Programmation & Administration des Bases de Données

Contenu

- Suite du S1 : basé sur SQL

- Bases de la programmation d'applications utilisant une bases de données SQL
  - Programmation PL/SQL
  - Programmation JAVA/JDBC

- Eléments d'administration : vues et contrôles d'accès

Partie 1 : PL/SQL

Extension de SQL permettant de programmer des enchainements de requêtes SQL

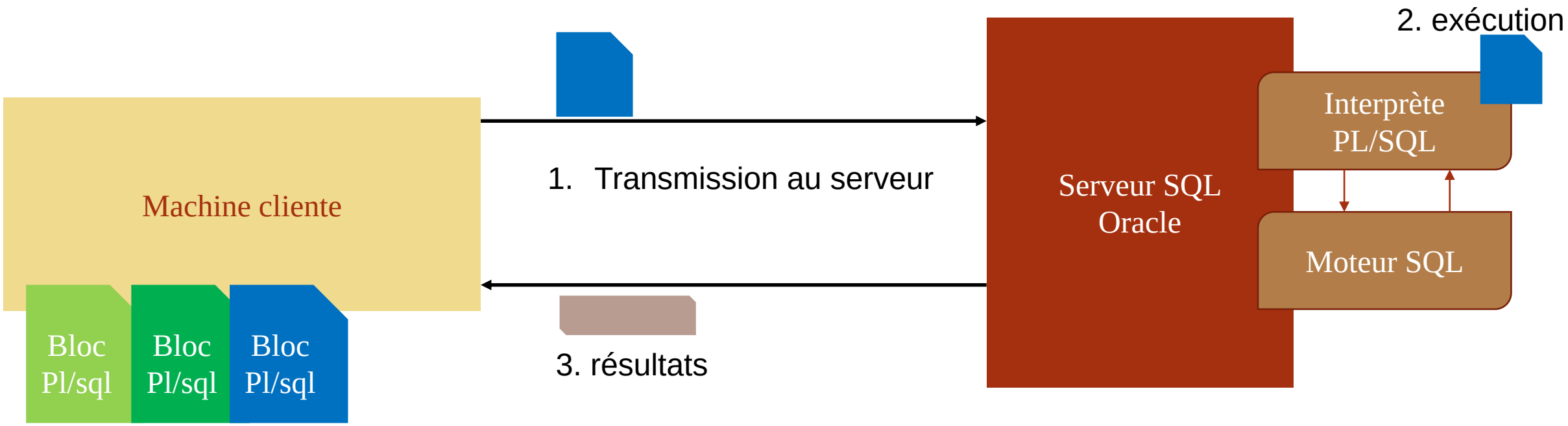
PL/SQL

- PL/SQL est une **extension de SQL** introduisant :
  - Des variables
  - Des structures de contrôles (for, while, if …)
  - Des procédures et des fonctions
  - De la gestion d'erreurs (exception)
- **Utilisation** : programmer des enchainements de requêtes

- 2 modes de **fonctionnement** :
  - **Scripts** stockés sur le client
  - **Procédures et fonctions stockées** sur le serveur

Scripts PL/SQL

- Programmes (bloc) stockés sur la machine cliente, comportant 1 ou plusieurs requêtes SQL
- Transmis en 1 fois au serveur qui exécute l'ensemble des instructions
- Retourne 1 réponse au client



