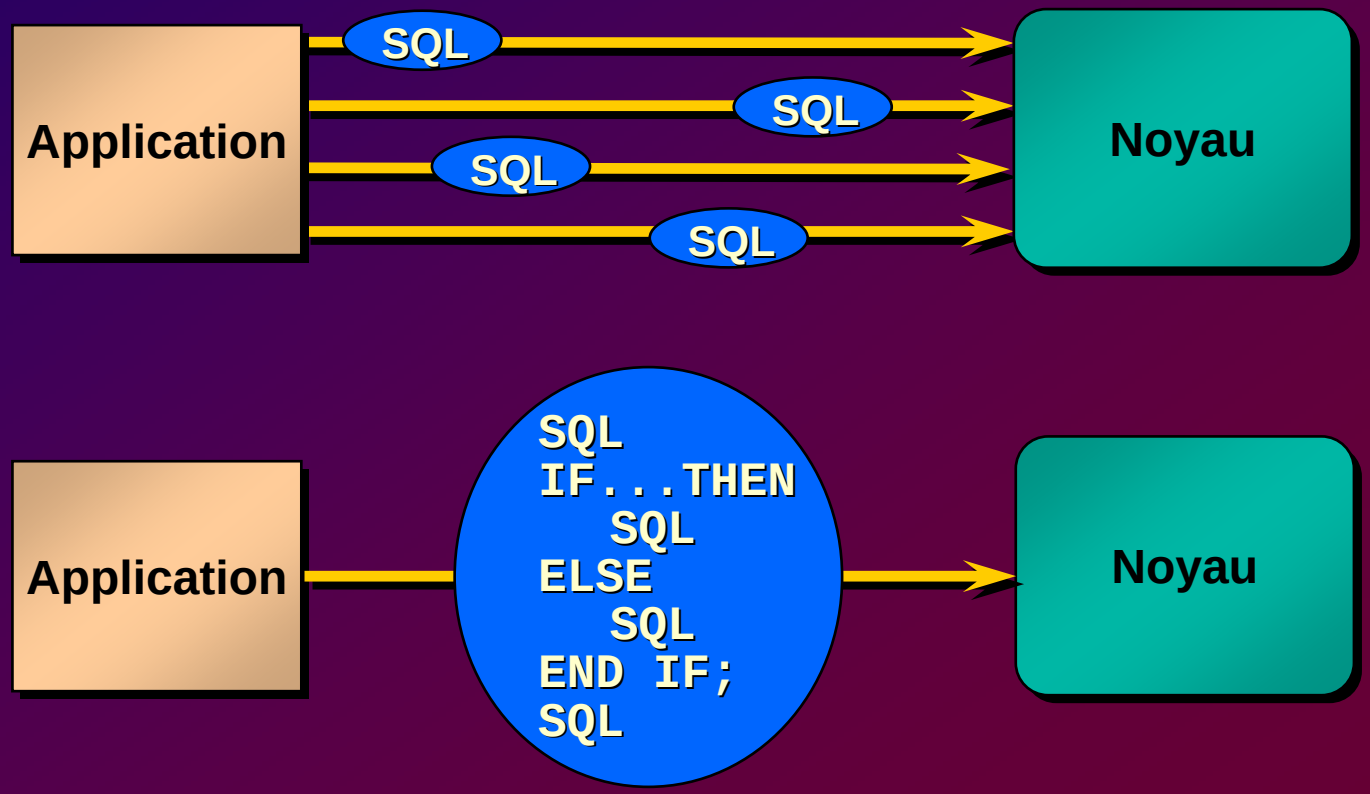
PL / SQL

Généralités - PL/SQL

- PL/SQL est une extension procédurale du langage SQL.
- Possibilité d ’inclure des requêtes et des ordres de manipulation des données à l ’intérieur d ’une structure algorithmique.

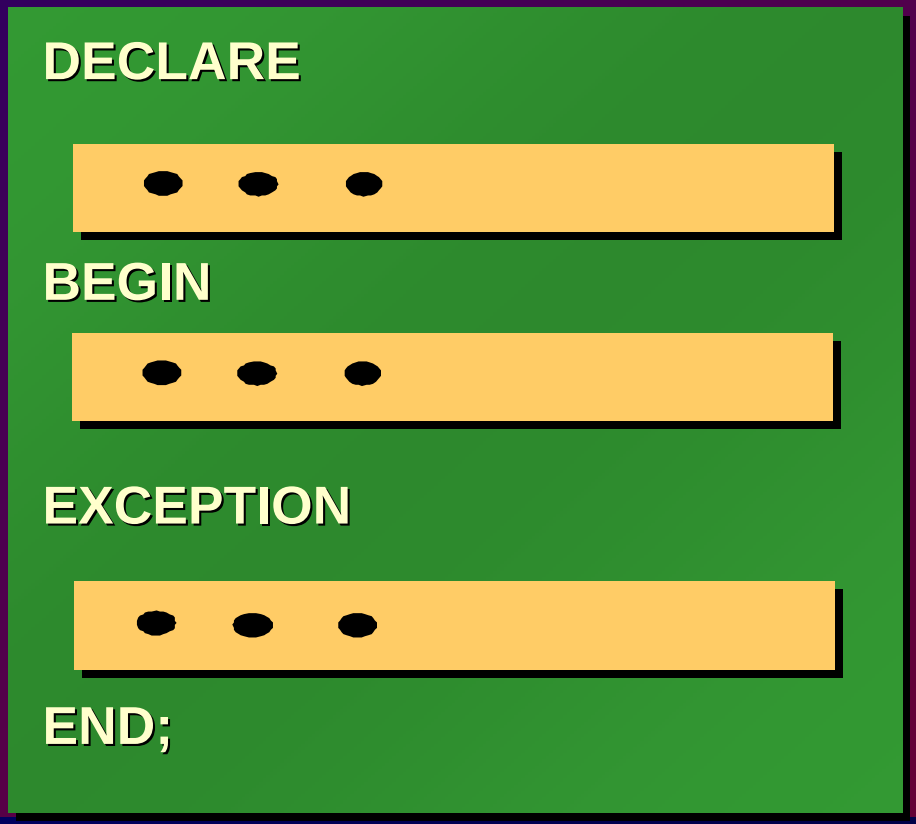
Intérêts du PL/SQL

Amélioration des Performances



Intérêts du PL/SQL

Développement MODULAIRE



Intérêts du PL/SQL

- Portabilité.
- Utilisation de variables.
- Structures de contrôle.
- Gestion des erreurs

PL/SQL Structure de BLOC

- DECLARE – Facultatif
  - Variables, curseurs, exceptions utilisateur
- BEGIN – Obligatoire
  - Ordres SQL
  - Instructions PL/SQL
- EXCEPTION – Facultatif
  - Actions à exécuter en cas d ’erreur
- END; – Obligatoire

Structure BLOC PL/SQL

```
DECLARE
  v_variable  VARCHAR2(5);
BEGIN
  SELECT      nom-colonne
    INTO      v_variable
    FROM      nom-table_;
EXCEPTION
  WHEN exception_nom-erreur THE
  ...
END;
```

Types de BLOC

Anonyme

```
[DECLARE]

BEGIN
  --statements

[EXCEPTION]

END;
```

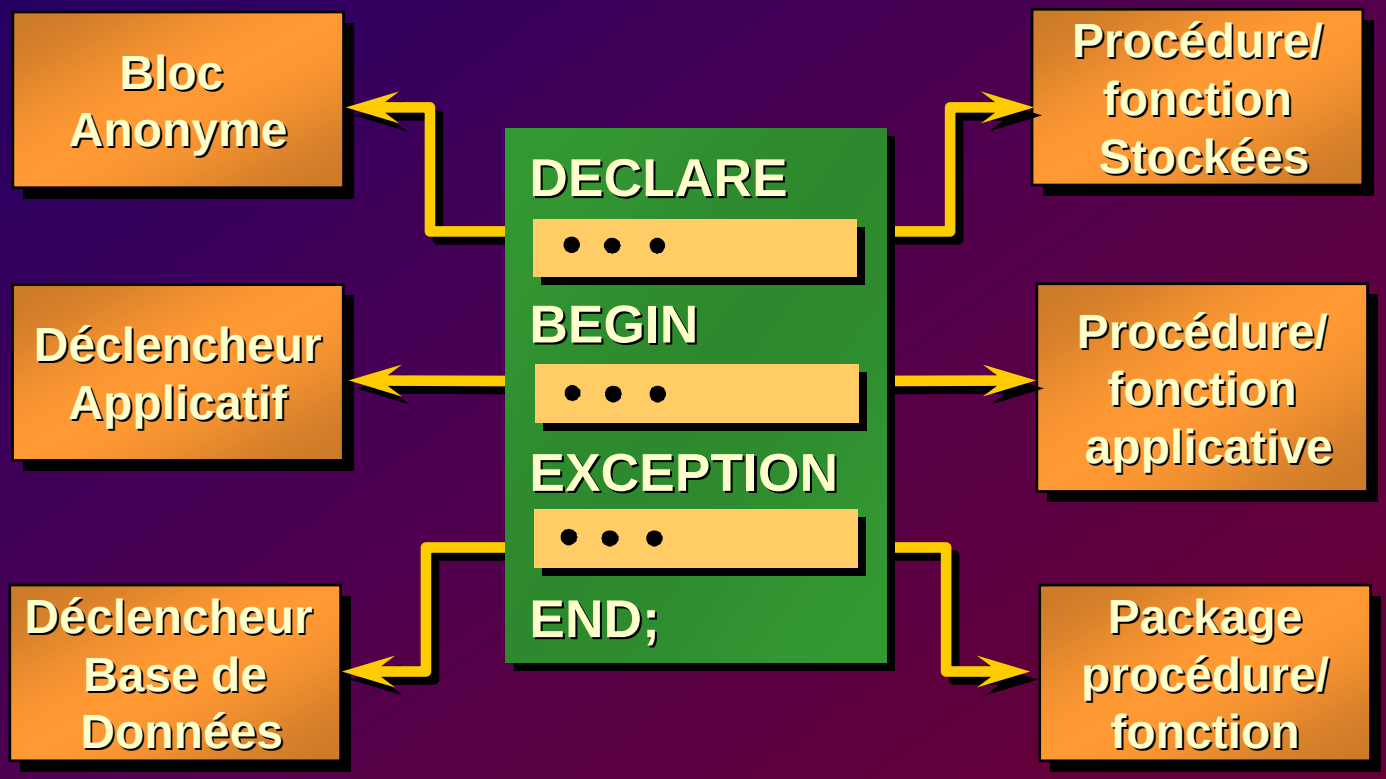
Procédure

```
PROCEDURE name
IS

BEGIN
  --statements

[EXCEPTION]

END;
```

