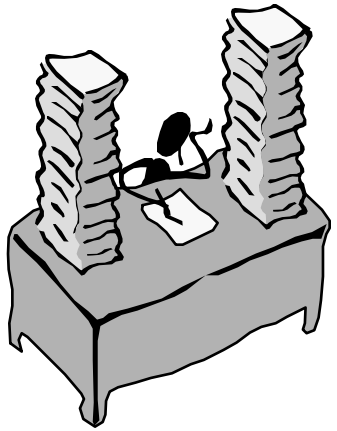
PL/SQL - 1

Objectifs du cours

- SQL - PL/SQL : quelle différence ?
- Environnement PL/SQL
- Bénéfices apportés par PL/SQL
- Caractéristiques de PL/SQL
- PL/SQL et ensuite ?



SQL-PL/SQL : quelle différence ?

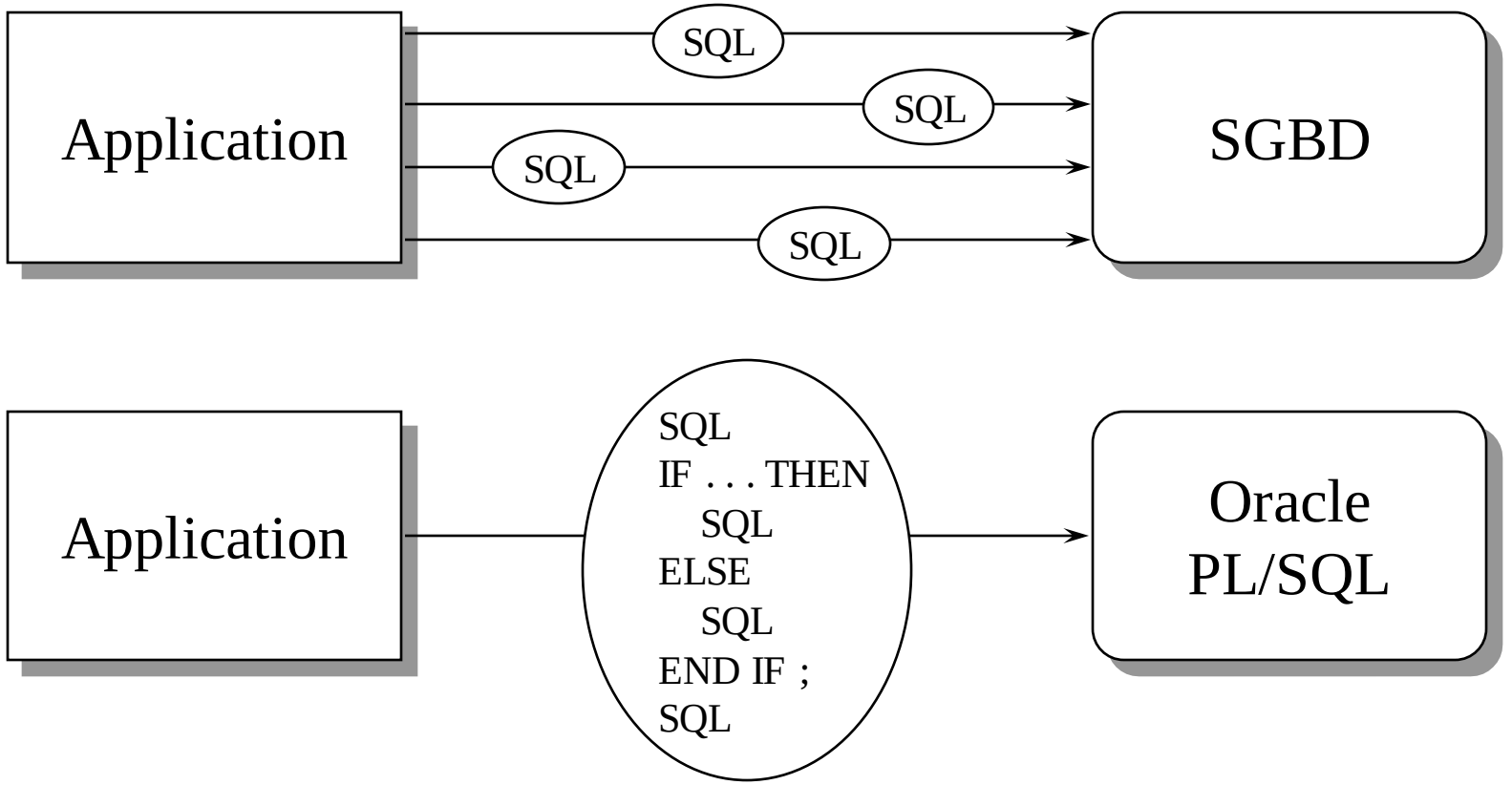
Clauses SQL ...