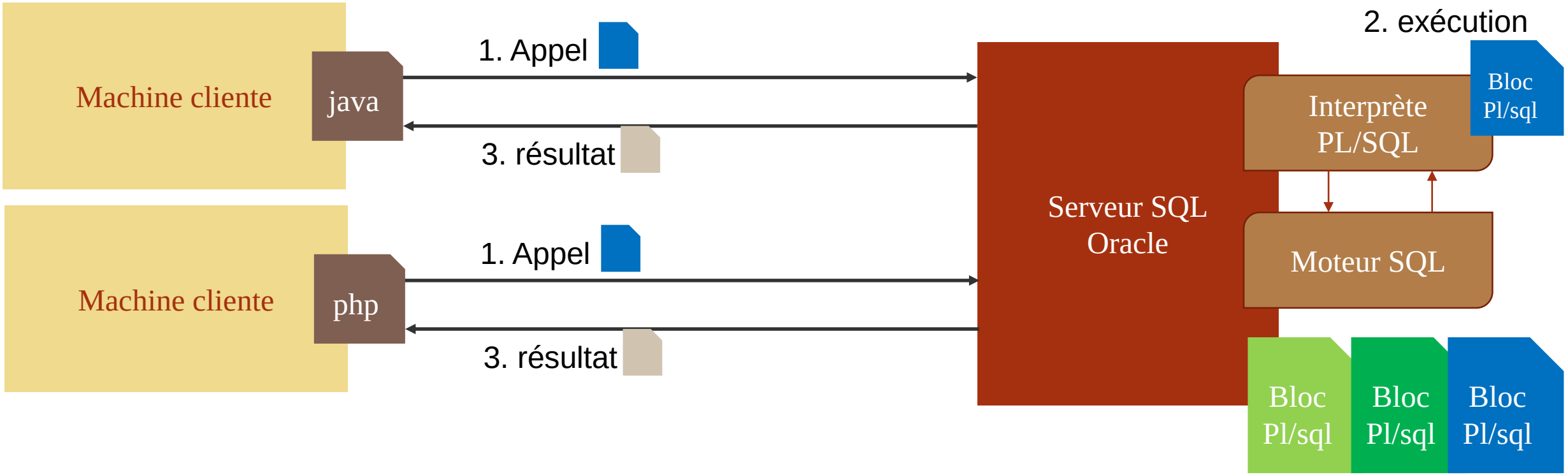
Utilisation des scripts PL/SQL

- Enchainer des requêtes SQL de toute nature (select, insert, update …) de manière automatisée afin de **réaliser** :
- des programmes exécutés de manière **ponctuelle**
- des opérations de **maintenance** / **surveillance** de la base de données
  - contrôler les données
  - transformer des données
- des opérations **d'administration** de la base
- des **mises au point** / essais / tests de programmes

- **Pas prévu pour développer des applications**

Procédures et fonctions stockées

- Blocs pl/sql compilés et stockés sur le serveur
- Peuvent être appelés par différents clients depuis différents langage de programmation



Utilisation des procédures/fonctions stockées

- Programmation de modules de manipulation de la BD utilisés par des applications
- Une procédure/fonction stockée peut être utilisée par un programme s'exécutant sur différentes machines clientes
- Une procédure/fonction stockée peut être utilisée par des programmes et applications différentes

Structure d’un bloc PL/SQL

```
DECLARE

Déclaration de variables, curseurs, exceptions

BEGIN

Corps du bloc PL/SQL
   instructions SQL + affectations + structures de
   contrôles

[EXCEPTION]
  [gestionnaire d’exceptions]

END ;
```

Exemple

```
DECLARE
  min_rang number(3) ;
  v_nomski  varchar2(30);

BEGIN
 v_nomski := &nomskieur;
 Select min(rang) into min_rang from classement
   where nomski = v_nomski ;
 If min_rang < 4 then
    Insert into resultat_podium values (v_nomski , min_rang) ;
 else
    Insert into hors_podium values (v_nomski , min_rang) ;
 end if ;
END ;
```

- les instructions PL/SQL sont séparées par un point virgule ;

- &nomskieur est une variable/paramètre dont la valeur est saisie lors de l'exécution

Déclaration des variables

- Toutes les variables doivent être **déclarées** et **typées**

- Elles peuvent être **initialisées** lors de la déclaration

- Types : **les types SQL** +
  - **BOOLEAN :** TRUE, FALSE, NULL
  - **BINARY_INTEGER :** les entiers signés de –2147483647 à 2147483647.
  - **NATURAL :** les entiers de 0 à 2147483647.
  - **POSITIVE :** les entiers de 1 à 2147483647.
  - *table.colonne*%**TYPE :** réutilisation du type d'une colonne existante de la base de données

Exemples

```
DECLARE
 v_fax_num char(12);
 v_total number(10,2);
 v_date_naiss DATE ;
 v_id_act acteur.idacteur%TYPE ;

 /* déclaration avec initialisation */

 v_count INT := 1337 ;
 v_mess VARCHAR (256) := 'les carottes sont cuites, je répète..';

BEGIN
END;
```

Instructions de base

- **Affectation** : l'opérateur `:=` permet d'affecter une valeur à une variable
- **Requête SQL** : toute requête SQL peut être utilisée dans un bloc PL/SQL
- Une **variable** PL/SQL peut-être utilisée dans une requête partout où l'on peut mettre une valeur ou une expression (`where`, `VALUES` …)
- **Toute valeur retournée par une requête select doit obligatoirement être stockée dans une variable PL/SQL, à condition que la requête retourne une seule et unique ligne**