Fonction

```
FUNCTION name
RETURN datatype
IS
BEGIN
  --statements
  RETURN value;
[EXCEPTION]

END;
```

Utilisation d ’un BLOC



Variables

Utilisation des Variables en PL/SQL

- Déclaration dans la section DECLARE.
- Affectation de valeurs dans la section exécution (ou à la déclaration).
- Passage de valeurs pour les procédures et fonctions.

Types de Variables

- Variables PL/SQL :
  - Scalaire
  - Structurée
  - Référence
  - LOB (Grands Objets - Large Object)
- Variables de liens (Non PL/SQL)

Declaration des Variables PL/SQL

Syntaxe

```
Nom_variable [CONSTANT] type-donnée [NOT NULL]
     [{:= | DEFAULT} expression];
```

Exemples

```
Declare
  v_hiredate      DATE;
  v_deptno        NUMBER(2) NOT NULL := 10;
  v_location      VARCHAR2(13) := 'Atlanta';
  c_comm          CONSTANT NUMBER := 1400;
```

Affectation de valeur

Syntaxe

```
Nom_variable := expr;
```

Exemples

Affecter une date d'embauche.

```
v_hiredate :=  To_DATE('03-JAN-2000','DD-MON-99');
```

Affecter un nom d 'employé.

```
v_ename := 'Maduro';
```

Initialisation d'une variable

Possible dans la section DECLARE par :

- **Opérateur d'affectation (:=)**
- **DEFAULT valeur**
- **NOT NULL**

***Exemples*:**

```
v_mgr NUMBER(4) DEFAULT 7839
v_loc VARCHAR2(50) NOT NULL := 'PARIS'
```

Type Scalaire

- VARCHAR2 (*longueur-maximale*)
  VARCHAR
- NUMBER [(*précision, décimales*)]
- DATE
- CHAR [(*longueur-maximale*)]
- LONG
- LONG RAW
- BOOLEAN
- BINARY_INTEGER
- PLS_INTEGER

Déclarations de type scalaire

Exemples

```
v_job          VARCHAR2(9);
v_count        BINARY_INTEGER := 0;
v_total_sal    NUMBER(9,2) := 0;
v_orderdate    DATE := SYSDATE + 7;
c_tax_rate     CONSTANT NUMBER(3,2) := 8.25;
v_valid        BOOLEAN NOT NULL := TRUE;
```

Déclaration de type par référence

- Déclarer une variable par référence à :
  - Une colonne de table.
  - Une autre variable déclarée.
- Utilisation du suffixe %TYPE après :
  - Nom-table.nom-colonne.
  - Nom-variable

Déclaration de type par référence

Exemples

```
...
  v_ename                emp.ename%TYPE;
  v_balance              NUMBER(7,2);
  v_min_balance          v_balance%TYPE := 10;
...
```

Déclaration de Type Booléen

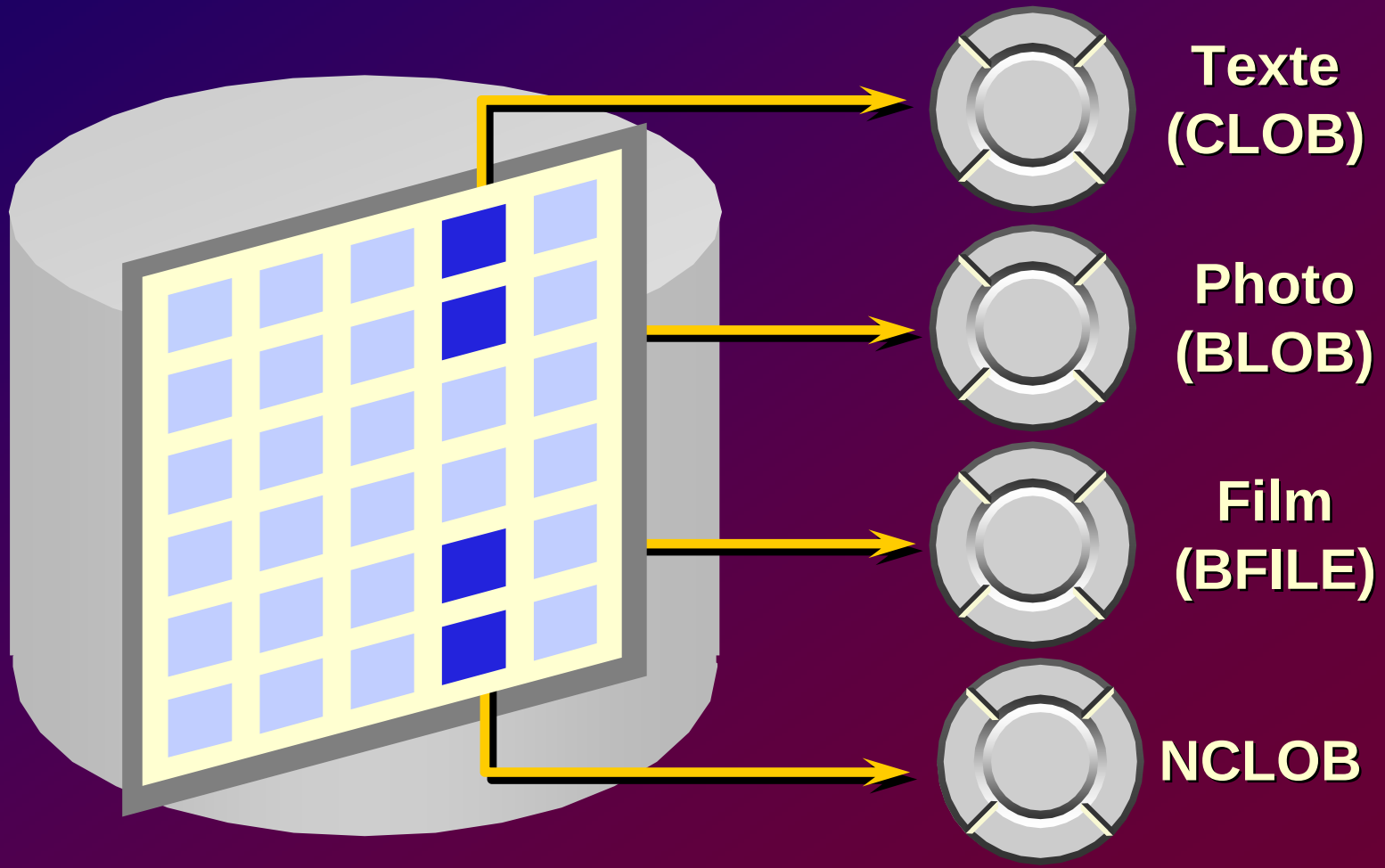
- Valeurs TRUE, FALSE ou NULL.
- Opérateurs AND, OR, et NOT.
- Possibilité d'obtenir une valeur booléenne à partir d'une expression arithmétique ou chaîne de caractères.

```
v_comm_sal BOOLEAN := (v_sal < v_comm);
```