- Intérroger des données :
  SELECT
- Manipuler les données
  INSERT
  UPDATE
  DELETE
- Définir des données
  CREATE, ALTER, DROP, RENAME, TRUNCATE
- Contrôler les transactions
  COMMIT
  ROLLBACK
  SAVEPOINT

... Et PL/SQL ?

- PL/SQL est une extension de SQL qui possède *en plus* les caractériques d'un langage de programmation
- Les clauses SQL de gestion des donées sont intégrées dans le code procédural
- PL/SQL est donc un langage de programmation (Oracle) qui :
  - intégre directement les clauses SQL d'interrogation, de manipulation, et de définition des données
  - supporte l'encapsulation des données dans du code
  - supporte les exceptions

Environnement PL/SQL



Bénéfices de PL/SQL

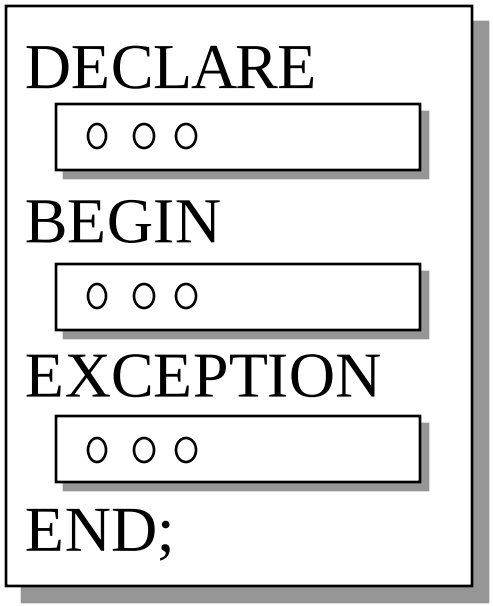
- PL/SQL regroupe les requêtes SQL *en un seul bloc* qui est envoyé au serveur *en un seul appel*
- PL/SQL améliore les performances (moins de communcations à travers le réseau)
- C'est un langage portable : il peut fonctionner sur toute plateforme supportant Oracle Server
- PL/SQL peut aussi coopérer avec les différents outils de développment d'application de Oracle Server (p.ex. Developer 2000)
- Permet de créer des bibliothèeques de code réutilisable

Environnement PL/SQL

Blocks Anonymes

Triggers d'application

Triggers de base de données



Procédures enregitrées

Procédures d'application

Packages

Développement de programmes MO-DU-LAIRE

Environnement PL/SQL

- *Block anonyme* :
  block PL/SQL imbriqué dans une application ou créé interactivement
- *Procédure enregistrée* :
  block nommé enregistré dans le serveur Oracle et qui peut être invoqué par son nom
- *Procédure d'application* :
  block nommé enregistré dans une application Developper 2000 ou dans une librairie partagée
- *Package* :
  module PL/SQL qui regroupe un ensemble de procédures
- *Trigger base de donnée* :
  block associé à une table et déclenché automatiquement lors d'une requête
- *Trigger d'application* :
  block PL/SQL associé à un événement d'application