Exemple

```
DECLARE
 v_nomski varchar(12);
 v_nomstat skieur.nomstat%type;

BEGIN
 v_nomski := ‘hischer’ ;
 select nomstat into v_nomstat from skieur
 where nomski = v_nomski ;

 update station set nbsk = nbsk+1
 where nomstat = v_nomstat ;

END;
```

Entrées/sorties : affichage de valeurs

- PL/SQL fournit un ensemble de commandes pour l'affichage de valeurs à partir d'un bloc PL/SQL

- La documentation Oracle indique que cet affichage est à utiliser pour des raisons de mise au point et de recherche d'erreurs

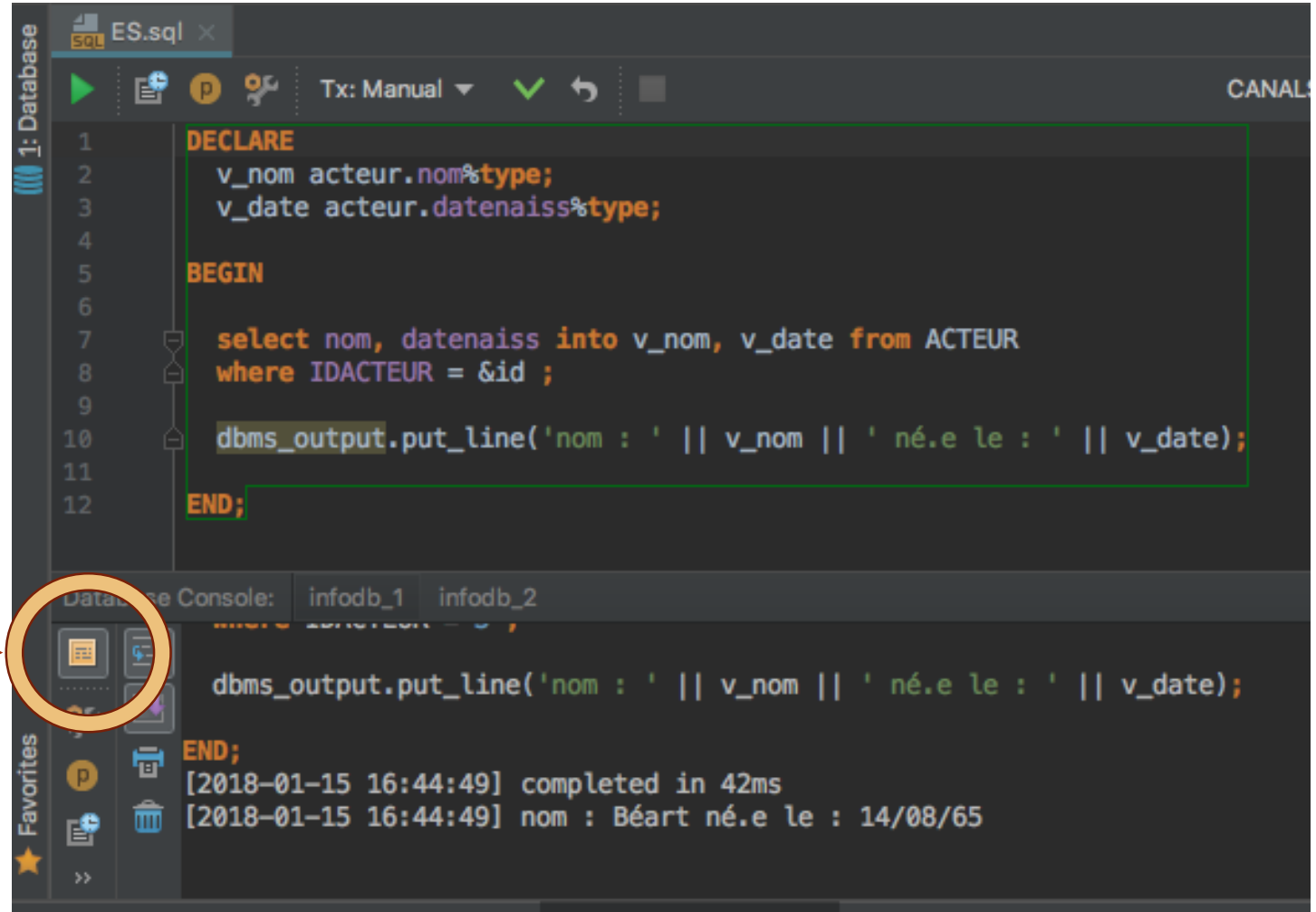
- Syntaxe : `dbms_output.put_line( <expression_chaine> )`