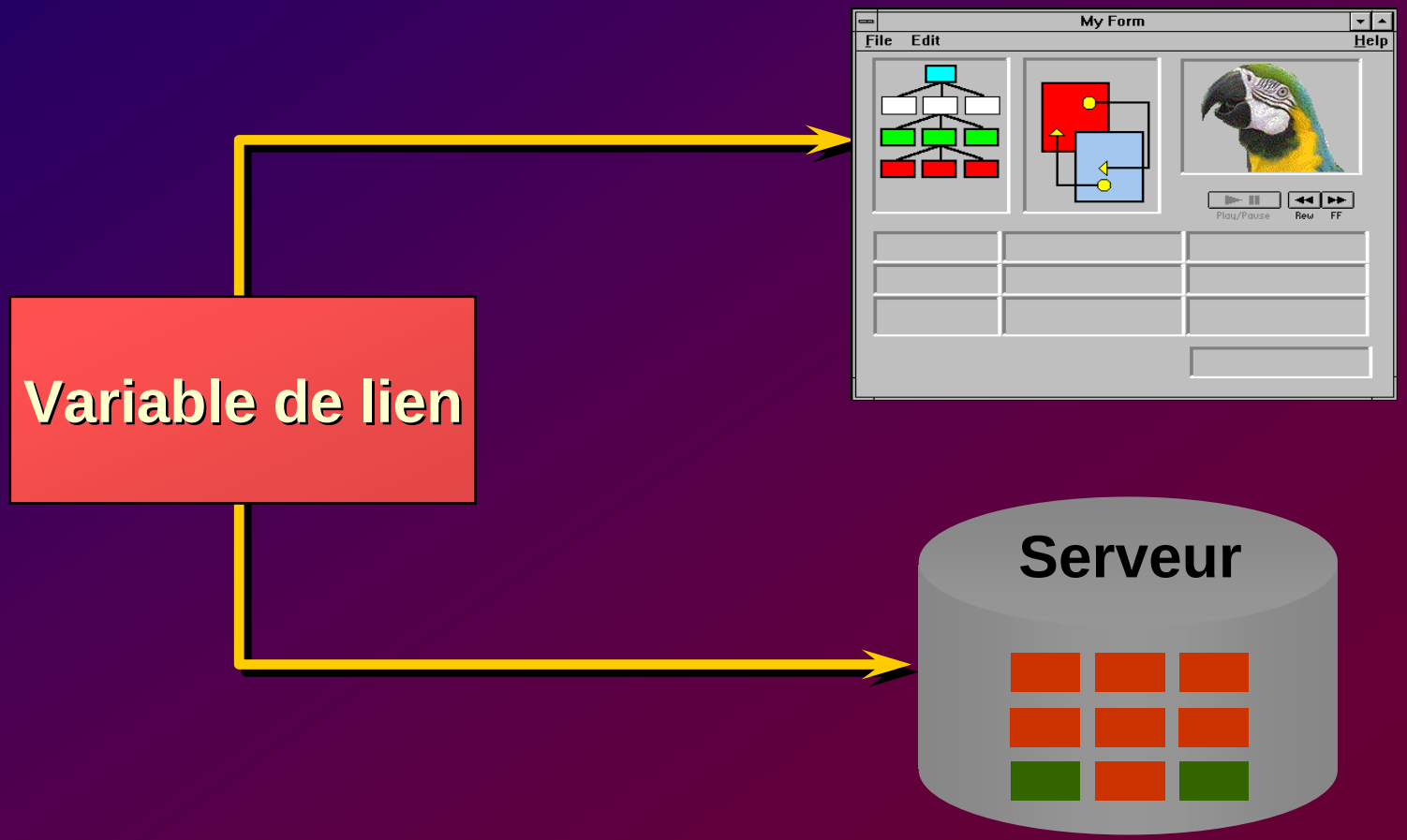
Types Structurés

- PL/SQL Table
- PL/SQL Enregistrement (RECORD)

Type LOB



Variables de lien



Référence à une variable de lien

Variable de lien ou Variable hôte

Préfixer le nom de variable par (:)

Exemple :

Ranger le salaire annuel dans une variable de lien :

```
:g_monthly_sal := v_sal / 12;
```

Visualisation des variables : DBMS_OUTPUT.PUT_LINE

- DBMS_OUTPUT : package fourni par Oracle
- Procédure PUT_LINE : affichage de la valeur d ’une variable.
- Utilisable sous SQL*PLUS avec l’option

  SET SERVEROUTPUT ON

```
DBMS_OUTPUT.PUT_LINE('Salaire mensuel : ' ||
TO_CHAR(v_sal,'99999.99'));
```

Instructions

BLOC PL/SQL Syntaxe

- Une instruction peut être écrite sur plusieurs lignes.
- Chaque instruction est terminée par (;)
- Identificateur :
  - Permet de référencer un élément PL/SQL.
  - Doit commencer par une lettre.
  - Maximum 30 caractères.

Syntaxe : Ligne Commentaire

- Une seule ligne : deux tirets (--) en début de ligne.
- Plusieurs lignes entre les symboles : /* et */.

Exemple

```
...
  v_sal NUMBER (9,2);
BEGIN
  /* calcul du salaire annuel à partir de données
fournies par l'utilisateur */
  v_sal := :p_monthly_sal * 12;
END; -- fin de la transaction
```

Fonctions SQL en PL/SQL

- Utilisables :
  - Fonction-ligne numérique
  - Fonction-ligne alphanumérique
  - Conversion de type
  - Date
- Non utilisables :
  - DECODE
  - Fonctions de groupe

Fonctions SQL en PL/SQL

Exemples

- Adresse complète d'une entreprise :

```
v_mailing_address := v_name||CHR(10)||
                     v_address||CHR(10)||v_state||
                     CHR(10)||v_zip;
```

- Mettre le nom d 'employé en lettres minuscules :

```
v_ename          := LOWER(v_ename);
```

Blocs Imbriqués (Nested Blocks)

Exemple

```
...
  x  BINARY_INTEGER;
BEGIN
  ...
  DECLARE
    y  NUMBER;
  BEGIN
    ...
  END;
  ...
END;
```

Bloc externe

Bloc imbriqué

Blocs Imbriqués

- Un bloc peut être inséré en lieu et place d'une instruction.
- Un bloc imbriqué correspond à une instruction.
- La section EXCEPTION peut contenir des blocs imbriqués.

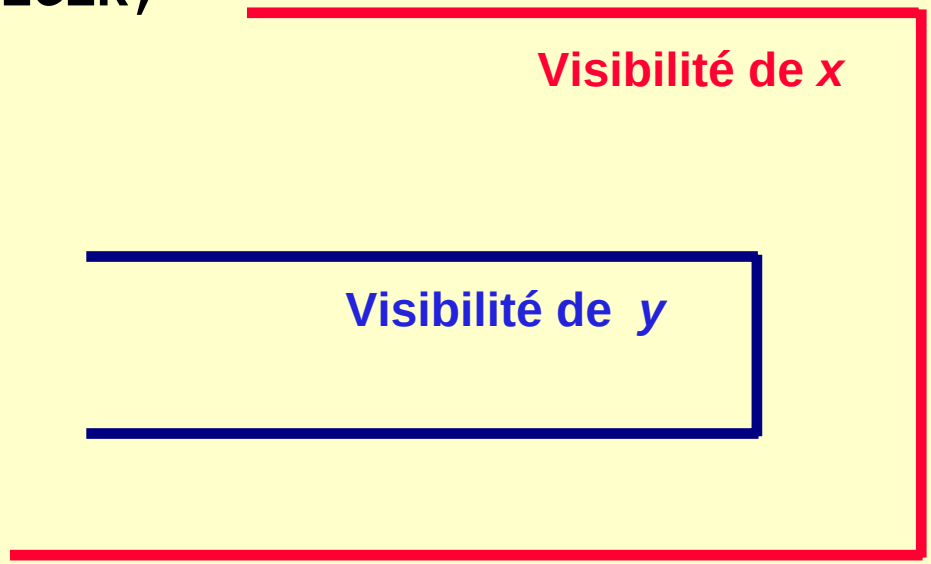
Blocs Imbriqués
Visibilité des variables

Un identificateur (variable, curseur) est visible dans tous les blocs imbriqués par rapport à celui où il est défini.

Blocs Imbriqués
Visibilité des variables

Exemple

```
...
  x  BINARY_INTEGER;
BEGIN
  ...
  DECLARE
    y  NUMBER;
  BEGIN
    ...
  END;
  ...
END;
```



Blocs Imbriqués
Visibilité des variables

```
...
DECLARE
V_SAL           NUMBER(7,2) := 60000;
V_COMM          NUMBER(7,2) := V_SAL * .20;
V_MESSAGE       VARCHAR2(255) := ' eligible for commission';
BEGIN ...
```

```
DECLARE
  V_SAL               NUMBER(7,2) := 50000;
  V_COMM              NUMBER(7,2) := 0;
  V_TOTAL_COMP        NUMBER(7,2) := V_SAL + V_COMM;
BEGIN ...
  V_MESSAGE := 'CLERK not'||V_MESSAGE;
END;
```

```
   V_MESSAGE := 'SALESMAN'||V_MESSAGE;
END;
```

Opérateurs en PL/SQL

- Logique
- Arithmétique
- Concaténation
- Parenthèses possibles
- Opérateur exponentiel (**)

Opérateurs en PL/SQL

Exemples

- Incrément de l'indice d'une boucle.

```
v_count      := v_count + 1;
```

- Initialisation de valeur pour un indicateur booléen.

```
v_equal      := (v_n1 = v_n2);
```

- Test de la valeur d'un numéro d'employé

```
v_valid      := (v_empno IS NOT NULL);
```

Accès aux données

Ordres SQL en PL/SQL

- Consultation par SELECT : une seule ligne peut être retournée.
- Modification des données par les ordres de manipulation INSERT, UPDATE DELETE.
- Contrôle des transactions par COMMIT, ROLLBACK, ou SAVEPOINT.
- Curseur implicite.

Ordre SELECT en PL/SQL

Consultation de la base de données.

