à contenir l'ensemble des lignes résultat de cette requête

Syntaxe

```
Create [or replace] trigger nom
   before | after
      insert | update [of col] | delete | insert or delete
         | ... or ...
   on table
   [ for each row ]
   [ when ( . . .) ]
   bloc PL/SQL
```

Exemple 2

- Agir en cas de baisse du prix :

```
create trigger post_maj_prix
    after update
    on article
    for each row
    when (old.prixht > new.prixht)
    begin
       dbms.output_putline('ancien prix' || :old.prixht);
    end;
```

```
Declare
   cursor c1 is
      select refart from article order by nom;
   art article.refart%type;
begin
   open c1;
   fetch c1 into art; -- affectation de variable (fetch into)
   dbms_output.putline(art);
   -- éventuels autres non parcourus
   close c1;
end;
```

Exemple 1

- Table article(refart, nom, prixht)
- Afficher à l'écran la référence du premier article par ordre alphabétique
- On suppose qu'il y en a au moins un
- Select into impossible car il peut y en avoir plusieurs

Exemple 1

- Créer un trigger qui prévienne d'une suppression :

```
create trigger avant_supp_cli
    before delete
    on client
    begin
       dbms.output_putline('je supprime un client');
    end;
```

Triggers

- « déclencheur » = bloc PL/SQL
  - Associé à une table
  - s 'exécute lors d'un ordre DML (mise à jour d'instance) sur cette table

```
Declare
   cursor c2 is
      select refart from article order by nom;
   art article.refart%type;
begin
   open c2; fetch c2 into art; dbms_output.putline(art);
   fetch c2 into art;
   if c2%found
      then dbms_output.putline(art);
   close c2;
end;
```

Exemple 2

- Afficher à l'écran la référence du premier article par ordre alphabétique, et du deuxième s'il existe
- On suppose qu'il y en a au moins un
- Attributs : indicateurs sur l'état du curseur
  - Ex : c2 curseur
    - c2%found : booléen, à true ssi le dernier fetch a ramené une ligne
    - c2%notfound
    - C2%isopen

PL/SQL, triggers, procédures stockées

- Principes
- Le langage
- Objets de la base utilisant PL/SQL
  - Triggers
  - Procédures et fonctions stockées
  - Packages

Le langage

- Variables
- Structures de contrôle
- Curseurs
- Gestion des exceptions

Exemple 3

- Afficher à l'écran la référence de tous les articles
- On ne suppose rien

Parcours du curseur

- Analogue à parcours de fichier séquentiel
- Notion de pointeur logique
- Après le open :
  - pointeur logique positionné avant la 1ère ligne
  - c%found est à vrai
- Fetch c into a :
  1. Avance le pointeur
  2. Lit la ligne pointée par le pointeur
  3. Si pas de ligne pointée (l'avant-dernier fetch avait lu

Exemple

```
Declare
   cursor c1 is select . . .; art c1%rowtype;
   pasdetable exception;
   pragma exception_init(pasdetable, -942);
begin -- dès erreur passage à la section exception
   open c1; fetch c1 into art;
   while c1%found loop
       dbms_output.putline(c1.refart || ' ' || c1.prixht);
       fetch c1 into art;
   end loop; close c1;
exception
   when pasdetable
      then dbms_output.putline('table n'existe pas !');
   -- puis sortie du bloc après traitement de l'exception
   when others then dbms_output.putline('autre erreur !');
```

Erreurs Oracle (2)

- En cas d'erreur (ex : table inexistante) :
  - Sous SQL*Plus :
    - Annonce par message à l'écran
    - Induit une réaction du programmeur en mode interactif
  - Dans PL/SQL : gestion par une exception :
    - Détection de l'erreur par le gestionnaire d'exception
    - Induit une réaction du programme définie dans la section de traitement des exceptions

Exemple

```
Declare
   cursor c3 is select refart from article order by nom;
   art article.refart%type;
begin
   open c3;
   fetch c3 into art;
   while c3%found loop
      dbms_output.putline(art);
      fetch c3 into art;
   end loop;
   close c3;
end;
```

Exemple 4

- Afficher la référence et le prix de tous les articles

Erreurs Oracle

- On dit qu'il y a erreur quand un ordre SQL ne réussit pas à s'exécuter
- Exemple : nom de table incorrect dans un select
- Toutes les erreurs SQL (et PL/SQL) ont un code
- Principe :
  - Associer un nom à ce code
  - Traiter l'erreur correspondante comme une exception utilisateur

```
declare
   cursor c1 is select . . .; art c1%rowtype;
   depassement exception;
begin
   open c1; fetch c1 into art;
   while c1%found loop
       dbms_output.putline(c1.refart || ' ' || c1.prixht);
       if c1.prixht > 10 000
          then raise depassement;
       fetch c1 into art;
   end loop; close c1;
exception
   when depassement then dbms_output.putline('c'est trop');
end;
```

Exemple

```
Declare
   cursor c4 is
      select refart, prixht
      from article;
   art c4%rowtype; -- type dérivé (curseur)
begin
   open c4;
   fetch c4 into art;
   while c4%found loop
       dbms_output.putline(c1.refart || ' ' || c1.prixht);
       fetch c4 into art;
   end loop;
   close c4;
end;
```

Curseur avec paramètre

- Afficher les références des articles dont le prix est passé en paramètre

Gestion des exceptions

- But : associer un traitement à des « exceptions » survenues lors de l'exécution d'un bloc PL/SQL
  - Exceptions définies par l'utilisateur
  - Erreurs Oracle (ordres SQL non exécutés)
- Même principe qu'en Java, mais moins général
- Exceptions utilisateur :
  - Exemple :
    - Afficher tous les articles
    - Si l'un dépasse 10 000 euros, le dire et arrêter

Le langage

- Variables
- Structures de contrôle
- Curseurs
- Gestion des exceptions

Exemple

```
Declare
    cursor c5 (p article.prixht%type) is
        select refart
        from article
        where prixht = p;
    art article.refart%type;
begin
    open c5(7.77);
    fetch c5 into art;
    dbms_output.putline(art);
    . . .
end;
```