Exemple :

```
dbms_output.put_line('nom du skieur'|| v_nomski);
```

- **ATTENTION** : l'utilisation effective des fonctionnalités d'affichage demande toujours une configuration particulière du client utilisé, qui doit accepter de recevoir des données particulières du serveur et les afficher

- **Sql*plus** : positionner les variables d'environnement SERVEROUTPUT et ECHO sur ON
  - `SET serveroutput ON`
- **Datagrip :** activer le bouton « Enable SYS.dbms_output » dans la partie output de la console



Entrées/sorties : saisie de valeurs

- Un script PL/SQL peut contenir des ***variables de substitution***, c-a-d des variables qui sont substituées par une valeur juste avant l'envoi du script au serveur
- Cette substitution est réalisée par le client utilisé pour dialoguer avec le serveur
- La syntaxe d'une variable de substitution dépend du client utilisé

Syntaxe sql*plus

```
V_nom := '&nom_du_skieur' ;
```

Syntaxe datagrip (configurable)

```
V_nom :=  &nom_du_skieur ;
V_stat := :nom_station ;
```

Les structures de contrôle

les conditionnelles

```
IF condition THEN
        instruction;
 …
        instructions;
[ ELSE
        instruction;
 …
        instructions;
]
END IF;
```

- **Condition** : utilise les opérateurs SQL (>,<,>=,<=,=,<>, !=, IS [NOT] NULL) avec les variables PL/SQL.
- Une instruction **IF** peut contenir plusieurs clauses **ELSIF**, mais une seule clause **ELSE**.
- Les clauses **ELSIF** et **ELSE** sont optionnelles.

Itérations

- L'instruction de base est la boucle : **`LOOP …. END LOOP ;`** qui réalise une itération infinie
- On peut programmer une sortie de la boucle sur une condition avec l'instruction **`: EXIT WHEN <`*cond*`>`**

```
i:=1;

LOOP
  DBMS_OUTPUT.put_line( i );
  i:=i+1;

  EXIT  WHEN i=10 ;

END LOOP;
```

- **Itération WHILE …. LOOP**
  - Condition de sortie évaluée avant chaque itération

```
i:= 1 ;
WHILE i < 10 LOOP
  DBMS_OUTPUT.put_line( i );
  i:=i+1;
END LOOP;
```

- **Itération FOR …. LOOP**
  - Condition de sortie gérée avec un compteur

```
FOR i IN 1 .. 9  LOOP

   DBMS_OUTPUT.put_line( i );

END LOOP ;
```

Instruction NULL

Permet dans certaines conditions d'indiquer qu'aucune action n'est à entreprendre (IF……THEN….ELSE……)

```
IF i<100 THEN
  i:=i+1;
ELSE NULL ;
END IF ;
```

Les variables composées RECORD

- Une variable composée est une variable structurée agrégeant plusieurs champs éventuellement de types différent
- Exemple : une **personne** =
  - nom
  - Prénom
  - Email
  - Date de naissance
  - No tel
  - Id skype
- Pour utiliser une variable composée RECORD :
  1. Déclarer un **TYPE**
  2. Déclarer une **variable** de ce type

Déclaration d'un type et d'une variable

```
TYPE <nom_type> IS RECORD
( <champ1> <type> [NOT NULL] [VALEUR PAR DEFAUT] ,
  <champ2> <type> [NOT NULL] [VALEUR PAR DEFAUT] ,
  …
 ) ;
```

```
DECLARE
TYPE personne IS RECORD (
  nom VARCHAR(64) NOT NULL,
  prenom VARCHAR(64),
  date_naiss DATE,
  email VARCHAR(128),
  tel VARCHAR(16)
);
v_pers personne ;
```