Syntaxe

```
SELECT liste de projection
INTO   {nom variable[, nom variable]...
       | nom enregistrement}
FROM   table
WHERE  condition;
```

Ordre SELECT en PL/SQL

Utilisation de la clause : INTO

Exemple

```
DECLARE
  v_deptno  NUMBER(2);
  v_loc     VARCHAR2(15);
BEGIN
  SELECT    deptno, loc
  INTO      v_deptno, v_loc
  FROM      dept
  WHERE     dname = 'SALES';
...
END;
```

Ordre SELECT en PL/SQL

Exemple

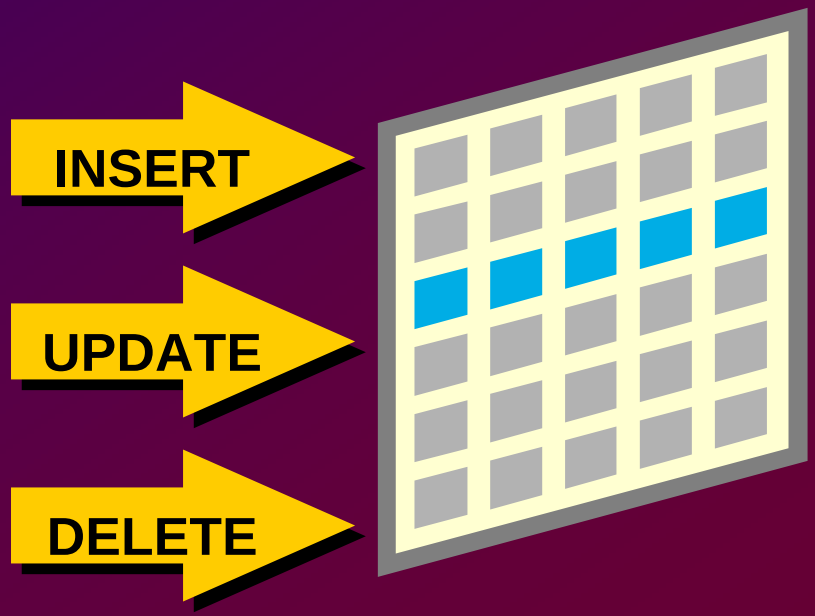
Montant total des salaires des employés d ’un département

```
DECLARE
  v_sum_sal   emp.sal%TYPE;
  v_deptno    NUMBER NOT NULL := 10;
BEGIN
  SELECT     SUM(sal)  -- fonction de groupe
  INTO       v_sum_sal
  FROM       emp
  WHERE      deptno = v_deptno;
END;
```

Mise à jour des données

Utilisation des ordres :

- **INSERT**
- **UPDATE**
- **DELETE**



Ajout de données

Exemple

Ajout d ’un nouvel employé dans la table EMP.

```
BEGIN
    INSERT INTO emp(empno, ename, job, deptno)
    VALUES(empno_sequence.NEXTVAL, 'HARDING',
          'CLERK', 10);
END;
```

Modification de données

Exemple

Modification de la valeur du salaire des employés 'ANALYST'.

```
DECLARE
  v_sal_increase   emp.sal%TYPE := 2000;
BEGIN
  UPDATE    emp
  SET       sal = sal + v_sal_increase
  WHERE     job = 'ANALYST';
END;
```

Suppression de données

Exemple

Supprimer les employés d'un département .

```
DECLARE
  v_deptno   emp.deptno%TYPE := 10;
BEGIN
  DELETE FROM   emp
  WHERE  deptno = v_deptno;
END;
```

Ordres COMMIT et ROLLBACK

- Début de transaction : premier ordre LMD.
- Fin de transaction explicite : COMMIT ou ROLLBACK.

Accès multilignes

Curseur SQL

- Zone de travail privée.
- Deux types de curseurs :
  - Implicite
  - Explicite (déclaré)
- Toute exécution d'ordre SQL utilise un curseur.
- Un code statut est positionné à la fin d 'exécution de l'ordre SQL.

Curseur IMPLICITE - Statut

Positionné à la fin d'exécution de l'ordre.

SQL%ROWCOUNT	Nombre de lignes traitées (entier)
SQL%FOUND	positionné à VRAI si l'ordre a traité une ou plusieurs lignes
SQL%NOTFOUND	positionné à VRAI si l'ordre n 'a traité de ligne

Curseur IMPLICITE - Statut

Exemple

Affichage du nombre de lignes supprimées.

```
VARIABLE rows_deleted VARCHAR2(30)
DECLARE
  v_ordid  NUMBER := 605;
BEGIN
  DELETE FROM  item
  WHERE  ordid = v_ordid;
  :rows_deleted := (SQL%ROWCOUNT ||
                        ' rows deleted.');
END;
/
PRINT rows_deleted
```

Structures de contrôle

Structures de contrôle

Deux structures :

- Alternative
- Répétitive.

Structures de contrôle

STRUCTURE ALTERNATIVE

Instruction IF

Trois formes :

- **IF-THEN-END IF**
- **IF-THEN-ELSE-END IF**
- **IF-THEN-ELSIF-END IF**

Instruction IF

Syntaxe

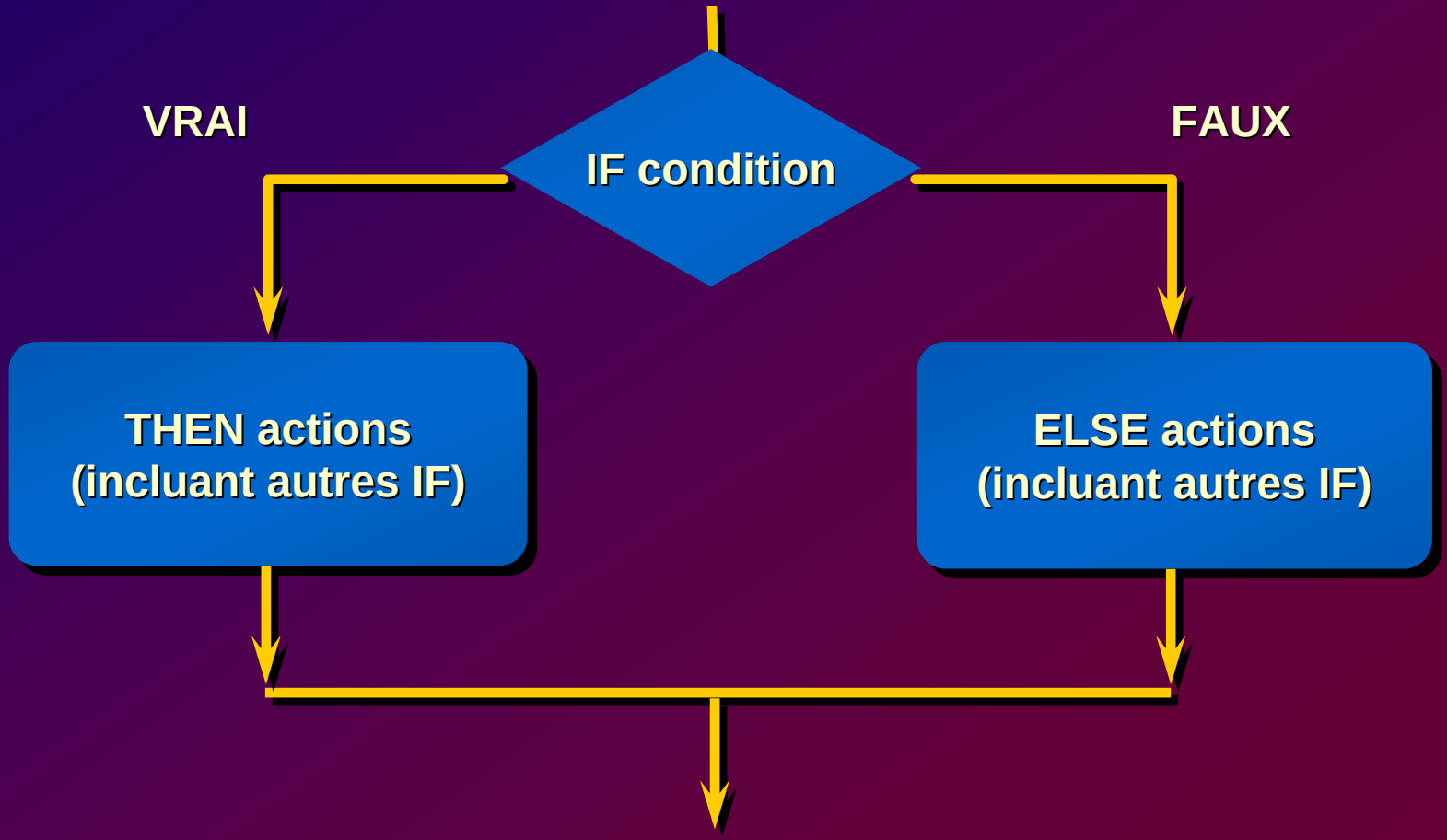
```
IF condition THEN
  instructions;
[ELSIF condition THEN
  instructions;]
[ELSE
  instructions;]
END IF;
```

Exemple :

N° manager = 22 si nom employé = Osborne.

```
IF v_ename = 'OSBORNE' THEN
  v_mgr := 22;
END IF;
```