Caractéristiques de PL/SQL

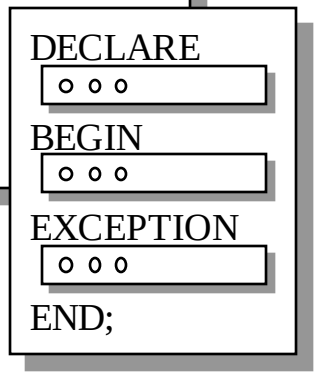
Tout ***block*** PL/SQL est composé de *3 sections*

1 DECLARE (*optionel*)
- variables, constantes, curseurs

2 BEGIN (*obligatoire*)
- clauses SQL
- instructions PL/SQL

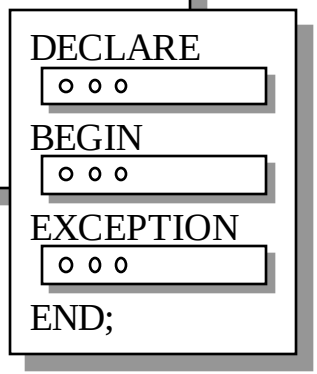
3 EXCEPTION (*optionel*)
- Actions à réaliser quand une exception est levée ou quand une terminaison anormale a lieu

END; (*obligatoire*)



Les blocks PL/SQL

```
DECLARE
        variable_v      VARCHAR2(5);
BEGIN
        SELECT          colonne_c
                INTO    variable_v
                FROM    table_t;
EXCEPTION
        WHEN exception_e THEN
        . . .
END;
```



Les différents types de blocks

Anonyme

```
[DECLARE]


BEGIN
. . .

[EXCEPTION]
. . .
END;
```

Procédure

```
PROCEDURE <nom>
IS

BEGIN
. . .

[EXCEPTION]
. . .
END;
```

Fonction

```
FUNCTION <nom>
RETURN <type>
IS
BEGIN
. . .
RETURN <valeur> ;
[EXCEPTION]
. . .
END;
```

Commentaires :

-- commentaires sur une ligne

/* commentaires sur
plusieurs lignes */

Exemple de block anonyme

```
SQL> serveroutput on ;
SQL> @C:\Mes Documents\monfichier.sql
BEGIN
    dbms_output.put_line('Hello World') ;
END ;
/
SQL > Hello World
```

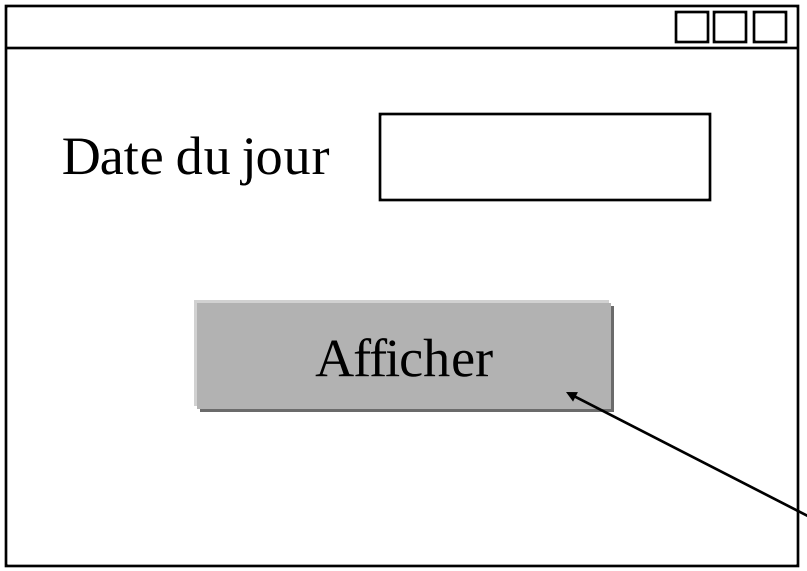
Execute le script contenu dans monfichier.sql

Contenu du fichier monfichier.sql

Block anonyme

Pour que le block soit exécuté dans SQLPlus

Autre exemple



Déclencheur d'application

```
BEGIN
    :block.champ :=
        :SYSTEM.SYSDATE;
END;
```