Utilisation d'une variable de type Record

```
DECLARE
TYPE personne IS RECORD ( … );
v_pers personne ;

BEGIN
 v_pers.nom := 'marcel' ;

 SELECT prenom, mail INTO v_pers.prenom, v_pers.email
   FROM client WHERE nom = v_pers.nom ;

 SELECT nom, prenom, datenaiss, mail, notel
 INTO v_pers
 FROM client WHERE nom = 'jules' ;

END ;
```

RECORD et LIGNES de Table

- Il est possible de déclarer une variable en lui donnant un type RECORD correspondant à la structure des lignes d'une table.
- Il est alors inutile de déclarer le TYPE RECORD

```
DECLARE
v_client client%rowtype ;

BEGIN

 SELECT * INTO v_client
 FROM client WHERE nom = 'samia' ;

END ;
```

Les curseurs

- Le **curseur** est un mécanisme permettant de traiter les requêtes SELECT générant plusieurs lignes résultat, même lorsque ce résultat est de grande taille,

- Un curseur est une structure de données associée au **résultat d'une requête** SQL

- Un curseur permet de charger et parcourir le résultat **ligne par ligne** et d'interagir avec chaque ligne :
  - charger la ligne dans une variable PL/SQL
  - modifier la ligne (update)
  - supprimer la ligne

Principe d’utilisation d’un curseur

1. **Déclaration du curseur**
   - on associe au nom du curseur une instruction SELECT
2. **Ouverture du curseur**
   - la requête associée au curseur est exécutée et les lignes résultat sont sélectionnées
3. **Parcours des lignes du curseur**
   - Les lignes du curseur sont chargées une par une par le programme PL/SQL
4. **Fermeture du curseur**
   - Les ressources allouées au curseur par le système sont libérées

Déclaration d'un curseur

CURSOR *<nom_curseur>* IS *select …* ;

```
DECLARE
Cursor ski_descente is
        Select nomski , nomstat
         from skieur
         where specialite='descente' ;
 …
```

Ouverture d'un curseur

- L'ouverture d'un curseur déclenche l'exécution de la requête associée et la création des ressources (mémoire) nécessaires

OPEN *<nom_curseur>*;

```
DECLARE
Cursor ski_descente is
        Select nomski , nomstat
         from skieur
         where specialite='descente' ;
BEGIN

OPEN ski_descente ;
```

Chargement d'une ligne

- FETCH charge la ligne courante du résultat et positionne le curseur sur la ligne suivante. Cette ligne doit obligatoirement être stockée dans des variables PL/SQL, ou une variable RECORD

```
FETCH <nom_curseur> INTO {<v1>,..,<vn> | <var_record>}
```

```
DECLARE
Cursor ski_descente is  Select nomski , nomstat from skieur
         where specialite='descente' ;
V_nom  skieur.nomski%type ;
V_stat skieur.nomstat%type ;

BEGIN
OPEN ski_descente
FETCH ski_descente into v_nom,v_stat ;
end ;
```

Fermeture d'un curseur

- Un curseur doit être fermé afin de libérer les ressources attibuées lors de l'ouverture.

```
DECLARE
Cursor ski_descente is
        Select nomski , nomstat
         from skieur
         where specialite='descente' ;
BEGIN
OPEN ski_descente ;
/* fetch ...
 */

CLOSE ski_descente ;
```

Contrôle d'un curseur

- On contrôle un curseur grâce à des **attributs** dont la valeur indique **l'état** du curseur :

Attribut	Type	Objet	exemple
%found	booléen	Vrai si fetch a retourné une ligne	Ski_descente%found
%notfound	booléen	Vrai si fetch n'a pas retourné de ligne	Ski_descente%notfound
%isopen	booléen	Vrai si le curseur est ouvert	Ski_descente%isopen
%rowcount	entier	Nombre de lignes traitées par le curseur	Ski_descente%rowcount

Parcourir les lignes d'un curseur

- On utilise une boucle pour itérer sur les lignes d'un curseur :
  - L'instruction FETCH charge une ligne et positionne le curseur sur la suivante
  - Les attributs `%found` et `%notfound` permettent de déterminer si le curseur est épuisé