Instruction IF-THEN-ELSE

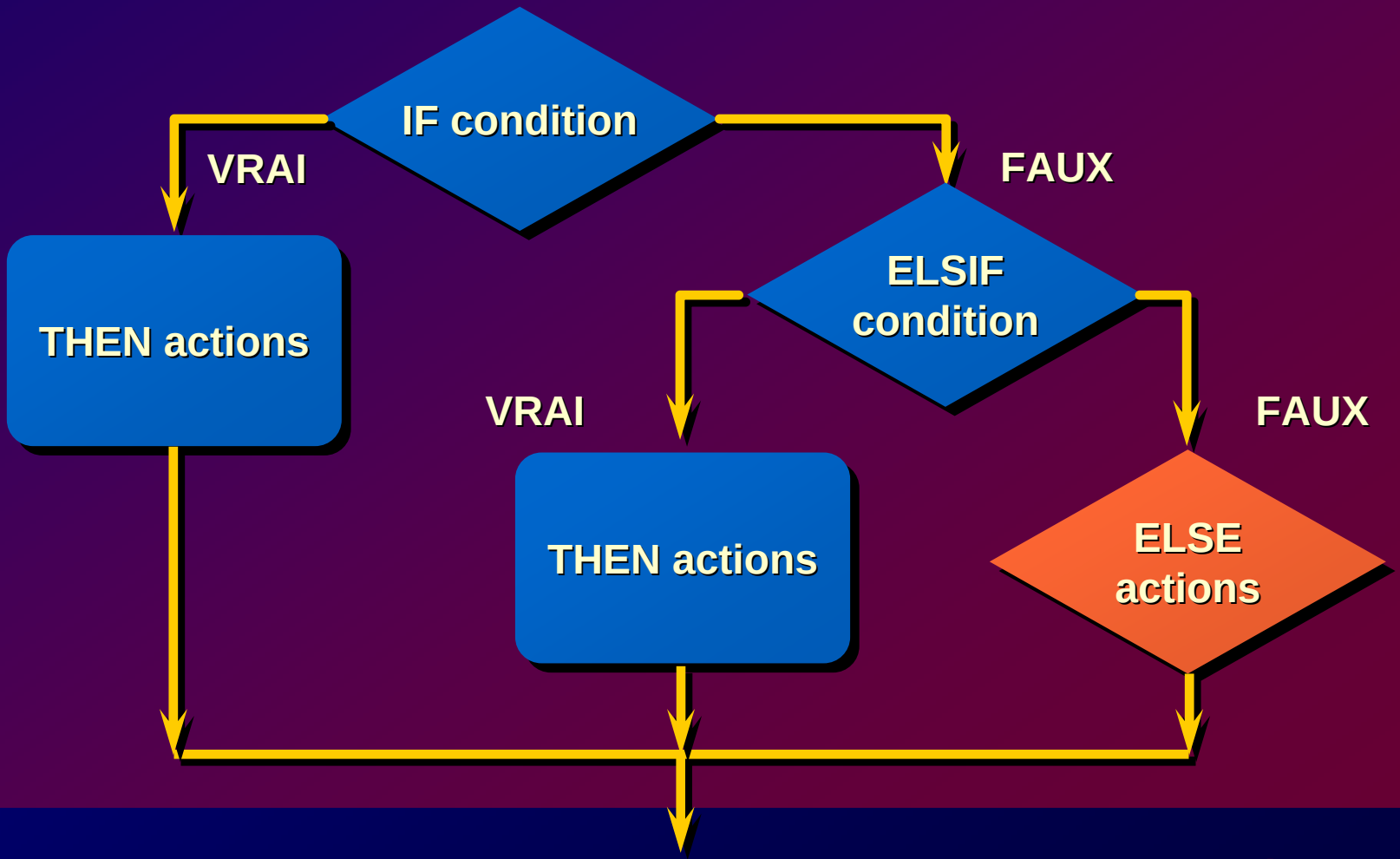


Instruction IF-THEN-ELSE

Exemple

```
...
IF v_shipdate - v_orderdate < 5 THEN
  v_ship_flag := 'Acceptable';
ELSE
  v_ship_flag := 'Unacceptable';
END IF;
...
```

Instruction IF-THEN-ELSIF



Instruction IF-THEN-ELSIF

Exemple

```
. . .
IF v_start > 100 THEN
  v_start := 2 * v_start;
ELSIF v_start >= 50 THEN
  v_start := .5 * v_start;
ELSE
  v_start := .1 * v_start;
END IF;
. . .
```

Structure répétitive

- Une boucle répète une *instruction* ou une *séquence d'instructions* plusieurs fois.
- Trois possibilités :
  - instruction LOOP
  - Instruction FOR
  - instruction WHILE

Instruction Loop

Syntaxe

```
LOOP                                    -- début de boucle
  instruction(s);                       -- instructions
  . . .
  EXIT [WHEN condition];                -- EXIT instruction
END LOOP;                               -- fin de boucle
```

Instruction Loop

Exemple

```
DECLARE
  v_ordid    item.ordid%TYPE := 601;
  v_counter  NUMBER(2) := 1;
BEGIN
  LOOP
    INSERT INTO item(ordid, itemid)
      VALUES(v_ordid, v_counter);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 10;
  END LOOP;
END;
```

Instruction FOR

Syntaxe

```
FOR indice in [REVERSE]
    borne-inférieure..borne-supérieure LOOP
  instruction 1;
  instruction 2;
  . . .
END LOOP;
```

- Le nombre de répétitions est contrôlé par l'indice.
- Ne pas déclarer l'indice, sa déclaration est implicite.

Instruction FOR

Règles :

- **L'indice n'est utilisable qu'à l'intérieur de la boucle.**
- **Il est interdit d'affecter une valeur à l'indice.**

Instruction FOR

Exemple

Création de 10 lignes pour la commande de n° 601.

```
DECLARE
  v_ordid   item.ordid%TYPE := 601;
BEGIN
  FOR i IN 1..10 LOOP
    INSERT INTO item(ordid, itemid)
      VALUES(v_ordid, i);
  END LOOP;
END;
```

Instruction WHILE

Syntaxe

```
WHILE condition LOOP
  instruction 1;
  instruction2;
  . . .
END LOOP;
```

La condition est évaluée en début de boucle.

Les instructions de la boucle sont répétées tant que la condition est vraie.

Instruction WHILE

Exemple

```
ACCEPT p_new_order PROMPT 'Enter the order number: '
ACCEPT p_items -
  PROMPT 'Enter the number of items in this order: '
DECLARE
v_count     NUMBER(2) := 1;
BEGIN
  WHILE v_count <= &p_items LOOP
    INSERT INTO item (ordid, itemid)
    VALUES (&p_new_order, v_count);
    v_count := v_count + 1;
  END LOOP;
  COMMIT;
END;
/
```

Structures imbriquées et étiquettes

- Plusieurs niveaux d'imbrication possibles.
- Utiliser des étiquettes pour différencier BLOC et Structure imbriquées.
- Possibilité de sortir d'une boucle interne par l 'ordre EXIT.

Structures imbriquées et étiquettes

```
...
BEGIN
  <<Boucle-externe>>
  LOOP
    v_counter := v_counter+1;
  EXIT WHEN v_counter>10;
    <<Boucle-interne>>
    LOOP
      ...
      EXIT Boucle-externe WHEN prédicat;
      -- Sortie des deux boucles
      EXIT WHEN prédicat;
      -- sortie de la boucle interne uniquement
      ...
    END LOOP boucle-interne;
    ...
  END LOOP Boucle-externe;
END;
```

Utilisation des Types Structurés : Enregistrement Tableau

Types Structurés

- Deux types:
  - Enregistrement (RECORD)
  - Tableau (TABLE PL/SQL)
- Contiennent des composants internes
- Sont réutilisables

Enregistrement PL/SQL

- Peut contenir un ou plusieurs composants de type : scalaire, RECORD ou TABLE PL/SQL.
- Identique à la structure enregistrement en L3G.
- Différent de la notion de ligne de table relationnelle.
- Considère un ensemble de champs comme une unité logique.
- Peut être utilisé pour recevoir une ligne de table.

Déclaration d'un type Enregistrement

Syntaxe

```
TYPE nom_type IS RECORD -- déclaration de type
(déclaration de champ[, déclaration de champ]…);
nom_variable nom_type;  -- déclaration de variable
```

Avec déclaration de champ :

```
Nom_champ {type_champ | variable%TYPE
    | table.colonne%TYPE | table%ROWTYPE}
    [[NOT NULL] {:= | DEFAULT} expression]
```

Déclaration d'un type Enregistrement

Exemple

Déclaration d 'une variable pour stocker nom, emploi, et salaire d'un employé.

```
...
  TYPE emp_record_type IS RECORD
    (ename        VARCHAR2(10),
     job          VARCHAR2(9),
     sal          NUMBER(7,2));
  emp_record      emp_record_type;
...
```

Utilisation de %ROWTYPE

- Permet de déclarer une variable de même structure qu'une ligne de table ou de vue.
- Nom_table%ROWTYPE.
- Les champs de l'enregistrement ont même nom et même type que ceux des colonnes de la table ou de la vue.

Utilisation de %ROWTYPE

Exemples

Déclarer une variable pour stocker la même information que celle définie dans la table DEPT.

```
dept_record    dept%ROWTYPE;
```

Déclare une variable pour stocker la même information que celle définie dans la table EMP.

```
emp_record     emp%ROWTYPE;
```

Avantage de %ROWTYPE

- Il n ’est pas nécessaire de connaître les caractéristiques des colonnes de la ligne de référence .
- Mise à jour automatique en cas de modification de la structure de la ligne de référence.
- Utilisable avec SELECT pour recueillir les données d’une ligne.