Les caractéristiques de PL/SQL

- *Gestion des variables*
  - variables locales
  - variables d'environnement extérieures à PL/SQL
- *Structures de contrôle*
  - traitements conditionnels
  - traitements répétitifs
- *Utilisation des curseurs*
  - définition
  - utilisation de curseurs explicites
  - attributs des curseurs
  - modification des valeurs d'un curseur
- *Gestion des erreurs*
  - anomalies programme utilisateur
  - erreurs Oracle
  - erreurs prédéfinies

Variables dans PL/SQL

Utiliser les variables pour :

- *L'enregistrement* temporaire des donnéees
  comme dans tout langage de programmation procédural
- La *manipulation* de données enregistrées
  afin de réalidser des caluls et autres manipulations sans accéder à la base de données
- La *réutilisabilité*
  une fois déclarée, une variable peut être utilisée plusieurs fois dans une application
- La facilité de *maintenance*
  en utilisant %TYPE et %ROWTYPE (voir plus tard), on déclare des variables ayant le même type que des attributs de tables; si la définition de la tble change, le type de la variable change de même.
  => réduction des coûts de maintenance

Types de variables

- *Scalaires*
  valeurs simples, les principaux types sont ceux de SQL
- *Composites*
  par ex. les enregistrements, permettent de définir des données structurées
- *Références*
  pointeurs désignant les éléments d'autres programmes
- LOBs (large objects)
  locateurs spécifiant l'meplacement de grands objets (par ex. images ou video)
- *Variables non-PL/SQL*
  variables déclarées en dehors de PL/SQL, par ex. dans un langage externe, dans des champs écran dans des applications Form, variables SQL*Plus

Déclaration de variables

- *Syntaxe* :

  <*nom*> [CONSTANT] <*type*> [NOT NULL]
  [:= | DEFAULT <*expression*>] ;

- *Exemple* :

```
DECLARE
    dateEmprunt_v       DATE ;
    noDept_v            NUMBER(2) NOT NULL := 10 ;
    lieu_v              VARCHAR2(13) := 'Paris' ;
    taux_c              CONSTANT NUMBER := 20 ;
```

- *Note* : constantes et variables NOT NULL doivent être immédiatement affectées

Conventions

- Déclarer au plus une variable par ligne
- Les noms de variables doivent :
  - commencer par une lettre
  - être composé d'au plus 30 caractères (lettres, nombres ou caractères spéciaux)
- Deux variables peuvent avoir le même nom dès lors qu'elles appartiennent à des blocs différents
- Ne pas utiliser le noms de tables ni les noms d'attributs
- Utiliser les suffixes :

  '_c' pour les constantes
  '_v' pour les variables
  '_g' pour les variables globales

Affectation de valeurs

- *Syntaxe* :

  <*nom_de_variable*> := <*expression*>

  ou

  SELECT . . .
  INTO <*nom_de_variable*>
  FROM . . . WHERE . . .

- *Initialisation des variables* :
  - Opérateur d'affectation ':='
    nom_v := 'Toto' ;
    dateEmprunt_v := '31-DEC-98' ;
  - DEFAULT
    chemin_g VARCHAR2(125) DEFAULT 'C:\progra~1\monAppli' ;
  - NOT NULL
    salaire_v NUMBER(4) NOT NULL := 0 ;

Affectations de valeurs

- Dans une affectation :

  <nom_de_variable> := *<expression>*

- <expression> peut être :
  - une constante
  - une variable
  - un calcul portant sur des constantes et de variables
- opérateurs de calcul :
  - opérateurs arithmétiques : + - * / ** (exponentation)
  - opérateur de concaténation : ||
  - opérateurs logiques :
    - comparaisons : < > = <= >= <>
    - connecteurs : AND OR NOT