- Plusieurs possibilités
  - `LOOP … EXIT WHEN` : permet de contrôler la sortie de la boucle sur une condition quelconque
  - `WHILE <cond> LOOP` : idem, mais impose le chargement de la 1ère ligne avant la boucle
  - La boucle spécialisée `FOR <var> IN <curseur> LOOP` : simplifie le parcours lorsque l'on souhaite parcourir toutes les lignes

```
DECLARE
v_s  skieur%rowtype ;
Cursor ski_descente is Select * from skieur where specialite='descente' ;
BEGIN
OPEN ski_descente ;
LOOP
  FETCH ski_descente INTO v_s ;
  EXIT WHEN ski_descente%notfound;       -- condition de sortie
                                                    -- en fin de curseur

  dbms_output.put_line( v_s.nom || 'de : ' || v_s.nomstat) ;

  EXIT WHEN ski_descente%rowcount > 20 ; -- condition de sortie
                                                -- (si utile)
END LOOP ;

dbms_output.put_line('nb : ' || ski_descente%rowcount ) ;
CLOSE ski_descente ;
END;
```

```
DECLARE
v_s  skieur%rowtype ;
Cursor ski_descente is Select * from skieur where specialite='descente' ;

BEGIN
OPEN ski_descente ;
FETCH ski_descente INTO v_s ;
WHILE ski_descente%found   LOOP
  dbms_output.put_line( v_s.nom || 'de : ' || v_s.nomstat) ;

  EXIT WHEN ski_descente%rowcount > 20 ;   -- condition de sortie
                                           -- (si utile)

  FETCH ski_descente into v_s ;
END LOOP ;

dbms_output.put_line('nb : ' || ski_descente%rowcount ) ;
CLOSE ski_descente ;
END;
```

La boucle de curseur FOR … LOOP

- Boucle spécialisée pour le parcours de toutes les lignes d'un curseur
- Regroupe en une seule instruction les 3 instructions : open, fetch, close
- La variable record qui récupère les données du curseur n’a pas besoin d’être déclarée et possède une structure de même type que celle du curseur.

```
DECLARE
Cursor nom_curs is
       ordre_select

BEGIN
  FOR rec_curs IN nom_curs LOOP
       /* traitement lignes curseur*/
  END LOOP;
```

```
DECLARE
Cursor ski_descente is
    Select * from skieur where specialite=’descente’ ;

BEGIN

FOR v_s IN ski_descente  LOOP
  dbms_output.put_line( v_s.nom || 'de : ' || v_s.nomstat) ;

  EXIT WHEN ski_descente%rowcount > 20 ;   -- condition de sortie
                                           -- (si utile)

END LOOP ;

END;
```

Mise à jour de données dans un curseur

- Il est possible de faire des mises à jour de données (UPDATE, DELETE) lors du parcours d'un curseur
- **Condition :** le curseur doit être déclaré « ***FOR UPDATE*** »

```
DECLARE
cursor c_skieurs is  select *  from skieur FOR UPDATE ;
cursor c_stations is select * from station FOR UPDATE OF capacite_accueil ;
```

- On peut alors utiliser une condition where dédiée permettant de sélectionner la ligne courante du curseur

```
update station set capacite_accueil = capacité_accueil + 1000
   WHERE CURRENT OF c_stations
```

```
DECLARE
cursor ski_descente is
    select nomski, age, categorie from skieur
    where specialite='descente'
    FOR UPDATE OF categorie ;

BEGIN

FOR skieur_row in ski_descente LOOP
  IF skieur_row.age > 32 THEN
    UPDATE skieur set categorie = 'veteran'
    WHERE current of ski_descente ;
  END IF ;
END LOOP;

END ;
```

Curseurs paramétrés

- Un curseur peut recevoir un paramètre qui doit être indiqué lors de la déclaration du curseur,
- Ce paramètre peut alors être utilisé dans la condition where de la requête associée
- On transmet 1 argument au curseur lors de son ouverture
- Intérêt : le même curseur peut être utilisé pour différentes requêtes

```
CURSOR nom_curseur (param1 type_param1[ :=val_defaut1], ……
                    paramN type_paramN[ :=val_defautN])
IS instruction_select ;


OPEN nom_curseur ( argt1 ,  …, argtN ) ;
```

```
DECLARE
V_specialite skieur.specialite%TYPE ;
cursor ski_specialite (p_special varchar2) is
    select nomski,nomstat
    from skieur
    where specialite = p_special ;

BEGIN
V_specialite := :specialite_du_skieur ;
FOR skieurs in ski_specialite( V_specialite ) LOOP
   dbms_output.put_line(skieurs.nomski || skieurs.nomstat);
end loop ;
END ;
```