Tables PL/SQL

- Composé de postes identiques de type :
  - Scalaire
  - Enregistrement
- Référence à un poste par clé primaire (PRIMARY KEY) de type BINARY_INTEGER

Déclaration d'un type Table

Syntaxe - poste de type scalaire

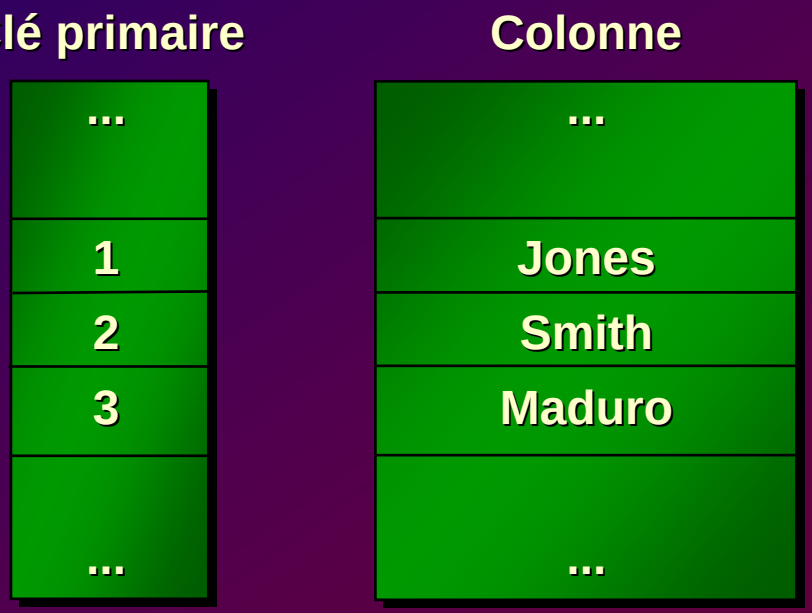
```
TYPE nom_type IS TABLE OF
  {type_colonne | variable%TYPE
  | table.colonne%TYPE} [NOT NULL]
  INDEX BY BINARY_INTEGER;
nom_variable  nom_type;
```

Exemple

Déclarer une table de noms.

```
...
TYPE nom_table_type IS TABLE OF emp.ename%TYPE
  INDEX BY BINARY_INTEGER;
table_nom  nom_table_type;
...
```

Structure Table PL/SQL



Création d'une Table PL/SQL

```
DECLARE
  TYPE ename_table_type IS TABLE OF emp.ename%TYPE
    INDEX BY BINARY_INTEGER;
  TYPE hiredate_table_type IS TABLE OF DATE
    INDEX BY BINARY_INTEGER;
  ename_table     ename_table_type;
  hiredate_table  hiredate_table_type;
BEGIN
  ename_table(1) := 'CAMERON';
  hiredate_table(8) := SYSDATE + 7;
    IF ename_table.EXISTS(1) THEN
      INSERT INTO ...
    ...
END;
```

Méthodes PL/SQL associées à la structure Table

Méthodes fournies en standard :

- EXISTS
- COUNT
- FIRST, LAST
- PRIOR
- NEXT
- EXTEND
- TRIM
- DELETE

Table d'enregistrements

Syntaxe - poste de type enregistrement

- Utilisation de %ROWTYPE.

Exemple

- Déclarer une variable pour recevoir les données de la table DEPT.

```
DECLARE
  TYPE dept_table_type IS TABLE OF dept%ROWTYPE
    INDEX BY BINARY_INTEGER;
  dept_table dept_table_type;
```

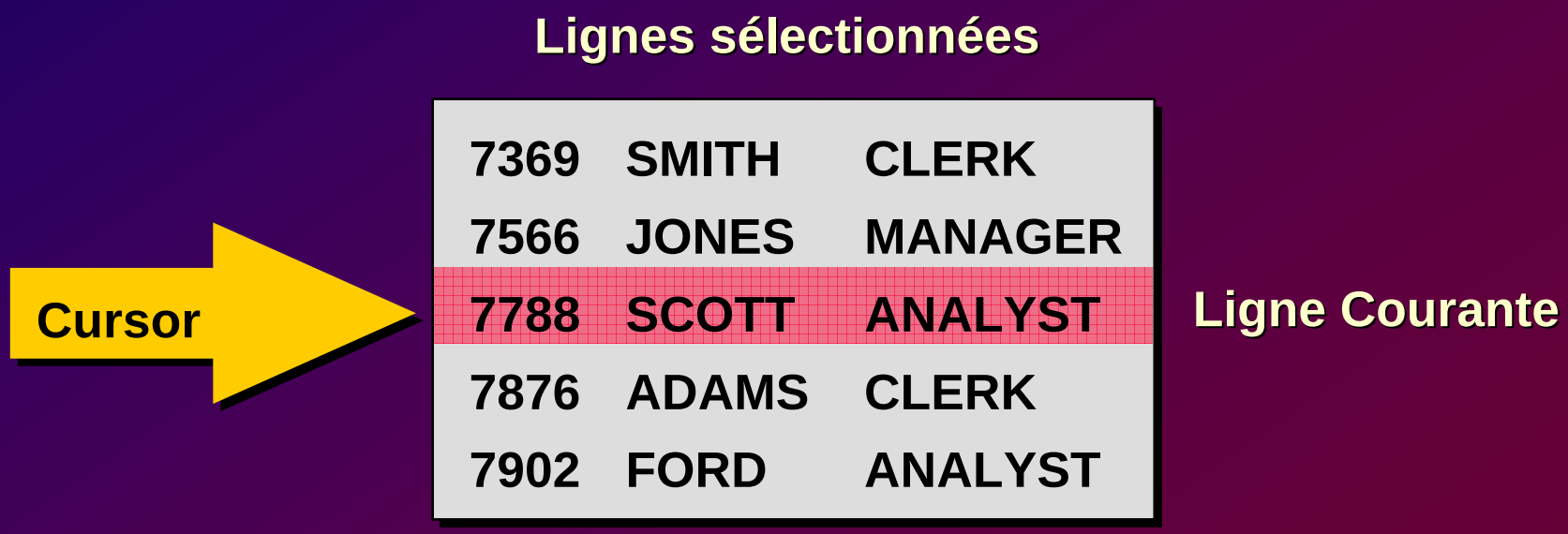
CURSEUR Explicite

Curseur

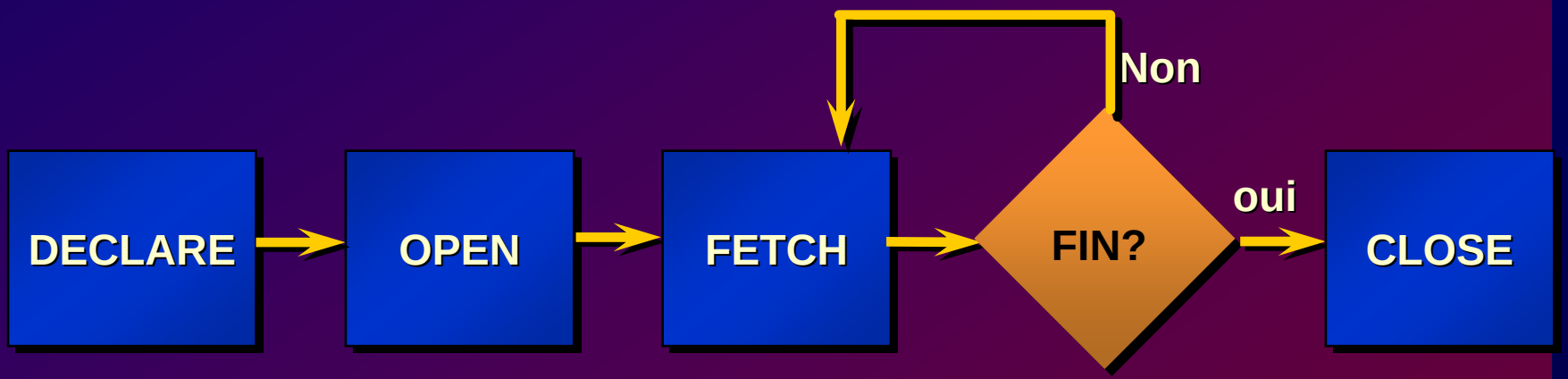
Tout ordre SQL utilise un curseur pour s'exécuter :

- curseur implicite
  - tout ordre LMD (DML)
  - SELECT ... INTO ... sous PL/SQL
- curseur explicite
  - déclaré dans un module