Types scalaires prédéfinis

- CHAR [(<*taille_max*>)]
  chaines de caractères de longueur fixe (max 32767)
- VARCHAR2 (<*taille_max*>)
  LONG
  chaines de caractères de longueur variable (max 32767)
- LONG RAW
  identique à LONG, mais contenu non interprété
- RAW [(<*taille_max*>)]
  donnée binaire ou chaîne de caractères (max 32767)
- NCHAR
  NVARCHAR2
  identiques à CHAR et VARCHAR2; gèrent les caractères nationaux
- STRING
  VARCHAR
  sous types de VARCHAR2, assurent la compatibilité ANSI/ISO

Types scalaires prédéfinis

- NUMBER [(<p>, <s>)]
  nombres réels, p chiffres en tout, s après la virgule
- BINARY_INTEGER
  type de base pour les entiers de -2.10^9 à 2.10^9
- PLS_INTEGER
  prennet moins de place et sont plus rapides que les valeurs de type number et binary_integer
- DEC, DECIMAL, NUMERIC, DOUBLE PRECISION, FLOAT, REAL INTEGER, INT, SMALLINT
  sous types de NUMBER, pour la compatibilité ANSI/ISO
- NATURAL, NATURALN, POSITIVE, POSITIVEN, SIGNTYPE
  sous types de BINARY_INTEGER, pour la compatibilité ANSI/ISO
- DATE
- BOOLEAN
  trois valeurs possibles : TRUE, FALSE et NULL

Particularités des Dates

- Le format par défaut des dates est ‘DD-MON-YY’
  par ex : insert into toto values(99, '31-may-98');
- On peut comparer deux dates
- La date courante peut être accédée par la variable d’environnement SYSDATE
- On peut soustraire deux dates, cela donne un nombre de jours (sous forme de *number*)
- On peut soustraire ou ajouter un nombre à une date, le nombre est interprété comme un nombre de jours
- On peut convertir une date en chaine de caractère avec la fonction TO_CHAR(<date>, '<format>')
  et inversement avec la fonction TO_DATE(<string>, '<format>')
- (voir les divers formats de date dans l’aide ou sur Internet)

Types définis par l'utilisateur

- *Syntaxe* :

```
TYPE  <nom_type>  IS  RECORD  (
      <nomchamps>  <type>  [ [NOT NULL]
                           [ :=  |  DEFAULT  <expression> ],
      . . . ) ;
```

- *Exemple* :

```
SQL>  DECLARE
      TYPE client_t IS RECORD (
            numero NUMBER(4),
            nom            CHAR(20),
            adresse        CHAR(20) ) ;
      client1_v  client_t ;
      BEGIN
      client1_v.numero := 2516 ;
      END ;
      /
```

L'attribut %TYPE

- A employer pour déclarer une variable en réutilisant :
  - la définition d'un attribut de table
  - la définition d'une autre variable déclarée précédemment

- *Exemple* :

```
. . .
nomEmploye_v        employe.nomEmp%TYPE ;
solde_v             NUMBER(7, 2) ;
soldeMinimal_v      solde_v%TYPE := -2000 ;
unEmploye_v         Employe%ROWTYPE ;
```

- *Note* :
  - %ROWTYPE : idem que %TYPE mais pour définir une variable de type enregistrement dont les champs correspondent à tous les attributs d'une table
  - les contraintes NOT NULL de la définition des attributs de tables ne sont pas réutilisées avec %TYPE

Exercices

- Déterminer lesquelles de ces déclarations sont légales ou pas, indiquer pourquoi :

```
DECLARE
    id_v                NUMBER(4) ;

    x_v, y_v, z_v       VARCHAR2(10) ;

    date_Naissance_v    DATE NOT NULL ;

    stock_v             BOOLEAN := 1 ;
```

Exercices

- Déterminer lesquelles de ces affectations sont légales ou pas, indiquer pourquoi :

  joursRestants_v := dateFin - SYSDATE ;

  emetteur_v := USER || ‘: ‘ || TO_CHAR(noDept_v) ;

  total_v := 100Frs + 124Frs ;

  flag_v := TRUE ;

  n1_v := n2_v > (2 * n3_v) ;

  val_v := NULL ;