Structure (simplifiée) du curseur

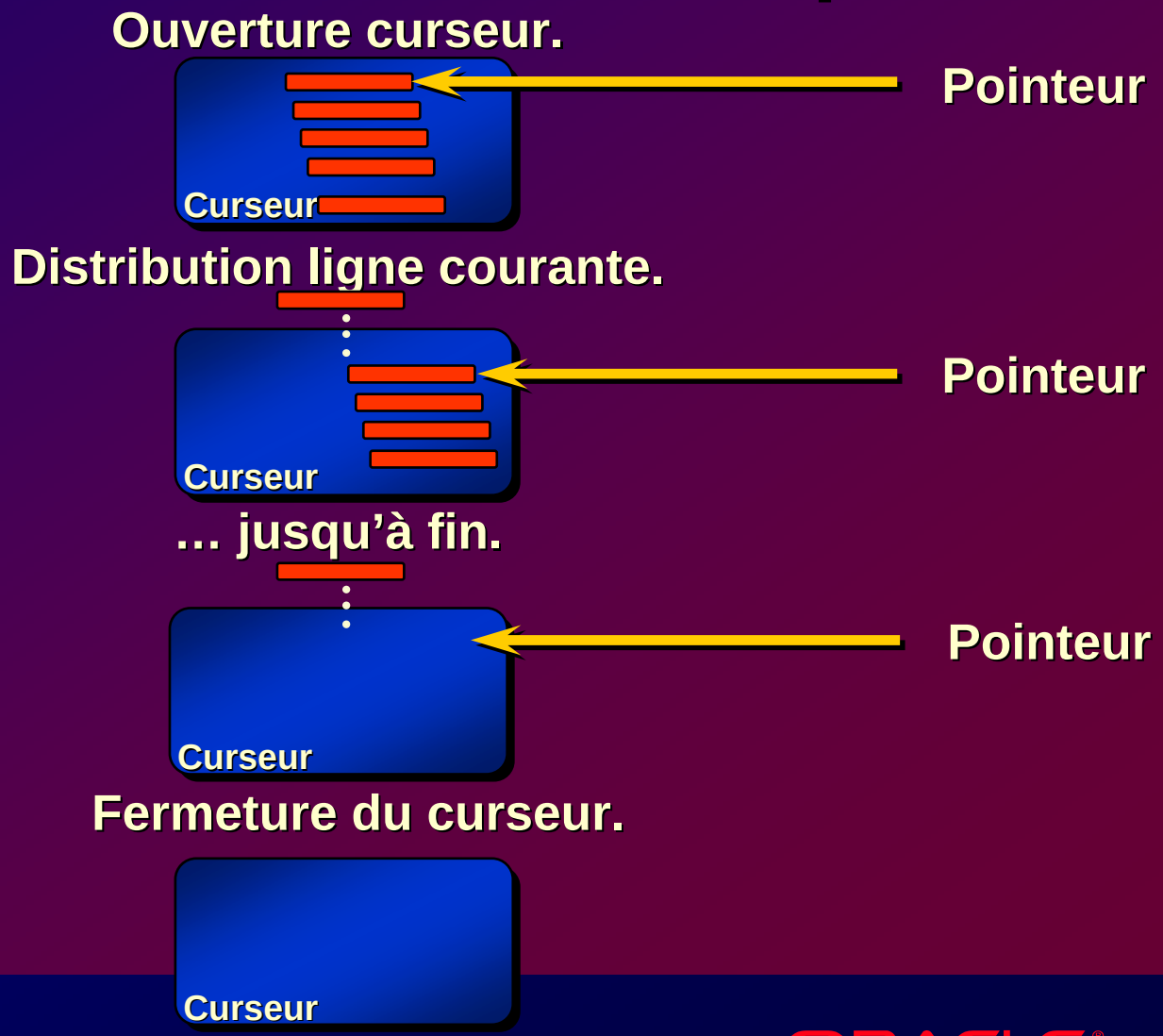


Mise en œuvre curseur explicite



- Déclaration requête SQL
- Ouverture et exécution
- Distribution ligne courante
- Teste existence ligne
- Libération du curseur

Mise en œuvre curseur explicite



Déclaration du curseur

Syntaxe

```
CURSOR nom_curseur IS
     requête;
```

- Requête sans clause INTO.
- Possibilité de clause ORDER BY.

Déclaration du curseur

Exemple

```
DECLARE
  CURSOR emp_cursor IS
    SELECT empno, ename
    FROM   emp;

BEGIN
  ...
```

Ouverture du curseur

Syntaxe

```
OPEN  nom_curseur;
```

- Exécution de la requête et génération des lignes résultats au niveau du serveur.
- Pas d'erreur si la requête ne sélectionne pas de ligne.
- Possibilité de tester le statut du curseur après exécution de l'ordre FETCH.

Distribution des lignes

Syntaxe

```
FETCH nom_curseur INTO [variable1, variable2, ...]
                       | [nom_enregistrement];
```

- Distribue les valeurs des colonnes de la ligne courante dans les variables de réception.
- Effectue une correspondance par position.
- Renvoie un code statut.

Mise en œuvre de l ’ordre FETCH

- Inclure l ’ordre FETCH dans une structure répétitive.
- Une ligne est distribuée à chaque itération.
- Utiliser %NOTFOUND ou %FOUND pour contrôler la sortie de la boucle.

Distribution des lignes

Exemples

```
FETCH emp_cursor INTO v_empno, v_ename;
```

```
...
OPEN nom_curseur;
LOOP
  FETCH nom_curseur INTO variables
  EXIT WHEN nom_curseur%NOTFOUND...;
  ...
    -- utilisation des valeurs distribuées à
       chaque itération
  ...
END LOOP;
```

Fermeture du curseur

Syntaxe

```
CLOSE     nom_curseur;
```

- Ferme le curseur et libère les ressources.
- Possibilité de ré-ouvrir le même curseur.

Codes statut d'un curseur

Informent sur l'état du curseur.

Code Mnémonique	Type	Description
%ISOPEN	Booléen	VRAI si le curseur est ouvert
%NOTFOUND	Booléen	VRAI si le dernier ordre fetch exécuté n'a pas distribué de ligne
%FOUND	Booléen	VRAI si le dernier ordre fetch exécuté a distribué une ligne - complément de %NOTFOUND
%ROWCOUNT	Nombre	Nombre de lignes distribuées

%ISOPEN

- La distribution de ligne ne s'effectue que pour un curseur ouvert.
- Permet de savoir si un curseur est ouvert avant d'exécuter un ordre fetch.

Exemple

```
IF NOT emp_cursor%ISOPEN THEN
    OPEN emp_cursor;
END IF;
LOOP
  FETCH emp_cursor...
```

NOTFOUND
ROWCOUNT et FOUND

- %ROWCOUNT donne, après chaque exécution de l'ordre fetch, le nombre de lignes distribuées.
- %NOTFOUND indique la fin de distribution des lignes d'un curseur.
- %FOUND testé après exécution du premier ordre fetch indique si la requête a sélectionné au moins une ligne.

Curseur et Enregistrement

Distribution des données de la ligne dans une structure RECORD.

Exemple

```
DECLARE
  CURSOR emp_cursor IS
    SELECT  empno, ename
    FROM    emp;
  emp_record   emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO emp_record;
  ...
```

Curseur et Enregistrement - utilisation de For -

Syntaxe

```
FOR nom_enregistrement IN nom_curseur LOOP
  instruction1; instruction2;
  . . .
END LOOP;
```

- Raccourci pour gérer la distribution des lignes.
- Exécute toutes les étapes (open, fetch, close).
- Déclaration implicite de l'enregistrement.

Curseur et Enregistrement - utilisation de For -

Exemple

```
DECLARE
  CURSOR emp_cursor IS
    SELECT ename, deptno
    FROM   emp;
BEGIN
  FOR emp_record IN emp_cursor LOOP
         -- open et fetch implicites
    IF emp_record.deptno = 30 THEN
       ...
  END LOOP; -- close implicite
END;
```

Curseur paramétré

Curseur avec paramètre(s)

Syntaxe

```
CURSOR nom_curseur
  [(mon_paramètre type, ...)]
IS
  requête;
```

- Affectation des valeurs des paramètres lors de l'ouverture du curseur.
- Le même curseur peut être ouvert plusieurs fois avec des valeurs de paramètres différentes.

Curseur avec paramètre(s)

Exemple

Donner le n° département et l'emploi sous forme de paramètres pour la clause WHERE.

```
DECLARE
  CURSOR emp_cursor
  (v_deptno NUMBER, v_job VARCHAR2) IS
    SELECT    empno, ename
    FROM      emp
    WHERE     deptno = v_deptno
     AND      job = v_job;
BEGIN
  OPEN emp_cursor(10, 'CLERK');
...
```

Mise à jour avec utilisation d'un curseur

Clause FOR UPDATE

Syntaxe

```
SELECT ...
FROM          ...
FOR UPDATE [OF nom_colonne][NOWAIT];
```

- Verrouille les lignes sélectionnées pour la durée de la transaction.
- Verrouille les lignes avant l'exécution d 'un ordre update ou delete.

Clause FOR UPDATE

Exemple

Sélectionner les employés du département 30.

```
DECLARE
  CURSOR emp_cursor IS
    SELECT empno, ename, sal
    FROM   emp
    WHERE  deptno = 30
    FOR UPDATE NOWAIT;
```

Clause WHERE CURRENT OF

Syntaxe

```
WHERE CURRENT OF nom_curseur;
```

- Curseur en vue de modifier ou supprimer les lignes sélectionnées.
- Utiliser la clause FOR UPDATE dans l'expression du curseur.
- Utiliser la clause WHERE CURRENT OF pour faire référence à la dernière ligne distribuée par le curseur.

Clause WHERE CURRENT OF

Exemple

```
DECLARE
  CURSOR sal_cursor IS
    SELECT    sal
    FROM      emp
    WHERE     deptno = 30
    FOR UPDATE of sal NOWAIT;
BEGIN
  FOR emp_record IN sal_cursor LOOP
    UPDATE    emp
    SET       sal = emp_record.sal * 1.10
    WHERE CURRENT OF sal_cursor;
  END LOOP;
  COMMIT;
END;
```

Gestion des exceptions

Gestion des exceptions en PL/SQL

- Exception ?
  - Tout événement qui survient pendant l'exécution d 'un ordre.
- Différents cas
  - Erreur diagnostiquée par le SGBDR.
  - Événement généré par le développeur.
- Gestions
  - Capture dans le module qui l'a détectée.
  - Propagation à l'environnement.

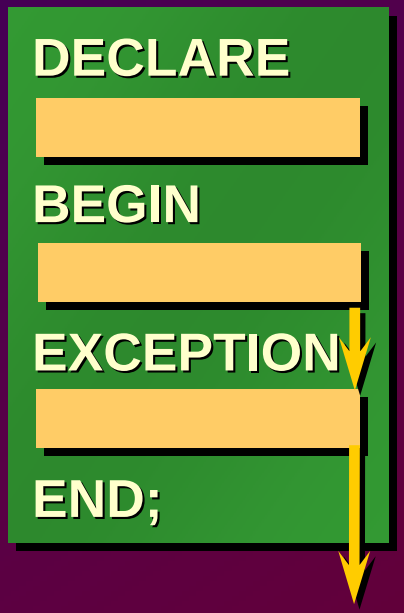
Gestion des exceptions en PL/SQL

Capture de l'exception

Création de l'exception

Capture de l'exception

Propagation de l'exception



Création de l'exception

L'exception n'est pas capturée

Exception propagée à l'environnement

Types d'Exceptions

- Erreur émise par le serveur
  - Prédéfinies
  - Non prédéfinies
- Exception générée par l'utilisateur

Capture des Exceptions

Syntaxe

```
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
    instruction1;
    instruction2;
    . . .
  [WHEN exception3 [OR exception4 . . .] THEN
    instruction1;
    instruction2;
    . . .]
  [WHEN OTHERS THEN
    instruction1;
    instruction2;
    . . .]
```

Capture des Exceptions

- WHEN OTHERS est la dernière clause.
- Le mot clé EXCEPTION introduit la section de gestion des exceptions.
- Plusieurs gestionnaires d'exception peuvent être définis dans un même bloc.
- Un seul gestionnaire d'exception est exécutée suite à la détection d 'une exception, avant de sortir du bloc.

Exceptions serveur prédéfinies

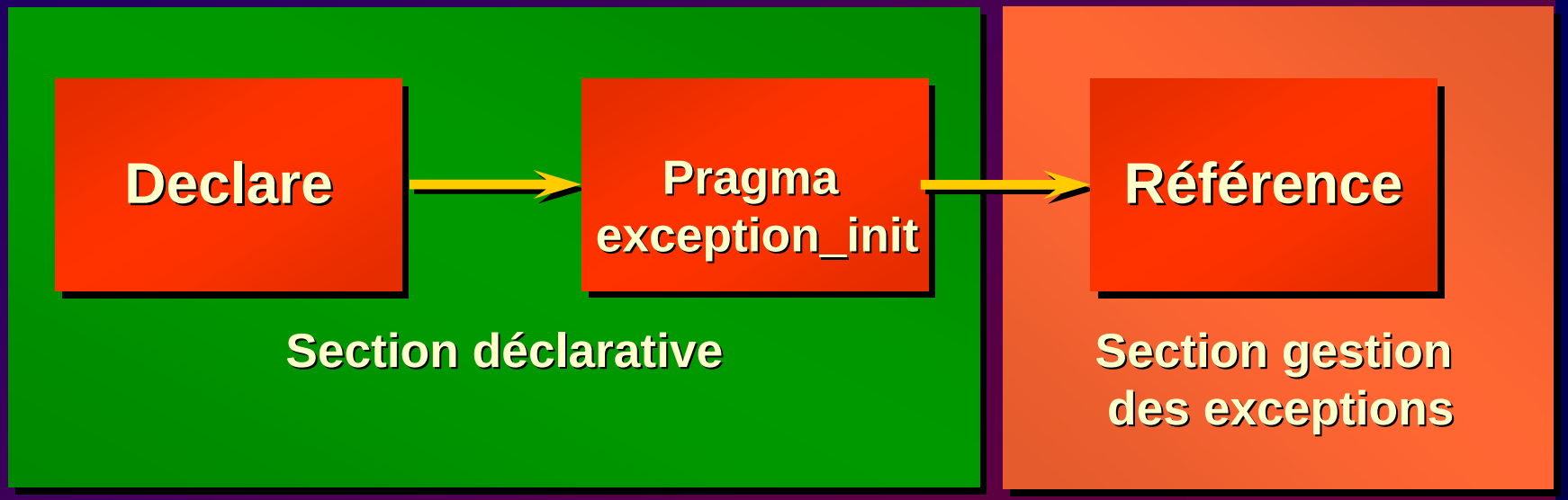
- Erreur émise par le serveur.
- Repérable par un nom-erreur.
- Exemple de nom-erreurs prédéfinies:
  - NO_DATA_FOUND
  - TOO_MANY_ROWS
  - INVALID_CURSOR
  - ZERO_DIVIDE
  - DUP_VAL_ON_INDEX

Utilisation des nom-erreurs

Syntaxe

```
BEGIN  SELECT ... COMMIT;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    instruction1;
    instruction2;
  WHEN TOO_MANY_ROWS THEN
    instruction1;
  WHEN OTHERS THEN
    instruction1;
    instruction2;
    instruction3;
END;
```

Exception serveur non prédéfinie



- Déclaration d'un nom d'exception
- Association à l'erreur
- Capture de l'exception

Exception serveur non prédéfinie

Déclaration d'un nom-erreur pour l'erreur n° -2292 (intégrité référentielle).

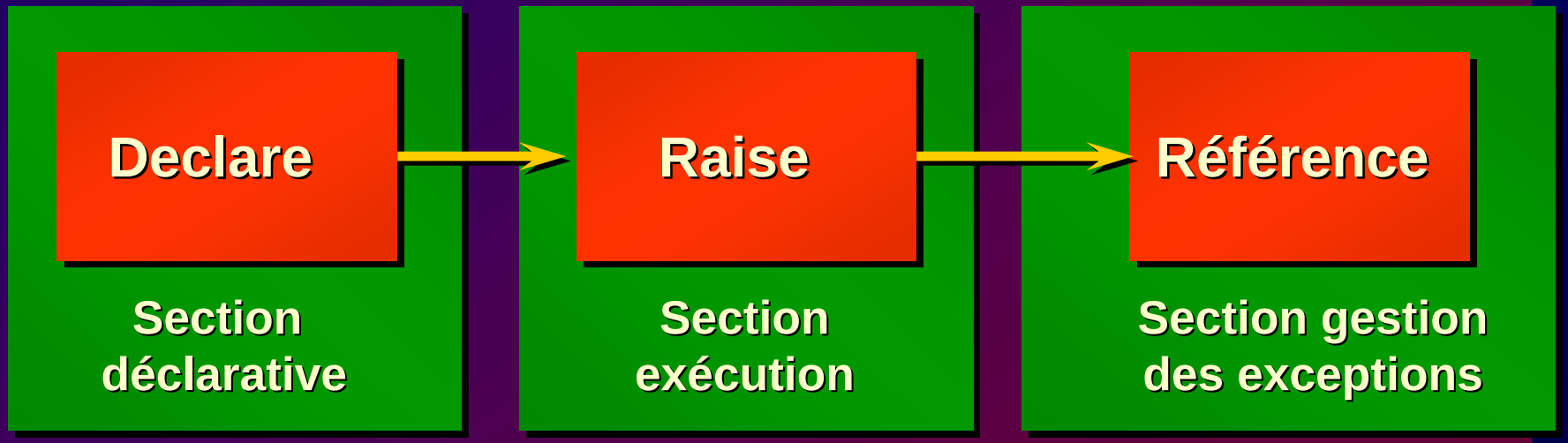
```
DECLARE
  e_emps_remaining    EXCEPTION;
  PRAGMA EXCEPTION_INIT (
              e_emps_remaining, -2292);
  v_deptno     dept.deptno%TYPE := &p_deptno;
BEGIN
  DELETE FROM dept
  WHERE        deptno = v_deptno;
  COMMIT;
EXCEPTION
  WHEN e_emps_remaining THEN
   DBMS_OUTPUT.PUT_LINE ( 'Suppression impossible
' ||TO_CHAR(v_deptno) || '.  Existence d'employés.
');
END;
```

1

2

3

Exception définie par l'utilisateur



- Déclaration d'un nom d'exception
- Génération de l'exception
- Traitement de l'exception

Exception définie par l'utilisateur

Syntaxe

```
DECLARE
  nom-exception EXCEPTION;
BEGIN
  …;
       RAISE nom-exception;
  ...;

EXCEPTION
  WHEN nom-exception  THEN
    ...;
END;
```

Exception définie par l'utilisateur

Exemple

```
DECLARE
  e_invalid_product  EXCEPTION;                                  1
BEGIN
  UPDATE      product
  SET         descrip = '&product_description'
  WHERE       prodid = &product_number;
  IF SQL%NOTFOUND THEN
    RAISE e_invalid_product;                                     2
  END IF;
  COMMIT;
EXCEPTION
  WHEN  e_invalid_product  THEN                                  3
    DBMS_OUTPUT.PUT_LINE(' N° produit inconnu.');
END;
```

Procédure RAISE_APPLICATION_ERROR

Syntaxe

```
raise_application_error (numéro-erreur,
                         message[, {TRUE | FALSE}]);
```

- Permet de définir une erreur (numéro [entre -20000 et -20999] et texte du message) dans un bloc PL/SQL.
- Utilisable dans les sections de code d'un bloc PL/SQL.

Procédure RAISE_APPLICATION_ERROR

- Utilisable :
  - dans la section Exécution
  - dans la section Exception
- La génération de l'erreur est conforme au standard du serveur et est traitable comme telle.

Procédure RAISE_APPLICATION_ERROR

Exemple

```
…
EXCEPTION
    WHEN NO_DATA_FOUND THEN
       RAISE_APPLICATION_ERROR (-20201,
              ‘ Ligne NON trouvée ’);

END;
```

Informations associées à toute erreur

- **SQLCODE**

  Valeur numérique de l'erreur

- **SQLERRM**

  Texte du message associé à l'erreur

Informations associées à une exception serveur

```
DECLARE
  v_error_code      NUMBER;
  v_error_message   VARCHAR2(255);
BEGIN
...
EXCEPTION
...
  WHEN OTHERS THEN
    ROLLBACK;
    v_error_code := SQLCODE ;
    v_error_message := SQLERRM ;
    INSERT INTO errors VALUES(v_error_code,
                              v_error_message);
END;
```

Propagation d'une exception

Un bloc peut gérer ses exceptions ou les transmettre au bloc de niveau supérieur.

```
DECLARE
  . . .
  e_no_rows       exception;
  e_integrity     exception;
  PRAGMA EXCEPTION_INIT (e_integrity, -2292);
BEGIN
  FOR c_record IN emp_cursor LOOP
    BEGIN
      SELECT ...
      UPDATE ...
      IF SQL%NOTFOUND THEN
        RAISE e_no_rows;
      END IF;
    EXCEPTION
      WHEN e_integrity THEN ...
      WHEN e_no_rows THEN ...
    END;
END LOOP;
EXCEPTION
  WHEN NO_DATA_FOUND THEN . . .
  WHEN TOO_MANY_ROWS THEN . . .
END;
```